

# The gcodepreview PythonSCAD library\*

Author: William F. Adams  
willadams at aol dot com

2025/02/14

### Abstract

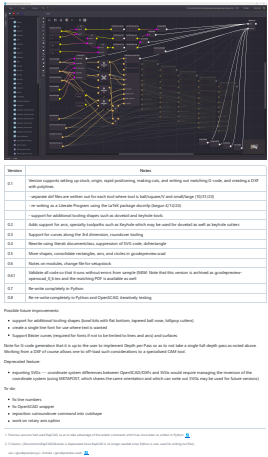
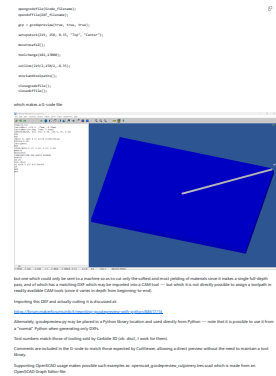
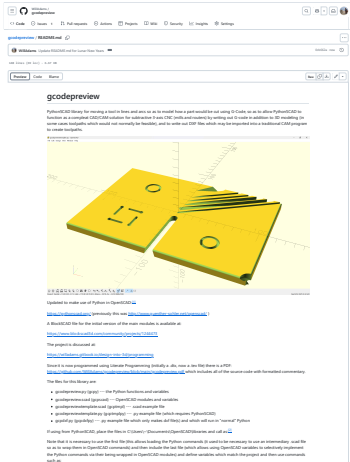
The gcodepreview library allows using PythonSCAD (OpenPythonSCAD) to move a tool in lines and arcs and output DXF and G-code files so as to work as a CAD/CAM program for CNC.

## Contents

<b>1</b>	<b>readme.md</b>	<b>2</b>
<b>2</b>	<b>Usage and Templates</b>	<b>6</b>
2.1	gcpdxf.py . . . . .	6
2.2	gcodepreviewtemplate.py . . . . .	8
2.3	gcodepreviewtemplate.scad . . . . .	14
<b>3</b>	<b>gcodepreview</b>	<b>18</b>
3.1	Module Naming Convention . . . . .	19
3.1.1	Parameters and Default Values . . . . .	21
3.2	Implementation files and gcodepreview class . . . . .	21
3.2.1	Position and Variables . . . . .	24
3.2.2	Initial Modules . . . . .	25
3.3	Tools and Changes . . . . .	28
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 . . . . .	32
3.3.2.2.2	Thread mills . . . . .	33
3.3.2.2.3	Dovetails . . . . .	33
3.3.2.3	Concave toolshapes . . . . .	33
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) . . . . .	34
3.3.3.3	Roundover (corner rounding) . . . . .	34
3.3.3.4	Dovetails . . . . .	34
3.3.3.5	toolchange routine . . . . .	34
3.3.4	tooldiameter . . . . .	36
3.3.5	Feeds and Speeds . . . . .	38
3.4	Movement and Cutting . . . . .	38
3.4.1	Lines . . . . .	40
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 . . . . .	49
3.4.4	Difference of Stock, Rapids, and Toolpaths . . . . .	58
3.5	Output files . . . . .	59
3.5.1	G-code Overview . . . . .	59
3.5.2	DXF Overview . . . . .	60
3.5.3	Python and OpenSCAD File Handling . . . . .	61
3.5.3.1	Writing to DXF files . . . . .	64
3.5.3.2	Closings . . . . .	69
<b>4</b>	<b>Notes</b>	<b>73</b>
	<b>Index</b>	<b>77</b>
	Routines . . . . .	78
	Variables . . . . .	79

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

114 rdme
115 rdme Deprecated feature:
116 rdme
117 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)
118 rdme
119 rdme To-do:
120 rdme
121 rdme - add conditional option to toggle between creation of manual and
        Literate Source
122 rdme - fix OpenSCAD wrapper and add any missing commands for Python
123 rdme - reposition cutroundover command into cutshape
124 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)
125 rdme - work on rotary axis option
```

---



```

7 gcpdxftp          True    # generatedxf
8 gcpdxftp          )
9 gcpdxftp
10 gcpdxftp # [Stock] */
11 gcpdxftp stockXwidth = 100
12 gcpdxftp # [Stock] */
13 gcpdxftp stockYheight = 50
14 gcpdxftp
15 gcpdxftp # [Export] */
16 gcpdxftp Base_filename = "dxfexport"
17 gcpdxftp
18 gcpdxftp
19 gcpdxftp # [CAM] */
20 gcpdxftp large_square_tool_num = 102
21 gcpdxftp # [CAM] */
22 gcpdxftp small_square_tool_num = 0
23 gcpdxftp # [CAM] */
24 gcpdxftp large_ball_tool_num = 0
25 gcpdxftp # [CAM] */
26 gcpdxftp small_ball_tool_num = 0
27 gcpdxftp # [CAM] */
28 gcpdxftp large_V_tool_num = 0
29 gcpdxftp # [CAM] */
30 gcpdxftp small_V_tool_num = 0
31 gcpdxftp # [CAM] */
32 gcpdxftp DT_tool_num = 374
33 gcpdxftp # [CAM] */
34 gcpdxftp KH_tool_num = 0
35 gcpdxftp # [CAM] */
36 gcpdxftp Roundover_tool_num = 0
37 gcpdxftp # [CAM] */
38 gcpdxftp MISC_tool_num = 0
39 gcpdxftp
40 gcpdxftp # [Design] */
41 gcpdxftp inset = 3
42 gcpdxftp # [Design] */
43 gcpdxftp radius = 6
44 gcpdxftp # [Design] */
45 gcpdxftp cornerstyle = "Fillet" # "Chamfer", "Flipped Fillet"
46 gcpdxftp
47 gcpdxftp gcp.opendxfile(Base_filename)
48 gcpdxftp #gcp.opendxfiles(Base_filename,
49 gcpdxftp #          large_square_tool_num,
50 gcpdxftp #          small_square_tool_num,
51 gcpdxftp #          large_ball_tool_num,
52 gcpdxftp #          small_ball_tool_num,
53 gcpdxftp #          large_V_tool_num,
54 gcpdxftp #          small_V_tool_num,
55 gcpdxftp #          DT_tool_num,
56 gcpdxftp #          KH_tool_num,
57 gcpdxftp #          Roundover_tool_num,
58 gcpdxftp #          MISC_tool_num)
59 gcpdxftp
60 gcpdxftp gcp.dxfrectangle(large_square_tool_num, 0, 0, stockXwidth,
61 gcpdxftp stockYheight)
62 gcpdxftp gcp.dxfarc(large_square_tool_num, inset, inset, radius, 0, 90)
63 gcpdxftp gcp.dxfarc(large_square_tool_num, stockXwidth - inset, inset,
64 gcpdxftp radius, 90, 180)
65 gcpdxftp gcp.dxfarc(large_square_tool_num, stockXwidth - inset, stockYheight
66 gcpdxftp - inset, radius, 180, 270)
67 gcpdxftp gcp.dxfarc(large_square_tool_num, inset, stockYheight - inset,
68 gcpdxftp radius, 270, 360)
69 gcpdxftp
70 gcpdxftp gcp.dxfline(large_square_tool_num, inset, inset + radius, inset,
71 gcpdxftp stockYheight - (inset + radius))
72 gcpdxftp gcp.dxfline(large_square_tool_num, inset + radius, inset,
73 gcpdxftp stockXwidth - (inset + radius), inset)
74 gcpdxftp gcp.dxfline(large_square_tool_num, stockXwidth - inset, inset +
75 gcpdxftp radius, stockXwidth - inset, stockYheight - (inset + radius))
76 gcpdxftp gcp.dxfline(large_square_tool_num, inset + radius, stockYheight -
77 gcpdxftp inset, stockXwidth - (inset + radius), stockYheight - inset)
78 gcpdxftp
79 gcpdxftp gcp.dxfrectangle(large_square_tool_num, radius +inset, radius,
80 gcpdxftp stockXwidth/2 - (radius * 4), stockYheight - (radius * 2),
81 gcpdxftp cornerstyle, radius)
82 gcpdxftp gcp.dxfrectangle(large_square_tool_num, stockXwidth/2 + (radius *
83 gcpdxftp 2) + inset, radius, stockXwidth/2 - (radius * 4), stockYheight -

```

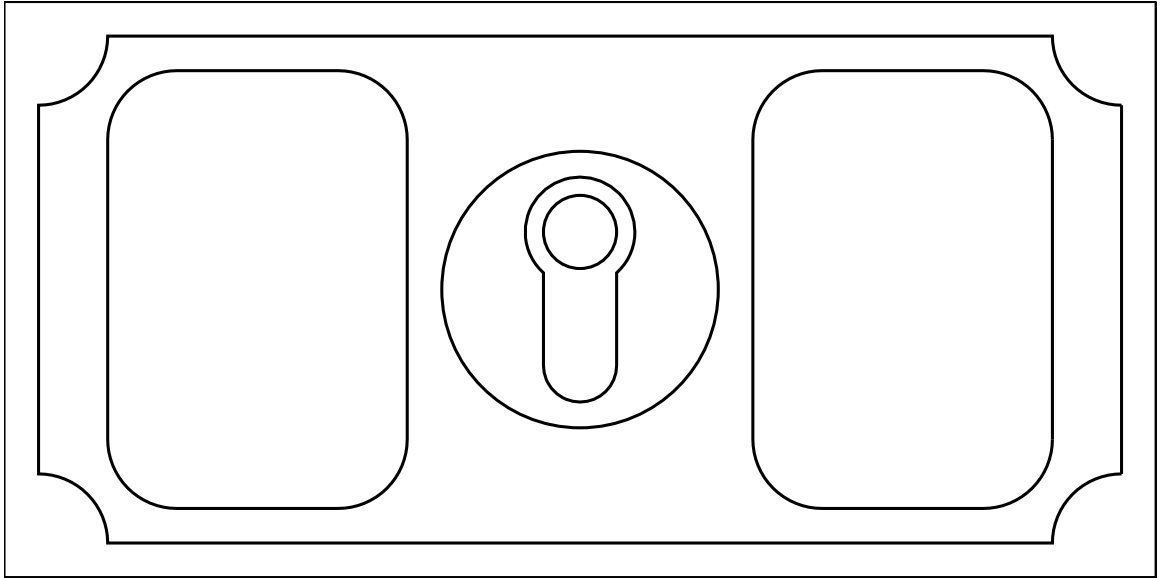
```

        (radius * 2), cornerstyle, radius)
74 gcpdxfpy #gcp.dxfrectangleround(large_square_tool_num, 64, 7, 24, 36, radius
    )
75 gcpdxfpy #gcp.dxfrectanglechamfer(large_square_tool_num, 64, 7, 24, 36,
    radius)
76 gcpdxfpy #gcp.dxfrectangleflippedfillet(large_square_tool_num, 64, 7, 24,
    36, radius)
77 gcpdxfpy
78 gcpdxfpy gcp.dxfcircle(large_square_tool_num, stockXwidth/2, stockYheight/2,
    radius * 2)
79 gcpdxfpy
80 gcpdxfpy gcp.dxfKH(374, stockXwidth/2, stockYheight/5*3, 0, -7, 270,
    11.5875)
81 gcpdxfpy
82 gcpdxfpy #gcp.closedxfiles()
83 gcpdxfpy 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)
- dxfarc (specifying arc center, radius, and beginning/ending angles)
- dxfKH (specifying origin, depth, angle, distance)

## 2.2 gcodepreviewtemplate.py

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

---

```

1 gcptmplpy #!/usr/bin/env python
2 gcptmplpy
3 gcptmplpy import sys
4 gcptmplpy
5 gcptmplpy 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
    /B0CPJPTMPP
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.opendxfile(Base_filename)
100 gcptmplpy gcp.opendxfiles(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,
112                             zeroheight, stockzero, retractheight)
113 gcptmplpy #print(pygcpversion())
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, -
121                                     stockZthickness)
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, -
142                                     stockZthickness)
143 gcptmplpy
144 gcptmplpy toolpaths = toolpaths.union(gcp.cutlinedxfgc(stockXwidth/2,
145                                                         stockYheight/2, -stockZthickness))
146 gcptmplpy
147 gcptmplpy gcp.rapidZ(retractheight)
148 gcptmplpy gcp.toolchange(201, 10000)
149 gcptmplpy gcp.rapidXY(0, stockYheight/16)
150 gcptmplpy gcp.rapidZ(0)
151 gcptmplpy toolpaths = toolpaths.union(gcp.cutlinedxfgc(stockXwidth/16*7,
152                                                         stockYheight/2, -stockZthickness))
153 gcptmplpy
154 gcptmplpy gcp.rapidZ(retractheight)
155 gcptmplpy gcp.toolchange(202, 10000)
156 gcptmplpy gcp.rapidXY(0, stockYheight/8)
157 gcptmplpy gcp.rapidZ(0)
158 gcptmplpy toolpaths = toolpaths.union(gcp.cutlinedxfgc(stockXwidth/16*6,
159                                                         stockYheight/2, -stockZthickness))
160 gcptmplpy
161 gcptmplpy gcp.rapidZ(retractheight)
162 gcptmplpy gcp.toolchange(101, 10000)
163 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.cutlineZgcfed(-1.531, plunge)
270 gcptmplpy toolpaths = toolpaths.union(gcp.cutline(gcp.xpos(), gcp.ypos(),
    -1.531))

```

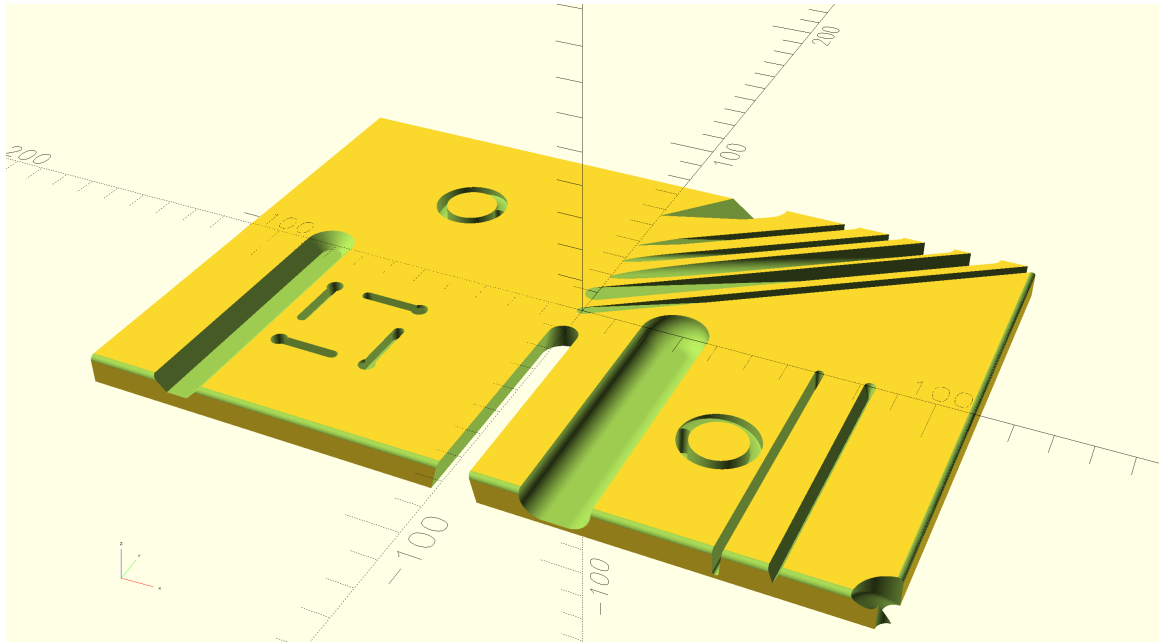
```

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 output (part)
308 gcptmplpy #output(test)
309 gcptmplpy #output (key)
310 gcptmplpy #output(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 #output(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/de-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 RO_ratio = 0.5; // [0.25:2]
80 gcptmpl /* [Feeds and Speeds] */
81 gcptmpl MISC_ratio = 0.5; // [0.25:2]
82 gcptmpl
83 gcptmpl thegeneratepaths = generatepaths == true ? 1 : 0;
84 gcptmpl thegeneratedxf = generatedxf == true ? 1 : 0;
85 gcptmpl thegenerategcode = generategcode == true ? 1 : 0;
86 gcptmpl
87 gcptmpl gcp = gcodepreview(thegeneratepaths,
88 gcptmpl                      thegenerategcode,
89 gcptmpl                      thegeneratedxf,
90 gcptmpl                      );
91 gcptmpl
92 gcptmpl.opengcodefile(Base_filename);
93 gcptmpl.opendxf(file(Base_filename);
94 gcptmpl.opendxf(files(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 cutlineZgcfed(-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

```

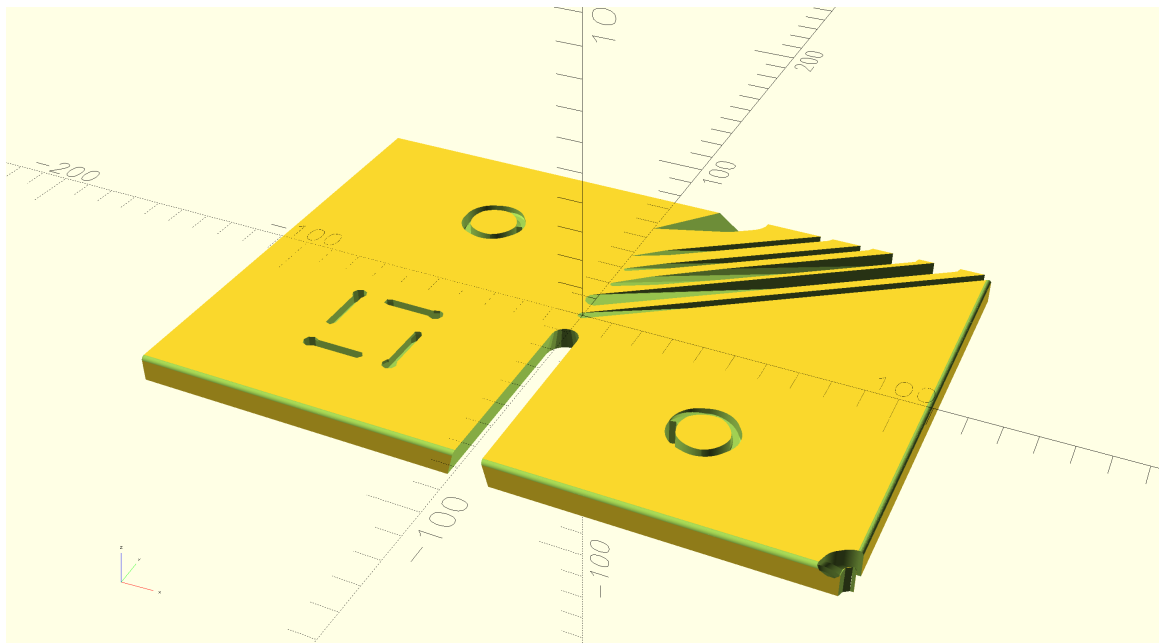
```

255 gcptmpl rapidZ(retractheight);
256 gcptmpl //#gcp.toolchange(56125, 10000)
257 gcptmpl cutlineZgcfed(-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 closedxfiles();
271 gcptmpl closedxfile();

```

---

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 4<sup>th</sup> 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
  - `dx` (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	dx	gcode	cut	dx	gcode
cut	cutline		cutlinegc	cutarc		cutarcgc
dx	cutlinedx	dxline		cutarcdx	dxarc	
gcode	cutlinegc		linegc	cutarcgc		arcgc
	cutlinedxfgc			cutarcdxfgc		

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

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)`, `dx`, `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 `dx` 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](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/](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 definitioon \DescribeSubroutine{FOOBAR}{pFOOBAR}

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

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

Further note that this style of definition might not have been necessary for some later modules since they are in turn calling internal modules which already use this structure. Lastly note that this style of programming was abandoned in favour of object-oriented dot notation after vo.6 (see below).

3.1.1 Parameters and Default Values

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

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

stepsizearc values, and thus afford the user/programmer the option of changing them after. See stepsizearc stepsizeroundover and stepsizeroundover.

3.2 Implementation files and gcodepreview class

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

---

```
1 gcpy #!/usr/bin/env python
2 gcpy #icon "C:\Program Files\PythonSCAD\bin\openscad.exe" --trust-python
3 gcpy #Currently tested with 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 gpcscad #!/OpenSCAD
2 gpcscad
3 gpcscad //gcodepreview version 0.8
4 gpcscad //
5 gpcscad //used via include <gcodepreview.scad>;
6 gpcscad //
7 gpcscad
8 gpcscad use <gcodepreview.py>
9 gpcscad
10 gpcscad module gcpversion(){
11 gpcscad echo(pygcpversion());
12 gpcscad }
13 gpcscad
14 gpcscad //function myfunc(var) = gcp.myfunc(var);
15 gpcscad //
16 gpcscad //function getvv() = gcp.getvv();
17 gpcscad //
18 gpcscad //module makecube(xdim, ydim, zdim){
19 gpcscad //gcp.makecube(xdim, ydim, zdim);
20 gpcscad //}
21 gpcscad //
22 gpcscad //module placecube(){
23 gpcscad //gcp.placecube();
24 gpcscad //}
25 gpcscad //
26 gpcscad //module instantiatecube(){
27 gpcscad //gcp.instantiatecube();
28 gpcscad //}
29 gpcscad //

```

---

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.

---

```

22 gcpy class gcodepreview:
23 gcpy
24 gcpy     def __init__(self, #basefilename = "export",
25 gcpy                     generatepaths = False,
26 gcpy                     generategcode = False,
27 gcpy                     generatedxf = False,
28 gcpy #                     stockXwidth = 25,
29 gcpy #                     stockYheight = 25,
30 gcpy #                     stockZthickness = 1,
31 gcpy #                     zeroheight = "Top",
32 gcpy #                     stockzero = "Lower-left",
33 gcpy #                     retractheight = 6,
34 gcpy #                     currenttoolnum = 102,
35 gcpy #                     toolradius = 3.175,
36 gcpy #                     plunge = 100,
37 gcpy #                     feed = 400,
38 gcpy #                     speed = 10000
39 gcpy                     ):
40 gcpy         """
41 gcpy         Initialize gcodepreview object.
42 gcpy
43 gcpy         Parameters
44 gcpy         -----
45 gcpy         generatepaths : boolean
46 gcpy                        Determines if toolpaths will be stored
47 gcpy                        internally or returned directly
48 gcpy
49 gcpy         generategcode : boolean
50 gcpy                        Enables writing out G-code.
51 gcpy
52 gcpy         generatedxf : boolean
53 gcpy                        Enables writing out DXF file(s).
54 gcpy
55 gcpy         Returns
56 gcpy         -----
57 gcpy         object
58 gcpy         The initialized gcodepreview object.
59 gcpy         """
60 gcpy         self.basefilename = basefilename
61 gcpy         if (generatepaths == 1):
62 gcpy             self.generatepaths = True
63 gcpy         if (generatepaths == 0):
64 gcpy             self.generatepaths = False
65 gcpy         else:
66 gcpy             self.generatepaths = generatepaths
67 gcpy         if (generategcode == 1):
68 gcpy             self.generategcode = True
69 gcpy         if (generategcode == 0):
70 gcpy             self.generategcode = False
71 gcpy         else:
72 gcpy             self.generategcode = generategcode
73 gcpy         if (generatedxf == 1):
74 gcpy             self.generatedxf = True
75 gcpy         if (generatedxf == 0):
76 gcpy             self.generatedxf = False
77 gcpy         else:
78 gcpy             self.generatedxf = generatedxf
79 gcpy         self.stockXwidth = stockXwidth
80 gcpy         self.stockYheight = stockYheight
81 gcpy         self.stockZthickness = stockZthickness
82 gcpy         self.zeroheight = zeroheight
83 gcpy         self.stockzero = stockzero
84 gcpy         self.retractheight = retractheight
85 gcpy         self.currenttoolnum = currenttoolnum
86 gcpy         self.toolradius = toolradius
87 gcpy         self.plunge = plunge
88 gcpy         self.feed = feed
89 gcpy         self.speed = speed
90 gcpy         global toolpaths
91 gcpy         if (openscadloaded == True):
92 gcpy             self.toolpaths = cylinder(0.1, 0.1)
93 gcpy         self.generatedxfs = False
94 gcpy
95 gcpy     def checkgeneratepaths():
96 gcpy         return self.generatepaths
97 gcpy
98 gcpy     def myfunc(self, var):
99 gcpy         self.vv = var * var
100 gcpy         return self.vv

```

```
99 gcpy # def getvv(self):
100 gcpy #     return self.vv
101 gcpy #
102 gcpy # def checkint(self):
103 gcpy #     return self.mc
104 gcpy #
105 gcpy # def makecube(self, xdim, ydim, zdim):
106 gcpy #     self.c=cube([xdim, ydim, zdim])
107 gcpy #
108 gcpy # def placecube(self):
109 gcpy #     output(self.c)
110 gcpy #
111 gcpy # def instantiatecube(self):
112 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

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

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

- setxpos
- setypos
- setzpos
- 130 gcpy def setxpos(self, newxpos):
- 131 gcpy # global mpx
- 132 gcpy self.mpx = newxpos
- 133 gcpy
- 134 gcpy def setypos(self, newypos):



```
135 gcpy #           global mpy
136 gcpy           self.mpy = newypos
137 gcpy
138 gcpy     def setzpos(self, newzpos):
139 gcpy #           global mpz
140 gcpy           self.mpz = newzpos
141 gcpy
142 gcpy #     def settpzinc(self, newtpzinc):
143 gcpy #           global tpzinc
144 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)).

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

`gcodepreview`     The first such setup subroutine is `gcodepreview setupstock` which is appropriately enough, to set up the stock, and perform other initializations — initially, the only thing done in Python was to set the value of the persistent (Python) variables (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.

```
162 gcpy     def setupstock(self, stockXwidth,
163 gcpy           stockYheight,
164 gcpy           stockZthickness,
165 gcpy           zeroheight,
166 gcpy           stockzero,
167 gcpy           retractheight):
168 gcpy         """
169 gcpy         Set up blank/stock for material and position/zero.
170 gcpy
171 gcpy         Parameters
172 gcpy         -----
173 gcpy         stockXwidth : float
174 gcpy                     X extent/dimension
175 gcpy         stockYheight : float
176 gcpy                     Y extent/dimension
177 gcpy         stockZthickness : boolean
178 gcpy                     Z extent/dimension
179 gcpy         zeroheight : string
180 gcpy                     Top or Bottom, determines if Z extent will
                        be positive or negative
181 gcpy         stockzero : string
182 gcpy                     Lower-Left, Center-Left, Top-Left, Center,
                        determines XY position of stock
183 gcpy         retractheight : float
184 gcpy                     Distance which tool retracts above surface
                        of stock.
185 gcpy
186 gcpy         Returns
187 gcpy         -----
188 gcpy         none
189 gcpy         """
```

---

```

190 gcpy          self.initializemachinestate()
191 gcpy          self.stockXwidth = stockXwidth
192 gcpy          self.stockYheight = stockYheight
193 gcpy          self.stockZthickness = stockZthickness
194 gcpy          self.zeroheight = zeroheight
195 gcpy          self.stockzero = stockzero
196 gcpy          self.retractheight = retractheight
197 gcpy #          global stock
198 gcpy          self.stock = cube([stockXwidth, stockYheight,
                                   stockZthickness])
199 gcpy %%WRITEGC          if self.generategcode == True:
200 gcpy %%WRITEGC          self.writegc("(Design File: " + self.
                                   basefilename + ")")
201 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.

---

```

202 gcpy          if self.zeroheight == "Top":
203 gcpy              if self.stockzero == "Lower-Left":
204 gcpy                  self.stock = self.stock.translate([0, 0, -self.
                                                             stockZthickness])
205 gcpy              if self.generategcode == True:
206 gcpy                  self.writegc("(stockMin:0.00mm,␣0.00mm,␣-", str
                                       (self.stockZthickness), "mm)")
207 gcpy                  self.writegc("(stockMax:", str(self.stockXwidth
                                                         ), "mm,␣", str(stockYheight), "mm,␣0.00mm)")
208 gcpy                  self.writegc("(STOCK/BLOCK,␣", str(self.
                                                             stockXwidth), ",␣", str(self.stockYheight),
                                                             ",␣", str(self.stockZthickness), ",␣0.00,␣
                                                             0.00,␣", str(self.stockZthickness), ")")
209 gcpy              if self.stockzero == "Center-Left":
210 gcpy                  self.stock = self.stock.translate([0, -stockYheight
                                                             / 2, -stockZthickness])
211 gcpy              if self.generategcode == True:
212 gcpy                  self.writegc("(stockMin:0.00mm,␣-", str(self.
                                                             stockYheight/2), "mm,␣-", str(self.
                                                             stockZthickness), "mm)")
213 gcpy                  self.writegc("(stockMax:", str(self.stockXwidth
                                                         ), "mm,␣", str(self.stockYheight/2), "mm,␣
                                                         0.00mm)")
214 gcpy                  self.writegc("(STOCK/BLOCK,␣", str(self.
                                                             stockXwidth), ",␣", str(self.stockYheight),
                                                             ",␣", str(self.stockZthickness), ",␣0.00,␣",
                                                             str(self.stockYheight/2), ",␣", str(self.
                                                             stockZthickness), ")");
215 gcpy              if self.stockzero == "Top-Left":
216 gcpy                  self.stock = self.stock.translate([0, -self.
                                                             stockYheight, -self.stockZthickness])
217 gcpy              if self.generategcode == True:
218 gcpy                  self.writegc("(stockMin:0.00mm,␣-", str(self.
                                                             stockYheight), "mm,␣-", str(self.
                                                             stockZthickness), "mm)")
219 gcpy                  self.writegc("(stockMax:", str(self.stockXwidth
                                                         ), "mm,␣0.00mm,␣0.00mm)")
220 gcpy                  self.writegc("(STOCK/BLOCK,␣", str(self.
                                                             stockXwidth), ",␣", str(self.stockYheight),
                                                             ",␣", str(self.stockZthickness), ",␣0.00,␣",
                                                             str(self.stockYheight), ",␣", str(self.
                                                             stockZthickness), ")")
221 gcpy              if self.stockzero == "Center":
222 gcpy                  self.stock = self.stock.translate([-self.
                                                             stockXwidth / 2, -self.stockYheight / 2, -self.
                                                             stockZthickness])
223 gcpy              if self.generategcode == True:
224 gcpy                  self.writegc("(stockMin:␣-", str(self.
                                                             stockXwidth/2), ",␣-", str(self.stockYheight
                                                             /2), "mm,␣-", str(self.stockZthickness), "mm
                                                             )")
225 gcpy                  self.writegc("(stockMax:", str(self.stockXwidth
                                                         /2), "mm,␣", str(self.stockYheight/2), "mm,␣
                                                         0.00mm)")
226 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), ")")
227 gcpy      if self.zeroheight == "Bottom":
228 gcpy      if self.stockzero == "Lower-Left":
229 gcpy      self.stock = self.stock.translate([0, 0, 0])
230 gcpy      if self.generategcode == True:
231 gcpy      self.writgc("(stockMin:0.00mm,␣0.00mm,␣0.00mm
        )")
232 gcpy      self.writgc("(stockMax:", str(self.
        stockXwidth), "mm,␣", str(self.stockYheight
        ), "mm,␣", str(self.stockZthickness), "mm"
        )
233 gcpy      self.writgc("(STOCK/BLOCK,␣", str(self.
        stockXwidth), ",␣", str(self.stockYheight),
        ",␣", str(self.stockZthickness), ",␣0.00,␣
        0.00,␣0.00)")
234 gcpy      if self.stockzero == "Center-Left":
235 gcpy      self.stock = self.stock.translate([0, -self.
        stockYheight / 2, 0])
236 gcpy      if self.generategcode == True:
237 gcpy      self.writgc("(stockMin:0.00mm,␣-", str(self.
        stockYheight/2), "mm,␣0.00mm)")
238 gcpy      self.writgc("(stockMax:", str(self.stockXwidth
        ), "mm,␣", str(self.stockYheight/2), "mm,␣-",
        , str(self.stockZthickness), "mm)")
239 gcpy      self.writgc("(STOCK/BLOCK,␣", str(self.
        stockXwidth), ",␣", str(self.stockYheight),
        ",␣", str(self.stockZthickness), ",␣0.00,␣",
        , str(self.stockYheight/2), ",␣0.00mm)");
240 gcpy      if self.stockzero == "Top-Left":
241 gcpy      self.stock = self.stock.translate([0, -self.
        stockYheight, 0])
242 gcpy      if self.generategcode == True:
243 gcpy      self.writgc("(stockMin:0.00mm,␣-", str(self.
        stockYheight), "mm,␣0.00mm)")
244 gcpy      self.writgc("(stockMax:", str(self.stockXwidth
        ), "mm,␣0.00mm,␣", str(self.stockZthickness)
        , "mm)")
245 gcpy      self.writgc("(STOCK/BLOCK,␣", str(self.
        stockXwidth), ",␣", str(self.stockYheight),
        ",␣", str(self.stockZthickness), ",␣0.00,␣",
        , str(self.stockYheight), ",␣0.00)")
246 gcpy      if self.stockzero == "Center":
247 gcpy      self.stock = self.stock.translate([-self.
        stockXwidth / 2, -self.stockYheight / 2, 0])
248 gcpy      if self.generategcode == True:
249 gcpy      self.writgc("(stockMin:␣-", str(self.
        stockXwidth/2), ",␣-", str(self.stockYheight
        /2), "mm,␣0.00mm)")
250 gcpy      self.writgc("(stockMax:", str(self.stockXwidth
        /2), "mm,␣", str(self.stockYheight/2), "mm,␣
        ", str(self.stockZthickness), "mm)")
251 gcpy      self.writgc("(STOCK/BLOCK,␣", str(self.
        stockXwidth), ",␣", str(self.stockYheight),
        ",␣", str(self.stockZthickness), ",␣", str(
        self.stockXwidth/2), ",␣", str(self.
        stockYheight/2), ",␣0.00)")
252 gcpy      if self.generategcode == True:
253 gcpy      self.writgc("G90");
254 gcpy      self.writgc("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 gpcpscad module setupstock(stockXwidth, stockYheight, stockZthickness,
        zeroheight, stockzero, retractheight) {
38 gpcpscad      gcp.setupstock(stockXwidth, stockYheight, stockZthickness,
        zeroheight, stockzero, retractheight);
39 gpcpscad }
```

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

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

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

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

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

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

```
264 gcpy      def shiftstock(self, shiftX, shiftY, shiftZ):
265 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:

```
267 gcpy      def addtostock(self, stockXwidth, stockYheight, stockZthickness
                                ,
268 gcpy      shiftX = 0,
269 gcpy      shiftY = 0,
270 gcpy      shiftZ = 0):
271 gcpy      addedpart = cube([stockXwidth, stockYheight,
                                stockZthickness])
272 gcpy      addedpart = addedpart.translate([shiftX, shiftY, shiftZ])
273 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

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:

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

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

1	2-3	4-5	6
endmill type radius/angle cutting diameter(and tip radius for tapered ball nose) cutting flute length			

- 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 (00 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<sup>nd</sup>
  - 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 (1/64th)
  - 3 - 0.0295
  - 4 - 0.03125 in (1/32nd)
  - 5 - 0.0335
  - 6 - 0.0354
  - 7 - 0.0625 in (1/16th)
  - 8 - 0.125 in (1/8th)
  - 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

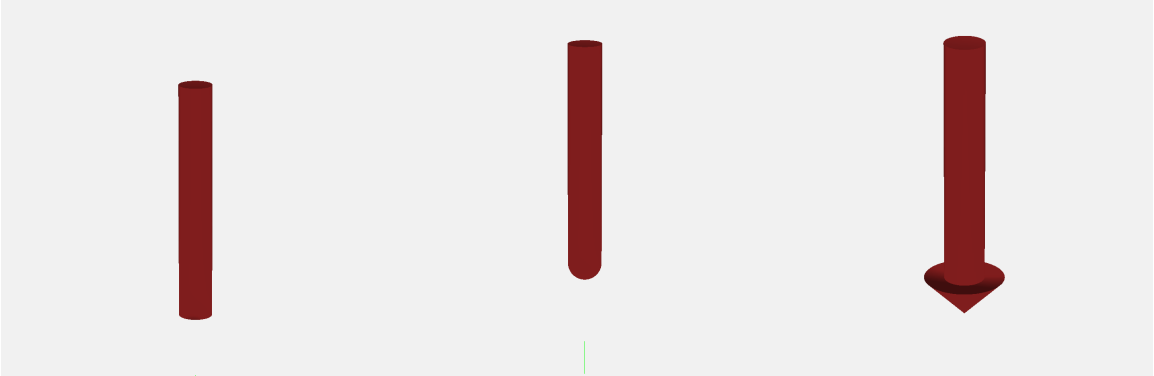
All of which reveals some notable limitations:

- 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:

```
287 gcpy      def endmill_square(self, es_diameter, es_flute_length):
288 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:

```
290 gcpy      def ballnose(self, es_diameter, es_flute_length):
291 gcpy          b = sphere(r=(es_diameter / 2))
292 gcpy          s = cylinder(r1=(es_diameter / 2), r2=(es_diameter / 2), h=
                        es_flute_length, center=False)
293 gcpy          p = union(b, s)
294 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):

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

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

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

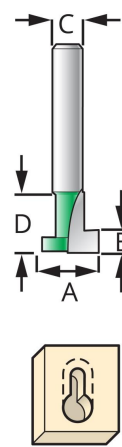
```
319 gcpy      def flat_V(self, es_tip, es_diameter, es_flute_length, es_angle
):
320 gcpy          c = cylinder(r1=(es_tip / 2), r2=(es_diameter / 2), h=
                                es_flute_length, center=False)
321 gcpy          return c
```

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 Router Bits

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

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

```
323 gcpy      def keyhole(self, es_diameter, es_flute_length):
324 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:

```
326 gcpy      def keyhole_shaft(self, es_diameter, es_flute_length):
327 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>.

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

```
341 gcpy      def dovetail(self, dt_bottomdiameter, dt_topdiameter, dt_height, dt_angle):
342 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 pre-viewing 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

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

```
348 gcpy     def toolchange(self, tool_number, speed = 10000):
349 gcpy #         global currenttoolshape
350 gcpy         self.currenttoolshape = self.endmill_square(0.001, 0.001)
351 gcpy
```

```

352 gcpy self.settool(tool_number)
353 gcpy if (self.generategcode == True):
354 gcpy     self.writegc("(Toolpath)")
355 gcpy     self.writegc("M05")
356 gcpy if (tool_number == 201):
357 gcpy     self.writegc("(TOOL/MILL,␣6.35,␣0.00,␣0.00,␣0.00)")
358 gcpy     self.currenttoolshape = self.endmill_square(6.35,
19.05)
359 gcpy elif (tool_number == 102):
360 gcpy     self.writegc("(TOOL/MILL,␣3.175,␣0.00,␣0.00,␣0.00)")
361 gcpy     self.currenttoolshape = self.endmill_square(3.175,
12.7)
362 gcpy elif (tool_number == 112):
363 gcpy     self.writegc("(TOOL/MILL,␣1.5875,␣0.00,␣0.00,␣0.00)")
364 gcpy     self.currenttoolshape = self.endmill_square(1.5875,
6.35)
365 gcpy elif (tool_number == 122):
366 gcpy     self.writegc("(TOOL/MILL,␣0.79375,␣0.00,␣0.00,␣0.00)")
367 gcpy     self.currenttoolshape = self.endmill_square(0.79375,
1.5875)
368 gcpy elif (tool_number == 202):
369 gcpy     self.writegc("(TOOL/MILL,␣6.35,␣3.175,␣0.00,␣0.00)")
370 gcpy     self.currenttoolshape = self.ballnose(6.35, 19.05)
371 gcpy elif (tool_number == 101):
372 gcpy     self.writegc("(TOOL/MILL,␣3.175,␣1.5875,␣0.00,␣0.00)")
373 gcpy     self.currenttoolshape = self.ballnose(3.175, 12.7)
374 gcpy elif (tool_number == 111):
375 gcpy     self.writegc("(TOOL/MILL,␣1.5875,␣0.79375,␣0.00,␣0.00)"
)
376 gcpy     self.currenttoolshape = self.ballnose(1.5875, 6.35)
377 gcpy elif (tool_number == 121):
378 gcpy     self.writegc("(TOOL/MILL,␣3.175,␣0.79375,␣0.00,␣0.00)")
379 gcpy     self.currenttoolshape = self.ballnose(0.79375, 1.5875)
380 gcpy elif (tool_number == 327):
381 gcpy     self.writegc("(TOOL/MILL,␣0.03,␣0.00,␣13.4874,␣30.00)")
382 gcpy     self.currenttoolshape = self.endmill_v(60, 26.9748)
383 gcpy elif (tool_number == 301):
384 gcpy     self.writegc("(TOOL/MILL,␣0.03,␣0.00,␣6.35,␣45.00)")
385 gcpy     self.currenttoolshape = self.endmill_v(90, 12.7)
386 gcpy elif (tool_number == 302):
387 gcpy     self.writegc("(TOOL/MILL,␣0.03,␣0.00,␣10.998,␣30.00)")
388 gcpy     self.currenttoolshape = self.endmill_v(60, 12.7)
389 gcpy elif (tool_number == 390):
390 gcpy     self.writegc("(TOOL/MILL,␣0.03,␣0.00,␣1.5875,␣45.00)")
391 gcpy     self.currenttoolshape = self.endmill_v(90, 3.175)
392 gcpy elif (tool_number == 374):
393 gcpy     self.writegc("(TOOL/MILL,␣9.53,␣0.00,␣3.17,␣0.00)")
394 gcpy elif (tool_number == 375):
395 gcpy     self.writegc("(TOOL/MILL,␣9.53,␣0.00,␣3.17,␣0.00)")
396 gcpy elif (tool_number == 376):
397 gcpy     self.writegc("(TOOL/MILL,␣12.7,␣0.00,␣4.77,␣0.00)")
398 gcpy elif (tool_number == 378):
399 gcpy     self.writegc("(TOOL/MILL,␣12.7,␣0.00,␣4.77,␣0.00)")
400 gcpy elif (tool_number == 814):
401 gcpy     self.writegc("(TOOL/MILL,␣12.7,␣6.367,␣12.7,␣0.00)")
402 gcpy     #dt_bottomdiameter, dt_topdiameter, dt_height, dt_angle
)
403 gcpy     #https://www.leevalley.com/en-us/shop/tools/power-tool-
accessories/router-bits/30172-dovetail-bits?item=18
J1607
404 gcpy     self.currenttoolshape = self.dovetail(12.7, 6.367,
12.7, 14)
405 gcpy #45828
406 gcpy elif (tool_number == 808079):
407 gcpy     self.writegc("(TOOL/MILL,␣12.7,␣6.816,␣20.95,␣0.00)")
408 gcpy     #http://www.amanatool.com/45828-carbide-tipped-dovetail
-8-deg-x-1-2-dia-x-825-x-1-4-inch-shank.html
409 gcpy     self.currenttoolshape = self.dovetail(12.7, 6.816,
20.95, 8)
410 gcpy elif (tool_number == 56125):#0.508/2, 1.531
411 gcpy     self.writegc("(TOOL/CRMILL,␣0.508,␣6.35,␣3.175,␣7.9375,
␣3.175)")
412 gcpy elif (tool_number == 56142):#0.508/2, 2.921
413 gcpy     self.writegc("(TOOL/CRMILL,␣0.508,␣3.571875,␣1.5875,␣
5.55625,␣1.5875)")
414 gcpy # elif (tool_number == 312):#1.524/2, 3.175
415 gcpy #     self.writegc("(TOOL/CRMILL, Diameter1, Diameter2,
Radius, Height, Length)")

```

```
416 gcpy          elif (tool_number == 1570):#0.507/2, 4.509
417 gcpy          self.writegc("(T00L/CRMILL,␣0.17018,␣9.525,␣4.7625,␣
                        12.7,␣4.7625)")
418 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
419 gcpy          elif (tool_number == 45982):#0.507/2, 4.509
420 gcpy          self.writegc("(T00L/MILL,␣15.875,␣6.35,␣19.05,␣0.00)")
421 gcpy          self.currenttoolshape = self.bowl_tool(6.35, 19.05,
                        15.875)
422 gcpy          elif (tool_number == 204):#
423 gcpy          self.writegc("()")
424 gcpy          self.currenttoolshape = self.tapered_ball(1.5875, 6.35,
                        38.1, 3.6)
425 gcpy          elif (tool_number == 304):#
426 gcpy          self.writegc("()")
427 gcpy          self.currenttoolshape = self.tapered_ball(3.175, 6.35,
                        38.1, 2.4)
428 gcpy          elif (tool_number == 501):#
429 gcpy          self.writegc("()")
430 gcpy          self.currenttoolshape = self.tapered_ball(0.127, 3.175,
                        2.688, 60)
431 gcpy          elif (tool_number == 502):#
432 gcpy          self.writegc("()")
433 gcpy          self.currenttoolshape = self.tapered_ball(0.127, 3.175,
                        4.25, 40)
434 gcpy          elif (tool_number == 13921):#
435 gcpy          self.writegc("()")
436 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.

```
432 gcpy          self.writegc("M6T", str(tool_number))
433 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 gcpscad function tool_diameter(td_tool, td_depth) = otool_diameter(td_tool,
                        td_depth);
```

tool diameter

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

```
435 gcpy          def tool_diameter(self, ptd_tool, ptd_depth):
436 gcpy # Square 122, 112, 102, 201
437 gcpy          if ptd_tool == 122:
438 gcpy              return 0.79375
439 gcpy          if ptd_tool == 112:
440 gcpy              return 1.5875
441 gcpy          if ptd_tool == 102:
442 gcpy              return 3.175
```

```

443 gcpy          if ptd_tool == 201:
444 gcpy              return 6.35
445 gcpy # Ball 121, 111, 101, 202
446 gcpy          if ptd_tool == 122:
447 gcpy              if ptd_depth > 0.396875:
448 gcpy                  return 0.79375
449 gcpy              else:
450 gcpy                  return ptd_tool
451 gcpy          if ptd_tool == 112:
452 gcpy              if ptd_depth > 0.79375:
453 gcpy                  return 1.5875
454 gcpy              else:
455 gcpy                  return ptd_tool
456 gcpy          if ptd_tool == 101:
457 gcpy              if ptd_depth > 1.5875:
458 gcpy                  return 3.175
459 gcpy              else:
460 gcpy                  return ptd_tool
461 gcpy          if ptd_tool == 202:
462 gcpy              if ptd_depth > 3.175:
463 gcpy                  return 6.35
464 gcpy              else:
465 gcpy                  return ptd_tool
466 gcpy # V 301, 302, 390
467 gcpy          if ptd_tool == 301:
468 gcpy              return ptd_tool
469 gcpy          if ptd_tool == 302:
470 gcpy              return ptd_tool
471 gcpy          if ptd_tool == 390:
472 gcpy              return ptd_tool
473 gcpy # Keyhole
474 gcpy          if ptd_tool == 374:
475 gcpy              if ptd_depth < 3.175:
476 gcpy                  return 9.525
477 gcpy              else:
478 gcpy                  return 6.35
479 gcpy          if ptd_tool == 375:
480 gcpy              if ptd_depth < 3.175:
481 gcpy                  return 9.525
482 gcpy              else:
483 gcpy                  return 8
484 gcpy          if ptd_tool == 376:
485 gcpy              if ptd_depth < 4.7625:
486 gcpy                  return 12.7
487 gcpy              else:
488 gcpy                  return 6.35
489 gcpy          if ptd_tool == 378:
490 gcpy              if ptd_depth < 4.7625:
491 gcpy                  return 12.7
492 gcpy              else:
493 gcpy                  return 8
494 gcpy # Dovetail
495 gcpy          if ptd_tool == 814:
496 gcpy              if ptd_depth > 12.7:
497 gcpy                  return 6.35
498 gcpy              else:
499 gcpy                  return ptd_tool
500 gcpy          if ptd_tool == 808079:
501 gcpy              if ptd_depth > 20.95:
502 gcpy                  return 6.816
503 gcpy              else:
504 gcpy                  return ptd_tool
505 gcpy # Bowl Bit
506 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
507 gcpy          if ptd_tool == 45982:
508 gcpy              if ptd_depth > 6.35:
509 gcpy                  return 15.875
510 gcpy              else:
511 gcpy                  return ptd_tool
512 gcpy # Tapered Ball Nose
513 gcpy          if ptd_tool == 204:
514 gcpy              if ptd_depth > 6.35:
515 gcpy                  return ptd_tool
516 gcpy          if ptd_tool == 304:
517 gcpy              if ptd_depth > 6.35:
518 gcpy                  return ptd_tool
519 gcpy              else:

```

520 gcpyreturn 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:

517 gcpydef tool\_radius(self, ptd\_tool, ptd\_depth):

518 gcpytr = self.tool\_diameter(ptd\_tool, ptd\_depth)/2

519 gcpyreturn 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 and speed are used, which may then be adjusted using various <tooldescriptor>\_ratio values, as an acknowledgement of the likelihood of a trim router being used as a spindle, the assumption is that the speed will remain unchanged.

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...

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.

r

cs

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:

521 gcpydef rcs(self, ex, ey, ez, shape):

522 gcpystart = shape

523 gcpyend = shape

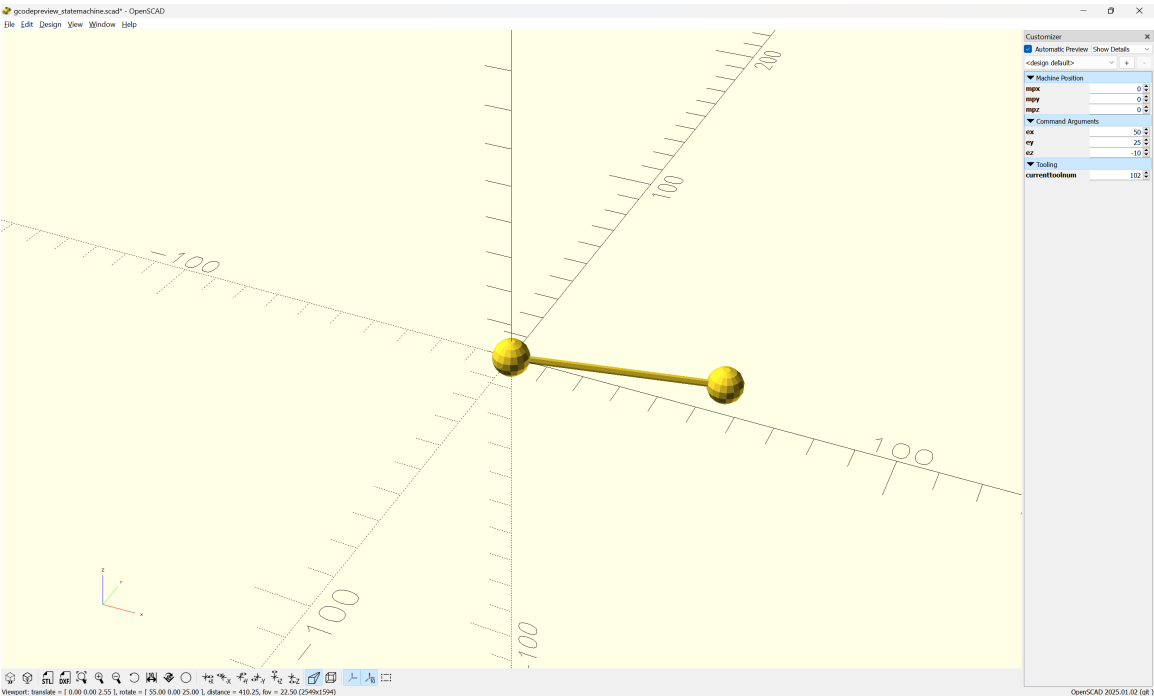
524 gcpytoolpath = hull(start.translate([self.xpos(), self.ypos(), self.zpos()]),

525 gcpyend.translate([ex, ey, ez]))

526 gcpyreturn toolpath

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

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



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

There are three different movements in G-code which will need to be handled. Rapid commands will be used for go movements and will not appear in DXFs but will appear in G-code files, while straight line cut (G1) and arc (G2/G3) commands may appear in both G-code and DXF files, depending on the specific command invoked.

```
528 gcpy      def rapid(self, ex, ey, ez):
529 gcpy      cts = self.currenttoolshape
530 gcpy      toolpath = self.rcs(ex, ey, ez, cts)
531 gcpy      self.setxpos(ex)
532 gcpy      self.setypos(ey)
533 gcpy      self.setzpos(ez)
534 gcpy      if self.generatepaths == True:
535 gcpy          self.rapids = self.rapids.union(toolpath)
536 gcpy #          return cylinder(0.01, 0, 0.01, center = False, fn = 3)
537 gcpy          return cube([0.001, 0.001, 0.001])
538 gcpy      else:
539 gcpy          return toolpath
540 gcpy
541 gcpy      def cutshape(self, ex, ey, ez):
542 gcpy      cts = self.currenttoolshape
543 gcpy      toolpath = self.rcs(ex, ey, ez, cts)
544 gcpy      if self.generatepaths == True:
545 gcpy          self.toolpaths = self.toolpaths.union(toolpath)
546 gcpy          return cube([0.001, 0.001, 0.001])
547 gcpy      else:
548 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:

```
550 gcpy      def movetosafeZ(self):
551 gcpy          rapid = self.rapid(self.xpos(), self.ypos(), self.
                    retractheight)
552 gcpy #          if self.generatepaths == True:
553 gcpy #              rapid = self.rapid(self.xpos(), self.ypos(), self.
                    retractheight)
554 gcpy #              self.rapids = self.rapids.union(rapid)
555 gcpy #          else:
556 gcpy #              if (generategcode == true) {
557 gcpy #                  writecomment("PREPOSITION FOR RAPID PLUNGE");Z25.650
558 gcpy #                  //G1Z24.663F381.0, "F", str(plunge)
```

```

559 gcpy          if self.generatepaths == False:
560 gcpy              return rapid
561 gcpy          else:
562 gcpy              return cube([0.001, 0.001, 0.001])
563 gcpy
564 gcpy          def rapidXYZ(self, ex, ey, ez):
565 gcpy              rapid = self.rapid(ex, ey, ez)
566 gcpy              if self.generatepaths == False:
567 gcpy                  return rapid
568 gcpy
569 gcpy          def rapidXY(self, ex, ey):
570 gcpy              rapid = self.rapid(ex, ey, self.zpos())
571 gcpy              if self.generatepaths == True:
572 gcpy                  self.rapids = self.rapids.union(rapid)
573 gcpy              else:
574 gcpy                  if self.generatepaths == False:
575 gcpy                      return rapid
576 gcpy
577 gcpy          def rapidXZ(self, ex, ez):
578 gcpy              rapid = self.rapid(ex, self.ypos(), ez)
579 gcpy              if self.generatepaths == False:
580 gcpy                  return rapid
581 gcpy
582 gcpy          def rapidYZ(self, ey, ez):
583 gcpy              rapid = self.rapid(self.xpos(), ey, ez)
584 gcpy              if self.generatepaths == False:
585 gcpy                  return rapid
586 gcpy
587 gcpy          def rapidX(self, ex):
588 gcpy              rapid = self.rapid(ex, self.ypos(), self.zpos())
589 gcpy              if self.generatepaths == False:
590 gcpy                  return rapid
591 gcpy
592 gcpy          def rapidY(self, ey):
593 gcpy              rapid = self.rapid(self.xpos(), ey, self.zpos())
594 gcpy              if self.generatepaths == False:
595 gcpy                  return rapid
596 gcpy
597 gcpy          def rapidZ(self, ez):
598 gcpy              rapid = self.rapid(self.xpos(), self.ypos(), ez)
599 gcpy              if self.generatepaths == True:
600 gcpy                  self.rapids = self.rapids.union(rapid)
601 gcpy              else:
602 gcpy                  if self.generatepaths == False:
603 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:



```

605 gcpy      def cutline(self, ex, ey, ez):\
606 gcpy #below will need to be integrated into if/then structure not yet
        copied
607 gcpy #          cts = self.currenttoolshape
608 gcpy          if (self.currenttoolnumber() == 374):
609 gcpy #              self.writegc("(TOOL/MILL, 9.53, 0.00, 3.17, 0.00)")
610 gcpy              self.currenttoolshape = self.keyhole(9.53/2, 3.175)
611 gcpy              toolpath = self.cutshape(ex, ey, ez)
612 gcpy              self.currenttoolshape = self.keyhole_shaft(6.35/2,
                    12.7)
613 gcpy              toolpath = toolpath.union(self.cutshape(ex, ey, ez))
614 gcpy #          elif (self.currenttoolnumber() == 375):
615 gcpy #              self.writegc("(TOOL/MILL, 9.53, 0.00, 3.17, 0.00)")
616 gcpy #          elif (self.currenttoolnumber() == 376):
617 gcpy #              self.writegc("(TOOL/MILL, 12.7, 0.00, 4.77, 0.00)")
618 gcpy #          elif (self.currenttoolnumber() == 378):
619 gcpy #              self.writegc("(TOOL/MILL, 12.7, 0.00, 4.77, 0.00)")
620 gcpy #          elif (self.currenttoolnumber() == 56125):#0.508/2, 1.531
621 gcpy #              self.writegc("(TOOL/CRMILL, 0.508, 6.35, 3.175,
5.9375, 3.175)")
622 gcpy          elif (self.currenttoolnumber() == 56142):#0.508/2, 2.921
623 gcpy #              self.writegc("(TOOL/CRMILL, 0.508, 3.571875, 1.5875,
5.55625, 1.5875)")
624 gcpy              toolpath = self.cutroundovertool(self.xpos(), self.ypos
                    (), self.zpos(), ex, ey, ez, 0.508/2, 1.531)
625 gcpy #          elif (self.currenttoolnumber() == 1570):#0.507/2, 4.509
626 gcpy #              self.writegc("(TOOL/CRMILL, 0.17018, 9.525, 4.7625,
12.7, 4.7625)")
627 gcpy          else:
628 gcpy              toolpath = self.cutshape(ex, ey, ez)
629 gcpy              self.setxpos(ex)
630 gcpy              self.setypos(ey)
631 gcpy              self.setzpos(ez)
632 gcpy #          if self.generatepaths == True:
633 gcpy #              self.toolpaths = union([self.toolpaths, toolpath])
634 gcpy #          else:
635 gcpy          if self.generatepaths == False:
636 gcpy              return toolpath
637 gcpy          else:
638 gcpy              return cube([0.001, 0.001, 0.001])
639 gcpy
640 gcpy      def cutlinedxfgc(self, ex, ey, ez):
641 gcpy          self.dxfline(self.currenttoolnumber(), self.xpos(), self.
                    ypos(), ex, ey)
642 gcpy          self.writegc("G01_X", str(ex), "Y", str(ey), "Z", str(ez)
                    )
643 gcpy #          if self.generatepaths == False:
644 gcpy          return self.cutline(ex, ey, ez)
645 gcpy
646 gcpy      def cutroundovertool(self, bx, by, bz, ex, ey, ez,
        tool_radius_tip, tool_radius_width, stepsizeroundover = 1):
647 gcpy #          n = 90 + fn*3
648 gcpy #          print("Tool dimensions", tool_radius_tip,
        tool_radius_width, "begin ", bx, by, bz, "end ", ex, ey, ez)
649 gcpy          step = 4 #360/n
650 gcpy          shaft = cylinder(step, tool_radius_tip, tool_radius_tip)
651 gcpy          toolpath = hull(shaft.translate([bx, by, bz]), shaft.
                    translate([ex, ey, ez]))
652 gcpy          shaft = cylinder(tool_radius_width*2, tool_radius_tip+
        tool_radius_width, tool_radius_tip+tool_radius_width)
653 gcpy          toolpath = toolpath.union(hull(shaft.translate([bx, by, bz+
        tool_radius_width]), shaft.translate([ex, ey, ez+
        tool_radius_width])))
654 gcpy          for i in range(1, 90, stepsizeroundover):
655 gcpy              angle = i
656 gcpy              dx = tool_radius_width*math.cos(math.radians(angle))
657 gcpy              dxx = tool_radius_width*math.cos(math.radians(angle+1))
658 gcpy              dzz = tool_radius_width*math.sin(math.radians(angle))
659 gcpy              dz = tool_radius_width*math.sin(math.radians(angle+1))
660 gcpy              dh = abs(dzz-dz)+0.0001
661 gcpy              slice = cylinder(dh, tool_radius_tip+tool_radius_width-
                    dx, tool_radius_tip+tool_radius_width-dxx)
662 gcpy              toolpath = toolpath.union(hull(slice.translate([bx, by,
                    bz+dz]), slice.translate([ex, ey, ez+dz])))
663 gcpy          if self.generatepaths == True:
664 gcpy              self.toolpaths = self.toolpaths.union(toolpath)
665 gcpy          else:
666 gcpy              return toolpath

```

```
667 gcpy
668 gcpy      def cutlineXYZwithfeed(self, ex, ey, ez, feed):
669 gcpy          return self.cutline(ex, ey, ez)
670 gcpy
671 gcpy      def cutlineXYZ(self, ex, ey, ez):
672 gcpy          return self.cutline(ex, ey, ez)
673 gcpy
674 gcpy      def cutlineXYwithfeed(self, ex, ey, feed):
675 gcpy          return self.cutline(ex, ey, self.zpos())
676 gcpy
677 gcpy      def cutlineXY(self, ex, ey):
678 gcpy          return self.cutline(ex, ey, self.zpos())
679 gcpy
680 gcpy      def cutlineXZwithfeed(self, ex, ez, feed):
681 gcpy          return self.cutline(ex, self.ypos(), ez)
682 gcpy
683 gcpy      def cutlineXZ(self, ex, ez):
684 gcpy          return self.cutline(ex, self.ypos(), ez)
685 gcpy
686 gcpy      def cutlineXwithfeed(self, ex, feed):
687 gcpy          return self.cutline(ex, self.ypos(), self.zpos())
688 gcpy
689 gcpy      def cutlineX(self, ex):
690 gcpy          return self.cutline(ex, self.ypos(), self.zpos())
691 gcpy
692 gcpy      def cutlineYZ(self, ey, ez):
693 gcpy          return self.cutline(self.xpos(), ey, ez)
694 gcpy
695 gcpy      def cutlineYwithfeed(self, ey, feed):
696 gcpy          return self.cutline(self.xpos(), ey, self.zpos())
697 gcpy
698 gcpy      def cutlineY(self, ey):
699 gcpy          return self.cutline(self.xpos(), ey, self.zpos())
700 gcpy
701 gcpy      def cutlineZgcfed(self, ez, feed):
702 gcpy          self.writegc("G01┘Z", str(ez), "F", str(feed))
703 gcpy      #          if self.generatepaths == False:
704 gcpy          return self.cutline(self.xpos(), self.ypos(), ez)
705 gcpy
706 gcpy      def cutlineZwithfeed(self, ez, feed):
707 gcpy          return self.cutline(self.xpos(), self.ypos(), ez)
708 gcpy
709 gcpy      def cutlineZ(self, ez):
710 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 cutlineZgcfed(ez, feed){
96 gcpscad     gcp.cutlineZgcfed(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) G2/3 ...
- DXF — dxfarc(xcenter, ycenter, radius, anglebegin, endangle, tn)
- approximation of arc using lines (OpenSCAD) in both clock-wise and counter-clock-wise directions

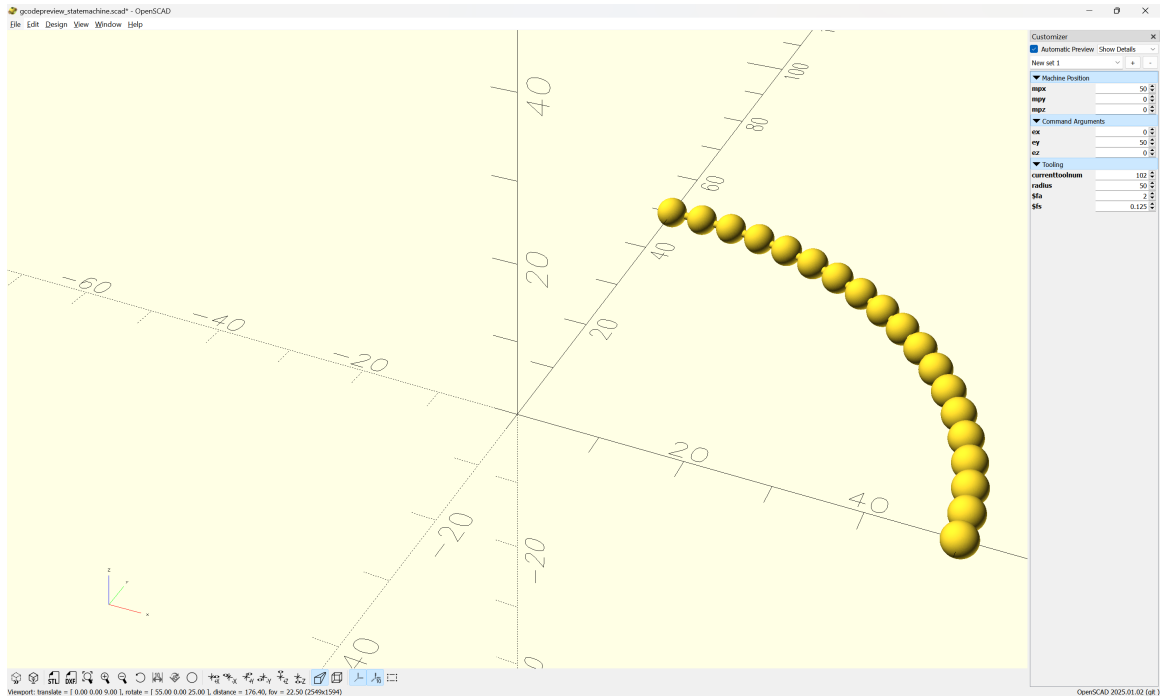
Cutting the quadrant arcs greatly simplifies the calculation and interface for the modules. A full set of 8 will be necessary, then circles will have a pair of modules (one for each cut direction) made for them.

Parameters which will need to be passed in are:

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

Since OpenSCAD does not have an arc movement command it is necessary to iterate through a cutarcCW loop: cutarcCW (clockwise) or cutarcCC (counterclockwise) to handle the drawing and processing 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.)

---

```

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

```

```
758 gcpy          else:
759 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:

```
757 gcpy      def cutarcCWdxf(self, barc, earc, xcenter, ycenter, radius,
758 gcpy          tpzreldim, stepsizearc=1):
759 gcpy          toolpath = self.cutarcCW(barc, earc, xcenter, ycenter,
760 gcpy              radius, tpzreldim, stepsizearc=1)
761 gcpy          self.dxfarc(self.currenttoolnumber(), xcenter, ycenter,
762 gcpy              radius, earc, barc)
763 gcpy          if self.generatepaths == False:
764 gcpy              return toolpath
765 gcpy          else:
766 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 sector<sup>1</sup>
  - 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

<sup>1</sup>[en.wikipedia.org/wiki/Hyperbolic\\_sector](https://en.wikipedia.org/wiki/Hyperbolic_sector) and [www.reddit.com/r/Geometry/comments/bkbzgh/is\\_there\\_a\\_name\\_for\\_a\\_3\\_pointed\\_figure\\_with\\_two](https://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:

```
802 gcpy      def dxfcircle(self, tool_num, xcenter, ycenter, radius):
803 gcpy          self.dxfarc(tool_num, xcenter, ycenter, radius, 0, 90)
804 gcpy          self.dxfarc(tool_num, xcenter, ycenter, radius, 90, 180)
805 gcpy          self.dxfarc(tool_num, xcenter, ycenter, radius, 180, 270)
806 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 forms for rectangles, square cornered and rounded:

```
808 gcpy      def dxfrectangle(self, tool_num, xorigin, yorigin, xwidth,
809 gcpy          yheight, corners = "Square", radius = 6):
810 gcpy          if corners == "Square":
811 gcpy              self.dxfline(tool_num, xorigin, yorigin, xorigin +
812 gcpy                  xwidth, yorigin)
813 gcpy              self.dxfline(tool_num, xorigin + xwidth, yorigin,
814 gcpy                  xorigin + xwidth, yorigin + yheight)
815 gcpy              self.dxfline(tool_num, xorigin + xwidth, yorigin +
816 gcpy                  yheight, xorigin, yorigin + yheight)
817 gcpy              self.dxfline(tool_num, xorigin, yorigin + yheight,
818 gcpy                  xorigin, yorigin)
819 gcpy          elif corners == "Fillet":
820 gcpy              self.dxfrectangleround(tool_num, xorigin, yorigin,
821 gcpy                  xwidth, yheight, radius)
822 gcpy          elif corners == "Chamfer":
823 gcpy              self.dxfrectanglechamfer(tool_num, xorigin, yorigin,
824 gcpy                  xwidth, yheight, radius)
825 gcpy          elif corners == "Flipped_Fillet":
826 gcpy              self.dxfrectangleflippedfillet(tool_num, xorigin,
827 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.

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

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



```
832 gcpy      def dxfrectanglechamfer(self, tool_num, xorigin, yorigin,
833 gcpy          xwidth, yheight, radius):
834 gcpy          self.dxfline(tool_num, xorigin + radius, yorigin, xorigin,
835 gcpy              yorigin + radius)
836 gcpy          self.dxfline(tool_num, xorigin, yorigin + yheight - radius,
837 gcpy              xorigin + radius, yorigin + yheight)
838 gcpy          self.dxfline(tool_num, xorigin + xwidth - radius, yorigin +
839 gcpy              yheight, xorigin + xwidth, yorigin + yheight - radius)
840 gcpy          self.dxfline(tool_num, xorigin + xwidth - radius, yorigin,
841 gcpy              xorigin + xwidth, yorigin + radius)
```

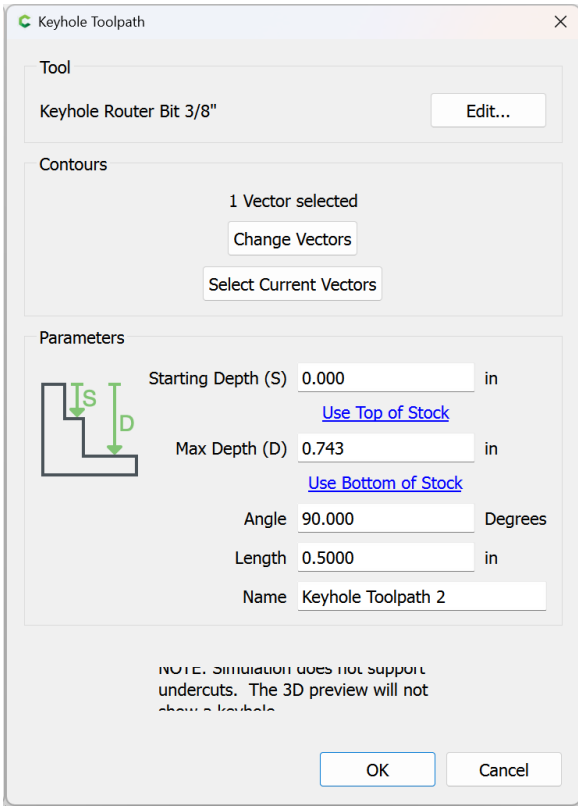
Flipped Fillet:

```
843 gcpy      def dxfrectangleflippedfillet(self, tool_num, xorigin, yorigin,
844 gcpy          xwidth, yheight, radius):
845 gcpy          self.dxfarc(tool_num, xorigin, yorigin, radius, 0, 90)
846 gcpy          self.dxfarc(tool_num, xorigin + xwidth, yorigin, radius,
847 gcpy              90, 180)
848 gcpy          self.dxfarc(tool_num, xorigin + xwidth, yorigin + yheight,
849 gcpy              radius, 180, 270)
850 gcpy          self.dxfarc(tool_num, xorigin, yorigin + yheight, radius,
851 gcpy              270, 360)
852 gcpy          self.dxfline(tool_num, xorigin + radius, yorigin, xorigin +
853 gcpy              xwidth - radius, yorigin)
854 gcpy          self.dxfline(tool_num, xorigin + xwidth, yorigin + radius,
855 gcpy              xorigin + xwidth, yorigin + yheight - radius)
856 gcpy          self.dxfline(tool_num, xorigin + xwidth - radius, yorigin +
857 gcpy              yheight, xorigin + radius, yorigin + yheight)
858 gcpy          self.dxfline(tool_num, xorigin, yorigin + yheight - radius,
859 gcpy              xorigin, yorigin + radius)
```

**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:

```
854 gcpy      def cutkeyholegdcxf(self, kh_tool_num, kh_start_depth,
855 gcpy          kh_max_depth, kht_direction, kh_distance):
856 gcpy          toolpath = self.cutKHgdcxf(kh_tool_num, kh_start_depth,
857 gcpy              kh_max_depth, 90, kh_distance)
858 gcpy          elif (kht_direction == "S"):
859 gcpy              toolpath = self.cutKHgdcxf(kh_tool_num, kh_start_depth,
860 gcpy                  kh_max_depth, 270, kh_distance)
861 gcpy          elif (kht_direction == "E"):
862 gcpy              toolpath = self.cutKHgdcxf(kh_tool_num, kh_start_depth,
863 gcpy                  kh_max_depth, 0, kh_distance)
864 gcpy          elif (kht_direction == "W"):
865 gcpy              toolpath = self.cutKHgdcxf(kh_tool_num, kh_start_depth,
866 gcpy                  kh_max_depth, 180, kh_distance)
867 gcpy          if self.generatepaths == True:
868 gcpy              self.toolpaths = union([self.toolpaths, toolpath])
869 gcpy              return toolpath
870 gcpy          else:
871 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:

869 gcpy def cutKHgcdxf(self, kh\_tool\_num, kh\_start\_depth, kh\_max\_depth, kh\_angle, kh\_distance):

870 gcpy oXpos = self.xpos()

871 gcpy oYpos = self.ypos()

872 gcpy self.dxfKH(kh\_tool\_num, self.xpos(), self.ypos(), kh\_start\_depth, kh\_max\_depth, kh\_angle, kh\_distance)

873 gcpy toolpath = self.cutline(self.xpos(), self.ypos(), - kh\_max\_depth)

874 gcpy self.setxpos(oXpos)

875 gcpy self.setypos(oYpos)

876 gcpy if self.generatepaths == False:

877 gcpy return toolpath

878 gcpy else:

879 gcpy return cube([0.001, 0.001, 0.001])

881 gcpy def dxfKH(self, kh\_tool\_num, oXpos, oYpos, kh\_start\_depth, kh\_max\_depth, kh\_angle, kh\_distance):

882 gcpy # oXpos = self.xpos()

883 gcpy # oYpos = self.ypos()

884 gcpy #Circle at entry hole

885 gcpy self.dxfarc(kh\_tool\_num, oXpos, oYpos, self.tool\_radius(kh\_tool\_num, 7), 0, 90)

886 gcpy self.dxfarc(kh\_tool\_num, oXpos, oYpos, self.tool\_radius(kh\_tool\_num, 7), 90, 180)

887 gcpy self.dxfarc(kh\_tool\_num, oXpos, oYpos, self.tool\_radius(kh\_tool\_num, 7), 180, 270)

888 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:

889 gcpy #pre-calculate needed values

890 gcpy r = self.tool\_radius(kh\_tool\_num, 7)

891 gcpy # print(r)

892 gcpy rt = self.tool\_radius(kh\_tool\_num, 1)

893 gcpy # print(rt)

894 gcpy ro = math.sqrt((self.tool\_radius(kh\_tool\_num, 1))\*\*2-(self.tool\_radius(kh\_tool\_num, 7))\*\*2)

895 gcpy # print(ro)

896 gcpy angle = math.degrees(math.acos(ro/rt))

897 gcpy #Outlines of entry hole and slot

898 gcpy if (kh\_angle == 0):

899 gcpy #Lower left of entry hole

900 gcpy self.dxfarc(kh\_tool\_num, self.xpos(), self.ypos(), self.tool\_radius(kh\_tool\_num, 1), 180, 270)

901 gcpy #Upper left of entry hole

902 gcpy self.dxfarc(kh\_tool\_num, self.xpos(), self.ypos(), self.tool\_radius(kh\_tool\_num, 1), 90, 180)

903 gcpy #Upper right of entry hole

904 gcpy # self.dxfarc(kh\_tool\_num, self.xpos(), self.ypos(), rt, 41.810, 90)

905 gcpy self.dxfarc(kh\_tool\_num, self.xpos(), self.ypos(), rt, angle, 90)

906 gcpy #Lower right of entry hole

907 gcpy self.dxfarc(kh\_tool\_num, self.xpos(), self.ypos(), rt, 270, 360-angle)

908 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))))

909 gcpy #Actual line of cut

910 gcpy # self.dxfline(kh\_tool\_num, self.xpos(), self.ypos(), self.xpos()+kh\_distance, self.ypos())

```

911 gcpy #upper right of end of slot (kh_max_depth+4.36))/2
912 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)
913 gcpy #lower right of end of slot
914 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)
915 gcpy #upper right slot
916 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))
917 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)
918 gcpy #end position at top of slot
919 gcpy #lower right slot
920 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))
921 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)
922 gcpy #end position at top of slot
923 gcpy #          hull(){
924 gcpy #              translate([xpos(), ypos(), zpos()]){
925 gcpy #                  keyhole_shaft(6.35, 9.525);
926 gcpy #              }
927 gcpy #              translate([xpos(), ypos(), zpos()-kh_max_depth]){
928 gcpy #                  keyhole_shaft(6.35, 9.525);
929 gcpy #              }
930 gcpy #          }
931 gcpy #          hull(){
932 gcpy #              translate([xpos(), ypos(), zpos()-kh_max_depth]){
933 gcpy #                  keyhole_shaft(6.35, 9.525);
934 gcpy #              }
935 gcpy #              translate([xpos()+kh_distance, ypos(), zpos()-kh_max_depth])
{
936 gcpy #                  keyhole_shaft(6.35, 9.525);
937 gcpy #              }
938 gcpy #          }
939 gcpy #          cutwithfeed(getxpos(), getypos(), -kh_max_depth, feed);
940 gcpy #          cutwithfeed(getxpos()+kh_distance, getypos(), -kh_max_depth,
                        feed);
941 gcpy #          setxpos(getxpos()-kh_distance);
942 gcpy #          } else if (kh_angle > 0 && kh_angle < 90) {
943 gcpy #          //echo(kh_angle);
944 gcpy #          dxsfarc(getxpos(), getypos(), tool_diameter(KH_tool_num, (
                        kh_max_depth))/2, 90+kh_angle, 180+kh_angle, KH_tool_num);
945 gcpy #          dxsfarc(getxpos(), getypos(), tool_diameter(KH_tool_num, (
                        kh_max_depth))/2, 180+kh_angle, 270+kh_angle, KH_tool_num);
946 gcpy #          dxsfarc(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);
947 gcpy #          dxsfarc(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);
948 gcpy #          dxsfarc(getxpos()+(kh_distance*cos(kh_angle)),
949 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);
950 gcpy #          dxsfarc(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);
951 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))),
952 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))),
953 gcpy # getxpos()+((kh_distance*cos(kh_angle))-((tool_diameter(KH_tool_num
        , (kh_max_depth+4.36))/2)*sin(kh_angle)),
954 gcpy # getypos()+((kh_distance*sin(kh_angle))+((tool_diameter(KH_tool_num
        , (kh_max_depth+4.36))/2)*cos(kh_angle)), KH_tool_num);
955 gcpy #//echo("a", tool_diameter(KH_tool_num, (kh_max_depth+4.36))/2);
956 gcpy #//echo("c", tool_diameter(KH_tool_num, (kh_max_depth))/2);
957 gcpy #echo("Aangle", asin((tool_diameter(KH_tool_num, (kh_max_depth
        +4.36))/2)/(tool_diameter(KH_tool_num, (kh_max_depth))/2)));
958 gcpy #//echo(kh_angle);
959 gcpy # cutwithfeed(getxpos()+((kh_distance*cos(kh_angle)), getypos()+((
        kh_distance*sin(kh_angle)), -kh_max_depth, feed);
960 gcpy #          toolpath = toolpath.union(self.cutline(self.xpos()+
        kh_distance, self.ypos(), -kh_max_depth))
961 gcpy          elif (kh_angle == 90):
962 gcpy #Lower left of entry hole
963 gcpy          self.dxfarc(kh_tool_num, oXpos, oYpos, self.tool_radius
        (kh_tool_num, 1), 180, 270)
964 gcpy #Lower right of entry hole
965 gcpy          self.dxfarc(kh_tool_num, oXpos, oYpos, self.tool_radius
        (kh_tool_num, 1), 270, 360)
966 gcpy #left slot
967 gcpy          self.dxfline(kh_tool_num, oXpos-r, oYpos+ro, oXpos-r,
        oYpos+kh_distance)
968 gcpy #right slot
969 gcpy          self.dxfline(kh_tool_num, oXpos+r, oYpos+ro, oXpos+r,
        oYpos+kh_distance)
970 gcpy #upper left of end of slot
971 gcpy          self.dxfarc(kh_tool_num, oXpos, oYpos+kh_distance, r,
        90, 180)
972 gcpy #upper right of end of slot
973 gcpy          self.dxfarc(kh_tool_num, oXpos, oYpos+kh_distance, r,
        0, 90)
974 gcpy #Upper right of entry hole
975 gcpy          self.dxfarc(kh_tool_num, oXpos, oYpos, rt, 0, 90-angle)
976 gcpy #Upper left of entry hole
977 gcpy          self.dxfarc(kh_tool_num, oXpos, oYpos, rt, 90+angle,
        180)
978 gcpy #          toolpath = toolpath.union(self.cutline(oXpos, oYpos+
        kh_distance, -kh_max_depth))
979 gcpy          elif (kh_angle == 180):
980 gcpy #Lower right of entry hole
981 gcpy          self.dxfarc(kh_tool_num, oXpos, oYpos, self.tool_radius
        (kh_tool_num, 1), 270, 360)
982 gcpy #Upper right of entry hole
983 gcpy          self.dxfarc(kh_tool_num, oXpos, oYpos, self.tool_radius
        (kh_tool_num, 1), 0, 90)
984 gcpy #Upper left of entry hole
985 gcpy          self.dxfarc(kh_tool_num, oXpos, oYpos, rt, 90, 180-
        angle)
986 gcpy #Lower left of entry hole
987 gcpy          self.dxfarc(kh_tool_num, oXpos, oYpos, rt, 180+angle,
        270)
988 gcpy #upper slot
989 gcpy          self.dxfline(kh_tool_num, oXpos-ro, oYpos-r, oXpos-
        kh_distance, oYpos-r)
990 gcpy #lower slot
991 gcpy          self.dxfline(kh_tool_num, oXpos-ro, oYpos+r, oXpos-
        kh_distance, oYpos+r)
992 gcpy #upper left of end of slot
993 gcpy          self.dxfarc(kh_tool_num, oXpos-kh_distance, oYpos, r,
        90, 180)
994 gcpy #lower left of end of slot
995 gcpy          self.dxfarc(kh_tool_num, oXpos-kh_distance, oYpos, r,
        180, 270)
996 gcpy #          toolpath = toolpath.union(self.cutline(oXpos-
        kh_distance, oYpos, -kh_max_depth))
997 gcpy          elif (kh_angle == 270):
998 gcpy #Upper left of entry hole
999 gcpy          self.dxfarc(kh_tool_num, oXpos, oYpos, self.tool_radius
        (kh_tool_num, 1), 90, 180)
1000 gcpy #Upper right of entry hole
1001 gcpy          self.dxfarc(kh_tool_num, oXpos, oYpos, self.tool_radius
        (kh_tool_num, 1), 0, 90)
1002 gcpy #left slot
1003 gcpy          self.dxfline(kh_tool_num, oXpos-r, oYpos-ro, oXpos-r,
        oYpos-kh_distance)

```

```

1004 gcpy #right slot
1005 gcpy          self.dxfline(kh_tool_num, oXpos+r, oYpos-ro, oXpos+r,
                             oYpos-kh_distance)
1006 gcpy #lower left of end of slot
1007 gcpy          self.dxfarc(kh_tool_num, oXpos, oYpos-kh_distance, r,
                             180, 270)
1008 gcpy #lower right of end of slot
1009 gcpy          self.dxfarc(kh_tool_num, oXpos, oYpos-kh_distance, r,
                             270, 360)
1010 gcpy #lower right of entry hole
1011 gcpy          self.dxfarc(kh_tool_num, oXpos, oYpos, rt, 180, 270-
                             angle)
1012 gcpy #lower left of entry hole
1013 gcpy          self.dxfarc(kh_tool_num, oXpos, oYpos, rt, 270+angle,
                             360)
1014 gcpy #          toolpath = toolpath.union(self.cutline(oXpos, oYpos-
                             kh_distance, -kh_max_depth))
1015 gcpy #          print(self.zpos())
1016 gcpy #          self.setxpos(oXpos)
1017 gcpy #          self.setypos(oYpos)
1018 gcpy #          if self.generatepaths == False:
1019 gcpy #              return toolpath
1020 gcpy
1021 gcpy # } else if (kh_angle == 90) {
1022 gcpy # //Lower left of entry hole
1023 gcpy # dxfarc(getxpos(), getypos(), 9.525/2, 180, 270, KH_tool_num);
1024 gcpy # //Lower right of entry hole
1025 gcpy # dxfarc(getxpos(), getypos(), 9.525/2, 270, 360, KH_tool_num);
1026 gcpy # //Upper right of entry hole
1027 gcpy # dxfarc(getxpos(), getypos(), 9.525/2, 0, acos(tool_diameter(
KH_tool_num, 5)/tool_diameter(KH_tool_num, 1)), KH_tool_num);
1028 gcpy # //Upper left of entry hole
1029 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
);
1030 gcpy # //Actual line of cut
1031 gcpy # dxfline(getxpos(), getypos(), getxpos(), getypos()+kh_distance
);
1032 gcpy # //upper right of slot
1033 gcpy # dxfarc(getxpos(), getypos()+kh_distance, tool_diameter(
KH_tool_num, (kh_max_depth+4.36))/2, 0, 90, KH_tool_num);
1034 gcpy # //upper left of slot
1035 gcpy # dxfarc(getxpos(), getypos()+kh_distance, tool_diameter(
KH_tool_num, (kh_max_depth+6.35))/2, 90, 180, KH_tool_num);
1036 gcpy # //right of slot
1037 gcpy # dxfline(
1038 gcpy #     getxpos()+tool_diameter(KH_tool_num, (kh_max_depth))/2,
1039 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,
1040 gcpy #     getxpos()+tool_diameter(KH_tool_num, (kh_max_depth))/2,
1041 gcpy #     //end position at top of slot
1042 gcpy #     getypos()+kh_distance,
1043 gcpy #     KH_tool_num);
1044 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);
1045 gcpy # hull(){
1046 gcpy #     translate([xpos(), ypos(), zpos()]){
1047 gcpy #         keyhole_shaft(6.35, 9.525);
1048 gcpy #     }
1049 gcpy #     translate([xpos(), ypos(), zpos()-kh_max_depth]){
1050 gcpy #         keyhole_shaft(6.35, 9.525);
1051 gcpy #     }
1052 gcpy # }
1053 gcpy # hull(){
1054 gcpy #     translate([xpos(), ypos(), zpos()-kh_max_depth]){
1055 gcpy #         keyhole_shaft(6.35, 9.525);
1056 gcpy #     }
1057 gcpy #     translate([xpos(), ypos()+kh_distance, zpos()-kh_max_depth])
{
1058 gcpy #         keyhole_shaft(6.35, 9.525);
1059 gcpy #     }
1060 gcpy # }
1061 gcpy # cutwithfeed(getxpos(), getypos(), -kh_max_depth, feed);
1062 gcpy # cutwithfeed(getxpos(), getypos()+kh_distance, -kh_max_depth,

```

```

        feed);
1063 gcpy #   setypos(getypos()-kh_distance);
1064 gcpy # } else if (kh_angle == 180) {
1065 gcpy #   //Lower right of entry hole
1066 gcpy #   dxfarc(getxpos(), getypos(), 9.525/2, 270, 360, KH_tool_num);
1067 gcpy #   //Upper right of entry hole
1068 gcpy #   dxfarc(getxpos(), getypos(), 9.525/2, 0, 90, KH_tool_num);
1069 gcpy #   //Upper left of entry hole
1070 gcpy #   dxfarc(getxpos(), getypos(), 9.525/2, 90, 90+acos(
tool_diameter(KH_tool_num, 5)/tool_diameter(KH_tool_num, 1)),
KH_tool_num);
1071 gcpy #   //Lower left of entry hole
1072 gcpy #   dxfarc(getxpos(), getypos(), 9.525/2, 270-acos(tool_diameter(
KH_tool_num, 5)/tool_diameter(KH_tool_num, 1)), 270, KH_tool_num
);
1073 gcpy #   //upper left of slot
1074 gcpy #   dxfarc(getxpos()-kh_distance, getypos(), tool_diameter(
KH_tool_num, (kh_max_depth+6.35))/2, 90, 180, KH_tool_num);
1075 gcpy #   //lower left of slot
1076 gcpy #   dxfarc(getxpos()-kh_distance, getypos(), tool_diameter(
KH_tool_num, (kh_max_depth+6.35))/2, 180, 270, KH_tool_num);
1077 gcpy #   //Actual line of cut
1078 gcpy #   dxfline(getxpos(), getypos(), getxpos()-kh_distance, getypos()
);
1079 gcpy #   //upper left slot
1080 gcpy #   dxfline(
1081 gcpy #       getxpos()-(sqrt((tool_diameter(KH_tool_num, 1)^2)-(
tool_diameter(KH_tool_num, 5)^2))/2),
1082 gcpy #       getypos()+tool_diameter(KH_tool_num, (kh_max_depth))/2,
//( (kh_max_depth-6.34))/2)^2-(tool_diameter(KH_tool_num, (
kh_max_depth-6.34))/2)^2,
1083 gcpy #       getxpos()-kh_distance,
1084 gcpy #   //end position at top of slot
1085 gcpy #       getypos()+tool_diameter(KH_tool_num, (kh_max_depth))/2,
1086 gcpy #       KH_tool_num);
1087 gcpy #   //lower right slot
1088 gcpy #   dxfline(
1089 gcpy #       getxpos()-(sqrt((tool_diameter(KH_tool_num, 1)^2)-(
tool_diameter(KH_tool_num, 5)^2))/2),
1090 gcpy #       getypos()-tool_diameter(KH_tool_num, (kh_max_depth))/2,
//( (kh_max_depth-6.34))/2)^2-(tool_diameter(KH_tool_num, (
kh_max_depth-6.34))/2)^2,
1091 gcpy #       getxpos()-kh_distance,
1092 gcpy #   //end position at top of slot
1093 gcpy #       getypos()-tool_diameter(KH_tool_num, (kh_max_depth))/2,
1094 gcpy #       KH_tool_num);
1095 gcpy #   hull(){
1096 gcpy #       translate([xpos(), ypos(), zpos()]){
1097 gcpy #           keyhole_shaft(6.35, 9.525);
1098 gcpy #       }
1099 gcpy #       translate([xpos(), ypos(), zpos()-kh_max_depth]){
1100 gcpy #           keyhole_shaft(6.35, 9.525);
1101 gcpy #       }
1102 gcpy #   }
1103 gcpy #   hull(){
1104 gcpy #       translate([xpos(), ypos(), zpos()-kh_max_depth]){
1105 gcpy #           keyhole_shaft(6.35, 9.525);
1106 gcpy #       }
1107 gcpy #       translate([xpos()-kh_distance, ypos(), zpos()-kh_max_depth])
{
1108 gcpy #           keyhole_shaft(6.35, 9.525);
1109 gcpy #       }
1110 gcpy #   }
1111 gcpy #   cutwithfeed(getxpos(), getypos(), -kh_max_depth, feed);
1112 gcpy #   cutwithfeed(getxpos()-kh_distance, getypos(), -kh_max_depth,
feed);
1113 gcpy #   setxpos(getxpos()+kh_distance);
1114 gcpy # } else if (kh_angle == 270) {
1115 gcpy #   //Upper right of entry hole
1116 gcpy #   dxfarc(getxpos(), getypos(), 9.525/2, 0, 90, KH_tool_num);
1117 gcpy #   //Upper left of entry hole
1118 gcpy #   dxfarc(getxpos(), getypos(), 9.525/2, 90, 180, KH_tool_num);
1119 gcpy #   //lower right of slot
1120 gcpy #   dxfarc(getxpos(), getypos()-kh_distance, tool_diameter(
KH_tool_num, (kh_max_depth+4.36))/2, 270, 360, KH_tool_num);
1121 gcpy #   //lower left of slot
1122 gcpy #   dxfarc(getxpos(), getypos()-kh_distance, tool_diameter(
KH_tool_num, (kh_max_depth+4.36))/2, 180, 270, KH_tool_num);

```

```

1123 gcpy # //Actual line of cut
1124 gcpy # dxfline(getxpos(), getypos(), getxpos(), getypos()-kh_distance
);
1125 gcpy # //right of slot
1126 gcpy # dxfline(
1127 gcpy #     getxpos()+tool_diameter(KH_tool_num, (kh_max_depth))/2,
1128 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,
1129 gcpy #     getxpos()+tool_diameter(KH_tool_num, (kh_max_depth))/2,
1130 gcpy #     //end position at top of slot
1131 gcpy #     getypos()-kh_distance,
1132 gcpy #     KH_tool_num);
1133 gcpy # //left of slot
1134 gcpy # dxfline(
1135 gcpy #     getxpos()-tool_diameter(KH_tool_num, (kh_max_depth))/2,
1136 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,
1137 gcpy #     getxpos()-tool_diameter(KH_tool_num, (kh_max_depth))/2,
1138 gcpy #     //end position at top of slot
1139 gcpy #     getypos()-kh_distance,
1140 gcpy #     KH_tool_num);
1141 gcpy # //Lower right of entry hole
1142 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
);
1143 gcpy # //Lower left of entry hole
1144 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);
1145 gcpy # hull(){
1146 gcpy #     translate([xpos(), ypos(), zpos()]){
1147 gcpy #         keyhole_shaft(6.35, 9.525);
1148 gcpy #     }
1149 gcpy #     translate([xpos(), ypos(), zpos()-kh_max_depth]){
1150 gcpy #         keyhole_shaft(6.35, 9.525);
1151 gcpy #     }
1152 gcpy # }
1153 gcpy # hull(){
1154 gcpy #     translate([xpos(), ypos(), zpos()-kh_max_depth]){
1155 gcpy #         keyhole_shaft(6.35, 9.525);
1156 gcpy #     }
1157 gcpy #     translate([xpos(), ypos()-kh_distance, zpos()-kh_max_depth])
{
1158 gcpy #         keyhole_shaft(6.35, 9.525);
1159 gcpy #     }
1160 gcpy # }
1161 gcpy # cutwithfeed(getxpos(), getypos(), -kh_max_depth, feed);
1162 gcpy # cutwithfeed(getxpos(), getypos()-kh_distance, -kh_max_depth,
feed);
1163 gcpy # setypos(getypos()+kh_distance);
1164 gcpy # }
1165 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.

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:

```

1167 gcpy     def cut_pins(self, Joint_Width, stockZthickness,
1168 gcpy         Number_of_Dovetails, Spacing, Proportion, DTT_diameter,
DTT_angle):
1169 gcpy         DTO = math.tan(math.radians(DTT_angle)) * (stockZthickness
* Proportion)
1170 gcpy         DTR = DTT_diameter/2 - DTO
1171 gcpy         cpr = self.rapidXY(0, stockZthickness + Spacing/2)
ctp = self.cutlinedxfgc(self.xpos(), self.ypos(), -
stockZthickness * Proportion)

```



```

1172 gcpy #         ctp = ctp.union(self.cutlinedxfgc(Joint_Width / (
                Number_of_Dovetails * 2), self.ypos(), -stockZthickness *
                Proportion))
1173 gcpy         i = 1
1174 gcpy         while i < Number_of_Dovetails * 2:
1175 gcpy #             print(i)
1176 gcpy             ctp = ctp.union(self.cutlinedxfgc(i * (Joint_Width / (
                Number_of_Dovetails * 2)), self.ypos(), -
                stockZthickness * Proportion))
1177 gcpy             ctp = ctp.union(self.cutlinedxfgc(i * (Joint_Width / (
                Number_of_Dovetails * 2)), (stockZthickness +
                Spacing) + (stockZthickness * Proportion) - (
                DTT_diameter/2), -(stockZthickness * Proportion)))
1178 gcpy             ctp = ctp.union(self.cutlinedxfgc(i * (Joint_Width / (
                Number_of_Dovetails * 2)), stockZthickness + Spacing
                /2, -(stockZthickness * Proportion)))
1179 gcpy             ctp = ctp.union(self.cutlinedxfgc((i + 1) * (
                Joint_Width / (Number_of_Dovetails * 2)),
                stockZthickness + Spacing/2, -(stockZthickness *
                Proportion)))
1180 gcpy             self.dxfrectangleround(self.currenttoolnumber(),
1181 gcpy                 i * (Joint_Width / (Number_of_Dovetails * 2))-DTR,
1182 gcpy                 stockZthickness + (Spacing/2) - DTR,
1183 gcpy                 DTR * 2,
1184 gcpy                 (stockZthickness * Proportion) + Spacing/2 + DTR *
                    2 - (DTT_diameter/2),
1185 gcpy                 DTR)
1186 gcpy             i += 2
1187 gcpy             self.rapidZ(0)
1188 gcpy             return ctp

```

---

and

```

1181 gcpy     def cut_tails(self, Joint_Width, stockZthickness,
                Number_of_Dovetails, Spacing, Proportion, DTT_diameter,
                DTT_angle):
1182 gcpy         DT0 = math.tan(math.radians(DTT_angle)) * (stockZthickness
                * Proportion)
1183 gcpy         DTR = DTT_diameter/2 - DT0
1184 gcpy         cpr = self.rapidXY(0, 0)
1185 gcpy         ctp = self.cutlinedxfgc(self.xpos(), self.ypos(), -
                stockZthickness * Proportion)
1186 gcpy         ctp = ctp.union(self.cutlinedxfgc(
1187 gcpy             Joint_Width / (Number_of_Dovetails * 2) - (DTT_diameter
                - DT0),
1188 gcpy             self.ypos(),
1189 gcpy             -stockZthickness * Proportion))
1190 gcpy         i = 1
1191 gcpy         while i < Number_of_Dovetails * 2:
1192 gcpy             ctp = ctp.union(self.cutlinedxfgc(
1193 gcpy                 i * (Joint_Width / (Number_of_Dovetails * 2)) - (
                    DTT_diameter - DT0),
1194 gcpy                 stockZthickness * Proportion - DTT_diameter / 2,
1195 gcpy                 -(stockZthickness * Proportion)))
1196 gcpy             ctp = ctp.union(self.cutarcCWdxf(180, 90,
1197 gcpy                 i * (Joint_Width / (Number_of_Dovetails * 2)),
1198 gcpy                 stockZthickness * Proportion - DTT_diameter / 2,
1199 gcpy #                 self.ypos(),
2000 gcpy                 DTT_diameter - DT0, 0, 1))
2001 gcpy             ctp = ctp.union(self.cutarcCWdxf(90, 0,
2002 gcpy                 i * (Joint_Width / (Number_of_Dovetails * 2)),
2003 gcpy                 stockZthickness * Proportion - DTT_diameter / 2,
2004 gcpy                 DTT_diameter - DT0, 0, 1))
2005 gcpy             ctp = ctp.union(self.cutlinedxfgc(
2006 gcpy                 i * (Joint_Width / (Number_of_Dovetails * 2)) + (
                    DTT_diameter - DT0),
2007 gcpy                 0,
2008 gcpy                 -(stockZthickness * Proportion)))
2009 gcpy             ctp = ctp.union(self.cutlinedxfgc(
2010 gcpy                 (i + 2) * (Joint_Width / (Number_of_Dovetails * 2))
                    - (DTT_diameter - DT0),
2011 gcpy                 0,
2012 gcpy                 -(stockZthickness * Proportion)))
2013 gcpy             i += 2
2014 gcpy             self.rapidZ(0)
2015 gcpy             self.rapidXY(0, 0)
2016 gcpy             ctp = ctp.union(self.cutlinedxfgc(self.xpos(), self.ypos(),
                -stockZthickness * Proportion))

```

```
1217 gcpy      self.dxfarc(self.currenttoolnumber(), 0, 0, DTR, 180, 270)
1218 gcpy      self.dxfline(self.currenttoolnumber(), -DTR, 0, -DTR,
                        stockZthickness + DTR)
1219 gcpy      self.dxfarc(self.currenttoolnumber(), 0, stockZthickness +
                        DTR, DTR, 90, 180)
1220 gcpy      self.dxfline(self.currenttoolnumber(), 0, stockZthickness +
                        DTR * 2, Joint_Width, stockZthickness + DTR * 2)
1221 gcpy      i = 0
1222 gcpy      while i < Number_of_Dovetails * 2:
1223 gcpy          ctp = ctp.union(self.cutline(i * (Joint_Width / (
                        Number_of_Dovetails * 2)), stockZthickness + DTO, -(
                        stockZthickness * Proportion)))
1224 gcpy          ctp = ctp.union(self.cutline((i+2) * (Joint_Width / (
                        Number_of_Dovetails * 2)), stockZthickness + DTO, -(
                        stockZthickness * Proportion)))
1225 gcpy          ctp = ctp.union(self.cutline((i+2) * (Joint_Width / (
                        Number_of_Dovetails * 2)), 0, -(stockZthickness *
                        Proportion)))
1226 gcpy      self.dxfarc(self.currenttoolnumber(), i * (Joint_Width
                        / (Number_of_Dovetails * 2)), 0, DTR, 270, 360)
1227 gcpy      self.dxfline(self.currenttoolnumber(),
1228 gcpy          i * (Joint_Width / (Number_of_Dovetails * 2)) + DTR
                        ,
1229 gcpy          0,
1230 gcpy          i * (Joint_Width / (Number_of_Dovetails * 2)) + DTR
                        , stockZthickness * Proportion - DTT_diameter /
                        2)
1231 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)
1232 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)
1233 gcpy      self.dxfline(self.currenttoolnumber(),
1234 gcpy          (i + 2) * (Joint_Width / (Number_of_Dovetails * 2))
                        - DTR,
1235 gcpy          0,
1236 gcpy          (i + 2) * (Joint_Width / (Number_of_Dovetails * 2))
                        - DTR, stockZthickness * Proportion -
                        DTT_diameter / 2)
1237 gcpy      self.dxfarc(self.currenttoolnumber(), (i + 2) * (
                        Joint_Width / (Number_of_Dovetails * 2)), 0, DTR,
                        180, 270)
1238 gcpy      i += 2
1239 gcpy      self.dxfarc(self.currenttoolnumber(), Joint_Width,
                        stockZthickness + DTR, DTR, 0, 90)
1240 gcpy      self.dxfline(self.currenttoolnumber(), Joint_Width + DTR,
                        stockZthickness + DTR, Joint_Width + DTR, 0)
1241 gcpy      self.dxfarc(self.currenttoolnumber(), Joint_Width, 0, DTR,
                        270, 360)
1242 gcpy      return ctp
```

which are used as:

toolpaths = gcp.cut\_pins(stockXwidth, stockZthickness, Number\_of\_Dovetails, Spacing, Proportion, DTT\_diameter)

toolpaths = toolpaths.union(gcp.cut\_tails(stockXwidth, stockZthickness, Number\_of\_Dovetails, Spacing, Proportion, DTT\_diameter))

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

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.

```
1207 gcpy      def stockandtoolpaths(self, option = "stockandtoolpaths"):
1208 gcpy          if option == "stock":
1209 gcpy              if self.generatepaths == False:
1210 gcpy                  output(self.stock)
1211 gcpy              print("Outputting stock")
1212 gcpy          else:
1213 gcpy              return self.stock
```

```
1214 gcpy          elif option == "toolpaths":
1215 gcpy              if self.generatepaths == False:
1216 gcpy                  output(self.toolpaths)
1217 gcpy              else:
1218 gcpy                  return self.toolpaths
1219 gcpy          elif option == "rapids":
1220 gcpy              if self.generatepaths == False:
1221 gcpy                  output(self.rapids)
1222 gcpy              else:
1223 gcpy                  return self.rapids
1224 gcpy          else:
1225 gcpy              part = self.stock.difference(self.toolpaths)
1226 gcpy              if self.generatepaths == False:
1227 gcpy                  output(part)
1228 gcpy              else:
1229 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)	writacomment(...)
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.

---

```

1231 gcpy      def writegc(self, *arguments):
1232 gcpy          if self.generategcode == True:
1233 gcpy              line_to_write = ""
1234 gcpy              for element in arguments:
1235 gcpy                  line_to_write += element
1236 gcpy              self.gc.write(line_to_write)
1237 gcpy              self.gc.write("\n")
1238 gcpy
1239 gcpy      def writedxf(self, toolnumber, *arguments):
1240 gcpy #          global dxfclosed
1241 gcpy          line_to_write = ""
1242 gcpy          for element in arguments:
1243 gcpy              line_to_write += element
1244 gcpy          if self.generateddxf == True:
1245 gcpy              if self.dxfclosed == False:
1246 gcpy                  self.dxf.write(line_to_write)
1247 gcpy                  self.dxf.write("\n")
1248 gcpy          if self.generateddxfs == True:
1249 gcpy              self.writedxfs(toolnumber, line_to_write)
1250 gcpy
1251 gcpy      def writedxfs(self, toolnumber, line_to_write):
1252 gcpy #          print("Processing writing toolnumber", toolnumber)
1253 gcpy #          line_to_write = ""
1254 gcpy #          for element in arguments:
1255 gcpy #              line_to_write += element
1256 gcpy          if (toolnumber == 0):
1257 gcpy              return
1258 gcpy          elif self.generateddxfs == True:
1259 gcpy              if (self.large_square_tool_num == toolnumber):
1260 gcpy                  self.dxf_lgsq.write(line_to_write)
1261 gcpy                  self.dxf_lgsq.write("\n")
1262 gcpy              if (self.small_square_tool_num == toolnumber):
1263 gcpy                  self.dxf_ssq.write(line_to_write)
1264 gcpy                  self.dxf_ssq.write("\n")
1265 gcpy              if (self.large_ball_tool_num == toolnumber):
1266 gcpy                  self.dxf_lgb.write(line_to_write)
1267 gcpy                  self.dxf_lgb.write("\n")
1268 gcpy              if (self.small_ball_tool_num == toolnumber):
1269 gcpy                  self.dxf_smb.write(line_to_write)
1270 gcpy                  self.dxf_smb.write("\n")
1271 gcpy              if (self.large_V_tool_num == toolnumber):
1272 gcpy                  self.dxf_lgv.write(line_to_write)
1273 gcpy                  self.dxf_lgv.write("\n")
1274 gcpy              if (self.small_V_tool_num == toolnumber):
1275 gcpy                  self.dxf_smv.write(line_to_write)
1276 gcpy                  self.dxf_smv.write("\n")
1277 gcpy              if (self.DT_tool_num == toolnumber):
1278 gcpy                  self.dxf_DT.write(line_to_write)
1279 gcpy                  self.dxf_DT.write("\n")
1280 gcpy              if (self.KH_tool_num == toolnumber):
1281 gcpy                  self.dxf_KH.write(line_to_write)
1282 gcpy                  self.dxf_KH.write("\n")
1283 gcpy              if (self.Roundover_tool_num == toolnumber):
1284 gcpy                  self.dxf_Rt.write(line_to_write)
1285 gcpy                  self.dxf_Rt.write("\n")
1286 gcpy              if (self.MISC_tool_num == toolnumber):
1287 gcpy                  self.dxf_Mt.write(line_to_write)

```

```
1288 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:

```
1290 gcpy def opengcodefile(self, basefilename = "export",
1291 gcpy     currenttoolnum = 102,
1292 gcpy     toolradius = 3.175,
1293 gcpy     plunge = 400,
1294 gcpy     feed = 1600,
1295 gcpy     speed = 10000
1296 gcpy ):
1297 gcpy     self.basefilename = basefilename
1298 gcpy     self.currenttoolnum = currenttoolnum
1299 gcpy     self.toolradius = toolradius
1300 gcpy     self.plunge = plunge
1301 gcpy     self.feed = feed
1302 gcpy     self.speed = speed
1303 gcpy     if self.generategcode == True:
1304 gcpy         self.gcodefilename = basefilename + ".nc"
1305 gcpy         self.gc = open(self.gcodefilename, "w")
1306 gcpy
1307 gcpy def opendxfile(self, basefilename = "export"):
1308 gcpy     self.basefilename = basefilename
1309 gcpy #     global generatedxfs
1310 gcpy #     global dxfclosed
1311 gcpy     self.dxfclosed = False
1312 gcpy     if self.generatedxif == True:
1313 gcpy         self.generatedxfs = False
1314 gcpy         self.dxffilename = basefilename + ".dxf"
1315 gcpy         self.dxf = open(self.dxffilename, "w")
1316 gcpy         self.dxfpreamble(-1)
1317 gcpy
1318 gcpy def opendxfiles(self, basefilename = "export",
1319 gcpy     large_square_tool_num = 0,
1320 gcpy     small_square_tool_num = 0,
1321 gcpy     large_ball_tool_num = 0,
1322 gcpy     small_ball_tool_num = 0,
1323 gcpy     large_V_tool_num = 0,
1324 gcpy     small_V_tool_num = 0,
1325 gcpy     DT_tool_num = 0,
1326 gcpy     KH_tool_num = 0,
```

```

1327 gcpy                                Roundover_tool_num = 0,
1328 gcpy                                MISC_tool_num = 0):
1329 gcpy #                                global generatedxfs
1330 gcpy                                self.basefilename = basefilename
1331 gcpy                                self.generatedxfs = True
1332 gcpy                                self.large_square_tool_num = large_square_tool_num
1333 gcpy                                self.small_square_tool_num = small_square_tool_num
1334 gcpy                                self.large_ball_tool_num = large_ball_tool_num
1335 gcpy                                self.small_ball_tool_num = small_ball_tool_num
1336 gcpy                                self.large_V_tool_num = large_V_tool_num
1337 gcpy                                self.small_V_tool_num = small_V_tool_num
1338 gcpy                                self.DT_tool_num = DT_tool_num
1339 gcpy                                self.KH_tool_num = KH_tool_num
1340 gcpy                                self.Roundover_tool_num = Roundover_tool_num
1341 gcpy                                self.MISC_tool_num = MISC_tool_num
1342 gcpy                                if self.generatedxf == True:
1343 gcpy                                    if (large_square_tool_num > 0):
1344 gcpy                                        self.dxfllsqfilename = basefilename + str(
                                            large_square_tool_num) + ".dxf"
1345 gcpy #                                        print("Opening ", str(self.dxfllsqfilename))
1346 gcpy                                        self.dxfllsq = open(self.dxfllsqfilename, "w")
1347 gcpy                                if (small_square_tool_num > 0):
1348 gcpy #                                    print("Opening small square")
1349 gcpy                                        self.dxfllsqfilename = basefilename + str(
                                            small_square_tool_num) + ".dxf"
1350 gcpy                                        self.dxfllsq = open(self.dxfllsqfilename, "w")
1351 gcpy                                if (large_ball_tool_num > 0):
1352 gcpy #                                    print("Opening large ball")
1353 gcpy                                        self.dxfllblfilename = basefilename + str(
                                            large_ball_tool_num) + ".dxf"
1354 gcpy                                        self.dxfllbl = open(self.dxfllblfilename, "w")
1355 gcpy                                if (small_ball_tool_num > 0):
1356 gcpy #                                    print("Opening small ball")
1357 gcpy                                        self.dxfllblfilename = basefilename + str(
                                            small_ball_tool_num) + ".dxf"
1358 gcpy                                        self.dxfllbl = open(self.dxfllblfilename, "w")
1359 gcpy                                if (large_V_tool_num > 0):
1360 gcpy #                                    print("Opening large V")
1361 gcpy                                        self.dxfllVfilename = basefilename + str(
                                            large_V_tool_num) + ".dxf"
1362 gcpy                                        self.dxfllV = open(self.dxfllVfilename, "w")
1363 gcpy                                if (small_V_tool_num > 0):
1364 gcpy #                                    print("Opening small V")
1365 gcpy                                        self.dxfllVfilename = basefilename + str(
                                            small_V_tool_num) + ".dxf"
1366 gcpy                                        self.dxfllV = open(self.dxfllVfilename, "w")
1367 gcpy                                if (DT_tool_num > 0):
1368 gcpy #                                    print("Opening DT")
1369 gcpy                                        self.dxfDTfilename = basefilename + str(DT_tool_num
                                            ) + ".dxf"
1370 gcpy                                        self.dxfDT = open(self.dxfDTfilename, "w")
1371 gcpy                                if (KH_tool_num > 0):
1372 gcpy #                                    print("Opening KH")
1373 gcpy                                        self.dxfKHfilename = basefilename + str(KH_tool_num
                                            ) + ".dxf"
1374 gcpy                                        self.dxfKH = open(self.dxfKHfilename, "w")
1375 gcpy                                if (Roundover_tool_num > 0):
1376 gcpy #                                    print("Opening Rt")
1377 gcpy                                        self.dxfRtfilename = basefilename + str(
                                            Roundover_tool_num) + ".dxf"
1378 gcpy                                        self.dxfRt = open(self.dxfRtfilename, "w")
1379 gcpy                                if (MISC_tool_num > 0):
1380 gcpy #                                    print("Opening Mt")
1381 gcpy                                        self.dxfMtfilename = basefilename + str(
                                            MISC_tool_num) + ".dxf"
1382 gcpy                                        self.dxfMt = open(self.dxfMtfilename, "w")

```

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

```

1383 gcpy                                if (large_square_tool_num > 0):
1384 gcpy                                    self.dxfpreamble(large_square_tool_num)
1385 gcpy                                if (small_square_tool_num > 0):
1386 gcpy                                    self.dxfpreamble(small_square_tool_num)
1387 gcpy                                if (large_ball_tool_num > 0):
1388 gcpy                                    self.dxfpreamble(large_ball_tool_num)
1389 gcpy                                if (small_ball_tool_num > 0):
1390 gcpy                                    self.dxfpreamble(small_ball_tool_num)

```

```
1391 gcpy          if (large_V_tool_num > 0):
1392 gcpy              self.dxfpreamble(large_V_tool_num)
1393 gcpy          if (small_V_tool_num > 0):
1394 gcpy              self.dxfpreamble(small_V_tool_num)
1395 gcpy          if (DT_tool_num > 0):
1396 gcpy              self.dxfpreamble(DT_tool_num)
1397 gcpy          if (KH_tool_num > 0):
1398 gcpy              self.dxfpreamble(KH_tool_num)
1399 gcpy          if (Roundover_tool_num > 0):
1400 gcpy              self.dxfpreamble(Roundover_tool_num)
1401 gcpy          if (MISC_tool_num > 0):
1402 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:

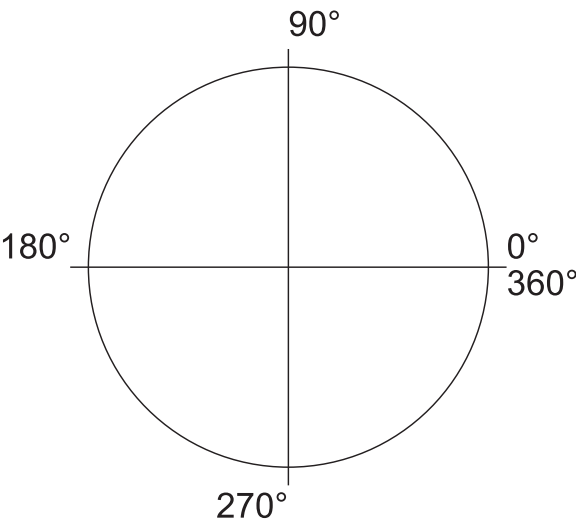
- writedxflgbl
- writedxfsmb1
- writedxflgsq
- writedxfsmsq
- writedxflgV
- writedxfsmV
- writedxfKH
- writedxfDT
- Ball nose, large (lgbl) writedxflgbl
  - Ball nose, small (smb1) writedxfsmb1
  - Square, large (lgsq) writedxflgsq
  - Square, small (smsq) writedxfsmsq
  - V, large (lgV) writedxflgV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

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.

```
1404 gcpy      def dxfpreamble(self, tn):
1405 gcpy #          self.writedxf(tn, str(tn))
1406 gcpy          self.writedxf(tn, "0")
1407 gcpy          self.writedxf(tn, "SECTION")
1408 gcpy          self.writedxf(tn, "2")
1409 gcpy          self.writedxf(tn, "ENTITIES")
```

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

- dxfline
- dxfarc
- a line dxfline
  - ARC — a notable option would be for the arc to close on itself, creating a circle: dxfarc
- DXF orders arcs counter-clockwise:





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

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

A further refinement would be to connect multiple line segments/arcs into a larger polyline, but since most CAM tools implicitly join elements on import, that is not necessary. There are three possible interactions for DXF elements and toolpaths:

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

---

```
1411 gcpy      def dxfline(self, tn, xbegin, ybegin, xend, yend):
1412 gcpy          self.writedxf(tn, "0")
1413 gcpy          self.writedxf(tn, "LWPOLYLINE")
1414 gcpy          self.writedxf(tn, "90")
1415 gcpy          self.writedxf(tn, "2")
1416 gcpy          self.writedxf(tn, "70")
1417 gcpy          self.writedxf(tn, "0")
1418 gcpy          self.writedxf(tn, "43")
1419 gcpy          self.writedxf(tn, "0")
1420 gcpy          self.writedxf(tn, "10")
1421 gcpy          self.writedxf(tn, str(xbegin))
1422 gcpy          self.writedxf(tn, "20")
1423 gcpy          self.writedxf(tn, str(ybegin))
1424 gcpy          self.writedxf(tn, "10")
1425 gcpy          self.writedxf(tn, str(xend))
1426 gcpy          self.writedxf(tn, "20")
1427 gcpy          self.writedxf(tn, str(yend))
```

---

There are specific commands for writing out the DXF and G-code files. Note that for the G-code version it will be necessary to calculate the end-position, and to determine if the arc is clockwise or no (G2 vs. G3).

---

```
1429 gcpy      def dxfarc(self, tn, xcenter, ycenter, radius, anglebegin,
1430 gcpy          endangle):
1431 gcpy          if (self.generatedxf == True):
1432 gcpy              self.writedxf(tn, "0")
1433 gcpy              self.writedxf(tn, "ARC")
1434 gcpy              self.writedxf(tn, "10")
1435 gcpy              self.writedxf(tn, str(xcenter))
1436 gcpy              self.writedxf(tn, "20")
1437 gcpy              self.writedxf(tn, str(ycenter))
1438 gcpy              self.writedxf(tn, "40")
1439 gcpy              self.writedxf(tn, str(radius))
1440 gcpy              self.writedxf(tn, "50")
1441 gcpy              self.writedxf(tn, str(anglebegin))
1442 gcpy              self.writedxf(tn, "51")
1443 gcpy              self.writedxf(tn, str(endangle))
1444 gcpy      def gcodearc(self, tn, xcenter, ycenter, radius, anglebegin,
1445 gcpy          endangle):
1446 gcpy          if (self.generategcode == True):
1447 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.

---

```
1448 gcpy      def cutarcNECCdxf(self, ex, ey, ez, xcenter, ycenter, radius):
1449 gcpy      #          global toolpath
1450 gcpy      #          toolpath = self.currenttool()
1451 gcpy      #          toolpath = toolpath.translate([self.xpos(), self.ypos(),
1452 gcpy      #          self.zpos()])
```

---

```

1452 gcpy          self.dxfarc(self.currenttoolnumber(), xcenter, ycenter,
                        radius, 0, 90)
1453 gcpy          if (self.zpos == ez):
1454 gcpy              self.settzpos(0)
1455 gcpy          else:
1456 gcpy              self.settzpos((self.zpos()-ez)/90)
1457 gcpy          # self.setxpos(ex)
1458 gcpy          # self.setypos(ey)
1459 gcpy          # self.setzpos(ez)
1460 gcpy          if self.generatepaths == True:
1461 gcpy              print("Unioning cutarcNECCdxf toolpath")
1462 gcpy              self.arcloop(1, 90, xcenter, ycenter, radius)
1463 gcpy          # self.toolpaths = self.toolpaths.union(toolpath)
1464 gcpy          else:
1465 gcpy              toolpath = self.arcloop(1, 90, xcenter, ycenter, radius
                        )
1466 gcpy          # print("Returning cutarcNECCdxf toolpath")
1467 gcpy          return toolpath
1468 gcpy
1469 gcpy          def cutarcNWCCdxf(self, ex, ey, ez, xcenter, ycenter, radius):
1470 gcpy              global toolpath
1471 gcpy              toolpath = self.currenttool()
1472 gcpy              toolpath = toolpath.translate([self.xpos(), self.ypos(),
self.zpos()])
1473 gcpy          self.dxfarc(self.currenttoolnumber(), xcenter, ycenter,
                        radius, 90, 180)
1474 gcpy          if (self.zpos == ez):
1475 gcpy              self.settzpos(0)
1476 gcpy          else:
1477 gcpy              self.settzpos((self.zpos()-ez)/90)
1478 gcpy          # self.setxpos(ex)
1479 gcpy          # self.setypos(ey)
1480 gcpy          # self.setzpos(ez)
1481 gcpy          if self.generatepaths == True:
1482 gcpy              self.arcloop(91, 180, xcenter, ycenter, radius)
1483 gcpy          # self.toolpaths = self.toolpaths.union(toolpath)
1484 gcpy          else:
1485 gcpy              toolpath = self.arcloop(91, 180, xcenter, ycenter,
                        radius)
1486 gcpy              return toolpath
1487 gcpy
1488 gcpy          def cutarcSWCCdxf(self, ex, ey, ez, xcenter, ycenter, radius):
1489 gcpy              global toolpath
1490 gcpy              toolpath = self.currenttool()
1491 gcpy              toolpath = toolpath.translate([self.xpos(), self.ypos(),
self.zpos()])
1492 gcpy          self.dxfarc(self.currenttoolnumber(), xcenter, ycenter,
                        radius, 180, 270)
1493 gcpy          if (self.zpos == ez):
1494 gcpy              self.settzpos(0)
1495 gcpy          else:
1496 gcpy              self.settzpos((self.zpos()-ez)/90)
1497 gcpy          # self.setxpos(ex)
1498 gcpy          # self.setypos(ey)
1499 gcpy          # self.setzpos(ez)
1500 gcpy          if self.generatepaths == True:
1501 gcpy              self.arcloop(181, 270, xcenter, ycenter, radius)
1502 gcpy          # self.toolpaths = self.toolpaths.union(toolpath)
1503 gcpy          else:
1504 gcpy              toolpath = self.arcloop(181, 270, xcenter, ycenter,
                        radius)
1505 gcpy              return toolpath
1506 gcpy
1507 gcpy          def cutarcSECCdxf(self, ex, ey, ez, xcenter, ycenter, radius):
1508 gcpy              global toolpath
1509 gcpy              toolpath = self.currenttool()
1510 gcpy              toolpath = toolpath.translate([self.xpos(), self.ypos(),
self.zpos()])
1511 gcpy          self.dxfarc(self.currenttoolnumber(), xcenter, ycenter,
                        radius, 270, 360)
1512 gcpy          if (self.zpos == ez):
1513 gcpy              self.settzpos(0)
1514 gcpy          else:
1515 gcpy              self.settzpos((self.zpos()-ez)/90)
1516 gcpy          # self.setxpos(ex)
1517 gcpy          # self.setypos(ey)
1518 gcpy          # self.setzpos(ez)
1519 gcpy          if self.generatepaths == True:

```

```

1520 gcpy                self.arcloop(271, 360, xcenter, ycenter, radius)
1521 gcpy #                self.toolpaths = self.toolpaths.union(toolpath)
1522 gcpy                else:
1523 gcpy                toolpath = self.arcloop(271, 360, xcenter, ycenter,
1524 gcpy                    radius)
1525 gcpy                return toolpath
1526 gcpy                def cutarcNECWdxf(self, ex, ey, ez, xcenter, ycenter, radius):
1527 gcpy #                    global toolpath
1528 gcpy #                    toolpath = self.currenttool()
1529 gcpy #                    toolpath = toolpath.translate([self.xpos(), self.ypos(),
self.zpos()])
1530 gcpy                self.dxfarc(self.currenttoolnumber(), xcenter, ycenter,
1531 gcpy                    radius, 0, 90)
1532 gcpy                if (self.zpos == ez):
1533 gcpy                    self.settzpos(0)
1534 gcpy                else:
1535 gcpy                    self.settzpos((self.zpos()-ez)/90)
1536 gcpy #                    self.setxpos(ex)
1537 gcpy #                    self.setypos(ey)
1538 gcpy #                    self.setzpos(ez)
1539 gcpy                if self.generatepaths == True:
1540 gcpy                    self.narcloop(89, 0, xcenter, ycenter, radius)
1541 gcpy                    self.toolpaths = self.toolpaths.union(toolpath)
1542 gcpy                else:
1543 gcpy                    toolpath = self.narcloop(89, 0, xcenter, ycenter,
1544 gcpy                        radius)
1545 gcpy                return toolpath
1546 gcpy                def cutarcSECWdxf(self, ex, ey, ez, xcenter, ycenter, radius):
1547 gcpy #                    global toolpath
1548 gcpy #                    toolpath = self.currenttool()
1549 gcpy #                    toolpath = toolpath.translate([self.xpos(), self.ypos(),
self.zpos()])
1550 gcpy                self.dxfarc(self.currenttoolnumber(), xcenter, ycenter,
1551 gcpy                    radius, 270, 360)
1552 gcpy                if (self.zpos == ez):
1553 gcpy                    self.settzpos(0)
1554 gcpy                else:
1555 gcpy                    self.settzpos((self.zpos()-ez)/90)
1556 gcpy #                    self.setxpos(ex)
1557 gcpy #                    self.setypos(ey)
1558 gcpy #                    self.setzpos(ez)
1559 gcpy                if self.generatepaths == True:
1560 gcpy                    self.narcloop(359, 270, xcenter, ycenter, radius)
1561 gcpy                    self.toolpaths = self.toolpaths.union(toolpath)
1562 gcpy                else:
1563 gcpy                    toolpath = self.narcloop(359, 270, xcenter, ycenter,
1564 gcpy                        radius)
1565 gcpy                return toolpath
1566 gcpy                def cutarcSWCWdxf(self, ex, ey, ez, xcenter, ycenter, radius):
1567 gcpy #                    global toolpath
1568 gcpy #                    toolpath = self.currenttool()
1569 gcpy #                    toolpath = toolpath.translate([self.xpos(), self.ypos(),
self.zpos()])
1570 gcpy                self.dxfarc(self.currenttoolnumber(), xcenter, ycenter,
1571 gcpy                    radius, 180, 270)
1572 gcpy                if (self.zpos == ez):
1573 gcpy                    self.settzpos(0)
1574 gcpy                else:
1575 gcpy                    self.settzpos((self.zpos()-ez)/90)
1576 gcpy #                    self.setxpos(ex)
1577 gcpy #                    self.setypos(ey)
1578 gcpy #                    self.setzpos(ez)
1579 gcpy                if self.generatepaths == True:
1580 gcpy                    self.narcloop(269, 180, xcenter, ycenter, radius)
1581 gcpy                    self.toolpaths = self.toolpaths.union(toolpath)
1582 gcpy                else:
1583 gcpy                    toolpath = self.narcloop(269, 180, xcenter, ycenter,
1584 gcpy                        radius)
1585 gcpy                return toolpath
1586 gcpy                def cutarcNWCWdxf(self, ex, ey, ez, xcenter, ycenter, radius):
1587 gcpy #                    global toolpath
1588 gcpy #                    toolpath = self.currenttool()
1589 gcpy #                    toolpath = toolpath.translate([self.xpos(), self.ypos(),
self.zpos()])

```

```
1587 gcpy          self.dxfarc(self.currenttoolnumber(), xcenter, ycenter,
                        radius, 90, 180)
1588 gcpy          if (self.zpos == ez):
1589 gcpy              self.settzpos(0)
1590 gcpy          else:
1591 gcpy              self.settzpos((self.zpos()-ez)/90)
1592 gcpy #          self.setxpos(ex)
1593 gcpy #          self.setypos(ey)
1594 gcpy #          self.setzpos(ez)
1595 gcpy          if self.generatepaths == True:
1596 gcpy              self.narcloop(179, 90, xcenter, ycenter, radius)
1597 gcpy #          self.toolpaths = self.toolpaths.union(toolpath)
1598 gcpy          else:
1599 gcpy              toolpath = self.narcloop(179, 90, xcenter, ycenter,
                        radius)
1600 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

---

```
1602 gcpy          def arcCCgc(self, ex, ey, ez, xcenter, ycenter, radius):
1603 gcpy              self.writegc("G03_X", str(ex), "Y", str(ey), "Z", str(ez)
                        , "R", str(radius))

1604 gcpy          def arcCWgc(self, ex, ey, ez, xcenter, ycenter, radius):
1605 gcpy              self.writegc("G02_X", str(ex), "Y", str(ey), "Z", str(ez)
1606 gcpy              , "R", str(radius))
```

---

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

---

```
1608 gcpy          def cutarcNECCdxfgc(self, ex, ey, ez, xcenter, ycenter, radius)
1609 gcpy              :
1610 gcpy              self.arcCCgc(ex, ey, ez, xcenter, ycenter, radius)
1611 gcpy              if self.generatepaths == True:
1612 gcpy                  self.cutarcNECCdxf(ex, ey, ez, xcenter, ycenter, radius
1613 gcpy                  )
1614 gcpy              else:
1615 gcpy                  return self.cutarcNECCdxf(ex, ey, ez, xcenter, ycenter,
                        radius)

1616 gcpy          def cutarcNWCCdxfgc(self, ex, ey, ez, xcenter, ycenter, radius)
1617 gcpy              :
1618 gcpy              self.arcCCgc(ex, ey, ez, xcenter, ycenter, radius)
1619 gcpy              if self.generatepaths == False:
1620 gcpy                  return self.cutarcNWCCdxf(ex, ey, ez, xcenter, ycenter,
                        radius)

1621 gcpy          def cutarcSWCCdxfgc(self, ex, ey, ez, xcenter, ycenter, radius)
1622 gcpy              :
1623 gcpy              self.arcCCgc(ex, ey, ez, xcenter, ycenter, radius)
1624 gcpy              if self.generatepaths == False:
1625 gcpy                  return self.cutarcSWCCdxf(ex, ey, ez, xcenter, ycenter,
                        radius)

1626 gcpy          def cutarcSECCdxfgc(self, ex, ey, ez, xcenter, ycenter, radius)
1627 gcpy              :
1628 gcpy              self.arcCCgc(ex, ey, ez, xcenter, ycenter, radius)
1629 gcpy              if self.generatepaths == False:
1630 gcpy                  return self.cutarcSECCdxf(ex, ey, ez, xcenter, ycenter,
                        radius)

1631 gcpy          def cutarcNECWdxfgc(self, ex, ey, ez, xcenter, ycenter, radius)
1632 gcpy              :
1633 gcpy              self.arcCWgc(ex, ey, ez, xcenter, ycenter, radius)
1634 gcpy              if self.generatepaths == False:
1635 gcpy                  return self.cutarcNECWdxf(ex, ey, ez, xcenter, ycenter,
                        radius)

1636 gcpy          def cutarcSECWdxfgc(self, ex, ey, ez, xcenter, ycenter, radius)
1637 gcpy              :
1638 gcpy              self.arcCWgc(ex, ey, ez, xcenter, ycenter, radius)
1639 gcpy              if self.generatepaths == False:
```

```
1638 gcpy                return self.cutarcSECWdx(x, y, z, xc, yc, r)
1639 gcpy
1640 gcpy                def cutarcSWCdxfgc(self, x, y, z, xc, yc, r):
1641 gcpy                    self.arcCWgc(x, y, z, xc, yc, r)
1642 gcpy                    if self.generatepaths == False:
1643 gcpy                        return self.cutarcSWCdx(x, y, z, xc, yc, r)
1644 gcpy
1645 gcpy                def cutarcNWCdxfgc(self, x, y, z, xc, yc, r):
1646 gcpy                    self.arcCWgc(x, y, z, xc, yc, r)
1647 gcpy                    if self.generatepaths == False:
1648 gcpy                        return self.cutarcNWCdx(x, y, z, xc, yc, r)
1649 gcpy
1650 gcpy
1651 gcpy                module cutarcNECCdxfgc(x, y, z, xc, yc, r){
1652 gcpy                    gcp.cutarcNECCdxfgc(x, y, z, xc, yc, r);
1653 gcpy                }
1654 gcpy
1655 gcpy                module cutarcNWCCdxfgc(x, y, z, xc, yc, r){
1656 gcpy                    gcp.cutarcNWCCdxfgc(x, y, z, xc, yc, r);
1657 gcpy                }
1658 gcpy
1659 gcpy                module cutarcSWCCdxfgc(x, y, z, xc, yc, r){
1660 gcpy                    gcp.cutarcSWCCdxfgc(x, y, z, xc, yc, r);
1661 gcpy                }
1662 gcpy
1663 gcpy                module cutarcSECCdxfgc(x, y, z, xc, yc, r){
1664 gcpy                    gcp.cutarcSECCdxfgc(x, y, z, xc, yc, r);
1665 gcpy                }
```

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

```
1650 gcpy                def dxftemplate(self, tn):
1651 gcpy                # self.writedxf(tn, str(tn))
1652 gcpy                self.writedxf(tn, "0")
1653 gcpy                self.writedxf(tn, "ENDSEC")
1654 gcpy                self.writedxf(tn, "0")
1655 gcpy                self.writedxf(tn, "EOF")
1656 gcpy
1657 gcpy
1658 gcpy                def gcodepostamble(self):
1659 gcpy                self.writegc("Z12.700")
1660 gcpy                self.writegc("M05")
1661 gcpy                self.writegc("M02")
1662 gcpy
```

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

```
1662 gcpy                def closegcodefile(self):
1663 gcpy                if self.generategcode == True:
1664 gcpy                    self.gcodepostamble()
1665 gcpy                    self.gc.close()
1666 gcpy
1667 gcpy                def closedxf(self):
1668 gcpy                if self.generatedxf == True:
1669 gcpy                # global dxfclosed
1670 gcpy                self.dxfpostamble(-1)
1671 gcpy                # self.dxfclosed = True
1672 gcpy                self.dxf.close()
1673 gcpy
1674 gcpy                def closedxf(self):
1675 gcpy                if self.generatedxfs == True:
1676 gcpy                    if (self.large_square_tool_num > 0):
1677 gcpy                        self.dxfpostamble(self.large_square_tool_num)
1678 gcpy                    if (self.small_square_tool_num > 0):
1679 gcpy                        self.dxfpostamble(self.small_square_tool_num)
1680 gcpy                    if (self.large_ball_tool_num > 0):
1681 gcpy                        self.dxfpostamble(self.large_ball_tool_num)
```

```
1682 gcpy          if (self.small_ball_tool_num > 0):
1683 gcpy              self.dxfpostamble(self.small_ball_tool_num)
1684 gcpy          if (self.large_V_tool_num > 0):
1685 gcpy              self.dxfpostamble(self.large_V_tool_num)
1686 gcpy          if (self.small_V_tool_num > 0):
1687 gcpy              self.dxfpostamble(self.small_V_tool_num)
1688 gcpy          if (self.DT_tool_num > 0):
1689 gcpy              self.dxfpostamble(self.DT_tool_num)
1690 gcpy          if (self.KH_tool_num > 0):
1691 gcpy              self.dxfpostamble(self.KH_tool_num)
1692 gcpy          if (self.Roundover_tool_num > 0):
1693 gcpy              self.dxfpostamble(self.Roundover_tool_num)
1694 gcpy          if (self.MISC_tool_num > 0):
1695 gcpy              self.dxfpostamble(self.MISC_tool_num)
1696 gcpy
1697 gcpy          if (self.large_square_tool_num > 0):
1698 gcpy              self.dxfllsq.close()
1699 gcpy          if (self.small_square_tool_num > 0):
1700 gcpy              self.dxfllsq.close()
1701 gcpy          if (self.large_ball_tool_num > 0):
1702 gcpy              self.dxfllbl.close()
1703 gcpy          if (self.small_ball_tool_num > 0):
1704 gcpy              self.dxfllbl.close()
1705 gcpy          if (self.large_V_tool_num > 0):
1706 gcpy              self.dxfllV.close()
1707 gcpy          if (self.small_V_tool_num > 0):
1708 gcpy              self.dxfllV.close()
1709 gcpy          if (self.DT_tool_num > 0):
1710 gcpy              self.dxfDT.close()
1711 gcpy          if (self.KH_tool_num > 0):
1712 gcpy              self.dxfKH.close()
1713 gcpy          if (self.Roundover_tool_num > 0):
1714 gcpy              self.dxfRt.close()
1715 gcpy          if (self.MISC_tool_num > 0):
1716 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.

```
155 gcpscad module closegcodefile(){
156 gcpscad     gcp.closegcodefile();
157 gcpscad }
158 gcpscad
159 gcpscad module closedxfiles(){
160 gcpscad     gcp.closedxfiles();
161 gcpscad }
162 gcpscad
163 gcpscad module closedxfile(){
164 gcpscad     gcp.closedxfile();
165 gcpscad }
```

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

previewgcodefile      Which simply needs to call the previewgcodefile command:

```
1717 gcpy          def previewgcodefile(self, gc_file):
1718 gcpy              gc_file = open(gc_file, 'r')
```

```

1719 gcpy          gcfilerecontents = []
1720 gcpy          with gc_file as file:
1721 gcpy              for line in file:
1722 gcpy                  command = line
1723 gcpy                  gcfilerecontents.append(line)
1724 gcpy
1725 gcpy          numlinesfound = 0
1726 gcpy          for line in gcfilerecontents:
1727 gcpy              print(line)
1728 gcpy              if line[:10] == "(stockMin:":
1729 gcpy                  subdivisions = line.split()
1730 gcpy                  extentleft = float(subdivisions[0][10:-3])
1731 gcpy                  extentfb = float(subdivisions[1][:-3])
1732 gcpy                  extentd = float(subdivisions[2][:-3])
1733 gcpy                  numlinesfound = numlinesfound + 1
1734 gcpy              if line[:13] == "(STOCK/BLOCK,":
1735 gcpy                  subdivisions = line.split()
1736 gcpy                  sizeX = float(subdivisions[0][13:-1])
1737 gcpy                  sizeY = float(subdivisions[1][:-1])
1738 gcpy                  sizeZ = float(subdivisions[4][:-1])
1739 gcpy                  numlinesfound = numlinesfound + 1
1740 gcpy              if line[:3] == "G21":
1741 gcpy                  units = "mm"
1742 gcpy                  numlinesfound = numlinesfound + 1
1743 gcpy              if numlinesfound >=3:
1744 gcpy                  break
1745 gcpy              print(numlinesfound)
1746 gcpy
1747 gcpy          self.setupcuttingarea(sizeX, sizeY, sizeZ, extentleft,
                                     extentfb, extentd)
1748 gcpy
1749 gcpy          commands = []
1750 gcpy          for line in gcfilerecontents:
1751 gcpy              Xc = 0
1752 gcpy              Yc = 0
1753 gcpy              Zc = 0
1754 gcpy              Fc = 0
1755 gcpy              Xp = 0.0
1756 gcpy              Yp = 0.0
1757 gcpy              Zp = 0.0
1758 gcpy              if line == "G53G0Z-5.000\n":
1759 gcpy                  self.movetosafeZ()
1760 gcpy              if line[:3] == "M6T":
1761 gcpy                  tool = int(line[3:])
1762 gcpy                  self.toolchange(tool)
1763 gcpy              if line[:2] == "G0":
1764 gcpy                  machinestate = "rapid"
1765 gcpy              if line[:2] == "G1":
1766 gcpy                  machinestate = "cutline"
1767 gcpy              if line[:2] == "G0" or line[:2] == "G1" or line[:1] ==
1768 gcpy                  "X" or line[:1] == "Y" or line[:1] == "Z":
1769 gcpy                  if "F" in line:
1770 gcpy                      Fplus = line.split("F")
1771 gcpy                      Fc = 1
1772 gcpy                      fr = float(Fplus[1])
1773 gcpy                      line = Fplus[0]
1774 gcpy                  if "Z" in line:
1775 gcpy                      Zplus = line.split("Z")
1776 gcpy                      Zc = 1
1777 gcpy                      Zp = float(Zplus[1])
1778 gcpy                      line = Zplus[0]
1779 gcpy                  if "Y" in line:
1780 gcpy                      Yplus = line.split("Y")
1781 gcpy                      Yc = 1
1782 gcpy                      Yp = float(Yplus[1])
1783 gcpy                      line = Yplus[0]
1784 gcpy                  if "X" in line:
1785 gcpy                      Xplus = line.split("X")
1786 gcpy                      Xc = 1
1787 gcpy                      Xp = float(Xplus[1])
1788 gcpy                  if Zc == 1:
1789 gcpy                      if Yc == 1:
1790 gcpy                          if Xc == 1:
1791 gcpy                              if machinestate == "rapid":
1792 gcpy                                  command = "rapidXYZ(" + str(Xp) + "
,␣" + str(Yp) + ",␣" + str(Zp) +
")"
self.rapidXYZ(Xp, Yp, Zp)

```

```
1793 gcpy                                     else:
1794 gcpy                                     command = "cutlineXYZ(" + str(Xp) +
                                                ",␣" + str(Yp) + ",␣" + str(Zp)
                                                + ")"
                                                self.cutlineXYZ(Xp, Yp, Zp)
1795 gcpy
1796 gcpy                                     else:
1797 gcpy                                     if machinestate == "rapid":
1798 gcpy                                     command = "rapidYZ(" + str(Yp) + ",
                                                ␣" + str(Zp) + ")"
                                                self.rapidYZ(Yp, Zp)
1799 gcpy                                     else:
1800 gcpy                                     command = "cutlineYZ(" + str(Yp) +
                                                ",␣" + str(Zp) + ")"
                                                self.cutlineYZ(Yp, Zp)
1801 gcpy
1802 gcpy
1803 gcpy                                     else:
1804 gcpy                                     if Xc == 1:
1805 gcpy                                     if machinestate == "rapid":
1806 gcpy                                     command = "rapidXZ(" + str(Xp) + ",
                                                ␣" + str(Zp) + ")"
                                                self.rapidXZ(Xp, Zp)
1807 gcpy                                     else:
1808 gcpy                                     command = "cutlineXZ(" + str(Xp) +
                                                ",␣" + str(Zp) + ")"
                                                self.cutlineXZ(Xp, Zp)
1809 gcpy
1810 gcpy                                     else:
1811 gcpy                                     if machinestate == "rapid":
1812 gcpy                                     command = "rapidZ(" + str(Zp) + ")"
                                                self.rapidZ(Zp)
1813 gcpy                                     else:
1814 gcpy                                     command = "cutlineZ(" + str(Zp) + "
                                                )"
                                                self.cutlineZ(Zp)
1815 gcpy
1816 gcpy
1817 gcpy                                     else:
1818 gcpy                                     if Yc == 1:
1819 gcpy                                     if Xc == 1:
1820 gcpy                                     if machinestate == "rapid":
1821 gcpy                                     command = "rapidXY(" + str(Xp) + ",
                                                ␣" + str(Yp) + ")"
                                                self.rapidXY(Xp, Yp)
1822 gcpy                                     else:
1823 gcpy                                     command = "cutlineXY(" + str(Xp) +
                                                ",␣" + str(Yp) + ")"
                                                self.cutlineXY(Xp, Yp)
1824 gcpy                                     else:
1825 gcpy                                     if machinestate == "rapid":
1826 gcpy                                     command = "rapidY(" + str(Yp) + ")"
                                                self.rapidY(Yp)
1827 gcpy                                     else:
1828 gcpy                                     command = "cutlineY(" + str(Yp) + "
                                                )"
                                                self.cutlineY(Yp)
1829 gcpy
1830 gcpy                                     else:
1831 gcpy                                     if Xc == 1:
1832 gcpy                                     if machinestate == "rapid":
1833 gcpy                                     command = "rapidX(" + str(Xp) + ")"
                                                self.rapidX(Xp)
1834 gcpy                                     else:
1835 gcpy                                     command = "cutlineX(" + str(Xp) + "
                                                )"
                                                self.cutlineX(Xp)
1836 gcpy
1837 gcpy                                     commands.append(command)
1838 gcpy                                     print(line)
1839 gcpy                                     print(command)
1840 gcpy                                     print(machinestate, Xc, Yc, Zc)
1841 gcpy                                     print(Xp, Yp, Zp)
1842 gcpy                                     print("/n")
1843 gcpy #
1844 gcpy #
1845 gcpy #
1846 gcpy #
1847 gcpy #
1848 gcpy
1849 gcpy #         for command in commands:
1850 gcpy #             print(command)
1851 gcpy
1852 gcpy         output(self.stockandtoolpaths())
```

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

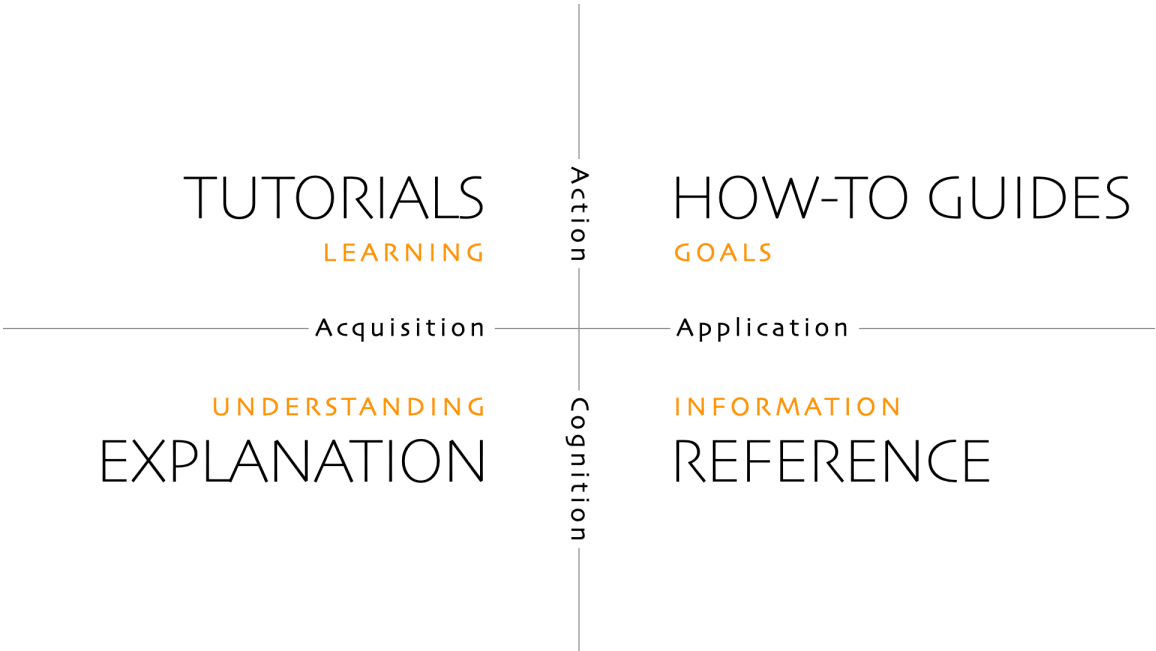
<https://thewhitetulip.gitbook.io/py/06-file-handling>

#### 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 documentation 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:

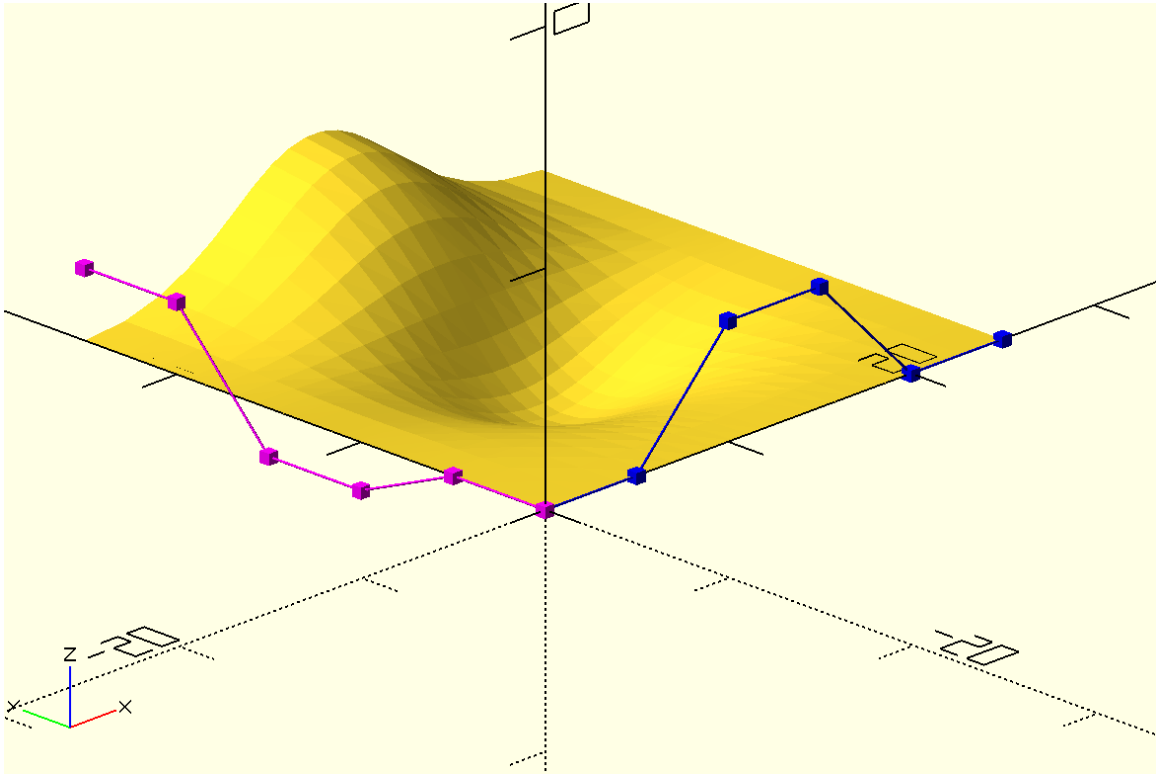
3 planes \* 3 Béziers \* (2 on-curve + 2 off-curve points) == 36 coordinate pairs

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/](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. <a href="https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=823374">https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=823374</a> <a href="https://www.nist.gov/publications/nist-rs274ngc-interpreter-version-3">https://www.nist.gov/publications/nist-rs274ngc-interpreter-version-3</a>
[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

- ballnose, 31
- bowl tool, 31
- closedxfile, 69, 70
- closegcodefile, 69, 70
- currenttoolnum, 24, 28
- currenttoolnumber, 28
- currenttoolshape, 34
- cut..., 38, 40
- cutarcCC, 43
- cutarcCW, 43
- cutkeyhole toolpath, 49
- cutKHgcdxf, 51
- cutline, 40
- dovetail, 33
- dxfar, 64
- dxflin, 64
- dxfpreamble, 69, 70
- dxfpreamble, 64
- dxfwrite, 64
- endmill square, 31
- endmill v, 31
- feed, 38
- flat V, 32
- gcodepreview, 22
  - writeln, 61
- gcp.setupstock, 25
- generatepaths, 18
- init, 22
- initializemachinestate(), 25
- keyhole, 33
- mpx, 24
- mpy, 24
- mpz, 24
- opendxfile, 62
- opengcodefile, 62
- plunge, 38
- previewgcodefile, 70
- rapid..., 38
- rcl, 38
- settool, 28
- setupstock, 25
  - gcodepreview, 25
- setxpos, 24
- setypos, 24
- setzpos, 24
- speed, 38
- stepsizearc, 21
- stepsizeroundover, 21
- stockandtoolpaths, 18
- subroutine
  - gcodepreview, 25
  - writeln, 61
- tapered ball, 32
- threadmill, 33
- tool diameter, 36
- tool radius, 38
- toolchange, 34
- toolpaths, 18
- tpzinc, 24
- writedxDT, 64
- writedxKH, 64
- writedxflgl, 64
- writedxflgsq, 64
- writedxflgV, 64
- writedxfsmb, 64
- writedxfsmsq, 64
- writedxfsmV, 64
- xpos, 24
- ypos, 24
- zpos, 24

# Routines

- ballnose, 31
- bowl tool, 31
- closedxfile, 69, 70
- closegcodefile, 69, 70
- currenttoolnumber, 28
- cut..., 38, 40
- cutarcCC, 43
- cutarcCW, 43
- cutkeyhole toolpath, 49
- cutKHgcdxf, 51
- cutline, 40
- dovetail, 33
- dxfarc, 64
- dxfline, 64
- dxfpreamble, 69, 70
- dxfpreamble, 64
- dxfwrite, 64
- endmill square, 31
- endmill v, 31
- flat V, 32
- gcodepreview, 22, 25
- gcp.setupstock, 25
- init, 22
- initializemachinestate(), 25
- keyhole, 33
- opendxfile, 62
- opengcodefile, 62
- previewgcodefile, 70
- rapid..., 38
- rcl, 38
- settool, 28
- setupstock, 25
- setxpos, 24
- setypos, 24
- setzpos, 24
- stockandtoolpaths, 18
- tapered ball, 32
- threadmill, 33
- tool diameter, 36
- tool radius, 38
- toolchange, 34
- writedxDT, 64
- writedxKH, 64
- writedxgbl, 64
- writedxgsg, 64
- writedxgV, 64
- writedxsmbl, 64
- writedxsgsg, 64
- writedxsmV, 64
- writeln, 61
- xpos, 24
- ypos, 24
- zpos, 24

# Variables

currenttoolnum, 24, 28	plunge, 38
currenttoolshape, 34	
feed, 38	speed, 38
	stepsizearc, 21
generatepaths, 18	stepsizeroundover, 21
mpx, 24	
mpy, 24	toolpaths, 18
mpz, 24	tpzinc, 24