

The gcodepreview PythonSCAD library*

Author: William F. Adams
willadams at aol dot com

2025/01/29

Abstract

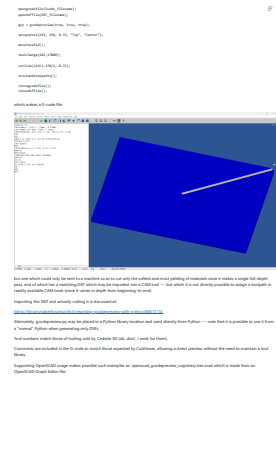
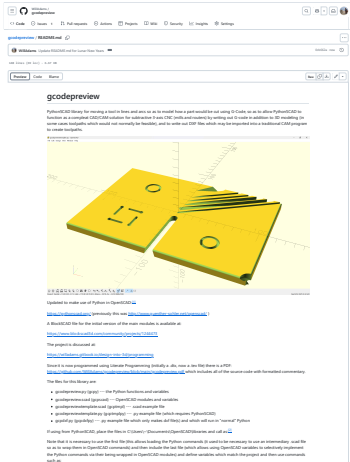
The gcodepreview library allows using PythonSCAD (OpenPythonSCAD) to move a tool in lines and arcs and output DXF and G-code files so as to work as a CAD/CAM program for CNC.

Contents

1	readme.md	2
2	Usage and Templates	5
2.1	gcpdxf.py	5
2.2	gcodepreviewtemplate.py	7
2.3	gcodepreviewtemplate.scad	13
3	gcodepreview	17
3.1	Module Naming Convention	18
3.1.1	Parameters and Default Values	20
3.2	Implementation files and gcodepreview class	20
3.2.1	Position and Variables	23
3.2.2	Initial Modules	24
3.3	Tools and Changes	26
3.3.1	3D Shapes for Tools	27
3.3.1.1	Normal Tooling/toolshapes	27
3.3.1.2	Tooling for Undercutting Toolpaths	28
3.3.1.2.1	Keyhole tools	28
3.3.1.2.2	Thread mills	29
3.3.1.2.3	Dovetails	29
3.3.1.3	Concave toolshapes	29
3.3.1.4	Roundover tooling	29
3.3.2	toolchange	30
3.3.2.1	Selecting Tools	30
3.3.2.2	Square and ball nose (including tapered ball nose)	30
3.3.2.3	Roundover (corner rounding)	30
3.3.2.4	Dovetails	30
3.3.2.5	toolchange routine	30
3.3.3	tooldiameter	32
3.3.4	Feeds and Speeds	34
3.4	Movement and Cutting	34
3.4.1	Lines	36
3.4.2	Arcs for toolpaths and DXFs	37
3.4.3	Cutting shapes and expansion	41
3.4.3.1	Building blocks	41
3.4.3.2	List of shapes	41
3.4.3.2.1	circles	42
3.4.3.2.2	rectangles	43
3.4.3.2.3	Keyhole toolpath and undercut tooling	44
3.4.4	Difference of Stock, Rapids, and Toolpaths	51
3.5	Output files	51
3.5.1	G-code Overview	51
3.5.2	DXF Overview	52
3.5.3	Python and OpenSCAD File Handling	53
3.5.3.1	Writing to DXF files	56
3.5.3.2	Closings	61
4	Notes	63
	Index	67
	Routines	68
	Variables	69

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

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 commentary.
23 rdme
24 rdme The files for this library are:
25 rdme
26 rdme - gcodepreview.py (gcpy) --- the Python functions and variables
27 rdme - gcodepreview.scad (gcpscad) --- OpenSCAD modules and variables
28 rdme - gcodepreviewtemplate.scad (gcptmpl) --- .scad example file
29 rdme - gcodepreviewtemplate.py (gcptmplpy) --- .py example file (which
  requires PythonSCAD)
30 rdme - gcpdxf.py (gcpdxfpy) --- .py example file which only makes dxf
  file(s) and which will run in "normal" Python
31 rdme
32 rdme If using from PythonSCAD, place the files in C:\Users\\~\Documents
  \OpenSCAD\libraries [^libraries]
33 rdme
34 rdme [^libraries]: C:\Users\\~\Documents\RapCAD\libraries is deprecated
  since RapCAD is no longer needed since Python is now used for
  writing out files.
35 rdme
36 rdme and call as:
37 rdme
38 rdme use <gcodepreview.py>
```

```
39 rdme      include <gcodepreview.scad>
40 rdme
41 rdme Note that it is necessary to use the first file (this allows
      loading the Python commands (it used to be necessary to use an
      intermediary .scad file so as to wrap them in OpenSCAD commands)
      and then include the last file (which allows using OpenSCAD
      variables to selectively implement the Python commands via their
      being wrapped in OpenSCAD modules) and define variables which
      match the project and then use commands such as:
42 rdme
43 rdme     .opengcodefile(Gcode_filename);
44 rdme     .opendxffile(DXF_filename);
45 rdme
46 rdme      gcp = gcodepreview(true, true, true);
47 rdme
48 rdme      setupstock(219, 150, 8.35, "Top", "Center");
49 rdme
50 rdme      movetosafeZ();
51 rdme
52 rdme      toolchange(102, 17000);
53 rdme
54 rdme      cutline(219/2, 150/2, -8.35);
55 rdme
56 rdme      stockandtoolpaths();
57 rdme
58 rdme      closegcodefile();
59 rdme      closedxfile();
60 rdme
61 rdme which makes a G-code file:
62 rdme
63 rdme ![OpenSCAD template G-code file](https://raw.githubusercontent.com/
      WillAdams/gcodepreview/main/gcodepreview_template.png?raw=true)
64 rdme
65 rdme but one which could only be sent to a machine so as to cut only the
      softest and most yielding of materials since it makes a single
      full-depth pass, and of which has a matching DXF which may be
      imported into a CAM tool --- but which it is not directly
      possible to assign a toolpath in readily available CAM tools (
      since it varies in depth from beginning-to-end).
66 rdme
67 rdme Importing this DXF and actually cutting it is discussed at:
68 rdme
69 rdme https://forum.makerforums.info/t/rewriting-gcodepreview-with-python
      /88617/14
70 rdme
71 rdme Alternately, gcodepreview.py may be placed in a Python library
      location and used directly from Python --- note that it is
      possible to use it from a "normal" Python when generating only
      DXFs.
72 rdme
73 rdme Tool numbers match those of tooling sold by Carbide 3D (ob. discl.,
      I work for them).
74 rdme
75 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.
76 rdme
77 rdme Supporting OpenSCAD usage makes possible such examples as:
      openscad_gcodepreview_cutjoinery.tres.scad which is made from an
      OpenSCAD Graph Editor file:
78 rdme
79 rdme ![OpenSCAD Graph Editor Cut Joinery File](https://raw.
      githubusercontent.com/WillAdams/gcodepreview/main/
      OSGE_cutjoinery.png?raw=true)
80 rdme
81 rdme | Version          | Notes          |
82 rdme | -----          | -----          |
83 rdme | 0.1              | Version  supports setting up stock, origin, rapid
      positioning, making cuts, and writing out matching G-code, and
      creating a DXF with polylines. |
84 rdme |                  | - separate dxf files are written out for each
      tool where tool is ball/square/V and small/large (10/31/23)
      |
85 rdme |                  | - re-writing as a Literate Program using the
      LaTeX package docmfp (begun 4/12/24)
      |
```

```

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

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

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

      |
89 rdme | 0.4           | Rewrite using literati documentclass, suppression
      of SVG code, dxfrectangle

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

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

      |
92 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|
93 rdme | 0.7           | Re-write completely in Python

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

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

      |
96 rdme
97 rdme Possible future improvements:
98 rdme
99 rdme - support for additional tooling shapes (tapered ball nose,
      lollipop cutters)
100 rdme - create a single line font for use where text is wanted
101 rdme - Support Bézier curves (required for fonts if not to be limited
      to lines and arcs) and surfaces
102 rdme
103 rdme Note for G-code generation that it is up to the user to implement
      Depth per Pass so as to not take a single full-depth pass as
      noted above. Working from a DXF of course allows one to off-load
      such considerations to a specialized CAM tool.
104 rdme
105 rdme Deprecated feature:
106 rdme
107 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)
108 rdme
109 rdme To-do:
110 rdme
111 rdme - fix OpenSCAD wrapper and add any missing commands for Python
112 rdme - reposition cutroundover command into cutshape
113 rdme - re-work architecture so that shaft is always defined/included (
      in a different colour so as to identify instances of rubbing)
114 rdme - work on rotary axis option

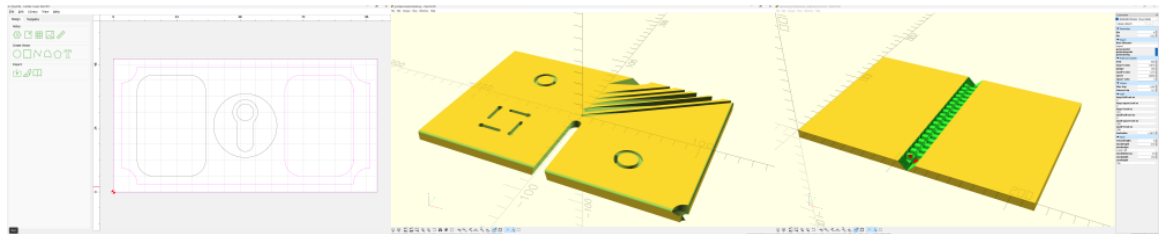
```

2 Usage and Templates

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

The various commands are shown all together in templates so as to provide examples of usage, and to ensure that the various files are used/included as necessary, all variables are set up with the correct names (note that the sparse template in `readme.md` eschews variables), and that files are opened before being written to, and that each is closed at the end in the correct order. Note that while the template files seem overly verbose, they specifically incorporate variables for each tool shape, possibly in two different sizes, and a feed rate parameter or ratio for each, which may be used (by setting a tool #) or ignored (by leaving the variable for a given tool at zero (0)).

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



Some comments on the templates:

- minimal — each is intended as a framework for a minimal working example (MWE) — it should be possible to comment out unused/unneeded portions and so arrive at code which tests any aspect of this project
- compleat — a quite wide variety of tools are listed (and probably more will be added in the future), but pre-defining them and having these “hooks” seems the easiest mechanism to handle everything.
- shortcuts — as the various examples show, while in real life it is necessary to make many passes with a tool, an expedient shortcut is to forgo the `loop` operation and just use a `hull()` operation and implementing Depth per Pass (but note that this will lose the previewing of scalloped tool marks in places where they might appear otherwise)

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

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

2.1 gcpdxf.py

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

```
1 gcpdxfpy from openscad import *
2 gcpdxfpy # nimport("https://raw.githubusercontent.com/WillAdams/gcodepreview
           /refs/heads/main/gcodepreview.py")
3 gcpdxfpy from gcodepreview import *
4 gcpdxfpy
5 gcpdxfpy gcp = gcodepreview(False, # generatepaths
6 gcpdxfpy                                False, # generategcode
7 gcpdxfpy                                True  # generatedxf
8 gcpdxfpy                                )
```

```

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.opendxftfile(Base_filename)
48 gcpdxftp #gcp.opendxftfiles(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, 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 -
84 gcpdxftp (radius * 2), cornerstyle, radius)
85 gcpdxftp #gcp.dxfrectangleround(large_square_tool_num, 64, 7, 24, 36, radius

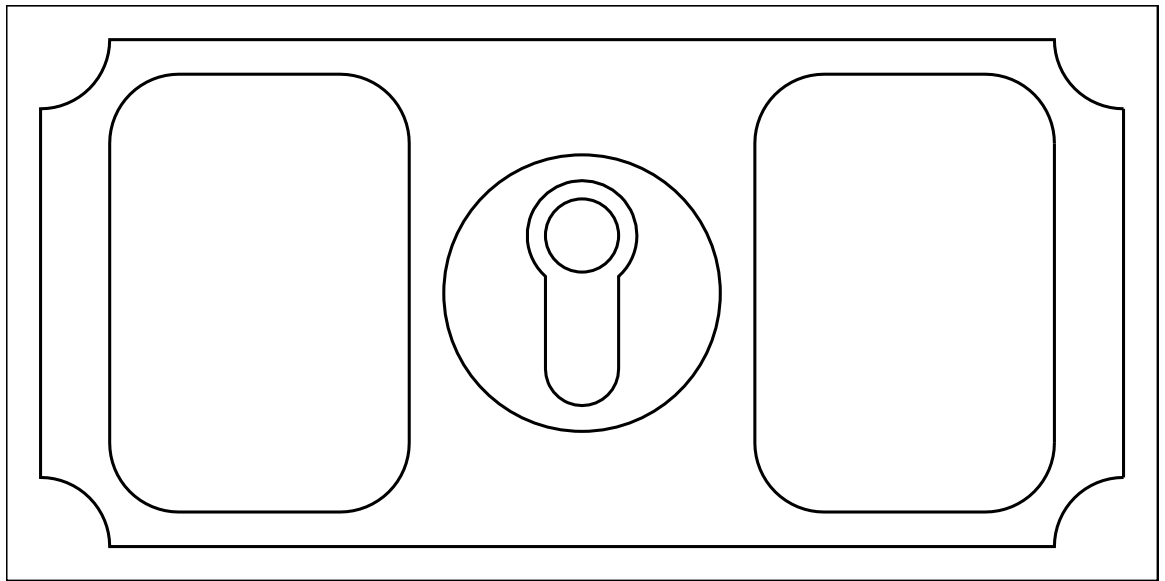
```

```

    )
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)
- dxfar (specifying arc center, radius, and beginning/ending angles)
- dxfKH (specifying origin, depth, angle, distance)

2.2 gcodepreviewtemplate.py

Note that since the v0.7 re-write, it is possible to directly use the underlying Python code. Using Python to generate 3D previews of how DXFs or G-code will cut requires the use of PythonSCAD.

```

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

```

```

15 gcptmplpy
16 gcptmplpy # [Export] */
17 gcptmplpy Base_filename = "aexport"
18 gcptmplpy # [Export] */
19 gcptmplpy generatepaths = False
20 gcptmplpy # [Export] */
21 gcptmplpy generatedxf = True
22 gcptmplpy # [Export] */
23 gcptmplpy generategcode = True
24 gcptmplpy
25 gcptmplpy # [Stock] */
26 gcptmplpy stockXwidth = 220
27 gcptmplpy # [Stock] */
28 gcptmplpy stockYheight = 150
29 gcptmplpy # [Stock] */
30 gcptmplpy stockZthickness = 8.35
31 gcptmplpy # [Stock] */
32 gcptmplpy zeroheight = "Top" # [Top, Bottom]
33 gcptmplpy # [Stock] */
34 gcptmplpy stockzero = "Center" # [Lower-Left, Center-Left, Top-Left, Center]
35 gcptmplpy # [Stock] */
36 gcptmplpy retractheight = 9
37 gcptmplpy
38 gcptmplpy # [CAM] */
39 gcptmplpy toolradius = 1.5875
40 gcptmplpy # [CAM] */
41 gcptmplpy large_square_tool_num = 201 # [0:0, 112:112, 102:102, 201:201]
42 gcptmplpy # [CAM] */
43 gcptmplpy small_square_tool_num = 102 # [0:0, 122:122, 112:112, 102:102]
44 gcptmplpy # [CAM] */
45 gcptmplpy large_ball_tool_num = 202 # [0:0, 111:111, 101:101, 202:202]
46 gcptmplpy # [CAM] */
47 gcptmplpy small_ball_tool_num = 101 # [0:0, 121:121, 111:111, 101:101]
48 gcptmplpy # [CAM] */
49 gcptmplpy large_V_tool_num = 301 # [0:0, 301:301, 690:690]
50 gcptmplpy # [CAM] */
51 gcptmplpy small_V_tool_num = 390 # [0:0, 390:390, 301:301]
52 gcptmplpy # [CAM] */
53 gcptmplpy DT_tool_num = 814 # [0:0, 814:814]
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
    /BOCPJPTMPP
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 R0_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.opendxxfile(Base_filename)
100 gcptmplpy gcp.opendxxfiles(Base_filename,
101 gcptmplpy                             large_square_tool_num,
102 gcptmplpy                             small_square_tool_num,
103 gcptmplpy                             large_ball_tool_num,
104 gcptmplpy                             small_ball_tool_num,
105 gcptmplpy                             large_V_tool_num,
106 gcptmplpy                             small_V_tool_num,
107 gcptmplpy                             DT_tool_num,
108 gcptmplpy                             KH_tool_num,
109 gcptmplpy                             Roundover_tool_num,
110 gcptmplpy                             MISC_tool_num)
111 gcptmplpy gcp.setupstock(stockXwidth, stockYheight, stockZthickness, "Top", "
    Center", retractheight)
112 gcptmplpy
113 gcptmplpy #print(pygcpcversion())
114 gcptmplpy
115 gcptmplpy #print(gcp.myfunc(4))
116 gcptmplpy
117 gcptmplpy #print(gcp.getvv())
118 gcptmplpy
119 gcptmplpy #ts = cylinder(12.7, 1.5875, 1.5875)
120 gcptmplpy #toolpaths = gcp.cutshape(stockXwidth/2, stockYheight/2, -
    stockZthickness)
121 gcptmplpy
122 gcptmplpy gcp.movetosafeZ()
123 gcptmplpy
124 gcptmplpy gcp.toolchange(102, 10000)
125 gcptmplpy
126 gcptmplpy #gcp.rapidXY(6, 12)
127 gcptmplpy gcp.rapidZ(0)
128 gcptmplpy
129 gcptmplpy #print (gcp.xpos())
130 gcptmplpy #print (gcp.ypos())
131 gcptmplpy #psetzpos(7)
132 gcptmplpy #gcp.setzpos(-12)
133 gcptmplpy #print (gcp.zpos())
134 gcptmplpy
135 gcptmplpy #print ("X", str(gcp.xpos()))
136 gcptmplpy #print ("Y", str(gcp.ypos()))
137 gcptmplpy #print ("Z", str(gcp.zpos()))
138 gcptmplpy
139 gcptmplpy toolpaths = gcp.currenttool()
140 gcptmplpy
141 gcptmplpy #toolpaths = gcp.cutline(stockXwidth/2, stockYheight/2, -
    stockZthickness)
142 gcptmplpy toolpaths = toolpaths.union(gcp.cutlinedxfgc(stockXwidth/2,
    stockYheight/2, -stockZthickness))
143 gcptmplpy
144 gcptmplpy gcp.rapidZ(retractheight)
145 gcptmplpy gcp.toolchange(201, 10000)
146 gcptmplpy gcp.rapidXY(0, stockYheight/16)
147 gcptmplpy gcp.rapidZ(0)
148 gcptmplpy toolpaths = toolpaths.union(gcp.cutlinedxfgc(stockXwidth/16*7,
    stockYheight/2, -stockZthickness))
149 gcptmplpy
150 gcptmplpy gcp.rapidZ(retractheight)
151 gcptmplpy gcp.toolchange(202, 10000)
152 gcptmplpy gcp.rapidXY(0, stockYheight/8)
153 gcptmplpy gcp.rapidZ(0)
154 gcptmplpy toolpaths = toolpaths.union(gcp.cutlinedxfgc(stockXwidth/16*6,
    stockYheight/2, -stockZthickness))
155 gcptmplpy
156 gcptmplpy gcp.rapidZ(retractheight)
157 gcptmplpy gcp.toolchange(101, 10000)
158 gcptmplpy gcp.rapidXY(0, stockYheight/16*3)
159 gcptmplpy gcp.rapidZ(0)
160 gcptmplpy toolpaths = toolpaths.union(gcp.cutlinedxfgc(stockXwidth/16*5,

```

```

        stockYheight/2, -stockZthickness))
161 gcptmplpy
162 gcptmplpy gcp.setzpos(retractheight)
163 gcptmplpy gcp.toolchange(390, 10000)
164 gcptmplpy gcp.rapidXY(0, stockYheight/16*4)
165 gcptmplpy gcp.rapidZ(0)
166 gcptmplpy toolpaths = toolpaths.union(gcp.cutlinedxfgc(stockXwidth/16*4,
        stockYheight/2, -stockZthickness))
167 gcptmplpy gcp.rapidZ(retractheight)
168 gcptmplpy
169 gcptmplpy gcp.toolchange(301, 10000)
170 gcptmplpy gcp.rapidXY(0, stockYheight/16*6)
171 gcptmplpy gcp.rapidZ(0)
172 gcptmplpy toolpaths = toolpaths.union(gcp.cutlinedxfgc(stockXwidth/16*2,
        stockYheight/2, -stockZthickness))
173 gcptmplpy
174 gcptmplpy rapids = gcp.rapid(gcp.xpos(), gcp.ypos(), retractheight)
175 gcptmplpy gcp.toolchange(102, 10000)
176 gcptmplpy
177 gcptmplpy rapids = gcp.rapid(-stockXwidth/4+stockYheight/16, +stockYheight/4,
        0)
178 gcptmplpy
179 gcptmplpy toolpaths = toolpaths.union(gcp.cutarcCC(0, 90, gcp.xpos()-
        stockYheight/16, gcp.ypos(), stockYheight/16, -stockZthickness
        /4))
180 gcptmplpy toolpaths = toolpaths.union(gcp.cutarcCC(90, 180, gcp.xpos(), gcp.
        ypos()-stockYheight/16, stockYheight/16, -stockZthickness/4))
181 gcptmplpy toolpaths = toolpaths.union(gcp.cutarcCC(180, 270, gcp.xpos()+
        stockYheight/16, gcp.ypos(), stockYheight/16, -stockZthickness
        /4))
182 gcptmplpy toolpaths = toolpaths.union(gcp.cutarcCC(270, 360, gcp.xpos(), gcp.
        ypos()+stockYheight/16, stockYheight/16, -stockZthickness/4))
183 gcptmplpy
184 gcptmplpy rapids = gcp.movetosafeZ()
185 gcptmplpy rapids = gcp.rapidXY(stockXwidth/4-stockYheight/16, -stockYheight
        /4)
186 gcptmplpy rapids = gcp.rapidZ(0)
187 gcptmplpy
188 gcptmplpy toolpaths = toolpaths.union(gcp.cutarcCW(180, 90, gcp.xpos()+
        stockYheight/16, gcp.ypos(), stockYheight/16, -stockZthickness
        /4))
189 gcptmplpy toolpaths = toolpaths.union(gcp.cutarcCW(90, 0, gcp.xpos(), gcp.
        ypos()-stockYheight/16, stockYheight/16, -stockZthickness/4))
190 gcptmplpy toolpaths = toolpaths.union(gcp.cutarcCW(360, 270, gcp.xpos()-
        stockYheight/16, gcp.ypos(), stockYheight/16, -stockZthickness
        /4))
191 gcptmplpy toolpaths = toolpaths.union(gcp.cutarcCW(270, 180, gcp.xpos(), gcp.
        ypos()+stockYheight/16, stockYheight/16, -stockZthickness/4))
192 gcptmplpy
193 gcptmplpy rapids = gcp.movetosafeZ()
194 gcptmplpy gcp.toolchange(201, 10000)
195 gcptmplpy rapids = gcp.rapidXY(stockXwidth/2, -stockYheight/2)
196 gcptmplpy rapids = gcp.rapidZ(0)
197 gcptmplpy
198 gcptmplpy toolpaths = toolpaths.union(gcp.cutlinedxfgc(gcp.xpos(), gcp.ypos()
        , -stockZthickness))
199 gcptmplpy #test = gcp.cutlinedxfgc(gcp.xpos(), gcp.ypos(), -stockZthickness)
200 gcptmplpy
201 gcptmplpy rapids = gcp.movetosafeZ()
202 gcptmplpy rapids = gcp.rapidXY(stockXwidth/2-6.34, -stockYheight/2)
203 gcptmplpy rapids = gcp.rapidZ(0)
204 gcptmplpy
205 gcptmplpy toolpaths = toolpaths.union(gcp.cutarcCW(180, 90, stockXwidth/2, -
        stockYheight/2, 6.34, -stockZthickness))
206 gcptmplpy
207 gcptmplpy rapids = gcp.movetosafeZ()
208 gcptmplpy gcp.toolchange(814, 10000)
209 gcptmplpy rapids = gcp.rapidXY(0, -(stockYheight/2+12.7))
210 gcptmplpy rapids = gcp.rapidZ(0)
211 gcptmplpy
212 gcptmplpy toolpaths = toolpaths.union(gcp.cutlinedxfgc(gcp.xpos(), gcp.ypos()
        , -stockZthickness))
213 gcptmplpy toolpaths = toolpaths.union(gcp.cutlinedxfgc(gcp.xpos(), -12.7, -
        stockZthickness))
214 gcptmplpy
215 gcptmplpy rapids = gcp.rapidXY(0, -(stockYheight/2+12.7))
216 gcptmplpy rapids = gcp.movetosafeZ()
217 gcptmplpy gcp.toolchange(374, 10000)

```

```

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

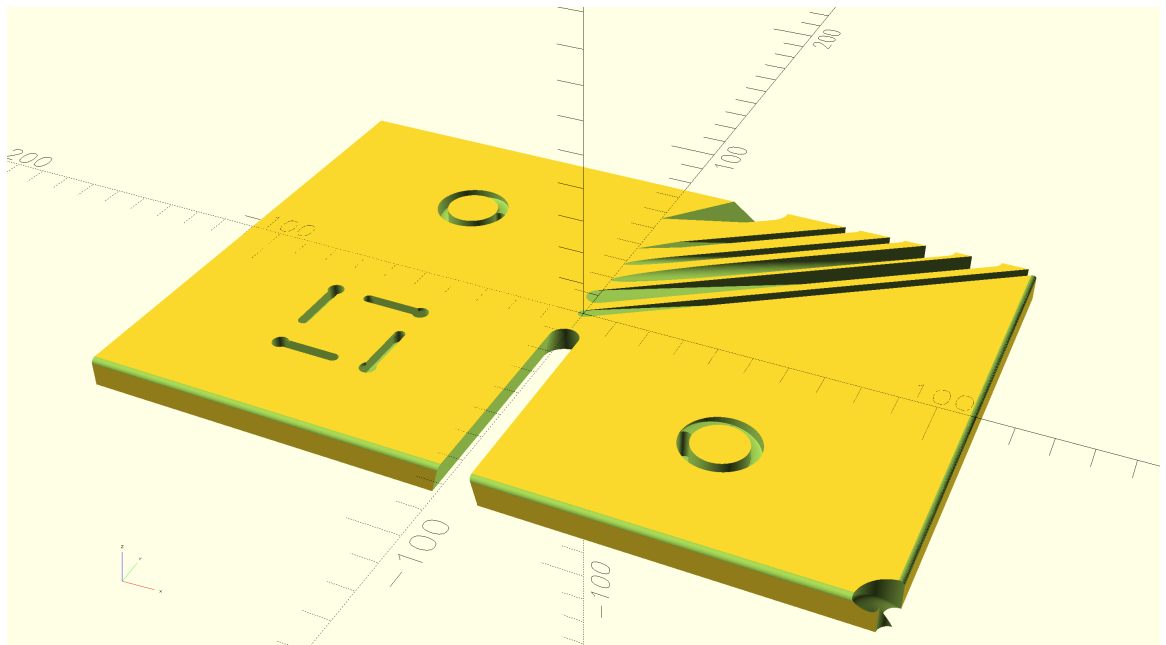
```

```

272 gcptmplpy
273 gcptmplpy gcp.rapidZ(retractheight)
274 gcptmplpy #gcp.toolchange(56125, 10000)
275 gcptmplpy #gcp.cutZgcfeed(-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>. A further consideration worth noting is that when called from OpenSCAD, Python will not halt for errors, but will run through to the end which is an expedient thing for viewing the end result of in-process code.

```

1 gcptmpl //!OpenSCAD
2 gcptmpl
3 gcptmpl use <gcodepreview.py>
4 gcptmpl include <gcodepreview.scad>
5 gcptmpl
6 gcptmpl $fa = 2;
7 gcptmpl $fs = 0.125;
8 gcptmpl fa = 2;
9 gcptmpl fs = 0.125;
10 gcptmpl
11 gcptmpl /* [Stock] */
12 gcptmpl stockXwidth = 219;
13 gcptmpl /* [Stock] */
14 gcptmpl stockYheight = 150;
15 gcptmpl /* [Stock] */
16 gcptmpl stockZthickness = 8.35;
17 gcptmpl /* [Stock] */
18 gcptmpl zeroheight = "Top"; // [Top, Bottom]
19 gcptmpl /* [Stock] */
20 gcptmpl stockzero = "Center"; // [Lower-Left, Center-Left, Top-Left, Center
    ]
21 gcptmpl /* [Stock] */
22 gcptmpl retractheight = 9;
23 gcptmpl
24 gcptmpl /* [Export] */
25 gcptmpl Base_filename = "export";
26 gcptmpl /* [Export] */
27 gcptmpl generatepaths = true;
28 gcptmpl /* [Export] */
29 gcptmpl generatedxf = true;
30 gcptmpl /* [Export] */
31 gcptmpl generategcode = true;
32 gcptmpl
33 gcptmpl /* [CAM] */
34 gcptmpl toolradius = 1.5875;
35 gcptmpl /* [CAM] */
36 gcptmpl large_square_tool_num = 0; // [0:0, 112:112, 102:102, 201:201]
37 gcptmpl /* [CAM] */
38 gcptmpl small_square_tool_num = 102; // [0:0, 122:122, 112:112, 102:102]
39 gcptmpl /* [CAM] */
40 gcptmpl large_ball_tool_num = 0; // [0:0, 111:111, 101:101, 202:202]
```

```

41 gcptmpl /* [CAM] */
42 gcptmpl small_ball_tool_num = 0; // [0:0, 121:121, 111:111, 101:101]
43 gcptmpl /* [CAM] */
44 gcptmpl large_V_tool_num = 0; // [0:0, 301:301, 690:690]
45 gcptmpl /* [CAM] */
46 gcptmpl small_V_tool_num = 0; // [0:0, 390:390, 301:301]
47 gcptmpl /* [CAM] */
48 gcptmpl DT_tool_num = 0; // [0:0, 814:814]
49 gcptmpl /* [CAM] */
50 gcptmpl KH_tool_num = 0; // [0:0, 374:374, 375:375, 376:376, 378:378]
51 gcptmpl /* [CAM] */
52 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.opendxxfile(Base_filename);
94 gcptmpl.opendxxfiles(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 cutZgcfed(-1.531, plunge);
252 gcptmpl //cutline(gcp.xpos(), gcp.ypos(), -1.531);

```

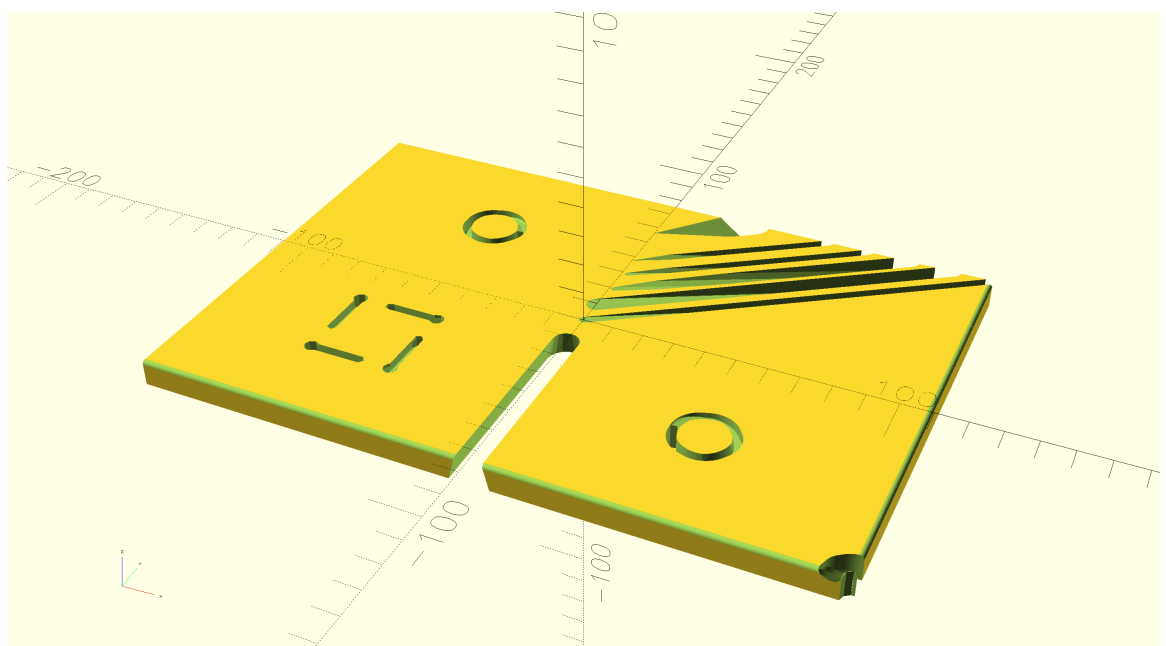


```

253 gcptmpl cutlinedxfgc(stockXwidth/2+0.508/2, -(stockYheight/2+0.508/2),
    -1.531);
254 gcptmpl
255 gcptmpl rapidZ(retractheight);
256 gcptmpl ##gcp.toolchange(56125, 10000)
257 gcptmpl cutZgcfeed(-1.531, plunge);
258 gcptmpl //toolpaths = toolpaths.union(gcp.cutline(gcp.xpos(), gcp.ypos(),
    -1.531))
259 gcptmpl cutlinedxfgc(stockXwidth/2+0.508/2, (stockYheight/2+0.508/2),
    -1.531);
260 gcptmpl
261 gcptmpl stockandtoolpaths();
262 gcptmpl //stockwotoolpaths();
263 gcptmpl //outputtoolpaths();
264 gcptmpl
265 gcptmpl //makecube(3, 2, 1);
266 gcptmpl
267 gcptmpl //instantiatecube();
268 gcptmpl
269 gcptmpl closegcodefile();
270 gcptmpl closedxffiles();
271 gcptmpl closedxffile();

```

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



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`)

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

3 *gcodepreview*

This library for PythonSCAD works by using Python code as a back-end so as to persistently store and access variables, and to write out files while both modeling the motion of a 3-axis CNC machine (note that at least a 4th additional axis may be worked up as a future option) and if desired, writing out DXF and/or G-code files (as opposed to the normal technique of rendering to a 3D model and writing out an STL or STEP or other model format and using a traditional CAM application). There are multiple modes for this, doing so requires two files:

- A Python file: `gcodepreview.py` (`gcpy`) — this has variables in the traditional sense which may be used for tracking machine position and so forth. Note that where it is placed/loaded from will depend on whether it is imported into a Python file:

```
import gcodepreview_standalone as gcp
```

or used in an OpenSCAD file:

```
use <gcodepreview.py>
```

with an additional OpenSCAD module which allows accessing it
- An OpenSCAD file: `gcodepreview.scad` (`gcpscad`) — which uses the Python file and which is included allowing it to access OpenSCAD variables for branching

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

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 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 3D object)
 - `rapid` (action — create 3D object so as to show a collision)
 - `open` (action (file))
 - `close` (action (file))
 - `set` (action/function) — note that the matching `get` is implicit in functions which return variables, e.g., `xpos()`
 - `current`
- Nouns
 - `arc`
 - `line`
 - `rectangle`
 - `circle`
- Suffixes
 - `feed` (parameter)
 - `gcode/gc` (filetype)
 - `pos` — position
 - `tool`
 - `loop`

- CC/CW
- number/num — note that num is used internally for variable names, making it straightforward to ensure that functions and variables have different names for purposes of scope

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

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

line			arc		
	cut	dxfgcode		cutdxfgcode	
cut	cutline	cutlinegc	cutarc	cutarcgc	
dxfgcode	cutlinedxf	dxflinedxf	cutarcdxf	dxfarcdxf	arcgc
	cutlinegc	linegc	cutarcgc	arcgc	
	cutlinedxfgc		cutarcdxfgc		

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

Principles for naming modules (and variables):

- minimize use of underscores (for convenience sake, underscores are not used for index entries)
- identify which aspect of the project structure is being worked with (`cut(ting)`, `dxfgcode`, `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 `dxfgcode` files
- OpenSCAD — see: <https://openscad.org/documentation.html>
- Programming in OpenSCAD with variables and calling Python — this requires 3 files and was originally used in the project as written up at: https://github.com/WillAdams/gcodepreview/blob/main/gcodepreview-openscad_0_6.pdf (for further details see below)
- Programming in OpenSCAD and calling Python where all variables as variables are held in Python classes (this is the technique used as of v0.8)
- Programming in Python and calling OpenSCAD — https://old.reddit.com/r/OpenPythonSCAD/comments/1heczmi/finally_using_scad_modules/

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

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

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

\end{writecode}
\addtocounter{gcpcscad}{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-arguments>

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

3.2 Implementation files and gcodepreview class

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

```
1 gcpy #!/usr/bin/env python
2 gcpy #icon "C:\Program Files\PythonSCAD\bin\openscad.exe" --trust-python
3 gcpy #Currently tested with PythonSCAD_nolibfive-2025.01.02-x86-64-Installer.exe and Python 3.11
4 gcpy #gcodepreview 0.8, for use with PythonSCAD,
5 gcpy #if using from PythonSCAD using OpenSCAD code, see gcodepreview.scad

6 gcpy
7 gcpy import sys
8 gcpy
9 gcpy # add math functions (using radians by default, convert to degrees
    where necessary)
10 gcpy import math
11 gcpy
12 gcpy # getting openscad functions into namespace
13 gcpy #https://github.com/gsohler/openscad/issues/39
14 gcpy try:
15 gcpy     from openscad import *
16 gcpy except ModuleNotFoundError as e:
17 gcpy     print("OpenSCAD module not loaded.")
18 gcpy
19 gcpy def pygcpversion():
20 gcpy     thegcpversion = 0.8
21 gcpy     return thegcpversion
```

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

```
1 gcpscad //!OpenSCAD
2 gcpscad
3 gcpscad //gcodepreview version 0.8
4 gcpscad //
5 gcpscad //used via include <gcodepreview.scad>;
6 gcpscad //
7 gcpscad
8 gcpscad use <gcodepreview.py>
9 gcpscad
10 gcpscad module gcpversion(){
11 gcpscad echo(pygcpversion());
12 gcpscad }
13 gcpscad
14 gcpscad //function myfunc(var) = gcp.myfunc(var);
15 gcpscad //
16 gcpscad //function getvv() = gcp.getvv();
17 gcpscad //
18 gcpscad //module makecube(xdim, ydim, zdim){
19 gcpscad //gcp.makecube(xdim, ydim, zdim);
```

```
20 gcpscad ///  
21 gcpscad ///  
22 gcpscad //module placecube(){  
23 gcpscad //gcp.placecube();  
24 gcpscad ///  
25 gcpscad ///  
26 gcpscad //module instantiatecube(){  
27 gcpscad //gcp.instantiatecube();  
28 gcpscad ///  
29 gcpscad ///
```

If all functions are to be handled within Python, then they will need to be gathered into a class which contains them and which is initialized so as to define shared variables, and 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
 - misc tool:
 - * MISC_tool_num, MISC_ratio

`gcodepreview` The class which is defined is `gcodepreview` which begins with the `init` method which allows passing in and defining the variables which will be used by the other methods in this class. Part of this includes handling various definitions for Boolean values.

```

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

```

```

41 gcpy          """
42 gcpy          Initialize gcodepreview object.
43 gcpy
44 gcpy          Parameters
45 gcpy          -----
46 gcpy          generatepaths : boolean
47 gcpy                          Determines if toolpaths will be stored
                                internally or returned directly
48 gcpy          generategcode : boolean
49 gcpy                          Enables writing out G-code.
50 gcpy          generatedxf   : boolean
51 gcpy                          Enables writing out DXF file(s).
52 gcpy
53 gcpy          Returns
54 gcpy          -----
55 gcpy          object
56 gcpy              The initialized gcodepreview object.
57 gcpy          """
58 gcpy #          self.basefilename = basefilename
59 gcpy          if (generatepaths == 1):
60 gcpy              self.generatepaths = True
61 gcpy          if (generatepaths == 0):
62 gcpy              self.generatepaths = False
63 gcpy          else:
64 gcpy              self.generatepaths = generatepaths
65 gcpy          if (generategcode == 1):
66 gcpy              self.generategcode = True
67 gcpy          if (generategcode == 0):
68 gcpy              self.generategcode = False
69 gcpy          else:
70 gcpy              self.generategcode = generategcode
71 gcpy          if (generatedxf == 1):
72 gcpy              self.generatedxf = True
73 gcpy          if (generatedxf == 0):
74 gcpy              self.generatedxf = False
75 gcpy          else:
76 gcpy              self.generatedxf = generatedxf
77 gcpy #          self.stockXwidth = stockXwidth
78 gcpy #          self.stockYheight = stockYheight
79 gcpy #          self.stockZthickness = stockZthickness
80 gcpy #          self.zeroheight = zeroheight
81 gcpy #          self.stockzero = stockzero
82 gcpy #          self.retractheight = retractheight
83 gcpy #          self.currenttoolnum = currenttoolnum
84 gcpy #          self.toolradius = toolradius
85 gcpy #          self.plunge = plunge
86 gcpy #          self.feed = feed
87 gcpy #          self.speed = speed
88 gcpy #          global toolpaths
89 gcpy #          if (openscadloaded == True):
90 gcpy #              self.toolpaths = cylinder(0.1, 0.1)
91 gcpy          self.generatedxfs = False
92 gcpy
93 gcpy          def checkgeneratepaths():
94 gcpy              return self.generatepaths
95 gcpy
96 gcpy #          def myfunc(self, var):
97 gcpy #              self.vv = var * var
98 gcpy #              return self.vv
99 gcpy #
100 gcpy #          def getvv(self):
101 gcpy #              return self.vv
102 gcpy #
103 gcpy #          def checkint(self):
104 gcpy #              return self.mc
105 gcpy #
106 gcpy #          def makecube(self, xdim, ydim, zdim):
107 gcpy #              self.c=cube([xdim, ydim, zdim])
108 gcpy #
109 gcpy #          def placecube(self):
110 gcpy #              output(self.c)
111 gcpy #
112 gcpy #          def instantiatecube(self):
113 gcpy #              return self.c
114 gcpy #

```

3.2.1 Position and Variables

In modeling the machine motion and G-code it will be necessary to have the machine track several variables for machine position, current tool, and the current depth in the current toolpath. This will be done using paired functions (which will set and return the matching variable) and a matching variable.

The first such variables are for xyz position:

- 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.

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

xpos
ypos
zpos

105 gcpy

def xpos(self):

106 gcpy

global mpx

107 gcpy

return self.mpx

108 gcpy

109 gcpy

def ypos(self):

110 gcpy

global mpy

111 gcpy

return self.mpy

112 gcpy

113 gcpy

def zpos(self):

114 gcpy

global mpz

115 gcpy

return self.mpz

116 gcpy

117 gcpy

def tpzinc(self):

118 gcpy

global tpzinc

119 gcpy

return self.tpzinc

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

30 gcpscad
31 gcpscad
32 gcpscad
33 gcpscad
34 gcpscad

function xpos()

= gcp.xpos();

function ypos()

= gcp.ypos();

function zpos()

= gcp.zpos();

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

setxpos
setypos
setzpos

121 gcpy

def setxpos(self, newxpos):

122 gcpy

global mpx

123 gcpy

self.mpx = newxpos

124 gcpy

125 gcpy

def setypos(self, newypos):

126 gcpy

global mpy

127 gcpy

self.mpy = newypos

128 gcpy

129 gcpy

def setzpos(self, newzpos):

130 gcpy

global mpz

131 gcpy

self.mpz = newzpos

132 gcpy

133 gcpy

def settpzinc(self, newtpzinc):

134 gcpy

global tpzinc

135 gcpy

self.tpzinc = newtpzinc

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

3.2.2 Initial Modules

gcodepreview

setupstock

gcp.setupstock

The first such routine, (actually a subroutine, see gcodepreview) setupstock will be appropriately enough, to set up the stock, and perform other initializations — initially, the only thing done in Python was to set the value of the persistent (Python) variables, but the rewritten standalone version handles all necessary actions.

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.

```
137 gcpy      def setupstock(self, stockXwidth,
138 gcpy                      stockYheight,
139 gcpy                      stockZthickness,
140 gcpy                      zeroheight,
141 gcpy                      stockzero,
142 gcpy                      retractheight):
143 gcpy      """
144 gcpy      Set up blank/stock for material and position/zero.
145 gcpy
146 gcpy      Parameters
147 gcpy      -----
148 gcpy      stockXwidth : float
149 gcpy                      X extent/dimension
150 gcpy      stockYheight : float
151 gcpy                      Y extent/dimension
152 gcpy      stockZthickness : boolean
153 gcpy                      Z extent/dimension
154 gcpy      zeroheight : string
155 gcpy                      Top or Bottom, determines if Z extent will
                        be positive or negative
156 gcpy      stockzero : string
157 gcpy                      Lower-Left, Center-Left, Top-Left, Center,
                        determines XY position of stock
158 gcpy      retractheight : float
159 gcpy                      Distance which tool retracts above surface
                        of stock.
160 gcpy
161 gcpy      Returns
162 gcpy      -----
163 gcpy      none
164 gcpy      """
165 gcpy      self.stockXwidth = stockXwidth
166 gcpy      self.stockYheight = stockYheight
167 gcpy      self.stockZthickness = stockZthickness
168 gcpy      self.zeroheight = zeroheight
169 gcpy      self.stockzero = stockzero
170 gcpy      self.retractheight = retractheight
171 gcpy      # global mpx
172 gcpy      self.mpx = float(0)
173 gcpy      # global mpy
174 gcpy      self.mpy = float(0)
175 gcpy      # global mpz
176 gcpy      self.mpz = float(0)
177 gcpy      # global tpz
178 gcpy      # self.tpzinc = float(0)
179 gcpy      # global currenttoolnum
180 gcpy      self.currenttoolnum = 102
181 gcpy      # global currenttoolshape
182 gcpy      self.currenttoolshape = cylinder(12.7, 1.5875)
183 gcpy      self.rapids = self.currenttoolshape
184 gcpy      # global stock
185 gcpy      self.stock = cube([stockXwidth, stockYheight,
                        stockZthickness])
186 gcpy      %%WRITEGC      if self.generategcode == True:
187 gcpy      %%WRITEGC      self.writegc("(Design File: " + self.
                        basefilename + ")")
188 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.

```
189 gcpy      if self.zeroheight == "Top":
```



```

190 gcpy          if self.stockzero == "Lower-Left":
191 gcpy              self.stock = stock.translate([0, 0, -self.
                    stockZthickness])
192 gcpy              if self.generategcode == True:
193 gcpy                  self.writegc("(stockMin:0.00mm,␣0.00mm,␣-", str
                    (self.stockZthickness), "mm)")
194 gcpy                  self.writegc("(stockMax:", str(self.stockXwidth
                    ), "mm,␣", str(stockYheight), "mm,␣0.00mm)")
195 gcpy                  self.writegc("(STOCK/BLOCK,␣", str(self.
                    stockXwidth), "␣", str(self.stockYheight),
                    "␣", str(self.stockZthickness), "␣0.00,␣
                    0.00,␣", str(self.stockZthickness), ")")
196 gcpy          if self.stockzero == "Center-Left":
197 gcpy              self.stock = self.stock.translate([0, -stockYheight
                    / 2, -stockZthickness])
198 gcpy              if self.generategcode == True:
199 gcpy                  self.writegc("(stockMin:0.00mm,␣-", str(self.
                    stockYheight/2), "mm,␣-", str(self.
                    stockZthickness), "mm)")
200 gcpy                  self.writegc("(stockMax:", str(self.stockXwidth
                    ), "mm,␣", str(self.stockYheight/2), "mm,␣
                    0.00mm)")
201 gcpy                  self.writegc("(STOCK/BLOCK,␣", str(self.
                    stockXwidth), "␣", str(self.stockYheight),
                    "␣", str(self.stockZthickness), "␣0.00,␣",
                    str(self.stockYheight/2), "␣", str(self.
                    stockZthickness), ")");
202 gcpy          if self.stockzero == "Top-Left":
203 gcpy              self.stock = self.stock.translate([0, -self.
                    stockYheight, -self.stockZthickness])
204 gcpy              if self.generategcode == True:
205 gcpy                  self.writegc("(stockMin:0.00mm,␣-", str(self.
                    stockYheight), "mm,␣-", str(self.
                    stockZthickness), "mm)")
206 gcpy                  self.writegc("(stockMax:", str(self.stockXwidth
                    ), "mm,␣0.00mm,␣0.00mm)")
207 gcpy                  self.writegc("(STOCK/BLOCK,␣", str(self.
                    stockXwidth), "␣", str(self.stockYheight),
                    "␣", str(self.stockZthickness), "␣0.00,␣",
                    str(self.stockYheight), "␣", str(self.
                    stockZthickness), ")")
208 gcpy          if self.stockzero == "Center":
209 gcpy              self.stock = self.stock.translate([-self.
                    stockXwidth / 2, -self.stockYheight / 2, -self.
                    stockZthickness])
210 gcpy              if self.generategcode == True:
211 gcpy                  self.writegc("(stockMin:␣-", str(self.
                    stockXwidth/2), "␣-", str(self.stockYheight
                    /2), "mm,␣-", str(self.stockZthickness), "mm
                    )")
212 gcpy                  self.writegc("(stockMax:", str(self.stockXwidth
                    /2), "mm,␣", str(self.stockYheight/2), "mm,␣
                    0.00mm)")
213 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), ")")
214 gcpy          if self.zeroheight == "Bottom":
215 gcpy              if self.stockzero == "Lower-Left":
216 gcpy                  self.stock = self.stock.translate([0, 0, 0])
217 gcpy                  if self.generategcode == True:
218 gcpy                      self.writegc("(stockMin:0.00mm,␣0.00mm,␣0.00mm
                    )")
219 gcpy                      self.writegc("(stockMax:", str(self.
                    stockXwidth), "mm,␣", str(self.stockYheight
                    ), "mm,␣", str(self.stockZthickness), "mm)"
                    )
220 gcpy                      self.writegc("(STOCK/BLOCK,␣", str(self.
                    stockXwidth), "␣", str(self.stockYheight),
                    "␣", str(self.stockZthickness), "␣0.00,␣
                    0.00,␣0.00)")
221 gcpy          if self.stockzero == "Center-Left":
222 gcpy              self.stock = self.stock.translate([0, -self.
                    stockYheight / 2, 0])
223 gcpy              if self.generategcode == True:
224 gcpy                  self.writegc("(stockMin:0.00mm,␣-", str(self.

```

```

        stockYheight/2), "mm,␣0.00mm)")
225 gcpy          self.writegc("(stockMax:", str(self.stockXwidth
        ), "mm,␣", str(self.stockYheight/2), "mm,␣-"
        , str(self.stockZthickness), "mm)")
226 gcpy          self.writegc("(STOCK/BLOCK,␣", str(self.
        stockXwidth), ",␣", str(self.stockYheight),
        ",␣", str(self.stockZthickness), ",␣0.00,␣",
        str(self.stockYheight/2), ",␣0.00mm)");
227 gcpy          if self.stockzero == "Top-Left":
228 gcpy              self.stock = self.stock.translate([0, -self.
        stockYheight, 0])
229 gcpy          if self.generategcode == True:
230 gcpy              self.writegc("(stockMin:0.00mm,␣-", str(self.
        stockYheight), "mm,␣0.00mm)")
231 gcpy          self.writegc("(stockMax:", str(self.stockXwidth
        ), "mm,␣0.00mm,␣", str(self.stockZthickness)
        , "mm)")
232 gcpy          self.writegc("(STOCK/BLOCK,␣", str(self.
        stockXwidth), ",␣", str(self.stockYheight),
        ",␣", str(self.stockZthickness), ",␣0.00,␣",
        str(self.stockYheight), ",␣0.00)")
233 gcpy          if self.stockzero == "Center":
234 gcpy              self.stock = self.stock.translate([-self.
        stockXwidth / 2, -self.stockYheight / 2, 0])
235 gcpy          if self.generategcode == True:
236 gcpy              self.writegc("(stockMin:␣-", str(self.
        stockXwidth/2), ",␣-", str(self.stockYheight
        /2), "mm,␣0.00mm)")
237 gcpy          self.writegc("(stockMax:", str(self.stockXwidth
        /2), "mm,␣", str(self.stockYheight/2), "mm,␣"
        , str(self.stockZthickness), "mm)")
238 gcpy          self.writegc("(STOCK/BLOCK,␣", str(self.
        stockXwidth), ",␣", str(self.stockYheight),
        ",␣", str(self.stockZthickness), ",␣", str(
        self.stockXwidth/2), ",␣", str(self.
        stockYheight/2), ",␣0.00)")
239 gcpy          if self.generategcode == True:
240 gcpy              self.writegc("G90");
241 gcpy              self.writegc("G21");
```

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

```

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

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

```

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

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

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

3.3 Tools and Changes

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

```

243 gcpy          def settool(self, tn):
244 gcpy #              global currenttoolnum
245 gcpy              self.currenttoolnum = tn
246 gcpy
247 gcpy          def currenttoolnumber(self):
248 gcpy #              global currenttoolnum
249 gcpy              return self.currenttoolnum
250 gcpy
251 gcpy #          def currentroundovertoolnumber(self):
252 gcpy #              global Roundover_tool_num
```

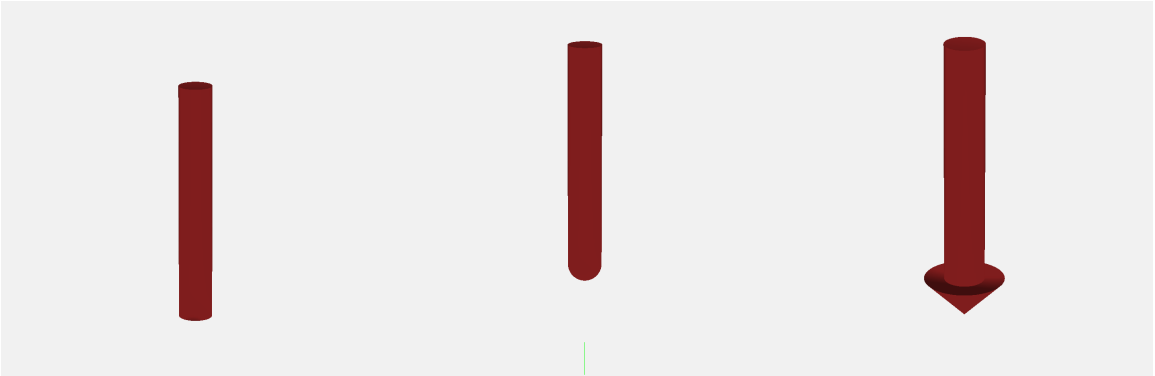
```
253 gcpy #          return self.Roundover_tool_num
```

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

3.3.1 3D Shapes for Tools

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

3.3.1.1 Normal Tooling/toolshapes Most tooling has quite standard shapes and are defined by their profile:



- Square (#201 and 102) — able to cut a flat bottom, perpendicular side and right angle their simple and easily understood geometry makes them a standard choice (a radiused form with a flat bottom, often described as a “bowl bit” is not implemented as-of-yet)
- Ballnose (#202 and 101) — rounded, they are the standard choice for concave and organic shapes
- V tooling (#301, 302 and 390) — pointed at the tip, they are available in a variety of angles and diameters and may be used for decorative V carving, or for chamfering or cutting specific angles (note that the commonly available radiused form is not implemented at this time, *e.g.*, #501 and 502)

Most tools are easily implemented with concise 3D descriptions which may be connected with a simple `hull` operation. 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:

```
255 gcpy      def endmill_square(self, es_diameter, es_flute_length):
256 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:

```
258 gcpy      def ballnose(self, es_diameter, es_flute_length):
259 gcpy          b = sphere(r=(es_diameter / 2))
260 gcpy          s = cylinder(r1=(es_diameter / 2), r2=(es_diameter / 2), h=
                        es_flute_length, center=False)
261 gcpy          p = union(b, s)
262 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):

```
264 gcpy      def endmill_v(self, es_v_angle, es_diameter):
265 gcpy          es_v_angle = math.radians(es_v_angle)
266 gcpy          v = cylinder(r1=0, r2=(es_diameter / 2), h=((es_diameter /
                        2) / math.tan((es_v_angle / 2))), center=False)
267 gcpy          s = cylinder(r1=(es_diameter / 2), r2=(es_diameter / 2), h=
                        ((es_diameter * 8) ), center=False)
268 gcpy          sh = s.translate([0, 0, ((es_diameter / 2) / math.tan((
                        es_v_angle / 2)))]])
269 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:

```
271 gcpy      def bowl_tool(self, radius, diameter, height):
272 gcpy          bts = cylinder(height - radius, diameter / 2, diameter / 2,
                                center=False)
273 gcpy          bts = bts.translate([0, 0, radius])
274 gcpy          bts = bts.union(cylinder(height, diameter / 2 - radius,
                                diameter / 2 - radius, center=False))
275 gcpy          for i in range(90):
276 gcpy #              print(math.sin(math.radians(i)))
277 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)
278 gcpy          bts = hull(bts, slice.translate([0, 0, (radius - radius
                                * math.cos(math.radians(i)))]))
279 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.

```
281 gcpy      def tapered_ball(self, es_tip, es_diameter, es_flute_length,
                                es_angle):
282 gcpy          b = sphere(r=(es_tip / 2))
283 gcpy          s = cylinder(r1=(es_tip / 2), r2=(es_diameter / 2), h=
                                es_flute_length, center=False)
284 gcpy          p = union(b, s)
285 gcpy          return p.translate([0, 0, (es_tip / 2)])
```

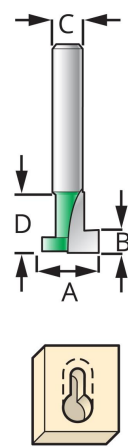
flat V The flat V tool is modeled as a cylinder with two different diameters, forming a truncated cone.

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

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

- Keyhole tools — intended to cut slots for retaining hardware used for picture hanging, they may be used to create slots for other purposes Note that it will be necessary to model these twice, once for the shaft, the second time for the actual keyhole cutting <https://assetssc.leevalley.com/en-gb/shop/tools/power-tool-accessories/router-bits/30113-keyhole-router-bits>
- Dovetail cutters — used for the joinery of the same name, they cut a large area at the bottom which slants up to a narrower region at a defined angle
- Lollipop cutters — normally used for 3D work, as their name suggests they are essentially a (cutting) ball on a narrow stick (the tool shaft), they are mentioned here only for completeness' sake and are not (at this time) implemented
- Threadmill — used for cutting threads, normally a single form geometry is used on a CNC.

3.3.1.2.1 Keyhole tools Keyhole toolpaths (see: subsection 3.4.3.2.3 are intended for use with tooling which projects beyond the the narrower shaft and so will cut usefully underneath the visible surface. Also described as “undercut” tooling, but see below.



Keyhole Router Bits

#	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:

```
293 gcpy      def keyhole(self, es_diameter, es_flute_length):
294 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:

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

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

```
299 gcpy      def threadmill(self, minor_diameter, major_diameter, cut_height):
300 gcpy          btm = cylinder(r1=(minor_diameter / 2), r2=(major_diameter / 2), h=cut_height, center = False)
301 gcpy          top = cylinder(r1=(major_diameter / 2), r2=(minor_diameter / 2), h=cut_height, center = False)
302 gcpy          top = top.translate([0, 0, cut_height/2])
303 gcpy          tm = btm.union(top)
304 gcpy          return tm
305 gcpy
306 gcpy      def threadmill_shaft(self, diameter, cut_height, height):
307 gcpy          shaft = cylinder(r1=(diameter / 2), r2=(diameter / 2), h=height, center = False)
308 gcpy          shaft = shaft.translate([0, 0, cut_height/2])
309 gcpy          return shaft
```

dovetail **3.3.1.2.3 Dovetails** The dovetail is modeled as a cylinder with the differing bottom and top diameters determining the angle (though dt_angle is still required as a parameter)

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

3.3.1.3 Concave toolshapes While normal tooling may be represented with a single hull operation betwixt two 3D toolshapes (or four in the instance of keyhole tools), concave tooling such as roundover/radius tooling require multiple sections or even slices of the tool shape to be modeled separately which are then hulled together. Something of this can be seen in the manual work-around for previewing them: <https://community.carbide3d.com/t/using-unsupported-tooling-in-carbide-create-roundover-cove-radius-bits/43723>.

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

3.3.1.4 Roundover tooling It is not possible to represent all tools using tool changes as coded above which require using a hull operation between 3D representations of the tools at the be-

gining and end points. Tooling which cannot be so represented will be implemented separately below, see paragraph 3.3.1.3.

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

3.3.2 toolchange

toolchange Then apply the appropriate commands for a toolchange. Note that it is expected that this code will be updated as needed when new tooling is introduced as additional modules which require specific tooling are added.

Note that the comments written out in G-code correspond to those used by the G-code pre-viewing tool CutViewer (which is unfortunately, no longer readily available).

A further concern is that early versions often passed the tool into a module using a parameter. That ceased to be necessary in the 2024.09.03 version of PythonSCAD, and all modules should read the tool # from currenttoolnumber().

Note that there are many varieties of tooling and not all will be implemented, especially in the early iterations of this project.

3.3.2.1 Selecting Tools The original implementation created the model for the tool at the current position, and a duplicate at the end position, wrapping the twain for each end of a given movement in a hull() command. This approach will not work within Python, so it will be necessary to instead assign and select the tool as part of the cutting command indirectly by first storing 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

```
314 gcpy     def currenttool(self):
315 gcpy #         global currenttoolshape
316 gcpy         return self.currenttoolshape
```

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

3.3.2.2 Square and ball nose (including tapered ball nose)

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

3.3.2.3 Roundover (corner rounding)

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

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

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

```
318 gcpy     def toolchange(self, tool_number, speed = 10000):
319 gcpy #         global currenttoolshape
320 gcpy         self.currenttoolshape = self.endmill_square(0.001, 0.001)
321 gcpy
322 gcpy         self.settool(tool_number)
323 gcpy         if (self.generategcode == True):
324 gcpy             self.writegc("(Toolpath)")
325 gcpy             self.writegc("M05")
```

```

326 gcpy          if (tool_number == 201):
327 gcpy              self.writegc("(TOOL/MILL,␣6.35,␣0.00,␣0.00,␣0.00)")
328 gcpy              self.currenttoolshape = self.endmill_square(6.35,
329 gcpy                  19.05)
330 gcpy          elif (tool_number == 102):
331 gcpy              self.writegc("(TOOL/MILL,␣3.175,␣0.00,␣0.00,␣0.00)")
332 gcpy              self.currenttoolshape = self.endmill_square(3.175,
333 gcpy                  12.7)
334 gcpy          elif (tool_number == 112):
335 gcpy              self.writegc("(TOOL/MILL,␣1.5875,␣0.00,␣0.00,␣0.00)")
336 gcpy              self.currenttoolshape = self.endmill_square(1.5875,
337 gcpy                  6.35)
338 gcpy          elif (tool_number == 122):
339 gcpy              self.writegc("(TOOL/MILL,␣0.79375,␣0.00,␣0.00,␣0.00)")
340 gcpy              self.currenttoolshape = self.endmill_square(0.79375,
341 gcpy                  1.5875)
342 gcpy          elif (tool_number == 202):
343 gcpy              self.writegc("(TOOL/MILL,␣6.35,␣3.175,␣0.00,␣0.00)")
344 gcpy              self.currenttoolshape = self.ballnose(6.35, 19.05)
345 gcpy          elif (tool_number == 101):
346 gcpy              self.writegc("(TOOL/MILL,␣3.175,␣1.5875,␣0.00,␣0.00)")
347 gcpy              self.currenttoolshape = self.ballnose(3.175, 12.7)
348 gcpy          elif (tool_number == 111):
349 gcpy              self.writegc("(TOOL/MILL,␣1.5875,␣0.79375,␣0.00,␣0.00)"
350 gcpy                  )
351 gcpy              self.currenttoolshape = self.ballnose(1.5875, 6.35)
352 gcpy          elif (tool_number == 121):
353 gcpy              self.writegc("(TOOL/MILL,␣3.175,␣0.79375,␣0.00,␣0.00)")
354 gcpy              self.currenttoolshape = self.ballnose(0.79375, 1.5875)
355 gcpy          elif (tool_number == 327):
356 gcpy              self.writegc("(TOOL/MILL,␣0.03,␣0.00,␣13.4874,␣30.00)")
357 gcpy              self.currenttoolshape = self.endmill_v(60, 26.9748)
358 gcpy          elif (tool_number == 301):
359 gcpy              self.writegc("(TOOL/MILL,␣0.03,␣0.00,␣6.35,␣45.00)")
360 gcpy              self.currenttoolshape = self.endmill_v(90, 12.7)
361 gcpy          elif (tool_number == 302):
362 gcpy              self.writegc("(TOOL/MILL,␣0.03,␣0.00,␣10.998,␣30.00)")
363 gcpy              self.currenttoolshape = self.endmill_v(60, 12.7)
364 gcpy          elif (tool_number == 390):
365 gcpy              self.writegc("(TOOL/MILL,␣0.03,␣0.00,␣1.5875,␣45.00)")
366 gcpy              self.currenttoolshape = self.endmill_v(90, 3.175)
367 gcpy          elif (tool_number == 374):
368 gcpy              self.writegc("(TOOL/MILL,␣9.53,␣0.00,␣3.17,␣0.00)")
369 gcpy          elif (tool_number == 375):
370 gcpy              self.writegc("(TOOL/MILL,␣9.53,␣0.00,␣3.17,␣0.00)")
371 gcpy          elif (tool_number == 376):
372 gcpy              self.writegc("(TOOL/MILL,␣12.7,␣0.00,␣4.77,␣0.00)")
373 gcpy          elif (tool_number == 378):
374 gcpy              self.writegc("(TOOL/MILL,␣12.7,␣0.00,␣4.77,␣0.00)")
375 gcpy          elif (tool_number == 814):
376 gcpy              self.writegc("(TOOL/MILL,␣12.7,␣6.367,␣12.7,␣0.00)")
377 gcpy              #dt_bottomdiameter, dt_topdiameter, dt_height, dt_angle
378 gcpy              )
379 gcpy              #https://www.leevalley.com/en-us/shop/tools/power-tool-
380 gcpy              accessories/router-bits/30172-dovetail-bits?item=18
381 gcpy              J1607
382 gcpy              self.currenttoolshape = self.dovetail(12.7, 6.367,
383 gcpy                  12.7, 14)
384 gcpy          elif (tool_number == 56125):#0.508/2, 1.531
385 gcpy              self.writegc("(TOOL/CRMILL,␣0.508,␣6.35,␣3.175,␣7.9375,
386 gcpy                  ␣3.175)")
387 gcpy          elif (tool_number == 56142):#0.508/2, 2.921
388 gcpy              self.writegc("(TOOL/CRMILL,␣0.508,␣3.571875,␣1.5875,␣
389 gcpy                  5.55625,␣1.5875)")
390 gcpy          elif (tool_number == 312):#1.524/2, 3.175
391 gcpy              self.writegc("(TOOL/CRMILL, Diameter1, Diameter2,
392 gcpy                  Radius, Height, Length)")
393 gcpy          elif (tool_number == 1570):#0.507/2, 4.509
394 gcpy              self.writegc("(TOOL/CRMILL,␣0.17018,␣9.525,␣4.7625,␣
395 gcpy                  12.7,␣4.7625)")
396 gcpy          #https://www.amanatool.com/45982-carbide-tipped-bowl-tray-1-4-
397 gcpy          radius-x-3-4-dia-x-5-8-x-1-4-inch-shank.html
398 gcpy          elif (tool_number == 45982):#0.507/2, 4.509
399 gcpy              self.writegc("(TOOL/MILL,␣15.875,␣6.35,␣19.05,␣0.00)")
400 gcpy              self.currenttoolshape = self.bowl_tool(6.35, 19.05,
401 gcpy                  15.875)
402 gcpy          elif (tool_number == 204):#
403 gcpy              self.writegc("(")

```

```

389 gcpy          self.currenttoolshape = self.tapered_ball(1.5875, 6.35,
390                  38.1, 3.6)
391 gcpy          elif (tool_number == 304):#
392 gcpy              self.writegc("(")
393 gcpy              self.currenttoolshape = self.tapered_ball(3.175, 6.35,
394                  38.1, 2.4)
395 gcpy          elif (tool_number == 501):#
396 gcpy              self.writegc("(")
397 gcpy              self.currenttoolshape = self.tapered_ball(0.127, 3.175,
398                  2.688, 60)
399 gcpy          elif (tool_number == 502):#
400 gcpy              self.writegc("(")
401 gcpy              self.currenttoolshape = self.tapered_ball(0.127, 3.175,
402                  4.25, 40)
403 gcpy          elif (tool_number == 13921):#
404 gcpy              self.writegc("(")
405 gcpy              self.currenttoolshape = self.flat_V(6.35, 31.75, 12.7,
406                  45)

```



```
417 gcpy                return ptd_tool
418 gcpy                if ptd_tool == 112:
419 gcpy                    if ptd_depth > 0.79375:
420 gcpy                        return 1.5875
421 gcpy                    else:
422 gcpy                        return ptd_tool
423 gcpy                if ptd_tool == 101:
424 gcpy                    if ptd_depth > 1.5875:
425 gcpy                        return 3.175
426 gcpy                    else:
427 gcpy                        return ptd_tool
428 gcpy                if ptd_tool == 202:
429 gcpy                    if ptd_depth > 3.175:
430 gcpy                        return 6.35
431 gcpy                    else:
432 gcpy                        return ptd_tool
433 gcpy # V 301, 302, 390
434 gcpy                if ptd_tool == 301:
435 gcpy                    return ptd_tool
436 gcpy                if ptd_tool == 302:
437 gcpy                    return ptd_tool
438 gcpy                if ptd_tool == 390:
439 gcpy                    return ptd_tool
440 gcpy # Keyhole
441 gcpy                if ptd_tool == 374:
442 gcpy                    if ptd_depth < 3.175:
443 gcpy                        return 9.525
444 gcpy                    else:
445 gcpy                        return 6.35
446 gcpy                if ptd_tool == 375:
447 gcpy                    if ptd_depth < 3.175:
448 gcpy                        return 9.525
449 gcpy                    else:
450 gcpy                        return 8
451 gcpy                if ptd_tool == 376:
452 gcpy                    if ptd_depth < 4.7625:
453 gcpy                        return 12.7
454 gcpy                    else:
455 gcpy                        return 6.35
456 gcpy                if ptd_tool == 378:
457 gcpy                    if ptd_depth < 4.7625:
458 gcpy                        return 12.7
459 gcpy                    else:
460 gcpy                        return 8
461 gcpy # Dovetail
462 gcpy                if ptd_tool == 814:
463 gcpy                    if ptd_depth > 12.7:
464 gcpy                        return 6.35
465 gcpy                    else:
466 gcpy                        return 12.7
467 gcpy # Bowl Bit
468 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
469 gcpy                if ptd_tool == 45982:
470 gcpy                    if ptd_depth > 6.35:
471 gcpy                        return 15.875
472 gcpy # Tapered Ball Nose
473 gcpy                if ptd_tool == 204:
474 gcpy                    if ptd_depth > 6.35:
475 gcpy                        return 0
476 gcpy                if ptd_tool == 304:
477 gcpy                    if ptd_depth > 6.35:
478 gcpy                        return 0
479 gcpy                    else:
480 gcpy                        return 0
```

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

```
482 gcpy    def tool_radius(self, ptd_tool, ptd_depth):
483 gcpy        tr = self.tool_diameter(ptd_tool, ptd_depth)/2
484 gcpy        return tr
```

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

3.3.4 Feeds and Speeds

feed There are several possibilities for handling feeds and speeds. Currently, base values for feed, plunge, 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

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

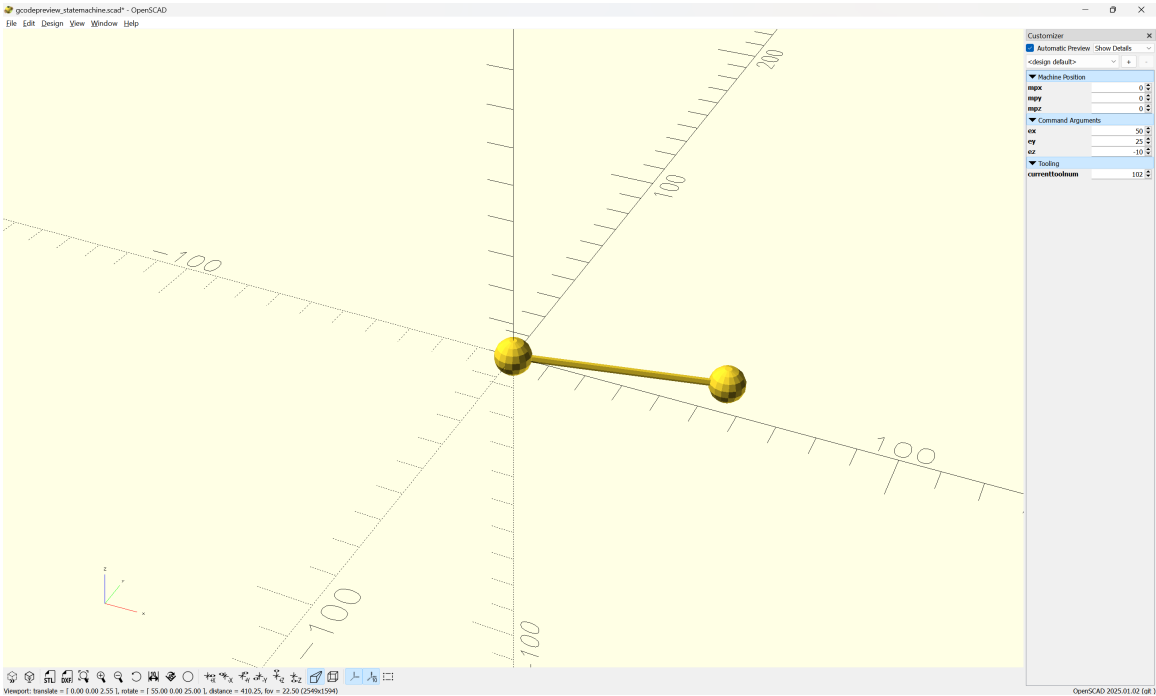
`rapid...` Note that the variables `self.rapids` and `self.toolpaths` are used to hold the accumulated (unioned) 3D models of the rapid motions and cuts so that they may be differenced from the stock when the value `generatepaths` is set to `True`.

In order to manage the various options when cutting it will be necessary to have a command where the actual cut is made, passing in the shape used for the cut as a parameter. Since the 3D aspect of rapid and cut operations are fundamentally the same, the command `rsc` 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:

```
486 gcpy      def rcs(self, ex, ey, ez, shape):
487 gcpy          start = shape
488 gcpy          end = shape
489 gcpy          toolpath = hull(start.translate([self.xpos(), self.ypos(),
490 gcpy                                     self.zpos()]),
491 gcpy                                     end.translate([ex, ey, ez]))
491 gcpy          return toolpath
```

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

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



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

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

```

493 gcpy      def rapid(self, ex, ey, ez):
494 gcpy          cts = self.currenttoolshape
495 gcpy          toolpath = self.rcs(ex, ey, ez, cts)
496 gcpy          self.setxpos(ex)
497 gcpy          self.setypos(ey)
498 gcpy          self.setzpos(ez)
499 gcpy          if self.generatepaths == True:
500 gcpy              self.rapids = self.rapids.union(toolpath)
501 gcpy          #          return cylinder(0.01, 0, 0.01, center = False, fn = 3)
502 gcpy          return cube([0.001, 0.001, 0.001])
503 gcpy      else:
504 gcpy          return toolpath
505 gcpy
506 gcpy      def cutshape(self, ex, ey, ez):
507 gcpy          cts = self.currenttoolshape
508 gcpy          toolpath = self.rcs(ex, ey, ez, cts)
509 gcpy          if self.generatepaths == True:
510 gcpy              self.toolpaths = self.toolpaths.union(toolpath)
511 gcpy              return cube([0.001, 0.001, 0.001])
512 gcpy          else:
513 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:

```

515 gcpy      def movetosafeZ(self):
516 gcpy          rapid = self.rapid(self.xpos(), self.ypos(), self.
                    retractheight)
517 gcpy      #          if self.generatepaths == True:
518 gcpy      #          rapid = self.rapid(self.xpos(), self.ypos(), self.
                    retractheight)
519 gcpy      #          self.rapids = self.rapids.union(rapid)
520 gcpy      #      else:
521 gcpy      #      if (generategcode == true) {
522 gcpy      #      //      writecomment("PREPOSITION FOR RAPID PLUNGE");Z25.650
523 gcpy      #      //G1Z24.663F381.0, "F", str(plunge)
524 gcpy          if self.generatepaths == False:
525 gcpy              return rapid
526 gcpy          else:
527 gcpy              return cube([0.001, 0.001, 0.001])
528 gcpy
529 gcpy      def rapidXY(self, ex, ey):
530 gcpy          rapid = self.rapid(ex, ey, self.zpos())
531 gcpy      #          if self.generatepaths == True:
532 gcpy      #          self.rapids = self.rapids.union(rapid)
533 gcpy      #      else:
534 gcpy          if self.generatepaths == False:
535 gcpy              return rapid
536 gcpy
537 gcpy      def rapidZ(self, ez):
538 gcpy          rapid = self.rapid(self.xpos(), self.ypos(), ez)
539 gcpy      #          if self.generatepaths == True:
540 gcpy      #          self.rapids = self.rapids.union(rapid)
541 gcpy      #      else:
542 gcpy          if self.generatepaths == False:
543 gcpy              return rapid

```

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

```

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

```

3.4.1 Lines

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

```

545 gcpy      def cutline(self, ex, ey, ez):\
546 gcpy      #below will need to be integrated into if/then structure not yet
              copied
547 gcpy      #          cts = self.currenttoolshape
548 gcpy      if (self.currenttoolnumber() == 374):
549 gcpy      #          self.writegc("(TOOL/MILL, 9.53, 0.00, 3.17, 0.00)")
550 gcpy      self.currenttoolshape = self.keyhole(9.53/2, 3.175)
551 gcpy      toolpath = self.cutshape(ex, ey, ez)
552 gcpy      self.currenttoolshape = self.keyhole_shaft(6.35/2,
              12.7)
553 gcpy      toolpath = toolpath.union(self.cutshape(ex, ey, ez))
554 gcpy      elif (self.currenttoolnumber() == 375):
555 gcpy      #          self.writegc("(TOOL/MILL, 9.53, 0.00, 3.17, 0.00)")
556 gcpy      #          elif (self.currenttoolnumber() == 376):
557 gcpy      #          self.writegc("(TOOL/MILL, 12.7, 0.00, 4.77, 0.00)")
558 gcpy      #          elif (self.currenttoolnumber() == 378):
559 gcpy      #          self.writegc("(TOOL/MILL, 12.7, 0.00, 4.77, 0.00)")
560 gcpy      #          elif (self.currenttoolnumber() == 56125):#0.508/2, 1.531
561 gcpy      #          self.writegc("(TOOL/CRMILL, 0.508, 6.35, 3.175,
              7.9375, 3.175)")
562 gcpy      elif (self.currenttoolnumber() == 56142):#0.508/2, 2.921
563 gcpy      #          self.writegc("(TOOL/CRMILL, 0.508, 3.571875, 1.5875,
              5.55625, 1.5875)")
564 gcpy      toolpath = self.cutroundovertool(self.xpos(), self.ypos
              (), self.zpos(), ex, ey, ez, 0.508/2, 1.531)
565 gcpy      #          elif (self.currenttoolnumber() == 1570):#0.507/2, 4.509
566 gcpy      #          self.writegc("(TOOL/CRMILL, 0.17018, 9.525, 4.7625,
              12.7, 4.7625)")
567 gcpy      else:
568 gcpy      toolpath = self.cutshape(ex, ey, ez)
569 gcpy      self.setxpos(ex)
570 gcpy      self.setypos(ey)
571 gcpy      self.setzpos(ez)
572 gcpy      #          if self.generatepaths == True:
573 gcpy      #          self.toolpaths = union([self.toolpaths, toolpath])
574 gcpy      #          else:
575 gcpy      if self.generatepaths == False:
576 gcpy      return toolpath
577 gcpy      else:
578 gcpy      return cube([0.001, 0.001, 0.001])
579 gcpy
580 gcpy      def cutlinedxfgc(self, ex, ey, ez):
581 gcpy      self.dxfline(self.currenttoolnumber(), self.xpos(), self.
              ypos(), ex, ey)
582 gcpy      self.writegc("G01_X", str(ex), "Y", str(ey), "Z", str(ez)
              )
583 gcpy      #          if self.generatepaths == False:
584 gcpy      return self.cutline(ex, ey, ez)
585 gcpy
586 gcpy      def cutroundovertool(self, bx, by, bz, ex, ey, ez,
              tool_radius_tip, tool_radius_width, stepsizeroundover = 1):
587 gcpy      #          n = 90 + fn*3
588 gcpy      #          print("Tool dimensions", tool_radius_tip,
              tool_radius_width, "begin ", bx, by, bz, "end ", ex, ey, ez)
589 gcpy      step = 4 #360/n

```

```
590 gcpy      shaft = cylinder(step, tool_radius_tip, tool_radius_tip)
591 gcpy      toolpath = hull(shaft.translate([bx, by, bz]), shaft.
                        translate([ex, ey, ez]))
592 gcpy      shaft = cylinder(tool_radius_width*2, tool_radius_tip+
                        tool_radius_width, tool_radius_tip+tool_radius_width)
593 gcpy      toolpath = toolpath.union(hull(shaft.translate([bx, by, bz+
                        tool_radius_width]), shaft.translate([ex, ey, ez+
                        tool_radius_width]))))
594 gcpy      for i in range(1, 90, stepsizeroundover):
595 gcpy          angle = i
596 gcpy          dx = tool_radius_width*math.cos(math.radians(angle))
597 gcpy          dxx = tool_radius_width*math.cos(math.radians(angle+1))
598 gcpy          dzz = tool_radius_width*math.sin(math.radians(angle))
599 gcpy          dz = tool_radius_width*math.sin(math.radians(angle+1))
600 gcpy          dh = abs(dzz-dz)+0.0001
601 gcpy          slice = cylinder(dh, tool_radius_tip+tool_radius_width-
                        dx, tool_radius_tip+tool_radius_width-dxx)
602 gcpy          toolpath = toolpath.union(hull(slice.translate([bx, by,
                        bz+dz]), slice.translate([ex, ey, ez+dz])))
603 gcpy      if self.generatepaths == True:
604 gcpy          self.toolpaths = self.toolpaths.union(toolpath)
605 gcpy      else:
606 gcpy          return toolpath
607 gcpy
608 gcpy      def cutZgcfeed(self, ez, feed):
609 gcpy          self.writegc("G01_Z", str(ez), "F", str(feed))
610 gcpy      #         if self.generatepaths == False:
611 gcpy          return self.cutline(self.xpos(), self.ypos(), ez)
```

The matching OpenSCAD command is a descriptor:

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

3.4.2 Arcs for toolpaths and DXFs

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

- cutarcCW — cut a partial arc described in a clock-wise direction
- cutarcCC — counter-clock-wise
- cutarcNWCW — cut the upper-left quadrant of a circle moving clockwise
- cutarcNWCC — upper-left quadrant counter-clockwise
- cutarcNECW
- cutarcNECC
- cutarcSECW
- cutarcSECC
- cutarcNECW
- cutarcNECC
- cutcircleCC — while it wont matter for generating a DXF, when G-code is implemented direction of cut will be a consideration for that
- cutcircleCW
- cutcircleCCdx
- cutcircleCWdx

It will be necessary to have two separate representations of arcs — the G-code and DXF may be easily and directly supported with a single command, but representing the matching tool movement in OpenSCAD will require a series of short line movements which approximate the arc cutting in each direction and at changing Z-heights so as to allow for threading and similar operations. Note that there are the following representations/interfaces for representing an arc:

- G-code — G2 (clockwise) and G3 (counter-clockwise) arcs may be specified, and since the endpoint is the positional requirement, it is most likely best to use the offset to the center (I and J), rather than the radius parameter (K) G2/G3 ...
- DXF — dxfarc(xcenter, ycenter, radius, anglebegin, endangle, tn)
- approximation of arc using lines (OpenSCAD) in both clock-wise and counter-clock-wise directions

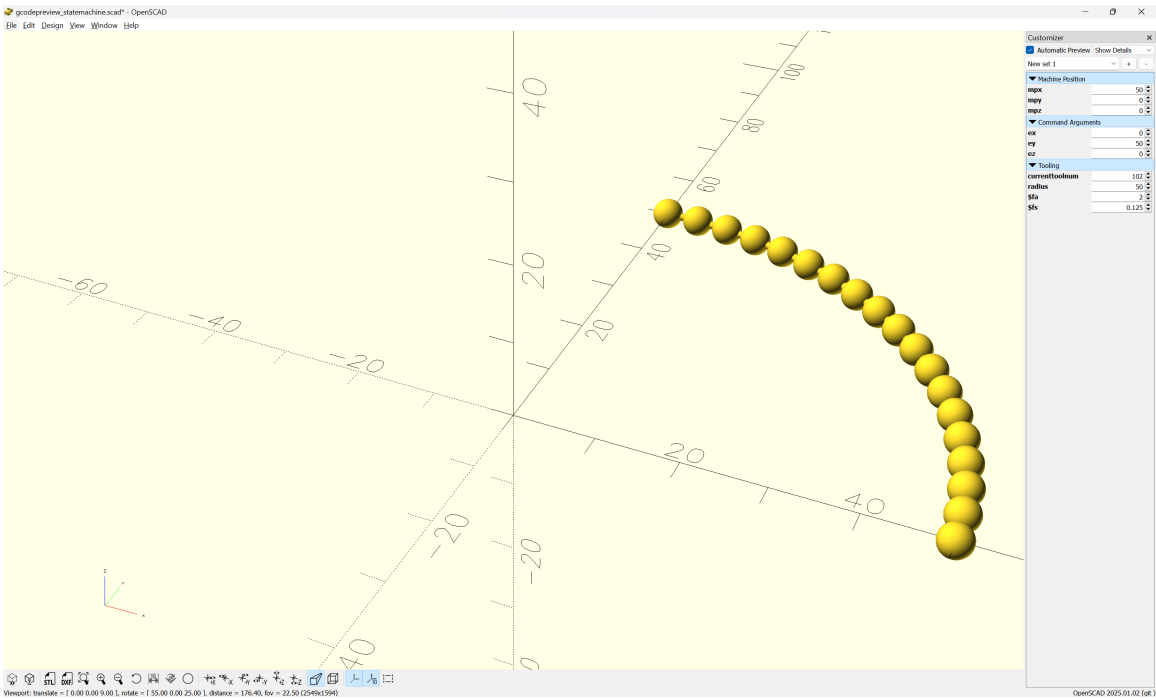
Cutting the quadrant arcs greatly simplifies the calculation and interface for the modules. A full set of 8 will be necessary, then circles will have a pair of modules (one for each cut direction) made for them.

Parameters which will need to be passed in are:

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

Since OpenSCAD does not have an arc movement command it is necessary to iterate through a cutarcCW loop: cutarcCW (clockwise) or cutarcCC (counterclockwise) to handle the drawing and processing cutarcCC of the cutline() toolpaths as short line segments which additionally affords a single point of control for adding additional features such as allowing the depth to vary as one cuts along an arc (the line version is used rather than shape so as to capture the changing machine positions with each step through the loop). Note that the definition matches the DXF definition of defining the center position with a matching radius, but it will be necessary to move the tool to the actual origin, and to calculate the end position when writing out a G2/G3 arc.

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



The code for which makes this obvious:

```

/* [Machine Position] */
mpx = 0;
/* [Machine Position] */
mpy = 0;
/* [Machine Position] */
mpz = 0;

/* [Command Arguments] */
ex = 50;
/* [Command Arguments] */
ey = 25;
/* [Command Arguments] */
ez = -10;

/* [Tooling] */
currenttoolnum = 102;

machine_extents();

radius = 50;
$fa = 2;
$fs = 0.125;

plot_arc(radius, 0, 0, 0, radius, 0, 0, 0, radius, 0, 90, 5);

module plot_arc(bx, by, bz, ex, ey, ez, acx, acy, radius, barc, earc, inc){
for (i = [barc : inc : earc-inc]) {
  union(){
    hull()
    {
      translate([acx + cos(i)*radius,
                  acy + sin(i)*radius,
                  0]){
        sphere(r=0.5);
      }
      translate([acx + cos(i+inc)*radius,
                  acy + sin(i+inc)*radius,
                  0]){
        sphere(r=0.5);
      }
    }
    translate([acx + cos(i)*radius,
                acy + sin(i)*radius,
                0]){
      sphere(r=2);
    }
    translate([acx + cos(i+inc)*radius,
                acy + sin(i+inc)*radius,
                0]){
      sphere(r=2);
    }
  }
}
}

module machine_extents(){
translate([-200, -200, 20]){
  cube([0.001, 0.001, 0.001], center=true);
}
translate([200, 200, 20]){
  cube([0.001, 0.001, 0.001], center=true);
}
}

module plot_cut(bx, by, bz, ex, ey, ez) {
  union(){
    translate([bx, by, bz]){
      sphere(r=5);
    }
    translate([ex, ey, ez]){
      sphere(r=5);
    }
  }
  hull(){
    translate([bx, by, bz]){
      sphere(r=1);
    }
    translate([ex, ey, ez]){
      sphere(r=1);
    }
  }
}

```

```

    }
  }
}
}

```

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

```

613 gcpy      def cutarcCC(self, barc, earc, xcenter, ycenter, radius,
tpzreldim, stepsizearc=1):
614 gcpy #      tpzinc = ez - self.zpos() / (earc - barc)
615 gcpy      tpzinc = tpzreldim / (earc - barc)
616 gcpy      cts = self.currenttoolshape
617 gcpy      toolpath = cts
618 gcpy      toolpath = toolpath.translate([self.xpos(), self.ypos(),
self.zpos()])
619 gcpy      i = barc
620 gcpy      while i < earc:
621 gcpy          toolpath = toolpath.union(self.cutline(xcenter + radius
* math.cos(math.radians(i)), ycenter + radius *
math.sin(math.radians(i)), self.zpos()+tpzinc))
622 gcpy          i += stepsizearc
623 gcpy      if self.generatepaths == False:
624 gcpy          return toolpath
625 gcpy      else:
626 gcpy          return cube([0.01, 0.01, 0.01])
627 gcpy
628 gcpy      def cutarcCW(self, barc, earc, xcenter, ycenter, radius,
tpzreldim, stepsizearc=1):
629 gcpy #          print(str(self.zpos()))
630 gcpy #          print(str(ez))
631 gcpy #          print(str(barc - earc))
632 gcpy #          tpzinc = ez - self.zpos() / (barc - earc)
633 gcpy #          print(str(tpzinc))
634 gcpy #          global toolpath
635 gcpy #          print("Entering n toolpath")
636 gcpy      tpzinc = tpzreldim / (barc - earc)
637 gcpy      cts = self.currenttoolshape
638 gcpy      toolpath = cts
639 gcpy      toolpath = toolpath.translate([self.xpos(), self.ypos(),
self.zpos()])
640 gcpy      i = barc
641 gcpy      while i > earc:
642 gcpy          toolpath = toolpath.union(self.cutline(xcenter + radius
* math.cos(math.radians(i)), ycenter + radius *
math.sin(math.radians(i)), self.zpos()+tpzinc))
643 gcpy #          self.setxpos(xcenter + radius * math.cos(math.radians(
i)))
644 gcpy #          self.setypos(ycenter + radius * math.sin(math.radians(
i)))
645 gcpy #          print(str(self.xpos()), str(self.ypos()), str(self.zpos
()))
646 gcpy #          self.setzpos(self.zpos()+tpzinc)
647 gcpy          i += abs(stepsizearc) * -1
648 gcpy #          self.dxfarc(self.currenttoolnumber(), xcenter, ycenter,
radius, barc, earc)
649 gcpy #          if self.generatepaths == True:
650 gcpy #              print("Unioning n toolpath")
651 gcpy #              self.toolpaths = self.toolpaths.union(toolpath)
652 gcpy #          else:
653 gcpy      if self.generatepaths == False:
654 gcpy          return toolpath
655 gcpy      else:
656 gcpy          return cube([0.01, 0.01, 0.01])

```

Matching OpenSCAD modules are easily made:

```

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

```

3.4.3 Cutting shapes and expansion

Certain basic shapes (arcs, circles, rectangles), will be incorporated in the main code. Other shapes will be added as they are developed, and of course the user is free to develop their own systems.

It is most expedient to test out new features in a new/separate file insofar as the file structures will allow (tool definitions for example will need to be consolidated in 3.3.2) which will need to be included in the projects which will make use of said features until such time as they are added into the main `gcodepreview.scad` file.

A basic requirement for two-dimensional regions will be to define them so as to cut them out. Two different geometric treatments will be necessary: modeling the geometry which defines the region to be cut out (output as a DXF); and modeling the movement of the tool, the toolpath which will be used in creating the 3D model and outputting the G-code.

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

- lines — `dxfline`
- arcs — `dxfarc`

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

3.4.3.2 List of shapes In the TUG presentation/paper: <http://tug.org/TUGboat/tb40-2/tb125adams-3d.pdf> a list of 2D shapes was put forward — which of these will need to be created, or if some more general solution will be put forward is uncertain. For the time being, shapes will be implemented on an as-needed basis, as modified by the interaction with the requirements of toolpaths.

- 0
 - circle — `dxfcircle`
 - ellipse (oval) (requires some sort of non-arc curve)
 - * egg-shaped
 - annulus (one circle within another, forming a ring) — handled by nested circles
 - superellipse (see astroid below)
- 1
 - cone with rounded end (arc)—see also “sector” under 3 below
- 2
 - semicircle/circular/half-circle segment (arc and a straight line); see also sector below
 - arch—curve possibly smoothly joining a pair of straight lines with a flat bottom
 - lens/vesica piscis (two convex curves)
 - lune/crescent (one convex, one concave curve)
 - heart (two curves)
 - tomoe (comma shape)—non-arc curves
- 3
 - triangle
 - * equilateral
 - * isosceles
 - * right triangle
 - * scalene
 - (circular) sector (two straight edges, one convex arc)
 - * quadrant (90°)
 - * sextants (60°)
 - * octants (45°)
 - deltoid curve (three concave arcs)
 - Reuleaux triangle (three convex arcs)
 - arbelos (one convex, two concave arcs)
 - two straight edges, one concave arc—an example is the hyperbolic sector¹
 - two convex, one concave arc
- 4
 - rectangle (including square) — `dxfrectangle`, `dxfrectangleround`

¹en.wikipedia.org/wiki/Hyperbolic_sector and www.reddit.com/r/Geometry/comments/bkbzgh/is_there_a_name_for_a_3_pointed_figure_with_two

- parallelogram
- rhombus
- trapezoid/trapezium
- kite
- ring/annulus segment (straight line, concave arc, straight line, convex arc)
- astroid (four concave arcs)
- salinon (four semicircles)
- three straight lines and one concave arc

Note that most shapes will also exist in a rounded form where sharp angles/points are replaced by arcs/portions of circles, with the most typical being `dxfrectangleround`.
Is the list of shapes for which there are not widely known names interesting for its lack of notoriety?

- two straight edges, one concave arc—oddly, an asymmetric form (hyperbolic sector) has a name, but not the symmetrical—while the colloquial/prosaic arrowhead was considered, it was rejected as being better applied to the shape below. (Its also the shape used for the spaceship in the game Asteroids (or Hyperspace), but that is potentially confusing with astroid.) At the conference, Dr. Knuth suggested dart as a suitable term.
- two convex, one concave arc—with the above named, the term arrowhead is freed up to use as the name for this shape.
- three straight lines and one concave arc.

The first in particular is sorely needed for this project (its the result of inscribing a circle in a square or other regular geometric shape). Do these shapes have names in any other languages which might be used instead?

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

- Contour — No Offset — the default, this is already supported in the existing code
- Contour — Outside Offset
- Contour — Inside Offset
- Pocket — such toolpaths/geometry should include the rounding of the tool at the corners, c.f., `dxfrectangleround`
- Drill — note that this is implemented as the plunging of a tool centered on a circle and normally that circle is the same diameter as the tool which is used.
- Keyhole — also beginning from a circle, the command for this also models the areas which should be cleared for the sake of reducing wear on the tool and ensuring chip clearance

Some further considerations:

- relationship of geometry to toolpath — arguably there should be an option for each toolpath (we will use Carbide Create as a reference implementation) which is to be supported. Note that there are several possibilities: modeling the tool movement, describing the outline which the tool will cut, modeling a reference shape for the toolpath
- tool geometry — it should be possible to include support for specialty tooling such as dovetail cutters and to get an accurate 3D model, esp. for tooling which undercuts since they cannot be modeled in Carbide Create.
- Starting and Max Depth — are there CAD programs which will make use of Z-axis information in a DXF? — would it be possible/necessary to further differentiate the DXF geometry? (currently written out separately for each toolpath in addition to one combined file)

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

```
658 gcpy      def dxfcircle(self, tool_num, xcenter, ycenter, radius):
659 gcpy          self.dxfarc(tool_num, xcenter, ycenter, radius, 0, 90)
660 gcpy          self.dxfarc(tool_num, xcenter, ycenter, radius, 90, 180)
661 gcpy          self.dxfarc(tool_num, xcenter, ycenter, radius, 180, 270)
662 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:

```
664 gcpy      def dxfrectangle(self, tool_num, xorigin, yorigin, xwidth,
665 gcpy          yheight, corners = "Square", radius = 6):
666 gcpy          if corners == "Square":
667 gcpy              self.dxfline(tool_num, xorigin, yorigin, xorigin +
668 gcpy                  xwidth, yorigin)
669 gcpy              self.dxfline(tool_num, xorigin + xwidth, yorigin,
670 gcpy                  xorigin + xwidth, yorigin + yheight)
671 gcpy              self.dxfline(tool_num, xorigin + xwidth, yorigin +
672 gcpy                  yheight, xorigin, yorigin + yheight)
673 gcpy              self.dxfline(tool_num, xorigin, yorigin + yheight,
674 gcpy                  xorigin, yorigin)
675 gcpy          elif corners == "Fillet":
676 gcpy              self.dxfrectangleround(tool_num, xorigin, yorigin,
677 gcpy                  xwidth, yheight, radius)
678 gcpy          elif corners == "Chamfer":
679 gcpy              self.dxfrectanglechamfer(tool_num, xorigin, yorigin,
680 gcpy                  xwidth, yheight, radius)
681 gcpy          elif corners == "Flipped_Fillet":
682 gcpy              self.dxfrectangleflippedfillet(tool_num, xorigin,
683 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.

```
677 gcpy      def dxfrectangleround(self, tool_num, xorigin, yorigin, xwidth,
678 gcpy          yheight, radius):
679 gcpy          self.dxfarc(tool_num, xorigin + xwidth - radius, yorigin +
680 gcpy              yheight - radius, radius, 0, 90)
681 gcpy          self.dxfarc(tool_num, xorigin + radius, yorigin + yheight -
682 gcpy              radius, radius, 90, 180)
683 gcpy          self.dxfarc(tool_num, xorigin + radius, yorigin + radius,
684 gcpy              radius, 180, 270)
685 gcpy          self.dxfarc(tool_num, xorigin + xwidth - radius, yorigin +
686 gcpy              radius, radius, 270, 360)
687 gcpy          self.dxfline(tool_num, xorigin + radius, yorigin, xorigin +
688 gcpy              xwidth - radius, yorigin)
689 gcpy          self.dxfline(tool_num, xorigin + xwidth, yorigin + radius,
690 gcpy              xorigin + xwidth, yorigin + yheight - radius)
691 gcpy          self.dxfline(tool_num, xorigin + xwidth - radius, yorigin +
692 gcpy              yheight, xorigin + xwidth, yorigin + yheight - radius)
693 gcpy          self.dxfline(tool_num, xorigin + xwidth - radius, yorigin,
694 gcpy              xorigin + xwidth, yorigin + radius)
```

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

```
688 gcpy      def dxfrectanglechamfer(self, tool_num, xorigin, yorigin,
689 gcpy          xwidth, yheight, radius):
690 gcpy          self.dxfline(tool_num, xorigin + radius, yorigin, xorigin,
691 gcpy              yorigin + radius)
692 gcpy          self.dxfline(tool_num, xorigin, yorigin + yheight - radius,
693 gcpy              xorigin + radius, yorigin + yheight)
694 gcpy          self.dxfline(tool_num, xorigin + xwidth - radius, yorigin +
695 gcpy              yheight, xorigin + xwidth, yorigin + yheight - radius)
696 gcpy          self.dxfline(tool_num, xorigin + xwidth - radius, yorigin,
697 gcpy              xorigin + xwidth, yorigin + radius)
```

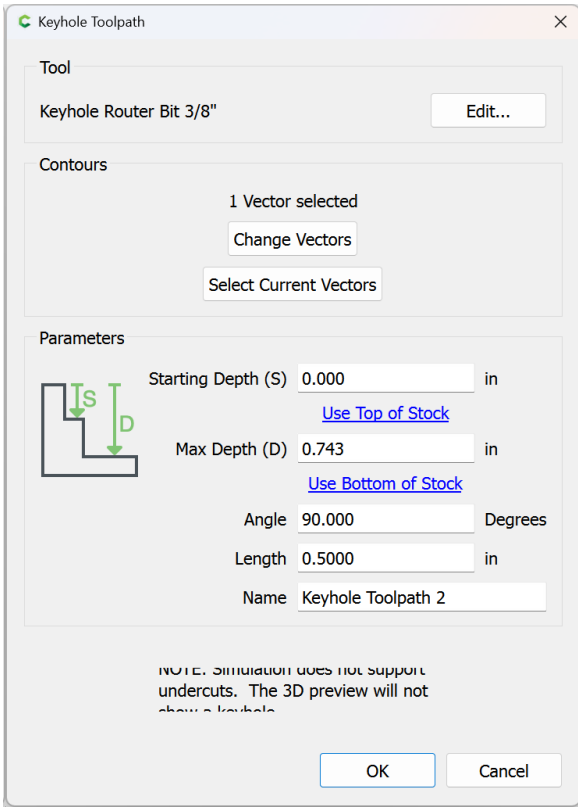
Flipped Fillet:

```
699 gcpy      def dxfrectangleflippedfillet(self, tool_num, xorigin, yorigin,
700 gcpy          xwidth, yheight, radius):
701 gcpy          self.dxfarc(tool_num, xorigin, yorigin, radius, 0, 90)
702 gcpy          self.dxfarc(tool_num, xorigin + xwidth, yorigin, radius,
703 gcpy              90, 180)
```

```
702 gcpy          self.dxfarc(tool_num, xorigin + xwidth, yorigin + yheight,
                        radius, 180, 270)
703 gcpy          self.dxfarc(tool_num, xorigin, yorigin + yheight, radius,
                        270, 360)
704 gcpy
705 gcpy          self.dxfline(tool_num, xorigin + radius, yorigin, xorigin +
                        xwidth - radius, yorigin)
706 gcpy          self.dxfline(tool_num, xorigin + xwidth, yorigin + radius,
                        xorigin + xwidth, yorigin + yheight - radius)
707 gcpy          self.dxfline(tool_num, xorigin + xwidth - radius, yorigin +
                        yheight, xorigin + radius, yorigin + yheight)
708 gcpy          self.dxfline(tool_num, xorigin, yorigin + yheight - radius,
                        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 will result in the creation of two different physical physical regions: the visible surface matching the union of the tool perimeter at the entry point and the linear movement of the shaft and the larger region of the tool perimeter at the depth which the tool is plunged to and moved along.

Tooling for such toolpaths is defined at paragraph 3.3.1.2
The interface which is being modeled is that of Carbide Create:



Hence the parameters:

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

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

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

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

```
710 gcpy      def cutkeyholegdcxf(self, kh_tool_num, kh_start_depth,
711 gcpy      kh_max_depth, kht_direction, kh_distance):
712 gcpy      toolpath = self.cutKHgdcxf(kh_tool_num, kh_start_depth,
713 gcpy      kh_max_depth, 90, kh_distance)
714 gcpy      elif (kht_direction == "S"):
715 gcpy      toolpath = self.cutKHgdcxf(kh_tool_num, kh_start_depth,
716 gcpy      kh_max_depth, 270, kh_distance)
717 gcpy      elif (kht_direction == "E"):
718 gcpy      toolpath = self.cutKHgdcxf(kh_tool_num, kh_start_depth,
719 gcpy      kh_max_depth, 0, kh_distance)
720 gcpy      elif (kht_direction == "W"):
721 gcpy      toolpath = self.cutKHgdcxf(kh_tool_num, kh_start_depth,
722 gcpy      kh_max_depth, 180, kh_distance)
723 gcpy      if self.generatepaths == True:
724 gcpy      self.toolpaths = union([self.toolpaths, toolpath])
725 gcpy      return toolpath
726 gcpy      else:
727 gcpy      return cube([0.01, 0.01, 0.01])

94 gcpscad module cutkeyholegdcxf(kh_tool_num, kh_start_depth, kh_max_depth,
95 gcpscad kht_direction, kh_distance){
96 gcpscad gcp.cutkeyholegdcxf(kh_tool_num, kh_start_depth, kh_max_depth,
97 gcpscad kht_direction, kh_distance);
98 gcpscad }
```

cutKHgdcxf The original version of the command, cutKHgdcxf 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:

```
725 gcpy      def cutKHgdcxf(self, kh_tool_num, kh_start_depth, kh_max_depth,
726 gcpy      kh_angle, kh_distance):
727 gcpy      oXpos = self.xpos()
728 gcpy      oYpos = self.ypos()
729 gcpy      self.dxfKH(kh_tool_num, self.xpos(), self.ypos(),
730 gcpy      kh_start_depth, kh_max_depth, kh_angle, kh_distance)
731 gcpy      toolpath = self.cutline(self.xpos(), self.ypos(), -
732 gcpy      kh_max_depth)
733 gcpy      self.setxpos(oXpos)
734 gcpy      self.setypos(oYpos)
735 gcpy      if self.generatepaths == False:
736 gcpy      return toolpath
737 gcpy      else:
738 gcpy      return cube([0.001, 0.001, 0.001])

739 gcpy      def dxfKH(self, kh_tool_num, oXpos, oYpos, kh_start_depth,
740 gcpy      kh_max_depth, kh_angle, kh_distance):
741 gcpy      oXpos = self.xpos()
742 gcpy      oYpos = self.ypos()
743 gcpy      #Circle at entry hole
744 gcpy      self.dxfarc(kh_tool_num, oXpos, oYpos, self.tool_radius(
745 gcpy      kh_tool_num, 7), 0, 90)
746 gcpy      self.dxfarc(kh_tool_num, oXpos, oYpos, self.tool_radius(
747 gcpy      kh_tool_num, 7), 90, 180)
748 gcpy      self.dxfarc(kh_tool_num, oXpos, oYpos, self.tool_radius(
749 gcpy      kh_tool_num, 7), 180, 270)
750 gcpy      self.dxfarc(kh_tool_num, oXpos, oYpos, self.tool_radius(
751 gcpy      kh_tool_num, 7), 270, 360)
```

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

```
746 gcpy #pre-calculate needed values
747 gcpy r = self.tool_radius(kh_tool_num, 7)
748 gcpy # print(r)
749 gcpy rt = self.tool_radius(kh_tool_num, 1)
750 gcpy # print(rt)
751 gcpy ro = math.sqrt((self.tool_radius(kh_tool_num, 1))**2-(self.
752 gcpy tool_radius(kh_tool_num, 7))**2)
```

```

752 gcpy #          print(ro)
753 gcpy          angle = math.degrees(math.acos(ro/rt))
754 gcpy #Outlines of entry hole and slot
755 gcpy          if (kh_angle == 0):
756 gcpy #Lower left of entry hole
757 gcpy          self.dxfarc(kh_tool_num, self.xpos(), self.ypos(), self
                          .tool_radius(kh_tool_num, 1), 180, 270)
758 gcpy #Upper left of entry hole
759 gcpy          self.dxfarc(kh_tool_num, self.xpos(), self.ypos(), self
                          .tool_radius(kh_tool_num, 1), 90, 180)
760 gcpy #Upper right of entry hole
761 gcpy #          self.dxfarc(kh_tool_num, self.xpos(), self.ypos(), rt,
                          41.810, 90)
762 gcpy          self.dxfarc(kh_tool_num, self.xpos(), self.ypos(), rt,
                          angle, 90)
763 gcpy #Lower right of entry hole
764 gcpy          self.dxfarc(kh_tool_num, self.xpos(), self.ypos(), rt,
                          270, 360-angle)
765 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))))
766 gcpy #Actual line of cut
767 gcpy #          self.dxfline(kh_tool_num, self.xpos(), self.ypos(),
                          self.xpos()+kh_distance, self.ypos())
768 gcpy #upper right of end of slot (kh_max_depth+4.36))/2
769 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)
770 gcpy #lower right of end of slot
771 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)
772 gcpy #upper right slot
773 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))
774 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)
775 gcpy #end position at top of slot
776 gcpy #lower right slot
777 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))
778 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)
779 gcpy #end position at top of slot
780 gcpy #          hull(){
781 gcpy #          translate([xpos(), ypos(), zpos()]){
782 gcpy #          keyhole_shaft(6.35, 9.525);
783 gcpy #          }
784 gcpy #          translate([xpos(), ypos(), zpos()-kh_max_depth]){
785 gcpy #          keyhole_shaft(6.35, 9.525);
786 gcpy #          }
787 gcpy #          }
788 gcpy #          hull(){
789 gcpy #          translate([xpos(), ypos(), zpos()-kh_max_depth]){
790 gcpy #          keyhole_shaft(6.35, 9.525);
791 gcpy #          }
792 gcpy #          translate([xpos()+kh_distance, ypos(), zpos()-kh_max_depth])
793 gcpy #          {
794 gcpy #          keyhole_shaft(6.35, 9.525);
795 gcpy #          }
796 gcpy #          cutwithfeed(getxpos(), getypos(), -kh_max_depth, feed);
797 gcpy #          cutwithfeed(getxpos()+kh_distance, getypos(), -kh_max_depth,
                          feed);

```

```

798 gcpy #      setxpos(getxpos()-kh_distance);
799 gcpy #  } else if (kh_angle > 0 && kh_angle < 90) {
800 gcpy #//echo(kh_angle);
801 gcpy #  dxfarf(getxpos(), getypos(), tool_diameter(KH_tool_num, (
      kh_max_depth))/2, 90+kh_angle, 180+kh_angle, KH_tool_num);
802 gcpy #  dxfarf(getxpos(), getypos(), tool_diameter(KH_tool_num, (
      kh_max_depth))/2, 180+kh_angle, 270+kh_angle, KH_tool_num);
803 gcpy #dxfarf(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);
804 gcpy #dxfarf(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);
805 gcpy #dxfarf(getxpos()+(kh_distance*cos(kh_angle)),
806 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);
807 gcpy #dxfarf(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);
808 gcpy #dxflin( 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))),
809 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))),
810 gcpy #  getxpos()+(kh_distance*cos(kh_angle))-((tool_diameter(KH_tool_num
      , (kh_max_depth+4.36))/2)*sin(kh_angle)),
811 gcpy #  getypos()+(kh_distance*sin(kh_angle))+((tool_diameter(KH_tool_num
      , (kh_max_depth+4.36))/2)*cos(kh_angle)), KH_tool_num);
812 gcpy #//echo("a", tool_diameter(KH_tool_num, (kh_max_depth+4.36))/2);
813 gcpy #//echo("c", tool_diameter(KH_tool_num, (kh_max_depth))/2);
814 gcpy #echo("Aangle", asin((tool_diameter(KH_tool_num, (kh_max_depth
      +4.36))/2)/(tool_diameter(KH_tool_num, (kh_max_depth))/2)));
815 gcpy #//echo(kh_angle);
816 gcpy #  cutwithfeed(getxpos()+(kh_distance*cos(kh_angle)), getypos()+(
      kh_distance*sin(kh_angle)), -kh_max_depth, feed);
817 gcpy #      toolpath = toolpath.union(self.cutline(self.xpos()+
      kh_distance, self.ypos(), -kh_max_depth))
818 gcpy      elif (kh_angle == 90):
819 gcpy #Lower left of entry hole
820 gcpy      self.dxfarc(kh_tool_num, oXpos, oYpos, self.tool_radius
      (kh_tool_num, 1), 180, 270)
821 gcpy #Lower right of entry hole
822 gcpy      self.dxfarc(kh_tool_num, oXpos, oYpos, self.tool_radius
      (kh_tool_num, 1), 270, 360)
823 gcpy #left slot
824 gcpy      self.dxfline(kh_tool_num, oXpos-r, oYpos+ro, oXpos-r,
      oYpos+kh_distance)
825 gcpy #right slot
826 gcpy      self.dxfline(kh_tool_num, oXpos+r, oYpos+ro, oXpos+r,
      oYpos+kh_distance)
827 gcpy #upper left of end of slot
828 gcpy      self.dxfarc(kh_tool_num, oXpos, oYpos+kh_distance, r,
      90, 180)
829 gcpy #upper right of end of slot
830 gcpy      self.dxfarc(kh_tool_num, oXpos, oYpos+kh_distance, r,
      0, 90)
831 gcpy #Upper right of entry hole
832 gcpy      self.dxfarc(kh_tool_num, oXpos, oYpos, rt, 0, 90-angle)
833 gcpy #Upper left of entry hole
834 gcpy      self.dxfarc(kh_tool_num, oXpos, oYpos, rt, 90+angle,
      180)
835 gcpy #      toolpath = toolpath.union(self.cutline(oXpos, oYpos+
      kh_distance, -kh_max_depth))
836 gcpy      elif (kh_angle == 180):
837 gcpy #Lower right of entry hole
838 gcpy      self.dxfarc(kh_tool_num, oXpos, oYpos, self.tool_radius
      (kh_tool_num, 1), 270, 360)
839 gcpy #Upper right of entry hole
840 gcpy      self.dxfarc(kh_tool_num, oXpos, oYpos, self.tool_radius
      (kh_tool_num, 1), 0, 90)
841 gcpy #Upper left of entry hole
842 gcpy      self.dxfarc(kh_tool_num, oXpos, oYpos, rt, 90, 180-
      angle)
843 gcpy #Lower left of entry hole
844 gcpy      self.dxfarc(kh_tool_num, oXpos, oYpos, rt, 180+angle,

```

```

270)
845 gcpy #upper slot
846 gcpy         self.dxfline(kh_tool_num, oXpos-ro, oYpos-r, oXpos-
                        kh_distance, oYpos-r)
847 gcpy #lower slot
848 gcpy         self.dxfline(kh_tool_num, oXpos-ro, oYpos+r, oXpos-
                        kh_distance, oYpos+r)
849 gcpy #upper left of end of slot
850 gcpy         self.dxfarc(kh_tool_num, oXpos-kh_distance, oYpos, r,
                        90, 180)
851 gcpy #lower left of end of slot
852 gcpy         self.dxfarc(kh_tool_num, oXpos-kh_distance, oYpos, r,
                        180, 270)
853 gcpy #         toolpath = toolpath.union(self.cutline(oXpos-
                        kh_distance, oYpos, -kh_max_depth))
854 gcpy         elif (kh_angle == 270):
855 gcpy #Upper left of entry hole
856 gcpy         self.dxfarc(kh_tool_num, oXpos, oYpos, self.tool_radius
                        (kh_tool_num, 1), 90, 180)
857 gcpy #Upper right of entry hole
858 gcpy         self.dxfarc(kh_tool_num, oXpos, oYpos, self.tool_radius
                        (kh_tool_num, 1), 0, 90)
859 gcpy #left slot
860 gcpy         self.dxfline(kh_tool_num, oXpos-r, oYpos-ro, oXpos-r,
                        oYpos-kh_distance)
861 gcpy #right slot
862 gcpy         self.dxfline(kh_tool_num, oXpos+r, oYpos-ro, oXpos+r,
                        oYpos-kh_distance)
863 gcpy #lower left of end of slot
864 gcpy         self.dxfarc(kh_tool_num, oXpos, oYpos-kh_distance, r,
                        180, 270)
865 gcpy #lower right of end of slot
866 gcpy         self.dxfarc(kh_tool_num, oXpos, oYpos-kh_distance, r,
                        270, 360)
867 gcpy #lower right of entry hole
868 gcpy         self.dxfarc(kh_tool_num, oXpos, oYpos, rt, 180, 270-
                        angle)
869 gcpy #lower left of entry hole
870 gcpy         self.dxfarc(kh_tool_num, oXpos, oYpos, rt, 270+angle,
                        360)
871 gcpy #         toolpath = toolpath.union(self.cutline(oXpos, oYpos-
                        kh_distance, -kh_max_depth))
872 gcpy #         print(self.zpos())
873 gcpy #         self.setxpos(oXpos)
874 gcpy #         self.setypos(oYpos)
875 gcpy #         if self.generatepaths == False:
876 gcpy #             return toolpath
877 gcpy
878 gcpy # } else if (kh_angle == 90) {
879 gcpy #     //Lower left of entry hole
880 gcpy #     dxfarc(getxpos(), getypos(), 9.525/2, 180, 270, KH_tool_num);
881 gcpy #     //Lower right of entry hole
882 gcpy #     dxfarc(getxpos(), getypos(), 9.525/2, 270, 360, KH_tool_num);
883 gcpy #     //Upper right of entry hole
884 gcpy #     dxfarc(getxpos(), getypos(), 9.525/2, 0, acos(tool_diameter(
                        KH_tool_num, 5)/tool_diameter(KH_tool_num, 1)), KH_tool_num);
885 gcpy #     //Upper left of entry hole
886 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
                        );
887 gcpy #     //Actual line of cut
888 gcpy #     dxfline(getxpos(), getypos(), getxpos(), getypos()+kh_distance
                        );
889 gcpy #     //upper right of slot
890 gcpy #     dxfarc(getxpos(), getypos()+kh_distance, tool_diameter(
                        KH_tool_num, (kh_max_depth+4.36))/2, 0, 90, KH_tool_num);
891 gcpy #     //upper left of slot
892 gcpy #     dxfarc(getxpos(), getypos()+kh_distance, tool_diameter(
                        KH_tool_num, (kh_max_depth+6.35))/2, 90, 180, KH_tool_num);
893 gcpy #     //right of slot
894 gcpy #     dxfline(
895 gcpy #         getxpos()+tool_diameter(KH_tool_num, (kh_max_depth))/2,
896 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,
897 gcpy #         getxpos()+tool_diameter(KH_tool_num, (kh_max_depth))/2,
898 gcpy #         //end position at top of slot
899 gcpy #         getypos()+kh_distance,

```



```

900 gcpy #         KH_tool_num);
901 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);
902 gcpy #     hull(){
903 gcpy #         translate([xpos(), ypos(), zpos()]){
904 gcpy #             keyhole_shaft(6.35, 9.525);
905 gcpy #         }
906 gcpy #         translate([xpos(), ypos(), zpos()-kh_max_depth]){
907 gcpy #             keyhole_shaft(6.35, 9.525);
908 gcpy #         }
909 gcpy #     }
910 gcpy #     hull(){
911 gcpy #         translate([xpos(), ypos(), zpos()-kh_max_depth]){
912 gcpy #             keyhole_shaft(6.35, 9.525);
913 gcpy #         }
914 gcpy #         translate([xpos(), ypos()+kh_distance, zpos()-kh_max_depth])
{
915 gcpy #             keyhole_shaft(6.35, 9.525);
916 gcpy #         }
917 gcpy #     }
918 gcpy #     cutwithfeed(getxpos(), getypos(), -kh_max_depth, feed);
919 gcpy #     cutwithfeed(getxpos(), getypos()+kh_distance, -kh_max_depth,
feed);
920 gcpy #     setypos(getypos()-kh_distance);
921 gcpy # } else if (kh_angle == 180) {
922 gcpy #     //Lower right of entry hole
923 gcpy #     dxfarc(getxpos(), getypos(), 9.525/2, 270, 360, KH_tool_num);
924 gcpy #     //Upper right of entry hole
925 gcpy #     dxfarc(getxpos(), getypos(), 9.525/2, 0, 90, KH_tool_num);
926 gcpy #     //Upper left of entry hole
927 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);
928 gcpy #     //Lower left of entry hole
929 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
);
930 gcpy #     //upper left of slot
931 gcpy #     dxfarc(getxpos()-kh_distance, getypos(), tool_diameter(
KH_tool_num, (kh_max_depth+6.35))/2, 90, 180, KH_tool_num);
932 gcpy #     //lower left of slot
933 gcpy #     dxfarc(getxpos()-kh_distance, getypos(), tool_diameter(
KH_tool_num, (kh_max_depth+6.35))/2, 180, 270, KH_tool_num);
934 gcpy #     //Actual line of cut
935 gcpy #     dxfline(getxpos(), getypos(), getxpos()-kh_distance, getypos()
);
936 gcpy #     //upper left slot
937 gcpy #     dxfline(
938 gcpy #         getxpos()-(sqrt((tool_diameter(KH_tool_num, 1)^2)-(
tool_diameter(KH_tool_num, 5)^2))/2),
939 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,
940 gcpy #         getxpos()-kh_distance,
941 gcpy #         //end position at top of slot
942 gcpy #         getypos()+tool_diameter(KH_tool_num, (kh_max_depth))/2,
943 gcpy #         KH_tool_num);
944 gcpy #     //lower right slot
945 gcpy #     dxfline(
946 gcpy #         getxpos()-(sqrt((tool_diameter(KH_tool_num, 1)^2)-(
tool_diameter(KH_tool_num, 5)^2))/2),
947 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,
948 gcpy #         getxpos()-kh_distance,
949 gcpy #         //end position at top of slot
950 gcpy #         getypos()-tool_diameter(KH_tool_num, (kh_max_depth))/2,
951 gcpy #         KH_tool_num);
952 gcpy #     hull(){
953 gcpy #         translate([xpos(), ypos(), zpos()]){
954 gcpy #             keyhole_shaft(6.35, 9.525);
955 gcpy #         }
956 gcpy #         translate([xpos(), ypos(), zpos()-kh_max_depth]){
957 gcpy #             keyhole_shaft(6.35, 9.525);
958 gcpy #         }

```

```

959 gcpy # }
960 gcpy # hull(){
961 gcpy #     translate([xpos(), ypos(), zpos()-kh_max_depth]){
962 gcpy #         keyhole_shaft(6.35, 9.525);
963 gcpy #     }
964 gcpy #     translate([xpos()-kh_distance, ypos(), zpos()-kh_max_depth])
965 gcpy # {
966 gcpy #     keyhole_shaft(6.35, 9.525);
967 gcpy # }
968 gcpy # cutwithfeed(getxpos(), getypos(), -kh_max_depth, feed);
969 gcpy # cutwithfeed(getxpos()-kh_distance, getypos(), -kh_max_depth,
970 gcpy # feed);
971 gcpy # setxpos(getxpos()+kh_distance);
972 gcpy # } else if (kh_angle == 270) {
973 gcpy # //Upper right of entry hole
974 gcpy # dxfarc(getxpos(), getypos(), 9.525/2, 0, 90, KH_tool_num);
975 gcpy # //Upper left of entry hole
976 gcpy # dxfarc(getxpos(), getypos(), 9.525/2, 90, 180, KH_tool_num);
977 gcpy # //lower right of slot
978 gcpy # dxfarc(getxpos(), getypos()-kh_distance, tool_diameter(
979 gcpy # KH_tool_num, (kh_max_depth+4.36))/2, 270, 360, KH_tool_num);
980 gcpy # //lower left of slot
981 gcpy # dxfarc(getxpos(), getypos()-kh_distance, tool_diameter(
982 gcpy # KH_tool_num, (kh_max_depth+4.36))/2, 180, 270, KH_tool_num);
983 gcpy # //Actual line of cut
984 gcpy # dxfline(getxpos(), getypos(), getxpos(), getypos()-kh_distance
985 gcpy # );
986 gcpy # //right of slot
987 gcpy # dxfline(
988 gcpy #     getxpos()+tool_diameter(KH_tool_num, (kh_max_depth))/2,
989 gcpy #     getypos()-(sqrt((tool_diameter(KH_tool_num, 1)^2)-(
990 gcpy # tool_diameter(KH_tool_num, 5)^2))/2), //( (kh_max_depth-6.34)
991 gcpy # /2)^2-(tool_diameter(KH_tool_num, (kh_max_depth-6.34))/2)^2,
992 gcpy #     getxpos()+tool_diameter(KH_tool_num, (kh_max_depth))/2,
993 gcpy #     //end position at top of slot
994 gcpy #     getypos()-kh_distance,
995 gcpy #     KH_tool_num);
996 gcpy # //left of slot
997 gcpy # dxfline(
998 gcpy #     getxpos()-tool_diameter(KH_tool_num, (kh_max_depth))/2,
999 gcpy #     getypos()-(sqrt((tool_diameter(KH_tool_num, 1)^2)-(
1000 gcpy # tool_diameter(KH_tool_num, 5)^2))/2), //( (kh_max_depth-6.34)
1001 gcpy # /2)^2-(tool_diameter(KH_tool_num, (kh_max_depth-6.34))/2)^2,
1002 gcpy #     getxpos()-tool_diameter(KH_tool_num, (kh_max_depth))/2,
1003 gcpy #     //end position at top of slot
1004 gcpy #     getypos()-kh_distance,
1005 gcpy #     KH_tool_num);
1006 gcpy # //Lower right of entry hole
1007 gcpy # dxfarc(getxpos(), getypos(), 9.525/2, 360-acos(tool_diameter(
1008 gcpy # KH_tool_num, 5)/tool_diameter(KH_tool_num, 1)), 360, KH_tool_num
1009 gcpy # );
1010 gcpy # //Lower left of entry hole
1011 gcpy # dxfarc(getxpos(), getypos(), 9.525/2, 180, 180+acos(
1012 gcpy # tool_diameter(KH_tool_num, 5)/tool_diameter(KH_tool_num, 1)),
1013 gcpy # KH_tool_num);
1014 gcpy # hull(){
1015 gcpy #     translate([xpos(), ypos(), zpos()]){
1016 gcpy #         keyhole_shaft(6.35, 9.525);
1017 gcpy #     }
1018 gcpy #     translate([xpos(), ypos(), zpos()-kh_max_depth]){
1019 gcpy #         keyhole_shaft(6.35, 9.525);
1020 gcpy #     }
1021 gcpy # }
1022 gcpy # hull(){
1023 gcpy #     translate([xpos(), ypos(), zpos()-kh_max_depth]){
1024 gcpy #         keyhole_shaft(6.35, 9.525);
1025 gcpy #     }
1026 gcpy #     translate([xpos(), ypos()-kh_distance, zpos()-kh_max_depth])
1027 gcpy # {
1028 gcpy #     keyhole_shaft(6.35, 9.525);
1029 gcpy # }
1030 gcpy # cutwithfeed(getxpos(), getypos(), -kh_max_depth, feed);
1031 gcpy # cutwithfeed(getxpos(), getypos()-kh_distance, -kh_max_depth,
1032 gcpy # feed);
1033 gcpy # setypos(getypos()+kh_distance);
1034 gcpy # }

```

```
1022 gcpy #}
```

3.4.4 Difference of Stock, Rapids, and Toolpaths

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

```
1024 gcpy      def stockandtoolpaths(self, option = "stockandtoolpaths"):
1025 gcpy          if option == "stock":
1026 gcpy              if self.generatepaths == False:
1027 gcpy                  output(self.stock)
1028 gcpy          #                  print("Outputting stock")
1029 gcpy              else:
1030 gcpy                  return self.stock
1031 gcpy          elif option == "toolpaths":
1032 gcpy              if self.generatepaths == False:
1033 gcpy                  output(self.toolpaths)
1034 gcpy              else:
1035 gcpy                  return self.toolpaths
1036 gcpy          elif option == "rapids":
1037 gcpy              if self.generatepaths == False:
1038 gcpy                  output(self.rapids)
1039 gcpy              else:
1040 gcpy                  return self.rapids
1041 gcpy          else:
1042 gcpy              part = self.stock.difference(self.toolpaths)
1043 gcpy              if self.generatepaths == False:
1044 gcpy                  output(part)
1045 gcpy              else:
1046 gcpy                  return part
```

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

3.5 Output files

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

3.5.1 G-code Overview

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

Command/Module	G-code
opengcodefile(s)(...); setupstock(...)	(export.nc) (stockMin: -109.5, -75mm, -8.35mm) (stockMax:109.5mm, 75mm, 0.00mm) (STOCK/BLOCK, 219, 150, 8.35, 109.5, 75, 8.35) G90 G21
movetosafez()	(Move to safe Z to avoid workholding) G53G0Z-5.000
toolchange(...);	(TOOL/MILL, 3.17, 0.00, 0.00, 0.00) M6T102 M03S16000
cutoneaxis_setfeed(...);	(PREPOSITION FOR RAPID PLUNGE) GOX0Y0 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) GOX0.000Y-152.400 Z0.250	writecomment(...) rapid(...) rapid(...)
G1Z-1.000F203.2 X109.500Y-77.400F508.0 X57.918Y16.302Z-0.726 Y22.023Z-1.023 X61.190Z-0.681 Y21.643 X57.681 Z12.700	cutwithfeed(...); cutwithfeed(...);
M05 M02	closegcodefile();

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

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 `dxfl` commands will be wrapped up with `if/elif` blocks which will write to additional file(s) based on tool number as set up above.

```

1048 gcpy      def writegc(self, *arguments):
1049 gcpy          if self.generategcode == True:
1050 gcpy              line_to_write = ""
1051 gcpy              for element in arguments:
1052 gcpy                  line_to_write += element
1053 gcpy                  self.gc.write(line_to_write)
1054 gcpy                  self.gc.write("\n")
1055 gcpy
1056 gcpy      def writedxf(self, toolnumber, *arguments):
1057 gcpy          # global dxfclosed
1058 gcpy          line_to_write = ""
1059 gcpy          for element in arguments:
1060 gcpy              line_to_write += element
1061 gcpy          if self.generatedxfl == True:
1062 gcpy              if self.dxfclosed == False:
1063 gcpy                  self.dxf.write(line_to_write)
1064 gcpy                  self.dxf.write("\n")
1065 gcpy          if self.generatedxfls == True:
1066 gcpy              self.writedxfls(toolnumber, line_to_write)
1067 gcpy
1068 gcpy      def writedxfls(self, toolnumber, line_to_write):
1069 gcpy          # print("Processing writing toolnumber", toolnumber)
1070 gcpy          # line_to_write = ""
1071 gcpy          # for element in arguments:
1072 gcpy          #     line_to_write += element
1073 gcpy          if (toolnumber == 0):
1074 gcpy              return
1075 gcpy          elif self.generatedxfls == True:
1076 gcpy              if (self.large_square_tool_num == toolnumber):
1077 gcpy                  self.dxfllsq.write(line_to_write)
1078 gcpy                  self.dxfllsq.write("\n")
1079 gcpy              if (self.small_square_tool_num == toolnumber):
1080 gcpy                  self.dxfllsq.write(line_to_write)
1081 gcpy                  self.dxfllsq.write("\n")
1082 gcpy              if (self.large_ball_tool_num == toolnumber):
1083 gcpy                  self.dxfllbl.write(line_to_write)
1084 gcpy                  self.dxfllbl.write("\n")
1085 gcpy              if (self.small_ball_tool_num == toolnumber):

```

```
1086 gcpy                self.dxfsmbl.write(line_to_write)
1087 gcpy                self.dxfsmbl.write("\n")
1088 gcpy                if (self.large_V_tool_num == toolnumber):
1089 gcpy                    self.dxfV.write(line_to_write)
1090 gcpy                    self.dxfV.write("\n")
1091 gcpy                if (self.small_V_tool_num == toolnumber):
1092 gcpy                    self.dxfsmV.write(line_to_write)
1093 gcpy                    self.dxfsmV.write("\n")
1094 gcpy                if (self.DT_tool_num == toolnumber):
1095 gcpy                    self.dxfDT.write(line_to_write)
1096 gcpy                    self.dxfDT.write("\n")
1097 gcpy                if (self.KH_tool_num == toolnumber):
1098 gcpy                    self.dxfKH.write(line_to_write)
1099 gcpy                    self.dxfKH.write("\n")
1100 gcpy                if (self.Roundover_tool_num == toolnumber):
1101 gcpy                    self.dxfRt.write(line_to_write)
1102 gcpy                    self.dxfRt.write("\n")
1103 gcpy                if (self.MISC_tool_num == toolnumber):
1104 gcpy                    self.dxfMt.write(line_to_write)
1105 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 and setting the associated defaults. There is a separate function for each type of file, and for DXFs, there are multiple file instances, one for each combination of different type and size of tool which it is expected a project will work with. Each such file will be suffixed with the tool number.

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

```
114 gpcscad module opendxfile(basefilename){
115 gpcscad     gcp.opendxfile(basefilename);
116 gpcscad }
117 gpcscad
118 gpcscad 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) {
119 gpcscad     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)
    ;
120 gpcscad }
```

opengcodefile With matching OpenSCAD commands: opengcodefile for OpenSCAD:

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

and Python:

```
1107 gcpy     def opengcodefile(self, basefilename = "export",
1108 gcpy         currenttoolnum = 102,
1109 gcpy         toolradius = 3.175,
1110 gcpy         plunge = 400,
1111 gcpy         feed = 1600,
1112 gcpy         speed = 10000
1113 gcpy         ):
1114 gcpy         self.basefilename = basefilename
1115 gcpy         self.currenttoolnum = currenttoolnum
1116 gcpy         self.toolradius = toolradius
1117 gcpy         self.plunge = plunge
1118 gcpy         self.feed = feed
1119 gcpy         self.speed = speed
1120 gcpy         if self.generategcode == True:
1121 gcpy             self.gcodefilename = basefilename + ".nc"
1122 gcpy             self.gc = open(self.gcodefilename, "w")
1123 gcpy
1124 gcpy     def opendxfile(self, basefilename = "export"):
```

```

1125 gcpy          self.basefilename = basefilename
1126 gcpy #          global generatedxfs
1127 gcpy #          global dxfclosed
1128 gcpy          self.dxfclosed = False
1129 gcpy          if self.generateddxf == True:
1130 gcpy              self.generatedxfs = False
1131 gcpy              self.dxffilename = basefilename + ".dxf"
1132 gcpy              self.dxf = open(self.dxffilename, "w")
1133 gcpy              self.dxfpreamble(-1)
1134 gcpy
1135 gcpy          def opendxfiles(self, basefilename = "export",
1136 gcpy                          large_square_tool_num = 0,
1137 gcpy                          small_square_tool_num = 0,
1138 gcpy                          large_ball_tool_num = 0,
1139 gcpy                          small_ball_tool_num = 0,
1140 gcpy                          large_V_tool_num = 0,
1141 gcpy                          small_V_tool_num = 0,
1142 gcpy                          DT_tool_num = 0,
1143 gcpy                          KH_tool_num = 0,
1144 gcpy                          Roundover_tool_num = 0,
1145 gcpy                          MISC_tool_num = 0):
1146 gcpy #              global generatedxfs
1147 gcpy          self.basefilename = basefilename
1148 gcpy          self.generatedxfs = True
1149 gcpy          self.large_square_tool_num = large_square_tool_num
1150 gcpy          self.small_square_tool_num = small_square_tool_num
1151 gcpy          self.large_ball_tool_num = large_ball_tool_num
1152 gcpy          self.small_ball_tool_num = small_ball_tool_num
1153 gcpy          self.large_V_tool_num = large_V_tool_num
1154 gcpy          self.small_V_tool_num = small_V_tool_num
1155 gcpy          self.DT_tool_num = DT_tool_num
1156 gcpy          self.KH_tool_num = KH_tool_num
1157 gcpy          self.Roundover_tool_num = Roundover_tool_num
1158 gcpy          self.MISC_tool_num = MISC_tool_num
1159 gcpy          if self.generateddxf == True:
1160 gcpy              if (large_square_tool_num > 0):
1161 gcpy                  self.dxf_lsqfilename = basefilename + str(
1162 gcpy                      large_square_tool_num) + ".dxf"
1163 gcpy                  print("Opening ", str(self.dxf_lsqfilename))
1164 gcpy                  self.dxf_lsq = open(self.dxf_lsqfilename, "w")
1165 gcpy              if (small_square_tool_num > 0):
1166 gcpy                  print("Opening small square")
1167 gcpy                  self.dxf_ssqfilename = basefilename + str(
1168 gcpy                      small_square_tool_num) + ".dxf"
1169 gcpy                  self.dxf_ssq = open(self.dxf_ssqfilename, "w")
1170 gcpy              if (large_ball_tool_num > 0):
1171 gcpy                  print("Opening large ball")
1172 gcpy                  self.dxf_lgbfilename = basefilename + str(
1173 gcpy                      large_ball_tool_num) + ".dxf"
1174 gcpy                  self.dxf_lgb = open(self.dxf_lgbfilename, "w")
1175 gcpy              if (small_ball_tool_num > 0):
1176 gcpy                  print("Opening small ball")
1177 gcpy                  self.dxf_smbfilename = basefilename + str(
1178 gcpy                      small_ball_tool_num) + ".dxf"
1179 gcpy                  self.dxf_smb = open(self.dxf_smbfilename, "w")
1180 gcpy              if (large_V_tool_num > 0):
1181 gcpy                  print("Opening large V")
1182 gcpy                  self.dxf_lgVfilename = basefilename + str(
1183 gcpy                      large_V_tool_num) + ".dxf"
1184 gcpy                  self.dxf_lgV = open(self.dxf_lgVfilename, "w")
1185 gcpy              if (small_V_tool_num > 0):
1186 gcpy                  print("Opening small V")
1187 gcpy                  self.dxf_smVfilename = basefilename + str(
1188 gcpy                      small_V_tool_num) + ".dxf"
1189 gcpy                  self.dxf_smV = open(self.dxf_smVfilename, "w")
1190 gcpy              if (DT_tool_num > 0):
1191 gcpy                  print("Opening DT")
1192 gcpy                  self.dxf_DTfilename = basefilename + str(DT_tool_num
1193 gcpy                      ) + ".dxf"
1194 gcpy                  self.dxf_DT = open(self.dxf_DTfilename, "w")
1195 gcpy              if (KH_tool_num > 0):
1196 gcpy                  print("Opening KH")
1197 gcpy                  self.dxf_KHfilename = basefilename + str(KH_tool_num
1198 gcpy                      ) + ".dxf"
1199 gcpy                  self.dxf_KH = open(self.dxf_KHfilename, "w")
1200 gcpy              if (Roundover_tool_num > 0):
1201 gcpy                  print("Opening Rt")
1202 gcpy                  self.dxf_Rtfilename = basefilename + str(

```

```

        Roundover_tool_num) + ".dxf"
1195 gcpy        self.dxfRt = open(self.dxfRtfilename, "w")
1196 gcpy        if (MISC_tool_num > 0):
1197 gcpy #            print("Opening Mt")
1198 gcpy            self.dxfMtfilename = basefilename + str(
                MISC_tool_num) + ".dxf"
1199 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:

```

1200 gcpy        if (large_square_tool_num > 0):
1201 gcpy            self.dxfpreamble(large_square_tool_num)
1202 gcpy        if (small_square_tool_num > 0):
1203 gcpy            self.dxfpreamble(small_square_tool_num)
1204 gcpy        if (large_ball_tool_num > 0):
1205 gcpy            self.dxfpreamble(large_ball_tool_num)
1206 gcpy        if (small_ball_tool_num > 0):
1207 gcpy            self.dxfpreamble(small_ball_tool_num)
1208 gcpy        if (large_V_tool_num > 0):
1209 gcpy            self.dxfpreamble(large_V_tool_num)
1210 gcpy        if (small_V_tool_num > 0):
1211 gcpy            self.dxfpreamble(small_V_tool_num)
1212 gcpy        if (DT_tool_num > 0):
1213 gcpy            self.dxfpreamble(DT_tool_num)
1214 gcpy        if (KH_tool_num > 0):
1215 gcpy            self.dxfpreamble(KH_tool_num)
1216 gcpy        if (Roundover_tool_num > 0):
1217 gcpy            self.dxfpreamble(Roundover_tool_num)
1218 gcpy        if (MISC_tool_num > 0):
1219 gcpy            self.dxfpreamble(MISC_tool_num)

```

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

3.5.3.1 Writing to DXF files When the command to open .dxf files is called it is passed all of the variables for the various tool types/sizes, and based on a value being greater than zero, the matching file is opened, and in addition, the main DXF which is always written to is opened as well. On the gripping hand, each element which may be written to a DXF file will have a user module as well as an internal module which will be called by it so as to write to the file for the current tool. It will be necessary for the dxfwrite command to evaluate the tool number which is passed in, and to use an appropriate command or set of commands to then write out to the appropriate file for a given tool (if positive) or not do anything (if zero), and to write to the master file if a negative value is passed in (this allows the various DXF template commands to be written only once and then called at need).

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

- writedxflgbl
- Ball nose, large (lgbl) writedxflgbl
- writedxfsmb1
- Ball nose, small (smb1) writedxfsmb1
- writedxflgsq
- Square, large (lgsq) writedxflgsq
- writedxfsmsq
- Square, small (smsq) writedxfsmsq
- writedxflgV
- V, large (lgV) writedxflgV
- writedx fsmV
- V, small (smV) writedx fsmV
- writedx fKH
- Keyhole (KH) writedx fKH
- writedx fDT
- Dovetail (DT) writedx fDT

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

```

1221 gcpy        def dxfpreamble(self, tn):
1222 gcpy #            self.writedxf(tn, str(tn))
1223 gcpy            self.writedxf(tn, "0")
1224 gcpy            self.writedxf(tn, "SECTION")
1225 gcpy            self.writedxf(tn, "2")
1226 gcpy            self.writedxf(tn, "ENTITIES")

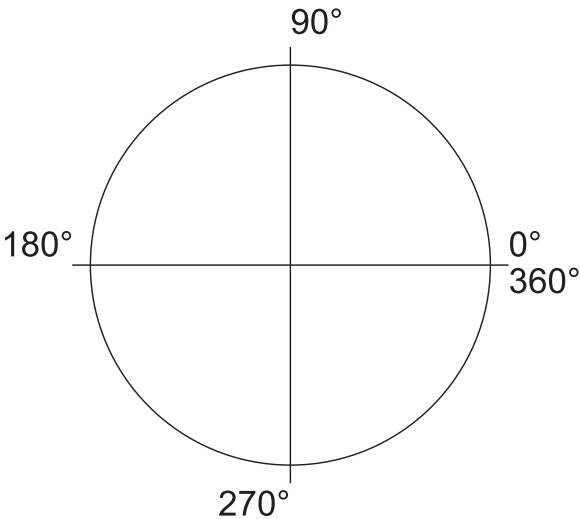
```


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.

```
1228 gcpy      def dxfline(self, tn, xbegin, ybegin, xend, yend):
1229 gcpy          self.writedxf(tn, "0")
1230 gcpy          self.writedxf(tn, "LWPOLYLINE")
1231 gcpy          self.writedxf(tn, "90")
1232 gcpy          self.writedxf(tn, "2")
1233 gcpy          self.writedxf(tn, "70")
1234 gcpy          self.writedxf(tn, "0")
1235 gcpy          self.writedxf(tn, "43")
1236 gcpy          self.writedxf(tn, "0")
1237 gcpy          self.writedxf(tn, "10")
1238 gcpy          self.writedxf(tn, str(xbegin))
1239 gcpy          self.writedxf(tn, "20")
1240 gcpy          self.writedxf(tn, str(ybegin))
1241 gcpy          self.writedxf(tn, "10")
1242 gcpy          self.writedxf(tn, str(xend))
1243 gcpy          self.writedxf(tn, "20")
1244 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).

```
1246 gcpy      def dxfarc(self, tn, xcenter, ycenter, radius, anglebegin,
1247 gcpy          endangle):
1248 gcpy          if (self.generatedxf == True):
1249 gcpy              self.writedxf(tn, "0")
```

```

1249 gcpy          self.writedxf(tn, "ARC")
1250 gcpy          self.writedxf(tn, "10")
1251 gcpy          self.writedxf(tn, str(xcenter))
1252 gcpy          self.writedxf(tn, "20")
1253 gcpy          self.writedxf(tn, str(ycenter))
1254 gcpy          self.writedxf(tn, "40")
1255 gcpy          self.writedxf(tn, str(radius))
1256 gcpy          self.writedxf(tn, "50")
1257 gcpy          self.writedxf(tn, str(anglebegin))
1258 gcpy          self.writedxf(tn, "51")
1259 gcpy          self.writedxf(tn, str(endangle))
1260 gcpy
1261 gcpy          def gcodearc(self, tn, xcenter, ycenter, radius, anglebegin,
endangle):
1262 gcpy              if (self.generategcode == True):
1263 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.

```

1265 gcpy          def cutarcNECCdxf(self, ex, ey, ez, xcenter, ycenter, radius):
1266 gcpy          #              global toolpath
1267 gcpy          #              toolpath = self.currenttool()
1268 gcpy          #              toolpath = toolpath.translate([self.xpos(), self.ypos(),
self.zpos()])
1269 gcpy          self.dxfarc(self.currenttoolnumber(), xcenter, ycenter,
radius, 0, 90)
1270 gcpy          if (self.zpos == ez):
1271 gcpy              self.settzpos(0)
1272 gcpy          else:
1273 gcpy              self.settzpos((self.zpos()-ez)/90)
1274 gcpy          #          self.setxpos(ex)
1275 gcpy          #          self.setypos(ey)
1276 gcpy          #          self.setzpos(ez)
1277 gcpy          if self.generatepaths == True:
1278 gcpy              print("Unioning cutarcNECCdxf toolpath")
1279 gcpy              self.arcloop(1, 90, xcenter, ycenter, radius)
1280 gcpy          #          self.toolpaths = self.toolpaths.union(toolpath)
1281 gcpy          else:
1282 gcpy              toolpath = self.arcloop(1, 90, xcenter, ycenter, radius
)
1283 gcpy          #          print("Returning cutarcNECCdxf toolpath")
1284 gcpy          return toolpath
1285 gcpy
1286 gcpy          def cutarcNWCCdxf(self, ex, ey, ez, xcenter, ycenter, radius):
1287 gcpy          #              global toolpath
1288 gcpy          #              toolpath = self.currenttool()
1289 gcpy          #              toolpath = toolpath.translate([self.xpos(), self.ypos(),
self.zpos()])
1290 gcpy          self.dxfarc(self.currenttoolnumber(), xcenter, ycenter,
radius, 90, 180)
1291 gcpy          if (self.zpos == ez):
1292 gcpy              self.settzpos(0)
1293 gcpy          else:
1294 gcpy              self.settzpos((self.zpos()-ez)/90)
1295 gcpy          #          self.setxpos(ex)
1296 gcpy          #          self.setypos(ey)
1297 gcpy          #          self.setzpos(ez)
1298 gcpy          if self.generatepaths == True:
1299 gcpy              self.arcloop(91, 180, xcenter, ycenter, radius)
1300 gcpy          #          self.toolpaths = self.toolpaths.union(toolpath)
1301 gcpy          else:
1302 gcpy              toolpath = self.arcloop(91, 180, xcenter, ycenter,
radius)
1303 gcpy          return toolpath
1304 gcpy
1305 gcpy          def cutarcSWCCdxf(self, ex, ey, ez, xcenter, ycenter, radius):
1306 gcpy          #              global toolpath
1307 gcpy          #              toolpath = self.currenttool()
1308 gcpy          #              toolpath = toolpath.translate([self.xpos(), self.ypos(),
self.zpos()])
1309 gcpy          self.dxfarc(self.currenttoolnumber(), xcenter, ycenter,
radius, 180, 270)
1310 gcpy          if (self.zpos == ez):
1311 gcpy              self.settzpos(0)
1312 gcpy          else:
1313 gcpy              self.settzpos((self.zpos()-ez)/90)
1314 gcpy          #          self.setxpos(ex)

```

```

1315 gcpy #         self.setypos(ey)
1316 gcpy #         self.setzpos(ez)
1317 gcpy         if self.generatepaths == True:
1318 gcpy             self.arcloop(181, 270, xcenter, ycenter, radius)
1319 gcpy #             self.toolpaths = self.toolpaths.union(toolpath)
1320 gcpy         else:
1321 gcpy             toolpath = self.arcloop(181, 270, xcenter, ycenter,
1322 gcpy                                     radius)
1323 gcpy             return toolpath
1324 gcpy         def cutarcSECCdx(self, ex, ey, ez, xcenter, ycenter, radius):
1325 gcpy             global toolpath
1326 gcpy             toolpath = self.currenttool()
1327 gcpy             toolpath = toolpath.translate([self.xpos(), self.ypos(),
1328 gcpy self.zpos()])
1329 gcpy             self.dxfarc(self.currenttoolnumber(), xcenter, ycenter,
1330 gcpy                             radius, 270, 360)
1331 gcpy             if (self.zpos == ez):
1332 gcpy                 self.settztzpos(0)
1333 gcpy             else:
1334 gcpy                 self.settztzpos((self.zpos()-ez)/90)
1335 gcpy                 self.setxpos(ex)
1336 gcpy                 self.setypos(ey)
1337 gcpy                 self.setzpos(ez)
1338 gcpy             if self.generatepaths == True:
1339 gcpy                 self.arcloop(271, 360, xcenter, ycenter, radius)
1340 gcpy                 self.toolpaths = self.toolpaths.union(toolpath)
1341 gcpy             else:
1342 gcpy                 toolpath = self.arcloop(271, 360, xcenter, ycenter,
1343 gcpy                                     radius)
1344 gcpy                 return toolpath
1345 gcpy         def cutarcNECWdx(self, ex, ey, ez, xcenter, ycenter, radius):
1346 gcpy             global toolpath
1347 gcpy             toolpath = self.currenttool()
1348 gcpy             toolpath = toolpath.translate([self.xpos(), self.ypos(),
1349 gcpy self.zpos()])
1350 gcpy             self.dxfarc(self.currenttoolnumber(), xcenter, ycenter,
1351 gcpy                             radius, 0, 90)
1352 gcpy             if (self.zpos == ez):
1353 gcpy                 self.settztzpos(0)
1354 gcpy             else:
1355 gcpy                 self.settztzpos((self.zpos()-ez)/90)
1356 gcpy                 self.setxpos(ex)
1357 gcpy                 self.setypos(ey)
1358 gcpy                 self.setzpos(ez)
1359 gcpy             if self.generatepaths == True:
1360 gcpy                 self.narcloop(89, 0, xcenter, ycenter, radius)
1361 gcpy                 self.toolpaths = self.toolpaths.union(toolpath)
1362 gcpy             else:
1363 gcpy                 toolpath = self.narcloop(89, 0, xcenter, ycenter,
1364 gcpy                                     radius)
1365 gcpy                 return toolpath
1366 gcpy         def cutarcSECWdx(self, ex, ey, ez, xcenter, ycenter, radius):
1367 gcpy             global toolpath
1368 gcpy             toolpath = self.currenttool()
1369 gcpy             toolpath = toolpath.translate([self.xpos(), self.ypos(),
1370 gcpy self.zpos()])
1371 gcpy             self.dxfarc(self.currenttoolnumber(), xcenter, ycenter,
1372 gcpy                             radius, 270, 360)
1373 gcpy             if (self.zpos == ez):
1374 gcpy                 self.settztzpos(0)
1375 gcpy             else:
1376 gcpy                 self.settztzpos((self.zpos()-ez)/90)
1377 gcpy                 self.setxpos(ex)
1378 gcpy                 self.setypos(ey)
1379 gcpy                 self.setzpos(ez)
1380 gcpy             if self.generatepaths == True:
1381 gcpy                 self.narcloop(359, 270, xcenter, ycenter, radius)
1382 gcpy                 self.toolpaths = self.toolpaths.union(toolpath)
1383 gcpy             else:
1384 gcpy                 toolpath = self.narcloop(359, 270, xcenter, ycenter,
1385 gcpy                                     radius)
1386 gcpy                 return toolpath
1387 gcpy         def cutarcSWCWdx(self, ex, ey, ez, xcenter, ycenter, radius):
1388 gcpy             global toolpath

```

```
1383 gcpy #         toolpath = self.currenttool()
1384 gcpy #         toolpath = toolpath.translate([self.xpos(), self.ypos(),
self.zpos()])
1385 gcpy         self.dxfarc(self.currenttoolnumber(), xcenter, ycenter,
radius, 180, 270)
1386 gcpy         if (self.zpos == ez):
1387 gcpy             self.settzpos(0)
1388 gcpy         else:
1389 gcpy             self.settzpos((self.zpos()-ez)/90)
1390 gcpy #         self.setxpos(ex)
1391 gcpy #         self.setypos(ey)
1392 gcpy #         self.setzpos(ez)
1393 gcpy         if self.generatepaths == True:
1394 gcpy             self.narcloop(269, 180, xcenter, ycenter, radius)
1395 gcpy #             self.toolpaths = self.toolpaths.union(toolpath)
1396 gcpy         else:
1397 gcpy             toolpath = self.narcloop(269, 180, xcenter, ycenter,
radius)
1398 gcpy             return toolpath
1399 gcpy
1400 gcpy         def cutarcNWCWdxf(self, ex, ey, ez, xcenter, ycenter, radius):
1401 gcpy             global toolpath
1402 gcpy #             toolpath = self.currenttool()
1403 gcpy #             toolpath = toolpath.translate([self.xpos(), self.ypos(),
self.zpos()])
1404 gcpy             self.dxfarc(self.currenttoolnumber(), xcenter, ycenter,
radius, 90, 180)
1405 gcpy             if (self.zpos == ez):
1406 gcpy                 self.settzpos(0)
1407 gcpy             else:
1408 gcpy                 self.settzpos((self.zpos()-ez)/90)
1409 gcpy #             self.setxpos(ex)
1410 gcpy #             self.setypos(ey)
1411 gcpy #             self.setzpos(ez)
1412 gcpy             if self.generatepaths == True:
1413 gcpy                 self.narcloop(179, 90, xcenter, ycenter, radius)
1414 gcpy #                 self.toolpaths = self.toolpaths.union(toolpath)
1415 gcpy             else:
1416 gcpy                 toolpath = self.narcloop(179, 90, xcenter, ycenter,
radius)
1417 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

```
1419 gcpy         def arcCCgc(self, ex, ey, ez, xcenter, ycenter, radius):
1420 gcpy             self.writegc("G03_X", str(ex), "Y", str(ey), "Z", str(ez)
, "R", str(radius))
1421 gcpy
1422 gcpy         def arcCWgc(self, ex, ey, ez, xcenter, ycenter, radius):
1423 gcpy             self.writegc("G02_X", str(ex), "Y", str(ey), "Z", str(ez)
, "R", str(radius))
```

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

```
1425 gcpy         def cutarcNECCdxfgc(self, ex, ey, ez, xcenter, ycenter, radius)
:
1426 gcpy             self.arcCCgc(ex, ey, ez, xcenter, ycenter, radius)
1427 gcpy             if self.generatepaths == True:
1428 gcpy                 self.cutarcNECCdxf(ex, ey, ez, xcenter, ycenter, radius
)
1429 gcpy             else:
1430 gcpy                 return self.cutarcNECCdxf(ex, ey, ez, xcenter, ycenter,
radius)
1431 gcpy
1432 gcpy         def cutarcNWCCdxfgc(self, ex, ey, ez, xcenter, ycenter, radius)
:
1433 gcpy             self.arcCCgc(ex, ey, ez, xcenter, ycenter, radius)
1434 gcpy             if self.generatepaths == False:
1435 gcpy                 return self.cutarcNWCCdxf(ex, ey, ez, xcenter, ycenter,
radius)
1436 gcpy
```

```

1437 gcpy      def cutarcSWCCdxfgc(self, ex, ey, ez, xcenter, ycenter, radius)
                :
1438 gcpy          self.arcCCGc(ex, ey, ez, xcenter, ycenter, radius)
1439 gcpy          if self.generatepaths == False:
1440 gcpy              return self.cutarcSWCCdxfc(ex, ey, ez, xcenter, ycenter,
                    radius)
1441 gcpy
1442 gcpy      def cutarcSECCdxfgc(self, ex, ey, ez, xcenter, ycenter, radius)
                :
1443 gcpy          self.arcCCGc(ex, ey, ez, xcenter, ycenter, radius)
1444 gcpy          if self.generatepaths == False:
1445 gcpy              return self.cutarcSECCdxfc(ex, ey, ez, xcenter, ycenter,
                    radius)
1446 gcpy
1447 gcpy      def cutarcNECWdxfgc(self, ex, ey, ez, xcenter, ycenter, radius)
                :
1448 gcpy          self.arcCWgc(ex, ey, ez, xcenter, ycenter, radius)
1449 gcpy          if self.generatepaths == False:
1450 gcpy              return self.cutarcNECWdxfc(ex, ey, ez, xcenter, ycenter,
                    radius)
1451 gcpy
1452 gcpy      def cutarcSECWdxfgc(self, ex, ey, ez, xcenter, ycenter, radius)
                :
1453 gcpy          self.arcCWgc(ex, ey, ez, xcenter, ycenter, radius)
1454 gcpy          if self.generatepaths == False:
1455 gcpy              return self.cutarcSECWdxfc(ex, ey, ez, xcenter, ycenter,
                    radius)
1456 gcpy
1457 gcpy      def cutarcSWCWdxfgc(self, ex, ey, ez, xcenter, ycenter, radius)
                :
1458 gcpy          self.arcCWgc(ex, ey, ez, xcenter, ycenter, radius)
1459 gcpy          if self.generatepaths == False:
1460 gcpy              return self.cutarcSWCWdxfc(ex, ey, ez, xcenter, ycenter,
                    radius)
1461 gcpy
1462 gcpy      def cutarcNWCWdxfgc(self, ex, ey, ez, xcenter, ycenter, radius)
                :
1463 gcpy          self.arcCWgc(ex, ey, ez, xcenter, ycenter, radius)
1464 gcpy          if self.generatepaths == False:
1465 gcpy              return self.cutarcNWCWdxfc(ex, ey, ez, xcenter, ycenter,
                    radius)

```

```

126 gcpscad module cutarcNECCdxfgc(ex, ey, ez, xcenter, ycenter, radius){
127 gcpscad     gcp.cutarcNECCdxfgc(ex, ey, ez, xcenter, ycenter, radius);
128 gcpscad }
129 gcpscad
130 gcpscad module cutarcNWCCdxfgc(ex, ey, ez, xcenter, ycenter, radius){
131 gcpscad     gcp.cutarcNWCCdxfgc(ex, ey, ez, xcenter, ycenter, radius);
132 gcpscad }
133 gcpscad
134 gcpscad module cutarcSWCCdxfgc(ex, ey, ez, xcenter, ycenter, radius){
135 gcpscad     gcp.cutarcSWCCdxfgc(ex, ey, ez, xcenter, ycenter, radius);
136 gcpscad }
137 gcpscad
138 gcpscad module cutarcSECCdxfgc(ex, ey, ez, xcenter, ycenter, radius){
139 gcpscad     gcp.cutarcSECCdxfgc(ex, ey, ez, xcenter, ycenter, radius);
140 gcpscad }

```

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

```

1467 gcpy      def dxfpostamble(self, tn):
1468 gcpy #      self.writedxf(tn, str(tn))
1469 gcpy      self.writedxf(tn, "0")
1470 gcpy      self.writedxf(tn, "ENDSEC")
1471 gcpy      self.writedxf(tn, "0")
1472 gcpy      self.writedxf(tn, "EOF")


---


1474 gcpy      def gcodepostamble(self):
1475 gcpy      self.writegc("Z12.700")
1476 gcpy      self.writegc("M05")

```

```
1477 gcpy self.writegc("M02")
```

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

```
1479 gcpy def closegcodefile(self):
1480 gcpy     self.gcodepreamble()
1481 gcpy     self.gc.close()
1482 gcpy
1483 gcpy def closedxfile(self):
1484 gcpy     if self.generatedxf == True:
1485 gcpy         global dxfclosed
1486 gcpy         self.dxfpreamble(-1)
1487 gcpy         self.dxfclosed = True
1488 gcpy         self.dxf.close()
1489 gcpy
1490 gcpy def closedxfiles(self):
1491 gcpy     if self.generatedxfs == True:
1492 gcpy         if (self.large_square_tool_num > 0):
1493 gcpy             self.dxfpreamble(self.large_square_tool_num)
1494 gcpy         if (self.small_square_tool_num > 0):
1495 gcpy             self.dxfpreamble(self.small_square_tool_num)
1496 gcpy         if (self.large_ball_tool_num > 0):
1497 gcpy             self.dxfpreamble(self.large_ball_tool_num)
1498 gcpy         if (self.small_ball_tool_num > 0):
1499 gcpy             self.dxfpreamble(self.small_ball_tool_num)
1500 gcpy         if (self.large_V_tool_num > 0):
1501 gcpy             self.dxfpreamble(self.large_V_tool_num)
1502 gcpy         if (self.small_V_tool_num > 0):
1503 gcpy             self.dxfpreamble(self.small_V_tool_num)
1504 gcpy         if (self.DT_tool_num > 0):
1505 gcpy             self.dxfpreamble(self.DT_tool_num)
1506 gcpy         if (self.KH_tool_num > 0):
1507 gcpy             self.dxfpreamble(self.KH_tool_num)
1508 gcpy         if (self.Roundover_tool_num > 0):
1509 gcpy             self.dxfpreamble(self.Roundover_tool_num)
1510 gcpy         if (self.MISC_tool_num > 0):
1511 gcpy             self.dxfpreamble(self.MISC_tool_num)
1512 gcpy
1513 gcpy         if (self.large_square_tool_num > 0):
1514 gcpy             self.dxfsq.close()
1515 gcpy         if (self.small_square_tool_num > 0):
1516 gcpy             self.dxfmsq.close()
1517 gcpy         if (self.large_ball_tool_num > 0):
1518 gcpy             self.dxfgbl.close()
1519 gcpy         if (self.small_ball_tool_num > 0):
1520 gcpy             self.dxfmb1.close()
1521 gcpy         if (self.large_V_tool_num > 0):
1522 gcpy             self.dxfV.close()
1523 gcpy         if (self.small_V_tool_num > 0):
1524 gcpy             self.dxfmV.close()
1525 gcpy         if (self.DT_tool_num > 0):
1526 gcpy             self.dxfDT.close()
1527 gcpy         if (self.KH_tool_num > 0):
1528 gcpy             self.dxfKH.close()
1529 gcpy         if (self.Roundover_tool_num > 0):
1530 gcpy             self.dxfRt.close()
1531 gcpy         if (self.MISC_tool_num > 0):
1532 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: dxfpreamble which when called provides
dxfpreamble the boilerplate needed at the end of their respective files.

```
142 gpcscad module closegcodefile(){
143 gpcscad     gcp.closegcodefile();
144 gpcscad }
145 gpcscad
146 gpcscad module closedxfiles(){
147 gpcscad     gcp.closedxfiles();
148 gpcscad }
149 gpcscad
150 gpcscad module closedxfile(){
151 gpcscad     gcp.closedxfile();
152 gpcscad }
```

4Notes

Other Resources

Coding Style

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

Red Flags to avoid include:

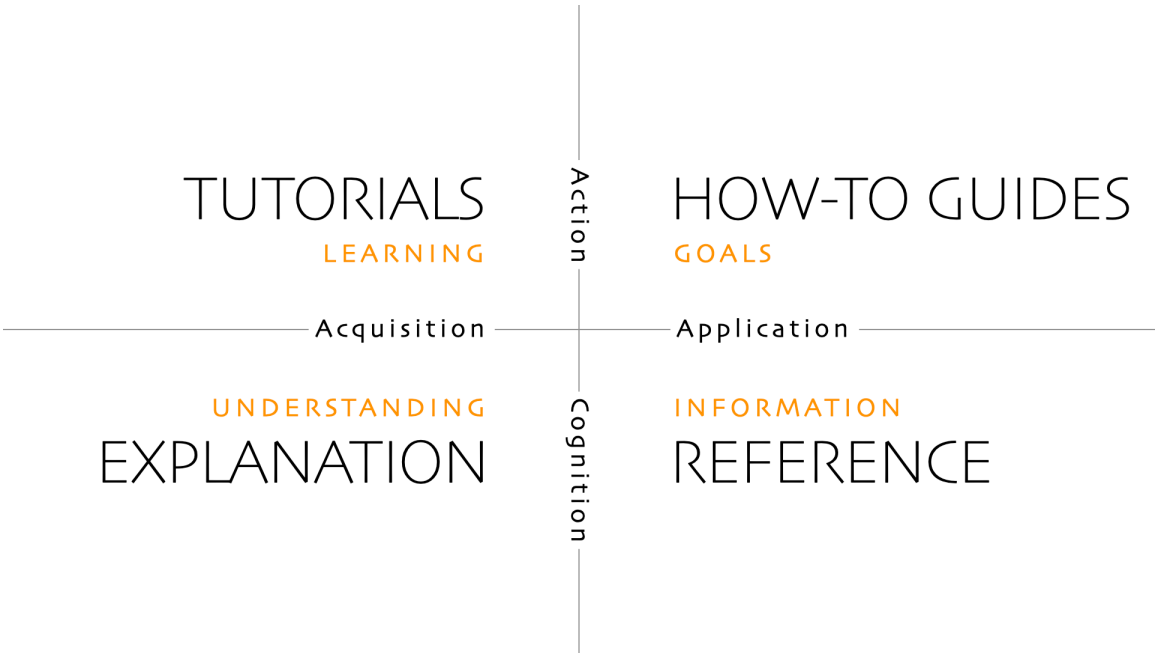
- Shallow Module
- Information Leakage
- Temporal Decomposition
- Overexposure
- Pass-Through Method
- Repetition
- Special-General Mixture
- Conjoined Methods
- Comment Repeats Code
- Implementation Documentation Contaminates Interface
- Vague Name
- Hard to Pick Name
- Hard to Describe
- Nonobvious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

Documentation Style

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

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

resulting in a matrix of:



where:

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

Holidays

Holidays are from <https://nationaltoday.com/>

DXFs

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

Future

Images

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

Import G-code

Use a tool to read in a G-code file, then create a 3D model which would serve as a preview of the cut?

- <https://stackoverflow.com/questions/34638372/simple-python-program-to-read-gcode-file>
- <https://pypi.org/project/gcodeparser/>
- <https://github.com/fragmuffin/pygcode/wiki>

Bézier curves in 2 dimensions

Take a Bézier curve definition and approximate it as arcs and write them into a DXF?

<https://pomax.github.io/bezierinfo/>
<https://ciechanow.ski/curves-and-surfaces/>
<https://www.youtube.com/watch?v=aVwxzDHniEw>
 c.f., <https://linuxcnc.org/docs/html/gcode/g-code.html#gcode:g5>

Bézier curves in 3 dimensions

One question is how many Bézier curves would it be necessary to have to define a surface in 3 dimensions. Attributes for this which are desirable/necessary:

- concise — a given Bézier curve should be represented by just the point coordinates, so two on-curve points, two off-curve points, each with a pair of coordinates
- For a given shape/region it will need to be possible to have a matching definition exactly match up with it so that one could piece together a larger more complex shape from smaller/simpler regions
- similarly it will be necessary for it to be possible to sub-divide a defined region — for example it should be possible if one had 4 adjacent regions, then the four quadrants at the intersection of the four regions could be used to construct a new region — is it possible to derive a new Bézier curve from half of two other curves?

For the three planes:

- XY
- XZ
- ZY

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

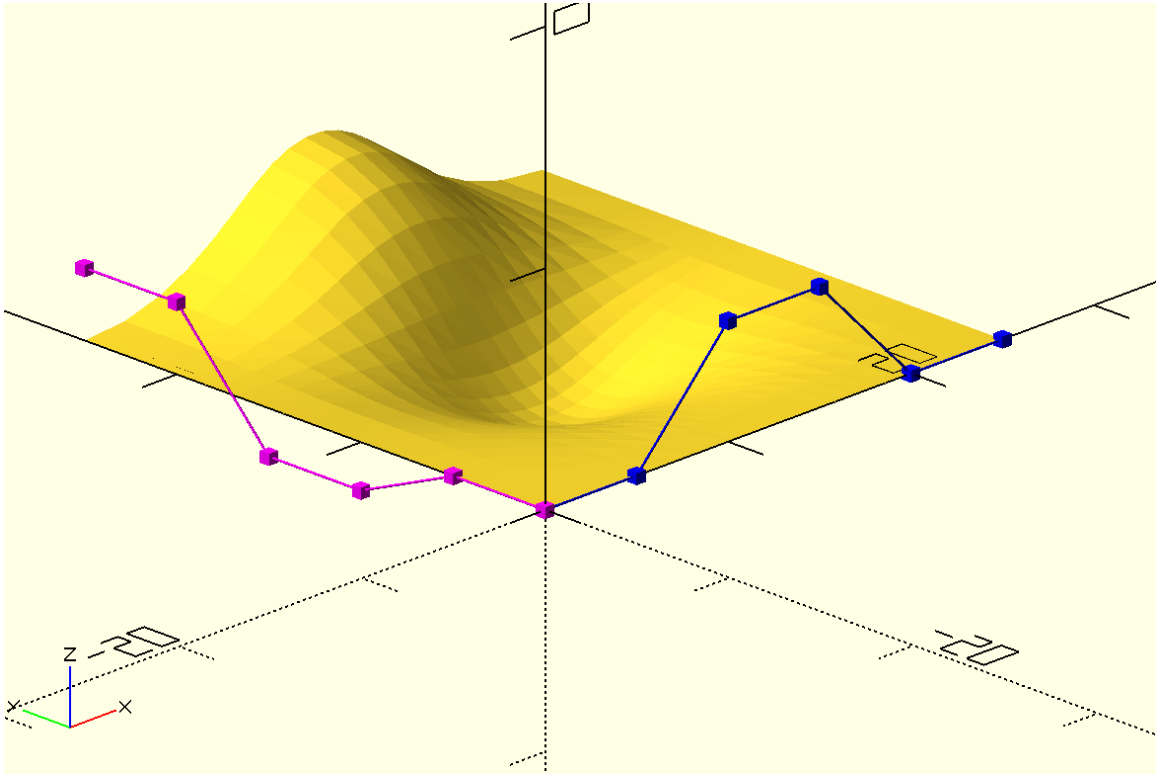
3 planes * 3 Béziers * (2 on-curve + 2 off-curve points) == 36 coordinate pairs

which is a marked contrast to representations such as:

<https://github.com/DavidPhillipOster/Teapot>

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

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



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

Mathematics

<https://elementsofprogramming.com/>

References

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

Command Glossary

settool settool(102). 25

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

Index

- ballnose, 27
- bowl tool, 27
- closedxfile, 61, 62
- closegcodefile, 61, 62
- currenttoolnum, 23
- currenttoolnumber, 26
- currenttoolshape, 30
- cut..., 34, 36
- cutarcCC, 38
- cutarcCW, 38
- cutkeyhole toolpath, 44
- cutKHgcdxf, 45
- cutline, 36
- dovetail, 29
- dxfarc, 57
- dxfline, 57
- dxfpreamble, 62
- dxfpreamble, 56
- dxfwrite, 56
- endmill square, 27
- endmill v, 27
- feed, 34
- flat V, 28
- gcodepreview, 21
 - writeln, 53
- gcp.setupstock, 24
- init, 21
- keyhole, 29
- mpx, 23
- mpy, 23
- mpz, 23
- opendxfile, 54
- opengcodefile, 54
- plunge, 34
- rapid..., 34
- rcl, 34
- settool, 26
- setupstock, 24
 - gcodepreview, 24
- setxpos, 23
- setypos, 23
- setzpos, 23
- speed, 34
- stepsizearc, 20
- stepsizearoundover, 20
- subroutine
 - gcodepreview, 24
 - writeln, 53
- tapered ball, 28
- threadmill, 29
- tool diameter, 32
- tool radius, 33
- toolchange, 30
- tpzinc, 23
- writedxDT, 56
- writedxKH, 56
- writedxflgbl, 56
- writedxflgsq, 56
- writedxflgV, 56
- writedxfsmb, 56
- writedxfsmsq, 56
- writedxfsmV, 56
- xpos, 23
- ypos, 23
- zpos, 23

Routines

- ballnose, 27
- bowl tool, 27
- closedxfile, 61, 62
- closegcodefile, 61, 62
- currenttoolnumber, 26
- cut..., 34, 36
- cutarcCC, 38
- cutarcCW, 38
- cutkeyhole toolpath, 44
- cutKHgcdxf, 45
- cutline, 36
- dovetail, 29
- dxfar, 57
- dxflin, 57
- dxfpreamble, 62
- dxfpreamble, 56
- dxfwrite, 56
- endmill square, 27
- endmill v, 27
- flat V, 28
- gcodepreview, 21, 24
- gcp.setupstock, 24
- init, 21
- keyhole, 29
- opendxfile, 54
- opengcodefile, 54
- rapid..., 34
- rcl, 34
- settool, 26
- setupstock, 24
- setxpos, 23
- setypos, 23
- setzpos, 23
- tapered ball, 28
- threadmill, 29
- tool diameter, 32
- tool radius, 33
- toolchange, 30
- writedxDT, 56
- writedxKH, 56
- writedxflgl, 56
- writedxflgsq, 56
- writedxflgV, 56
- writedxfsmb, 56
- writedxfsmsq, 56
- writedxsmV, 56
- writeln, 53
- xpos, 23
- ypos, 23
- zpos, 23

Variables

currenttoolnum, 23	plunge, 34
currenttoolshape, 30	
feed, 34	speed, 34
mpx, 23	stepsizearc, 20
mpy, 23	stepsizeroundover, 20
mpz, 23	tpzinc, 23