

# The gcodepreview PythonSCAD library\*

Author: William F. Adams  
willadams at aol dot com

2025/01/29

## Abstract

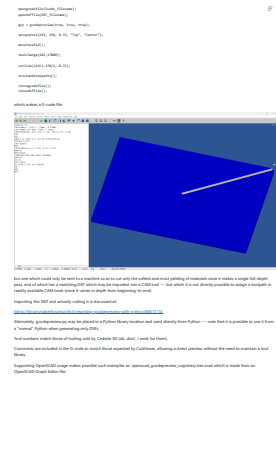
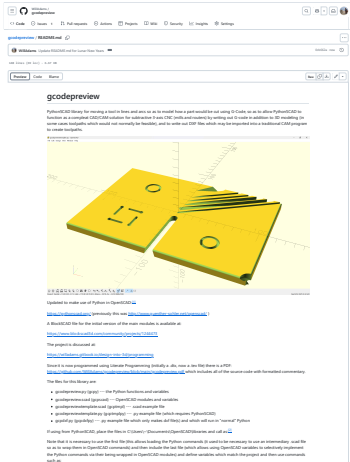
The gcodepreview library allows using PythonSCAD (OpenPythonSCAD) to move a tool in lines and arcs and output DXF and G-code files so as to work as a CAD/CAM program for CNC.

## Contents

<b>1</b>	<b>readme.md</b>	<b>2</b>
<b>2</b>	<b>Usage and Templates</b>	<b>5</b>
2.1	gcpdxf.py	5
2.2	gcodepreviewtemplate.py	7
2.3	gcodepreviewtemplate.scad	12
<b>3</b>	<b>gcodepreview</b>	<b>17</b>
3.1	Module Naming Convention	17
3.1.1	Parameters and Default Values	19
3.2	Implementation files and gcodepreview class	19
3.2.1	Position and Variables	22
3.2.2	Initial Modules	23
3.3	Tools and Changes	25
3.3.1	3D Shapes for Tools	26
3.3.1.1	Normal Tooling/toolshapes	26
3.3.1.2	Tooling for Undercutting Toolpaths	27
3.3.1.2.1	Keyhole tools	27
3.3.1.2.2	Thread mills	27
3.3.1.2.3	Dovetails	28
3.3.1.3	Concave toolshapes	28
3.3.1.4	Roundover tooling	28
3.3.2	toolchange	28
3.3.2.1	Selecting Tools	28
3.3.2.2	Square and ball nose (including tapered ball nose)	29
3.3.2.3	Roundover (corner rounding)	29
3.3.2.4	Dovetails	29
3.3.2.5	toolchange routine	29
3.3.3	tooldiameter	30
3.3.4	Feeds and Speeds	32
3.4	Movement and Cutting	32
3.4.1	Lines	34
3.4.2	Arcs for toolpaths and DXFs	35
3.4.3	Cutting shapes and expansion	39
3.4.3.1	Building blocks	39
3.4.3.2	List of shapes	39
3.4.3.2.1	circles	41
3.4.3.2.2	rectangles	41
3.4.3.2.3	Keyhole toolpath and undercut tooling	42
3.4.4	Difference of Stock, Rapids, and Toolpaths	49
3.5	Output files	49
3.5.1	G-code Overview	50
3.5.2	DXF Overview	50
3.5.3	Python and OpenSCAD File Handling	51
3.5.3.1	Writing to DXF files	54
3.5.3.2	Closings	59
<b>4</b>	<b>Notes</b>	<b>61</b>
	<b>Index</b>	<b>65</b>
	Routines	66
	Variables	67

\*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 - work on rotary axis option

```

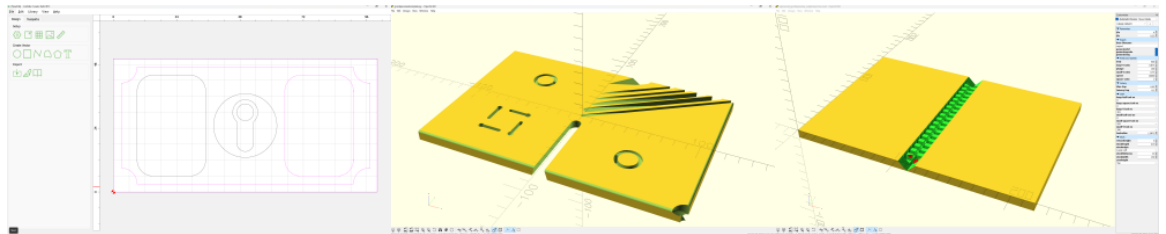
---

## 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 topic). To a great degree, the basic implementation of this tool will use G-code as a reference implementation, simultaneously using the abstraction from the mechanical task of machining which it affords as a decomposed version of that task, and creating what is in essence, both a front-end, and a tool, and an API for working with G-code programmatically. This then requires an architecture which allows 3D modeling (OpenSCAD), and writing out files (Python).

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

### 2.1 gcpdxf.py

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

---

```
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                           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                       radius, 90, 180)
65 gcpdxftp gcp.dxfarc(large_square_tool_num, stockXwidth - inset, stockYheight
66                       - inset, radius, 180, 270)
67 gcpdxftp gcp.dxfarc(large_square_tool_num, inset, stockYheight - inset,
68                       radius, 270, 360)
69 gcpdxftp
70 gcpdxftp gcp.dxfline(large_square_tool_num, inset, inset + radius, inset,
71                       stockYheight - (inset + radius))
72 gcpdxftp gcp.dxfline(large_square_tool_num, inset + radius, inset,
73                       stockXwidth - (inset + radius), inset)
74 gcpdxftp gcp.dxfline(large_square_tool_num, stockXwidth - inset, inset +
75                       radius, stockYheight - (inset + radius))
76 gcpdxftp gcp.dxfline(large_square_tool_num, inset + radius, stockYheight -
77                       inset, stockXwidth - (inset + radius), stockYheight - inset)
78 gcpdxftp
79 gcpdxftp gcp.dxfrectangle(large_square_tool_num, radius + inset, radius,
80                           stockXwidth/2 - (radius * 4), stockYheight - (radius * 2),
81                           cornerstyle, radius)
82 gcpdxftp gcp.dxfrectangle(large_square_tool_num, stockXwidth/2 + (radius *
83                           2) + inset, radius, stockXwidth/2 - (radius * 4), stockYheight -
84                           (radius * 2), cornerstyle, radius)
85 gcpdxftp #gcp.dxfrectangleround(large_square_tool_num, 64, 7, 24, 36, radius

```

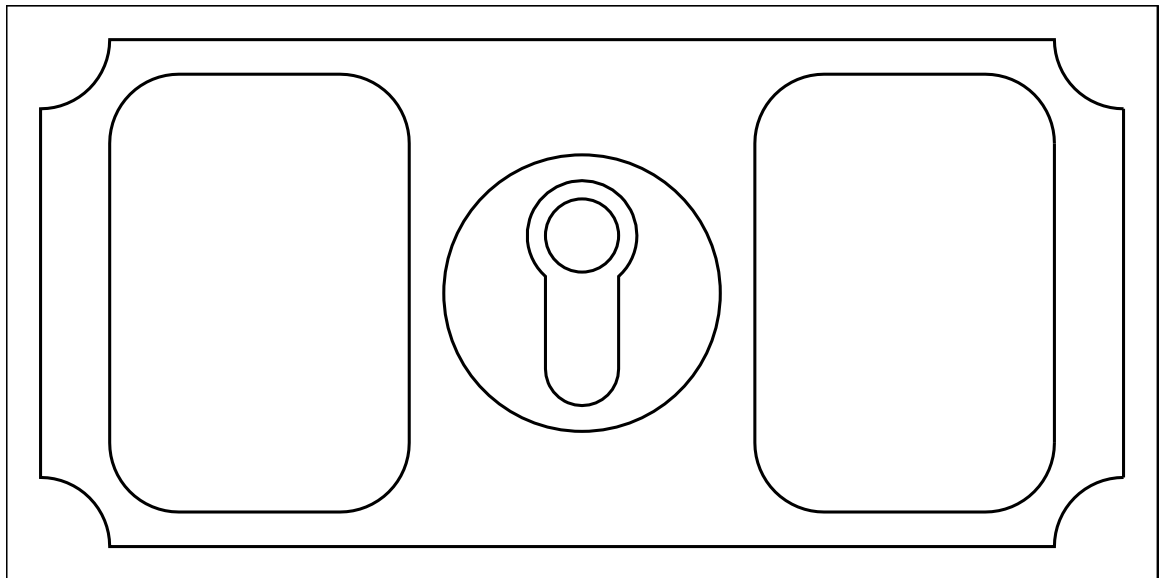
```

    )
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 # [648:648, 45982:45982]
60 gcptmplpy #648 threadmill_shaft(2.4, 0.75, 18)
61 gcptmplpy #45982 Carbide Tipped Bowl & Tray 1/4 Radius x 3/4 Dia x 5/8 x 1/4
    Inch Shank
62 gcptmplpy
63 gcptmplpy # [Feeds and Speeds] */
64 gcptmplpy plunge = 100
65 gcptmplpy # [Feeds and Speeds] */
66 gcptmplpy feed = 400
67 gcptmplpy # [Feeds and Speeds] */
68 gcptmplpy speed = 16000
69 gcptmplpy # [Feeds and Speeds] */
70 gcptmplpy small_square_ratio = 0.75 # [0.25:2]
71 gcptmplpy # [Feeds and Speeds] */
72 gcptmplpy large_ball_ratio = 1.0 # [0.25:2]
73 gcptmplpy # [Feeds and Speeds] */
74 gcptmplpy small_ball_ratio = 0.75 # [0.25:2]
75 gcptmplpy # [Feeds and Speeds] */
76 gcptmplpy large_V_ratio = 0.875 # [0.25:2]
77 gcptmplpy # [Feeds and Speeds] */
78 gcptmplpy small_V_ratio = 0.625 # [0.25:2]
79 gcptmplpy # [Feeds and Speeds] */
80 gcptmplpy DT_ratio = 0.75 # [0.25:2]
81 gcptmplpy # [Feeds and Speeds] */
82 gcptmplpy KH_ratio = 0.75 # [0.25:2]
83 gcptmplpy # [Feeds and Speeds] */
84 gcptmplpy RO_ratio = 0.5 # [0.25:2]
85 gcptmplpy # [Feeds and Speeds] */
86 gcptmplpy MISC_ratio = 0.5 # [0.25:2]
87 gcptmplpy
88 gcptmplpy gcp = gcodepreview(generatepaths,
89 gcptmplpy                    generategcode,
90 gcptmplpy                    generatedxf,
91 gcptmplpy                    )

```



```

92 gcptmplpy
93 gcptmplpy gcp.opengcodefile(Base_filename)
94 gcptmplpy gcp.opendxfile(Base_filename)
95 gcptmplpy gcp.opendxfiles(Base_filename,
96 gcptmplpy         large_square_tool_num,
97 gcptmplpy         small_square_tool_num,
98 gcptmplpy         large_ball_tool_num,
99 gcptmplpy         small_ball_tool_num,
100 gcptmplpy         large_V_tool_num,
101 gcptmplpy         small_V_tool_num,
102 gcptmplpy         DT_tool_num,
103 gcptmplpy         KH_tool_num,
104 gcptmplpy         Roundover_tool_num,
105 gcptmplpy         MISC_tool_num)
106 gcptmplpy gcp.setupstock(stockXwidth,stockYheight,stockZthickness,"Top",
        Center",retractheight)

107 gcptmplpy
108 gcptmplpy #print(pygcpversion())
109 gcptmplpy
110 gcptmplpy #print(gcp.myfunc(4))
111 gcptmplpy
112 gcptmplpy #print(gcp.getvv())
113 gcptmplpy
114 gcptmplpy #ts = cylinder(12.7, 1.5875, 1.5875)
115 gcptmplpy #toolpaths = gcp.cutshape(stockXwidth/2,stockYheight/2,-
        stockZthickness)

116 gcptmplpy
117 gcptmplpy gcp.movetosafeZ()
118 gcptmplpy
119 gcptmplpy gcp.toolchange(102,10000)
120 gcptmplpy
121 gcptmplpy #gcp.rapidXY(6,12)
122 gcptmplpy gcp.rapidZ(0)
123 gcptmplpy
124 gcptmplpy #print (gcp.xpos())
125 gcptmplpy #print (gcp.ypos())
126 gcptmplpy #psetzpos(7)
127 gcptmplpy #gcp.setzpos(-12)
128 gcptmplpy #print (gcp.zpos())
129 gcptmplpy
130 gcptmplpy #print ("X", str(gcp.xpos()))
131 gcptmplpy #print ("Y", str(gcp.ypos()))
132 gcptmplpy #print ("Z", str(gcp.zpos()))
133 gcptmplpy
134 gcptmplpy toolpaths = gcp.currenttool()
135 gcptmplpy
136 gcptmplpy #toolpaths = gcp.cutline(stockXwidth/2,stockYheight/2,-
        stockZthickness)
137 gcptmplpy toolpaths = toolpaths.union(gcp.cutlinedxfgc(stockXwidth/2,
        stockYheight/2, -stockZthickness))

138 gcptmplpy
139 gcptmplpy gcp.rapidZ(retractheight)
140 gcptmplpy gcp.toolchange(201,10000)
141 gcptmplpy gcp.rapidXY(0, stockYheight/16)
142 gcptmplpy gcp.rapidZ(0)
143 gcptmplpy toolpaths = toolpaths.union(gcp.cutlinedxfgc(stockXwidth/16*7,
        stockYheight/2, -stockZthickness))

144 gcptmplpy
145 gcptmplpy gcp.rapidZ(retractheight)
146 gcptmplpy gcp.toolchange(202,10000)
147 gcptmplpy gcp.rapidXY(0, stockYheight/8)
148 gcptmplpy gcp.rapidZ(0)
149 gcptmplpy toolpaths = toolpaths.union(gcp.cutlinedxfgc(stockXwidth/16*6,
        stockYheight/2, -stockZthickness))

150 gcptmplpy
151 gcptmplpy gcp.rapidZ(retractheight)
152 gcptmplpy gcp.toolchange(101,10000)
153 gcptmplpy gcp.rapidXY(0, stockYheight/16*3)
154 gcptmplpy gcp.rapidZ(0)
155 gcptmplpy toolpaths = toolpaths.union(gcp.cutlinedxfgc(stockXwidth/16*5,
        stockYheight/2, -stockZthickness))

156 gcptmplpy
157 gcptmplpy gcp.setzpos(retractheight)
158 gcptmplpy gcp.toolchange(390,10000)
159 gcptmplpy gcp.rapidXY(0, stockYheight/16*4)
160 gcptmplpy gcp.rapidZ(0)
161 gcptmplpy toolpaths = toolpaths.union(gcp.cutlinedxfgc(stockXwidth/16*4,
        stockYheight/2, -stockZthickness))

```

```

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

```

```

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

```

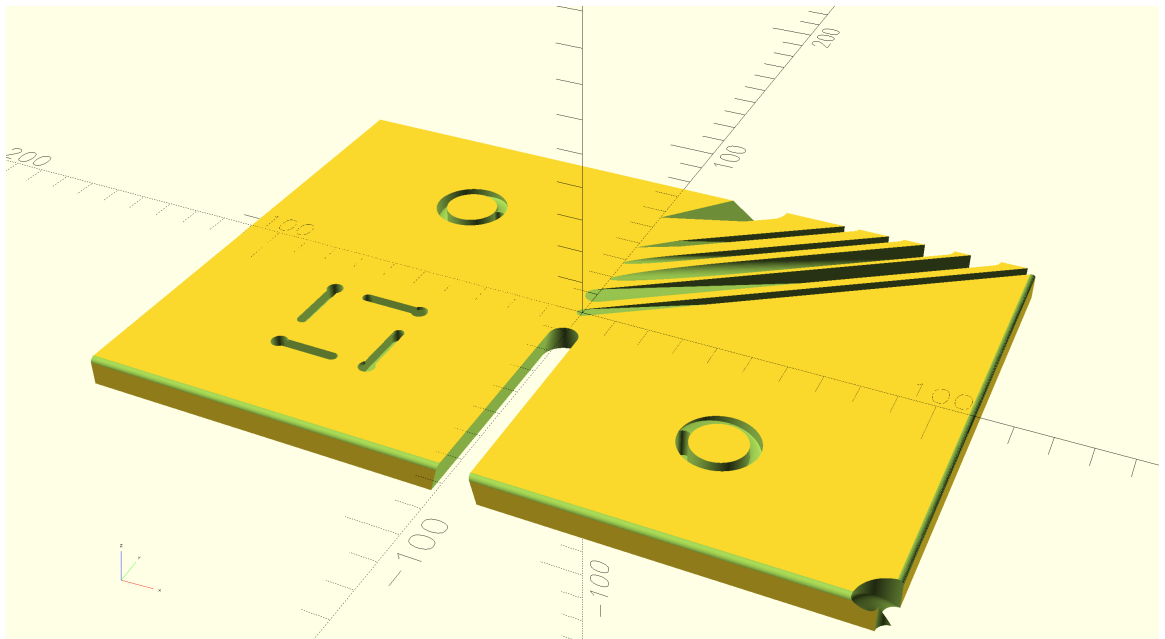
```

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

```

---

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



### 2.3 gcodepreviewtemplate.scad

Since the project began in OpenSCAD, having an implementation in that language has always been a goal. This is quite straight-forward since the Python code when imported into OpenSCAD may be accessed by quite simple modules which are for the most part, a series of decorators/descriptors which wrap up the Python definitions as OpenSCAD modules. Moreover, such an implementation will facilitate usage by tools intended for this application such as OpenSCAD Graph Editor: <https://github.com/derkork/openscad-graph-editor>. 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.opendxfile(Base_filename);
94 gcptmpl.opendxfiles(Base_filename,
95 gcptmpl                      large_square_tool_num,
96 gcptmpl                      small_square_tool_num,
97 gcptmpl                      large_ball_tool_num,
98 gcptmpl                      small_ball_tool_num,
99 gcptmpl                      large_V_tool_num,
100 gcptmpl                     small_V_tool_num,
101 gcptmpl                     DT_tool_num,
102 gcptmpl                     KH_tool_num,
103 gcptmpl                     Roundover_tool_num,
104 gcptmpl                     MISC_tool_num);
105 gcptmpl
106 gcptmpl.setupstock(stockXwidth, stockYheight, stockZthickness, zeroheight,
107 gcptmpl                      stockzero);
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);

```

```

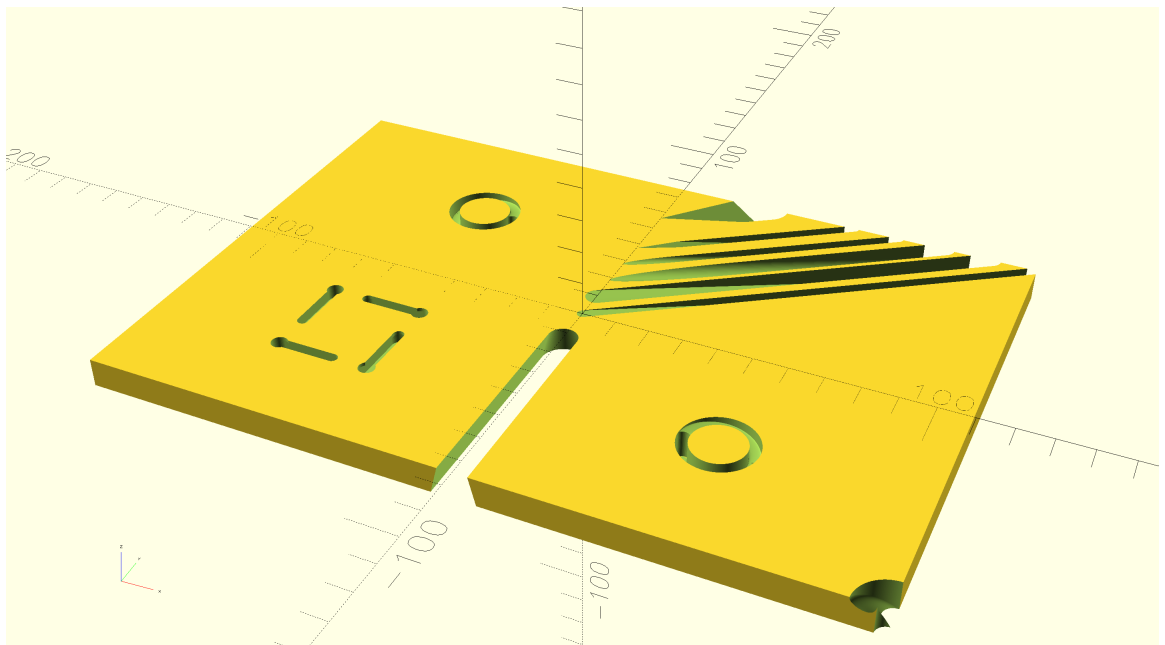
218 gcptmpl cutlinedxfgc(xpos() + stockYheight /9, ypos(), zpos());
219 gcptmpl cutline(xpos() - stockYheight /9, ypos(), zpos());
220 gcptmpl cutline(xpos(), ypos(), 0);
221 gcptmpl
222 gcptmpl rapidZ(retractheight);
223 gcptmpl rapidXY(-stockXwidth/4+stockXwidth/16, -(stockYheight/4+
      stockYheight/16));
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 cutZgcfeed(-1.531,plunge);
252 gcptmpl //cutline(gcp.xpos(), gcp.ypos(), -1.531);
253 gcptmpl cutlinedxfgc(stockXwidth/2+0.508/2, -(stockYheight/2+0.508/2),
      -1.531);
254 gcptmpl
255 gcptmpl rapidZ(retractheight);
256 gcptmpl //gcp.toolchange(56125,10000)
257 gcptmpl cutZgcfeed(-1.531,plunge);
258 gcptmpl //toolpaths = toolpaths.union(gcp.cutline(gcp.xpos(), gcp.ypos(),
      -1.531))
259 gcptmpl cutlinedxfgc(stockXwidth/2+0.508/2, (stockYheight/2+0.508/2),
      -1.531);
260 gcptmpl
261 gcptmpl stockandtoolpaths();
262 gcptmpl //stockwotoolpaths();
263 gcptmpl //outputtoolpaths();
264 gcptmpl
265 gcptmpl //makecube(3, 2, 1);
266 gcptmpl
267 gcptmpl //instantiatecube();
268 gcptmpl
269 gcptmpl closegcodefile();
270 gcptmpl closedxfiles();
271 gcptmpl closedxfile();

```

---

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





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

Note that there are several possible ways to work with the 3D models of the cuts, either directly displaying the returned 3D model when explicitly called for after storing it in a variable or calling it up as a calculation (Python command `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 4<sup>th</sup> additional axis may be worked up as a future option) and if desired, writing out DXF and/or G-code files (as opposed to the normal technique of rendering to a 3D model and writing out an STL or STEP or other model format and using a traditional CAM application). There are multiple modes for this, doing so requires two files:

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

```
import gcodepreview_standalone as gcp
```

 or used in an OpenSCAD file:  

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

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

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

#### 3.1 Module Naming Convention

The original implementation required three files and used a convention for prefacing commands with `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 straight-forward to ensure that functions and variables have different names for purposes of scope

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

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

	line			arc		
	cut	dx	gcode	cut	dx	gcode
cut	cutline		cutlinegc	cutarc		cutarcgc
dx	cutlinedx	dxline		cutarcdx	dxfar	
gcode	cutlinegc		linegc	cutarcgc		arcgc
	cutlinedxfgc			cutarcdxfgc		

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

Principles for naming modules (and variables):

- minimize use of underscores (for convenience sake, underscores are not used for index entries)

- identify which aspect of the project structure is being worked with (cut(ting), dxf, gcode, tool, etc.) note the `gcodepreview` class which will normally be imported as `gcp` so that module `<foo>` will be called as `gcp.<foo>` from Python and by the same `<foo>` in OpenSCAD

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

There are multiple modes for programming PythonSCAD:

- Python — in `gcodepreview` this allows writing out dxf files
- OpenSCAD — see: <https://openscad.org/documentation.html>
- Programming in OpenSCAD with variables and calling Python — this requires 3 files and was originally used in the project as written up at: [https://github.com/WillAdams/gcodepreview/blob/main/gcodepreview-openscad\\_0\\_6.pdf](https://github.com/WillAdams/gcodepreview/blob/main/gcodepreview-openscad_0_6.pdf) (for further details see below)
- Programming in OpenSCAD and calling Python where all variables as variables are held in Python classes (this is the technique used as of vo.8)
- Programming in Python and calling OpenSCAD — [https://old.reddit.com/r/OpenPythonSCAD/comments/1heczmi/finally\\_using\\_scad\\_modules/](https://old.reddit.com/r/OpenPythonSCAD/comments/1heczmi/finally_using_scad_modules/)

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

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

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

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

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

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

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

which in turn calls the internal Python definition `\DescribeSubroutine{FOOBAR}{pFOOBAR}`

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

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

Further note that this style of definition might not have been necessary for some later modules since they are in turn calling internal modules which already use this structure.

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

### 3.1.1 Parameters and Default Values

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

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

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

## 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 # getting openscad functions into namespace
10 gcpy #https://github.com/gsohler/openscad/issues/39
11 gcpy try:
12 gcpy     from openscad import *
13 gcpy except ModuleNotFoundError as e:
14 gcpy     print("OpenSCAD module not loaded.")
15 gcpy
16 gcpy # add math functions (using radians by default, convert to degrees
   where necessary)
17 gcpy import math
18 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
- misctool:
  - \* 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 #         self.basefilename = basefilename
42 gcpy if (generatepaths == 1):
43 gcpy     self.generatepaths = True
44 gcpy if (generatepaths == 0):
45 gcpy     self.generatepaths = False
46 gcpy else:
47 gcpy     self.generatepaths = generatepaths
48 gcpy if (generategcode == 1):
49 gcpy     self.generategcode = True
50 gcpy if (generategcode == 0):
51 gcpy     self.generategcode = False
52 gcpy else:
53 gcpy     self.generategcode = generategcode
54 gcpy if (generatedxf == 1):
55 gcpy     self.generatedxf = True
56 gcpy if (generatedxf == 0):
57 gcpy     self.generatedxf = False
58 gcpy else:
59 gcpy     self.generatedxf = generatedxf
60 gcpy #     self.stockXwidth = stockXwidth
61 gcpy #     self.stockYheight = stockYheight
62 gcpy #     self.stockZthickness = stockZthickness
63 gcpy #     self.zeroheight = zeroheight
64 gcpy #     self.stockzero = stockzero
65 gcpy #     self.retractheight = retractheight
66 gcpy #     self.currenttoolnum = currenttoolnum
67 gcpy #     self.toolradius = toolradius
68 gcpy #     self.plunge = plunge
69 gcpy #     self.feed = feed
70 gcpy #     self.speed = speed

```

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

---

3.2.1 Position and Variables

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

The first such variables are for xyz position:

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

---

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

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

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

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

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

3.2.2 Initial Modules

gcodepreview  
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.

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

---

```

160 gcpy          if self.zeroheight == "Top":
161 gcpy              if self.stockzero == "Lower-Left":
162 gcpy                  self.stock = stock.translate([0,0,-self.
                        stockZthickness])
163 gcpy              if self.generategcode == True:
164 gcpy                  self.writegc("(stockMin:0.00mm,␣0.00mm,␣-",str(
                        self.stockZthickness),"mm)")
165 gcpy                  self.writegc("(stockMax:",str(self.stockXwidth)
                        ,"mm,␣",str(stockYheight),"mm,␣0.00mm)")
166 gcpy                  self.writegc("(STOCK/BLOCK,␣",str(self.
                        stockXwidth),"␣",str(self.stockYheight),"␣
                        ",str(self.stockZthickness),"␣0.00,␣0.00,␣"
                        ,str(self.stockZthickness),")")
167 gcpy              if self.stockzero == "Center-Left":
168 gcpy                  self.stock = self.stock.translate([0,-stockYheight
                        / 2,-stockZthickness])
169 gcpy              if self.generategcode == True:
170 gcpy                  self.writegc("(stockMin:0.00mm,␣-",str(self.
                        stockYheight/2),"mm,␣-",str(self.
                        stockZthickness),"mm)")
171 gcpy                  self.writegc("(stockMax:",str(self.stockXwidth)
                        ,"mm,␣",str(self.stockYheight/2),"mm,␣0.00mm
                        )")
172 gcpy                  self.writegc("(STOCK/BLOCK,␣",str(self.
                        stockXwidth),"␣",str(self.stockYheight),"␣
                        ",str(self.stockZthickness),"␣0.00,␣",str(
                        self.stockYheight/2),"␣",str(self.
                        stockZthickness),")");
173 gcpy              if self.stockzero == "Top-Left":
174 gcpy                  self.stock = self.stock.translate([0,-self.
                        stockYheight,-self.stockZthickness])
175 gcpy              if self.generategcode == True:
176 gcpy                  self.writegc("(stockMin:0.00mm,␣-",str(self.
                        stockYheight),"mm,␣-",str(self.
                        stockZthickness),"mm)")
177 gcpy                  self.writegc("(stockMax:",str(self.stockXwidth)
                        ,"mm,␣0.00mm,␣0.00mm)")
178 gcpy                  self.writegc("(STOCK/BLOCK,␣",str(self.
                        stockXwidth),"␣",str(self.stockYheight),"␣
                        ",str(self.stockZthickness),"␣0.00,␣",str(
                        self.stockYheight),"␣",str(self.
                        stockZthickness),")")
179 gcpy              if self.stockzero == "Center":
180 gcpy                  self.stock = self.stock.translate([-self.
                        stockXwidth / 2,-self.stockYheight / 2,-self.
                        stockZthickness])
181 gcpy              if self.generategcode == True:
182 gcpy                  self.writegc("(stockMin:␣-",str(self.
                        stockXwidth/2),"␣-",str(self.stockYheight
                        /2),"mm,␣-",str(self.stockZthickness),"mm)")
183 gcpy                  self.writegc("(stockMax:",str(self.stockXwidth
                        /2),"mm,␣",str(self.stockYheight/2),"mm,␣
                        0.00mm)")
184 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),")")
185 gcpy          if self.zeroheight == "Bottom":
186 gcpy              if self.stockzero == "Lower-Left":
187 gcpy                  self.stock = self.stock.translate([0,0,0])
188 gcpy              if self.generategcode == True:
189 gcpy                  self.writegc("(stockMin:0.00mm,␣0.00mm,␣0.00mm
                        )")
190 gcpy                  self.writegc("(stockMax:",str(self.stockXwidth)
                        ),"mm,␣",str(self.stockYheight),"mm,␣␣",str
                        (self.stockZthickness),"mm)")
191 gcpy                  self.writegc("(STOCK/BLOCK,␣",str(self.
                        stockXwidth),"␣",str(self.stockYheight),"
                        ␣",str(self.stockZthickness),"␣0.00,␣0.00,
                        ␣0.00)")
192 gcpy              if self.stockzero == "Center-Left":
193 gcpy                  self.stock = self.stock.translate([0,-self.
                        stockYheight / 2,0])
194 gcpy              if self.generategcode == True:

```



```
195 gcpy                self.writegc("(stockMin:0.00mm,␣-",str(self.
                        stockYheight/2),"mm,␣0.00mm)")
196 gcpy                self.writegc("(stockMax:",str(self.stockXwidth)
                        ,"mm,␣",str(self.stockYheight/2),"mm,␣-",str
                        (self.stockZthickness),"mm)")
197 gcpy                self.writegc("(STOCK/BLOCK,␣",str(self.
                        stockXwidth),"␣",str(self.stockYheight),"␣
                        ",str(self.stockZthickness),"␣0.00,␣",str(
                        self.stockYheight/2),"␣0.00mm)");
198 gcpy                if self.stockzero == "Top-Left":
199 gcpy                    self.stock = self.stock.translate([0,-self.
                        stockYheight,0])
200 gcpy                if self.generategcode == True:
201 gcpy                    self.writegc("(stockMin:0.00mm,␣-",str(self.
                        stockYheight),"mm,␣0.00mm)")
202 gcpy                self.writegc("(stockMax:",str(self.stockXwidth)
                        ,"mm,␣0.00mm,␣",str(self.stockZthickness),"
                        mm)")
203 gcpy                self.writegc("(STOCK/BLOCK,␣",str(self.
                        stockXwidth),"␣",str(self.stockYheight),"␣
                        ",str(self.stockZthickness),"␣0.00,␣",str(
                        self.stockYheight),"␣0.00)")
204 gcpy                if self.stockzero == "Center":
205 gcpy                    self.stock = self.stock.translate([-self.
                        stockXwidth / 2,-self.stockYheight / 2,0])
206 gcpy                if self.generategcode == True:
207 gcpy                    self.writegc("(stockMin:␣-",str(self.
                        stockXwidth/2),"␣-",str(self.stockYheight
                        /2),"mm,␣0.00mm)")
208 gcpy                self.writegc("(stockMax:",str(self.stockXwidth
                        /2),"mm,␣",str(self.stockYheight/2),"mm,␣",
                        str(self.stockZthickness),"mm)")
209 gcpy                self.writegc("(STOCK/BLOCK,␣",str(self.
                        stockXwidth),"␣",str(self.stockYheight),"␣
                        ",str(self.stockZthickness),"␣",str(self.
                        stockXwidth/2),"␣", str(self.stockYheight
                        /2),"␣0.00)")
210 gcpy                if self.generategcode == True:
211 gcpy                    self.writegc("G90");
212 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

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

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

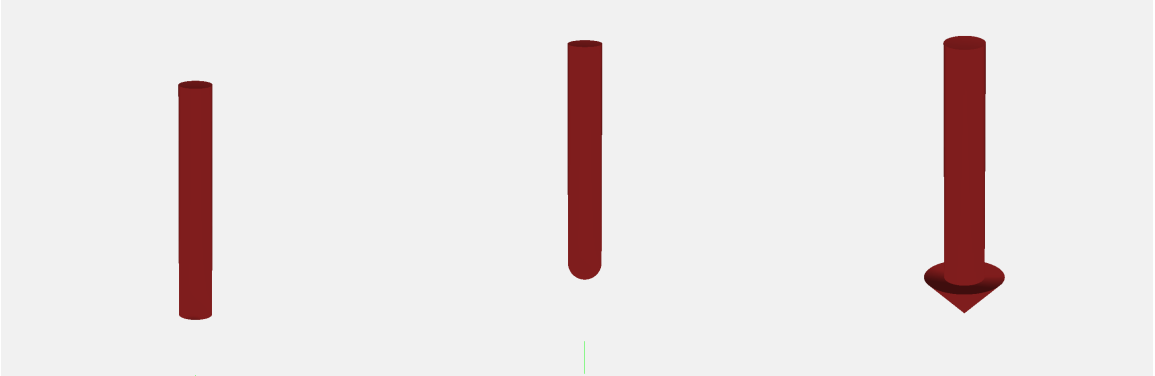
```
223 gcpy #         global Roundover_tool_num
224 gcpy #         return self.Roundover_tool_num
```

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

3.3.1 3D Shapes for Tools

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

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



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

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

endmill square      The endmill square is a simple cylinder:

```
226 gcpy         def endmill_square(self, es_diameter, es_flute_length):
227 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:

```
229 gcpy         def ballnose(self, es_diameter, es_flute_length):
230 gcpy             b = sphere(r=(es_diameter / 2))
231 gcpy             s = cylinder(r1=(es_diameter / 2), r2=(es_diameter / 2), h=
                               es_flute_length, center=False)
232 gcpy             p = union(b,s)
233 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):

```
235 gcpy         def endmill_v(self, es_v_angle, es_diameter):
236 gcpy             es_v_angle = math.radians(es_v_angle)
237 gcpy             v = cylinder(r1=0, r2=(es_diameter / 2), h=((es_diameter /
                               2) / math.tan((es_v_angle / 2))), center=False)
238 gcpy             s = cylinder(r1=(es_diameter / 2), r2=(es_diameter / 2), h=
                               ((es_diameter * 8) ), center=False)
239 gcpy             sh = s.translate([0, 0, ((es_diameter / 2) / math.tan((
                               es_v_angle / 2)))])
240 gcpy             return union(v,sh)
```

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

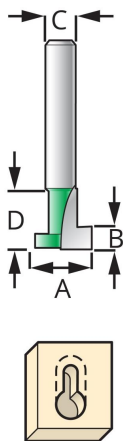
```
242 gcpy         def bowl_tool(self, radius, diameter, height):
243 gcpy             bts = cylinder(height - radius, diameter / 2, diameter / 2,
                                   center=False)
```

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

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

- Keyhole tools — intended to cut slots for retaining hardware used for picture hanging, they may be used to create slots for other purposes Note that it will be necessary to model these twice, once for the shaft, the second time for the actual keyhole cutting <https://assetssc.leevalley.com/en-gb/shop/tools/power-tool-accessories/router-bits/30113-keyhole-router-bits>
- Dovetail cutters — used for the joinery of the same name, they cut a large area at the bottom which slants up to a narrower region at a defined angle
- Lollipop cutters — normally used for 3D work, as their name suggests they are essentially a (cutting) ball on a narrow stick (the tool shaft), they are mentioned here only for 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:

```
252 gcpy      def keyhole(self, es_diameter, es_flute_length):
253 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:

```
255 gcpy      def keyhole_shaft(self, es_diameter, es_flute_length):
256 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>.

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

```

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

```
270 gcpy          def dovetail(self, dt_bottomdiameter, dt_topdiameter, dt_height
                        , dt_angle):
271 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 beginning and end points. Tooling which cannot be so represented will be implemented separately below, see paragraph 3.3.1.3.

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

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

currenttoolshape

cutline module and handle the tooling in a separate module as is currently done for roundover tooling.

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

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

3.3.2.2 Square and ball nose (including tapered ball nose)

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

3.3.2.3 Roundover (corner rounding)

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

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

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

```
277 gcpy      def toolchange(self,tool_number,speed = 10000):
278 gcpy #          global currenttoolshape
279 gcpy          self.currenttoolshape = self.endmill_square(0.001, 0.001)
280 gcpy
281 gcpy          self.settool(tool_number)
282 gcpy          if (self.generategcode == True):
283 gcpy              self.writegc("(Toolpath)")
284 gcpy              self.writegc("M05")
285 gcpy          if (tool_number == 201):
286 gcpy              self.writegc("(TOOL/MILL,6.35,0.00,0.00,0.00)")
287 gcpy              self.currenttoolshape = self.endmill_square(6.35,
288 gcpy                  19.05)
289 gcpy          elif (tool_number == 102):
290 gcpy              self.writegc("(TOOL/MILL,3.175,0.00,0.00,0.00)")
291 gcpy              self.currenttoolshape = self.endmill_square(3.175,
292 gcpy                  12.7)
293 gcpy          elif (tool_number == 112):
294 gcpy              self.writegc("(TOOL/MILL,1.5875,0.00,0.00,0.00)")
295 gcpy              self.currenttoolshape = self.endmill_square(1.5875,
296 gcpy                  6.35)
297 gcpy          elif (tool_number == 122):
298 gcpy              self.writegc("(TOOL/MILL,0.79375,0.00,0.00,0.00)")
299 gcpy              self.currenttoolshape = self.endmill_square(0.79375,
300 gcpy                  1.5875)
301 gcpy          elif (tool_number == 202):
302 gcpy              self.writegc("(TOOL/MILL,6.35,3.175,0.00,0.00)")
303 gcpy              self.currenttoolshape = self.ballnose(6.35, 19.05)
304 gcpy          elif (tool_number == 101):
305 gcpy              self.writegc("(TOOL/MILL,3.175,1.5875,0.00,0.00)")
306 gcpy              self.currenttoolshape = self.ballnose(3.175, 12.7)
307 gcpy          elif (tool_number == 111):
308 gcpy              self.writegc("(TOOL/MILL,1.5875,0.79375,0.00,0.00)")
309 gcpy              self.currenttoolshape = self.ballnose(1.5875, 6.35)
310 gcpy          elif (tool_number == 121):
311 gcpy              self.writegc("(TOOL/MILL,3.175,0.79375,0.00,0.00)")
312 gcpy              self.currenttoolshape = self.ballnose(0.79375, 1.5875)
313 gcpy          elif (tool_number == 327):
314 gcpy              self.writegc("(TOOL/MILL,0.03,0.00,13.4874,30.00)")
315 gcpy              self.currenttoolshape = self.endmill_v(60, 26.9748)
316 gcpy          elif (tool_number == 301):
317 gcpy              self.writegc("(TOOL/MILL,0.03,0.00,6.35,45.00)")
318 gcpy              self.currenttoolshape = self.endmill_v(90, 12.7)
319 gcpy          elif (tool_number == 302):
320 gcpy              self.writegc("(TOOL/MILL,0.03,0.00,10.998,30.00)")
321 gcpy              self.currenttoolshape = self.endmill_v(60, 12.7)
322 gcpy          elif (tool_number == 390):
323 gcpy              self.writegc("(TOOL/MILL,0.03,0.00,1.5875,45.00)")
324 gcpy              self.currenttoolshape = self.endmill_v(90, 3.175)
```

```
321 gcpy          elif (tool_number == 374):
322 gcpy              self.writegc("(TOOL/MILL,9.53,␣0.00,␣3.17,␣0.00)")
323 gcpy          elif (tool_number == 375):
324 gcpy              self.writegc("(TOOL/MILL,9.53,␣0.00,␣3.17,␣0.00)")
325 gcpy          elif (tool_number == 376):
326 gcpy              self.writegc("(TOOL/MILL,12.7,␣0.00,␣4.77,␣0.00)")
327 gcpy          elif (tool_number == 378):
328 gcpy              self.writegc("(TOOL/MILL,12.7,␣0.00,␣4.77,␣0.00)")
329 gcpy          elif (tool_number == 814):
330 gcpy              self.writegc("(TOOL/MILL,12.7,␣6.367,␣12.7,␣0.00)")
331 gcpy              #dt_bottomdiameter, dt_topdiameter, dt_height, dt_angle
332 gcpy              )
333 gcpy              #https://www.leevalley.com/en-us/shop/tools/power-tool-
334 gcpy              accessories/router-bits/30172-dovetail-bits?item=18
335 gcpy              J1607
336 gcpy              self.currenttoolshape = self.dovetail(12.7, 6.367,
337 gcpy                  12.7, 14)
338 gcpy          elif (tool_number == 56125):#0.508/2, 1.531
339 gcpy              self.writegc("(TOOL/CRMILL,␣0.508,␣6.35,␣3.175,␣7.9375,
340 gcpy                  ␣3.175)")
341 gcpy          elif (tool_number == 56142):#0.508/2, 2.921
342 gcpy              self.writegc("(TOOL/CRMILL,␣0.508,␣3.571875,␣1.5875,␣
343 gcpy                  5.55625,␣1.5875)")
344 gcpy          elif (tool_number == 312):#1.524/2, 3.175
345 gcpy              self.writegc("(TOOL/CRMILL, Diameter1, Diameter2,
346 gcpy                  Radius, Height, Length)")
347 gcpy          elif (tool_number == 1570):#0.507/2, 4.509
348 gcpy              self.writegc("(TOOL/CRMILL,␣0.17018,␣9.525,␣4.7625,␣
349 gcpy                  12.7,␣4.7625)")
350 gcpy          #https://www.amanatool.com/45982-carbide-tipped-bowl-tray-1-4-
351 gcpy          radius-x-3-4-dia-x-5-8-x-1-4-inch-shank.html
352 gcpy          elif (tool_number == 45982):#0.507/2, 4.509
353 gcpy              self.writegc("(TOOL/MILL,␣15.875,␣6.35,␣19.05,␣0.00)")
354 gcpy              self.currenttoolshape = self.bowl_tool(6.35, 19.05,
355 gcpy                  15.875)
```

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

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

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

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

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

For example:

```
toolchange(small_square_tool_num,speed);
```

(the assumption is that all speed rates in a file will be the same, so as to account for the most frequent use case of a trim router with speed controlled by a dial setting and feed rates/ratios being calculated to provide the correct chipload at that setting.)

### 3.3.3 tooldiameter

It will also be necessary to be able to provide the diameter of the current tool. Arguably, this would be much easier using an object-oriented programming style/dot notation.

One aspect of tool parameters which will need to be supported is shapes which create different profiles based on how deeply the tool is cutting into the surface of the material at a given point. To accommodate this, it will be necessary to either track the thickness of uncut material at any given point, or, to specify the depth of cut as a parameter which is what the initial version will implement.

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

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

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

---

```

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

---

```

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

```

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

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

### 3.3.4 Feeds and Speeds

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

The tools which need to be calculated thus are those in addition to the large\_square tool:

- small\_square\_ratio
- small\_ball\_ratio
- large\_ball\_ratio
- small\_V\_ratio
- large\_V\_ratio
- KH\_ratio
- DT\_ratio

## 3.4 Movement and Cutting

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.

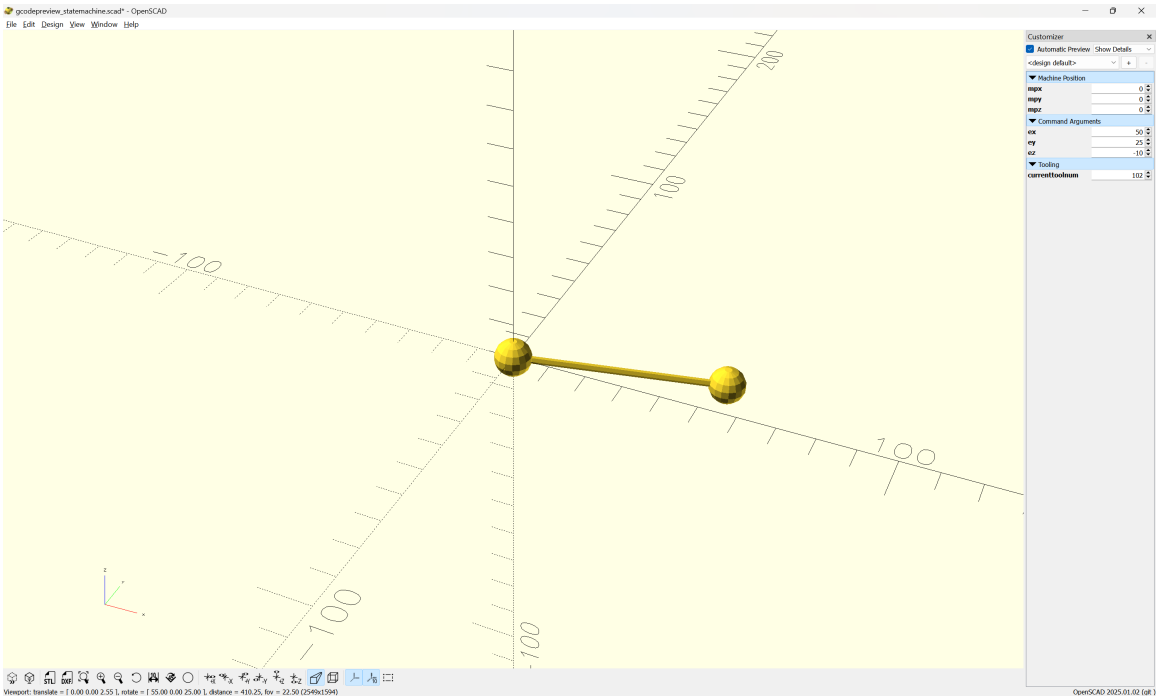
In order to manage the various options when cutting it will be necessary to have a command where the actual cut is made, passing in the shape used for the cut as a parameter. Since the 3D aspect of rapid and cut operations are fundamentally the same, the command rcs which returns the hull of the begin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

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

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

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





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

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

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

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

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

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

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

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

```

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

---

```

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

```

```
508 gcpy          self.setxpos(ex)
509 gcpy          self.setypos(ey)
510 gcpy          self.setzpos(ez)
511 gcpy #          if self.generatepaths == True:
512 gcpy #              self.toolpaths = union([self.toolpaths, toolpath])
513 gcpy #          else:
514 gcpy          if self.generatepaths == False:
515 gcpy              return toolpath
516 gcpy          else:
517 gcpy              return cube([0.001,0.001,0.001])
518 gcpy
519 gcpy          def cutlinedxfgc(self,ex, ey, ez):
520 gcpy              self.dxfline(self.currenttoolnumber(), self.xpos(), self.
                    ypos(), ex, ey)
521 gcpy              self.writegc("G01_X", str(ex), "Y", str(ey), "Z", str(ez)
                    )
522 gcpy #          if self.generatepaths == False:
523 gcpy          return self.cutline(ex, ey, ez)
524 gcpy
525 gcpy          def cutroundovertool(self, bx, by, bz, ex, ey, ez,
                    tool_radius_tip, tool_radius_width, stepsizeroundover = 1):
526 gcpy #              n = 90 + fn*3
527 gcpy #              print("Tool dimensions", tool_radius_tip,
                    tool_radius_width, "begin ",bx, by, bz,"end ", ex, ey, ez)
528 gcpy              step = 4 #360/n
529 gcpy              shaft = cylinder(step,tool_radius_tip,tool_radius_tip)
530 gcpy              toolpath = hull(shaft.translate([bx,by,bz]), shaft.
                    translate([ex,ey,ez]))
531 gcpy              shaft = cylinder(tool_radius_width*2,tool_radius_tip+
                    tool_radius_width,tool_radius_tip+tool_radius_width)
532 gcpy              toolpath = toolpath.union(hull(shaft.translate([bx,by,bz+
                    tool_radius_width]), shaft.translate([ex,ey,ez+
                    tool_radius_width])))
533 gcpy              for i in range(1, 90, stepsizeroundover):
534 gcpy                  angle = i
535 gcpy                  dx = tool_radius_width*math.cos(math.radians(angle))
536 gcpy                  dxx = tool_radius_width*math.cos(math.radians(angle+1))
537 gcpy                  dzz = tool_radius_width*math.sin(math.radians(angle))
538 gcpy                  dz = tool_radius_width*math.sin(math.radians(angle+1))
539 gcpy                  dh = abs(dzz-dz)+0.0001
540 gcpy                  slice = cylinder(dh,tool_radius_tip+tool_radius_width-
                    dx,tool_radius_tip+tool_radius_width-dxx)
541 gcpy                  toolpath = toolpath.union(hull(slice.translate([bx,by,
                    bz+dz]), slice.translate([ex,ey,ez+dz])))
542 gcpy              if self.generatepaths == True:
543 gcpy                  self.toolpaths = self.toolpaths.union(toolpath)
544 gcpy              else:
545 gcpy                  return toolpath
546 gcpy
547 gcpy          def cutZgcfeed(self, ez, feed):
548 gcpy              self.writegc("G01_Z", str(ez), "F",str(feed))
549 gcpy #              if self.generatepaths == False:
550 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
- cutcircleCCdxf
- cutcircleCWdxf

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

- G-code — G2 (clockwise) and G3 (counter-clockwise) arcs may be specified, and since the endpoint is the positional requirement, it is most likely best to use the offset to the center (I and J), rather than the radius parameter (K) G2/3 ...
- DXF — dxfarc(xcenter, ycenter, radius, anglebegin, endangle, tn)
- approximation of arc using lines (OpenSCAD) in both clock-wise and counter-clock-wise directions

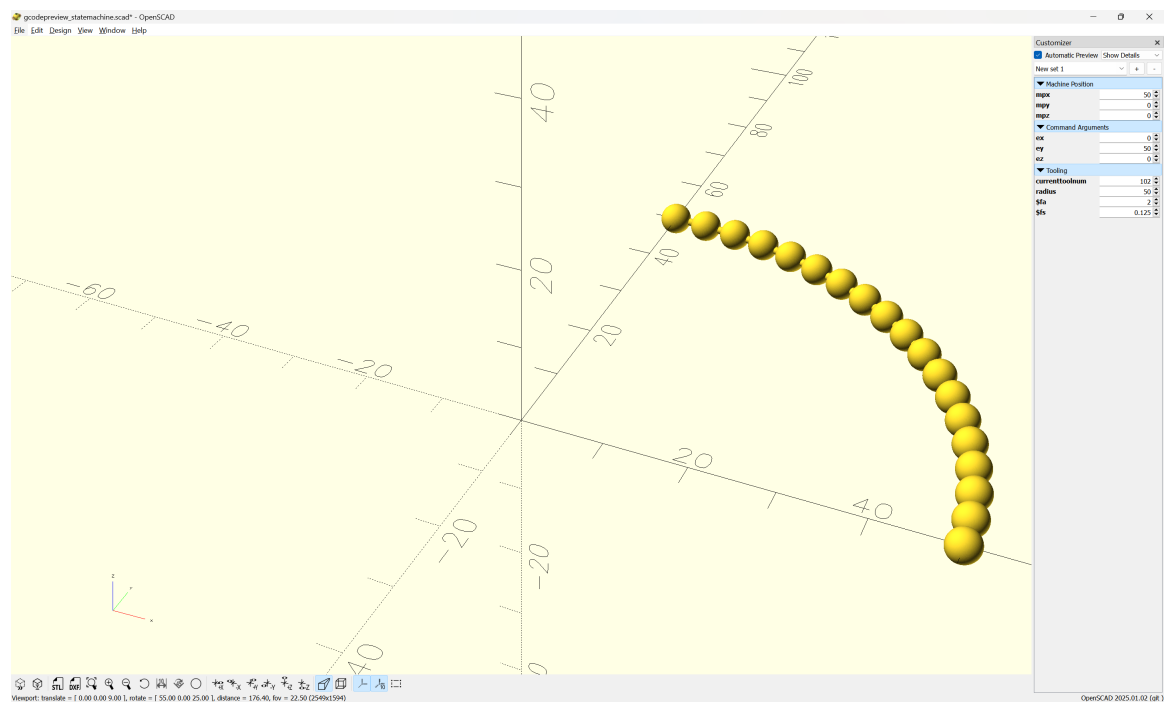
Cutting the quadrant arcs greatly simplifies the calculation and interface for the modules. A full set of 8 will be necessary, then circles will have a pair of modules (one for each cut direction) made for them.

Parameters which will need to be passed in are:

- ex — note that the matching origins (bx, by, bz) as well as the (current) toolnumber are accessed using the appropriate commands
- ey
- ez — allowing a different Z position will make possible threading and similar helical 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 loop: cutarcCW (clockwise) or cutarcCC (counterclockwise) to handle the drawing and processing of the cutline() toolpaths as short line segments which additionally affords a single point of control for adding additional features such as allowing the depth to vary as one cuts along an arc (the line version is used rather than shape so as to capture the changing machine positions with each step through the loop). Note that the definition matches the DXF definition of defining the center position with a matching radius, but it will be necessary to move the tool to the actual origin, and to calculate the end position when writing out a G2/G3 arc.

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



The code for which makes this obvious:

```

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

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

/* [Tooling] */
currenttoolnum = 102;

machine_extents();

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

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

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

```

```

}
}

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

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.)

---

```

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

```

---

```

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

```

---

Matching OpenSCAD modules are easily made:

---

```

86 gpcscad module cutarcCC(barC, earC, xcenter, ycenter, radius, tpzreldim){
87 gpcscad     gcp.cutarcCC(barC, earC, xcenter, ycenter, radius, tpzreldim);
88 gpcscad }
89 gpcscad
90 gpcscad module cutarcCW(barC, earC, xcenter, ycenter, radius, tpzreldim){
91 gpcscad     gcp.cutarcCW(barC, earC, xcenter, ycenter, radius, tpzreldim);
92 gpcscad }

```

---

### 3.4.3 Cutting shapes and expansion

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

It is most expedient to test out new features in a new/separate file insofar as the file structures will allow (tool definitions for example will need to be consolidated in 3.3.2) 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 sector<sup>1</sup>
  - two convex, one concave arc
- 4
  - rectangle (including square) — `dxfrectangle`, `dxfrectangleround`
  - parallelogram
  - rhombus
  - trapezoid/trapezium
  - kite
  - ring/annulus segment (straight line, concave arc, straight line, convex arc)
  - astroid (four concave arcs)
  - salinon (four semicircles)
  - three straight lines and one concave arc

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

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

- two straight edges, one concave arc—oddly, an asymmetric form (hyperbolic sector) has a name, but not the symmetrical—while the colloquial/prosaic arrowhead was considered, it was rejected as being better applied to the shape below. (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:

---

<sup>1</sup>[en.wikipedia.org/wiki/Hyperbolic\\_sector](https://en.wikipedia.org/wiki/Hyperbolic_sector) and [www.reddit.com/r/Geometry/comments/bkzbzgh/is\\_there\\_a\\_name\\_for\\_a\\_3\\_pointed\\_figure\\_with\\_two](https://www.reddit.com/r/Geometry/comments/bkzbzgh/is_there_a_name_for_a_3_pointed_figure_with_two)



- 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 dove-tail 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:

```
597 gcpy      def dxfcircle(self, tool_num, xcenter, ycenter, radius):
598 gcpy          self.dxfarc(tool_num, xcenter, ycenter, radius, 0, 90)
599 gcpy          self.dxfarc(tool_num, xcenter, ycenter, radius, 90, 180)
600 gcpy          self.dxfarc(tool_num, xcenter, ycenter, radius, 180, 270)
601 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:

```
603 gcpy      def dxfrectangle(self, tool_num, xorigin, yorigin, xwidth,
604 gcpy          yheight, corners = "Square", radius = 6):
605 gcpy          if corners == "Square":
606 gcpy              self.dxfline(tool_num, xorigin, yorigin, xorigin +
607 gcpy                  xwidth, yorigin)
608 gcpy              self.dxfline(tool_num, xorigin + xwidth, yorigin,
609 gcpy                  xorigin + xwidth, yorigin + yheight)
610 gcpy              self.dxfline(tool_num, xorigin + xwidth, yorigin +
611 gcpy                  yheight, xorigin, yorigin + yheight)
612 gcpy              self.dxfline(tool_num, xorigin, yorigin + yheight,
613 gcpy                  xorigin, yorigin)
614 gcpy          elif corners == "Fillet":
615 gcpy              self.dxfrectangleround(tool_num, xorigin, yorigin,
616 gcpy                  xwidth, yheight, radius)
617 gcpy          elif corners == "Chamfer":
618 gcpy              self.dxfrectanglechamfer(tool_num, xorigin, yorigin,
619 gcpy                  xwidth, yheight, radius)
620 gcpy          elif corners == "Flipped_Fillet":
621 gcpy              self.dxfrectangleflippedfillet(tool_num, xorigin,
622 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.

```
616 gcpy      def dxfrectangleround(self, tool_num, xorigin, yorigin, xwidth,
617 gcpy          yheight, radius):
618 gcpy          self.dxfarc(tool_num, xorigin + xwidth - radius, yorigin +
619 gcpy              yheight - radius, radius, 0, 90)
620 gcpy          self.dxfarc(tool_num, xorigin + radius, yorigin + yheight -
621 gcpy              radius, radius, 90, 180)
622 gcpy          self.dxfarc(tool_num, xorigin + radius, yorigin + radius,
623 gcpy              radius, 180, 270)
624 gcpy          self.dxfarc(tool_num, xorigin + xwidth - radius, yorigin +
625 gcpy              radius, radius, 270, 360)
626 gcpy          self.dxfline(tool_num, xorigin + radius, yorigin, xorigin +
627 gcpy              xwidth - radius, yorigin)
628 gcpy          self.dxfline(tool_num, xorigin + xwidth, yorigin + radius,
629 gcpy              xorigin + xwidth, yorigin + yheight - radius)
630 gcpy          self.dxfline(tool_num, xorigin + xwidth - radius, yorigin +
631 gcpy              yheight, xorigin + radius, yorigin + yheight)
632 gcpy          self.dxfline(tool_num, xorigin, yorigin + yheight - radius,
633 gcpy              xorigin, yorigin + radius)
```

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

```
627 gcpy      def dxfrectanglechamfer(self, tool_num, xorigin, yorigin,
628 gcpy          xwidth, yheight, radius):
629 gcpy          self.dxfline(tool_num, xorigin + radius, yorigin, xorigin,
630 gcpy              yorigin + radius)
```

```
629 gcpy          self.dxfline(tool_num, xorigin, yorigin + yheight - radius,
                             xorigin + radius, yorigin + yheight)
630 gcpy          self.dxfline(tool_num, xorigin + xwidth - radius, yorigin +
                             yheight, xorigin + xwidth, yorigin + yheight - radius)
631 gcpy          self.dxfline(tool_num, xorigin + xwidth - radius, yorigin,
                             xorigin + xwidth, yorigin + radius)

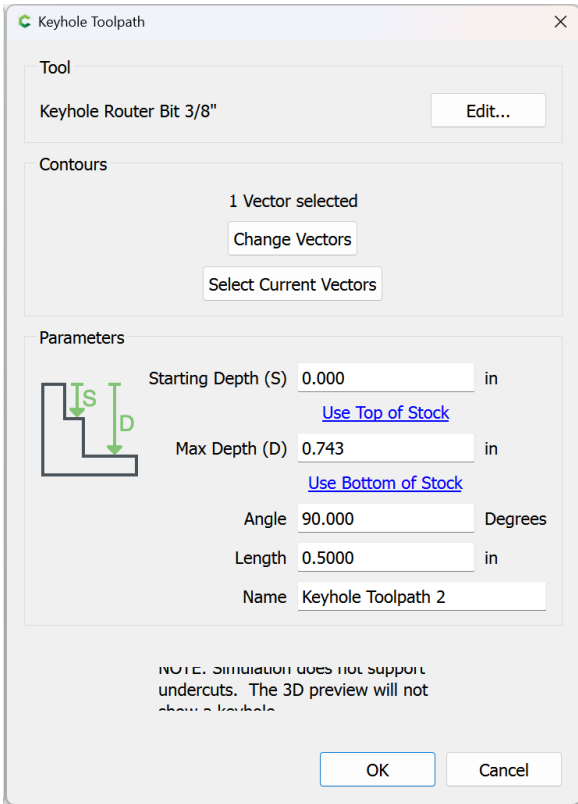
632 gcpy
633 gcpy          self.dxfline(tool_num, xorigin + radius, yorigin, xorigin +
                             xwidth - radius, yorigin)
634 gcpy          self.dxfline(tool_num, xorigin + xwidth, yorigin + radius,
                             xorigin + xwidth, yorigin + yheight - radius)
635 gcpy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

Flipped Fillet:

```
638 gcpy          def dxfrectangleflippedfillet(self, tool_num, xorigin, yorigin,
                             xwidth, yheight, radius):
639 gcpy          self.dxfarc(tool_num, xorigin, yorigin, radius, 0, 90)
640 gcpy          self.dxfarc(tool_num, xorigin + xwidth, yorigin, radius,
                             90, 180)
641 gcpy          self.dxfarc(tool_num, xorigin + xwidth, yorigin + yheight,
                             radius, 180, 270)
642 gcpy          self.dxfarc(tool_num, xorigin, yorigin + yheight, radius,
                             270, 360)

643 gcpy
644 gcpy          self.dxfline(tool_num, xorigin + radius, yorigin, xorigin +
                             xwidth - radius, yorigin)
645 gcpy          self.dxfline(tool_num, xorigin + xwidth, yorigin + radius,
                             xorigin + xwidth, yorigin + yheight - radius)
646 gcpy          self.dxfline(tool_num, xorigin + xwidth - radius, yorigin +
                             yheight, xorigin + radius, yorigin + yheight)
647 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:

```
649 gcpy      def cutkeyholegcdxf(self, kh_tool_num, kh_start_depth,
650 gcpy          kh_max_depth, kht_direction, kh_distance):
651 gcpy          toolpath = self.cutKHgcdxf(kh_tool_num, kh_start_depth,
652 gcpy              kh_max_depth, 90, kh_distance)
653 gcpy          elif (kht_direction == "S"):
654 gcpy              toolpath = self.cutKHgcdxf(kh_tool_num, kh_start_depth,
655 gcpy                  kh_max_depth, 270, kh_distance)
656 gcpy          elif (kht_direction == "E"):
657 gcpy              toolpath = self.cutKHgcdxf(kh_tool_num, kh_start_depth,
658 gcpy                  kh_max_depth, 0, kh_distance)
659 gcpy          elif (kht_direction == "W"):
660 gcpy              toolpath = self.cutKHgcdxf(kh_tool_num, kh_start_depth,
661 gcpy                  kh_max_depth, 180, kh_distance)
662 gcpy          if self.generatepaths == True:
663 gcpy              self.toolpaths = union([self.toolpaths, toolpath])
664 gcpy              return toolpath
665 gcpy          else:
666 gcpy              return cube([0.01,0.01,0.01])

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

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

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

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

---

```

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

```

---

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

---

```

685 gcpy #pre-calculate needed values
686 gcpy      r = self.tool_radius(kh_tool_num, 7)
687 gcpy #      print(r)
688 gcpy      rt = self.tool_radius(kh_tool_num, 1)
689 gcpy #      print(rt)
690 gcpy      ro = math.sqrt((self.tool_radius(kh_tool_num, 1))**2-(self.
                tool_radius(kh_tool_num, 7))**2)
691 gcpy #      print(ro)
692 gcpy      angle = math.degrees(math.acos(ro/rt))
693 gcpy #Outlines of entry hole and slot
694 gcpy      if (kh_angle == 0):
695 gcpy #Lower left of entry hole
696 gcpy          self.dxfarc(kh_tool_num, self.xpos(),self.ypos(),self.
                tool_radius(kh_tool_num, 1),180,270)
697 gcpy #Upper left of entry hole
698 gcpy          self.dxfarc(kh_tool_num, self.xpos(),self.ypos(),self.
                tool_radius(kh_tool_num, 1),90,180)
699 gcpy #Upper right of entry hole
700 gcpy #      self.dxfarc(kh_tool_num, self.xpos(), self.ypos(), rt,
                41.810, 90)
701 gcpy          self.dxfarc(kh_tool_num, self.xpos(), self.ypos(), rt,
                angle, 90)
702 gcpy #Lower right of entry hole
703 gcpy          self.dxfarc(kh_tool_num, self.xpos(), self.ypos(), rt,
                270, 360-angle)
704 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))))
705 gcpy #Actual line of cut
706 gcpy #      self.dxfline(kh_tool_num, self.xpos(),self.ypos(),self
                .xpos()+kh_distance,self.ypos())
707 gcpy #upper right of end of slot (kh_max_depth+4.36))/2
708 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)
709 gcpy #lower right of end of slot
710 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)
711 gcpy #upper right slot
712 gcpy          self.dxfline(kh_tool_num, self.xpos()+ro, self.ypos()-(
                self.tool_diameter(kh_tool_num,7)/2), self.xpos()+
                kh_distance, self.ypos()-(self.tool_diameter(
                kh_tool_num,7)/2))
713 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)
714 gcpy #end position at top of slot
715 gcpy #lower right slot
716 gcpy          self.dxfline(kh_tool_num, self.xpos()+ro, self.ypos()+(
                self.tool_diameter(kh_tool_num,7)/2), self.xpos()+
                kh_distance, self.ypos()+(self.tool_diameter(
                kh_tool_num,7)/2))
717 gcpy #      dxline(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)
718 gcpy #end position at top of slot
719 gcpy #    hull(){
720 gcpy #        translate([xpos(), ypos(), zpos()]){
721 gcpy #            keyhole_shaft(6.35, 9.525);
722 gcpy #        }
723 gcpy #        translate([xpos(), ypos(), zpos()-kh_max_depth]){
724 gcpy #            keyhole_shaft(6.35, 9.525);
725 gcpy #        }
726 gcpy #    }
727 gcpy #    hull(){
728 gcpy #        translate([xpos(), ypos(), zpos()-kh_max_depth]){
729 gcpy #            keyhole_shaft(6.35, 9.525);
730 gcpy #        }
731 gcpy #        translate([xpos()+kh_distance, ypos(), zpos()-kh_max_depth])
732 gcpy #    {
733 gcpy #        keyhole_shaft(6.35, 9.525);
734 gcpy #    }
735 gcpy #    cutwithfeed(getxpos(),getypos(),-kh_max_depth,feed);
736 gcpy #    cutwithfeed(getxpos()+kh_distance,getypos(),-kh_max_depth,feed
737 gcpy #    );
738 gcpy #    setxpos(getxpos()-kh_distance);
739 gcpy # } else if (kh_angle > 0 && kh_angle < 90) {
740 gcpy #    dxfarc(getxpos(),getypos(),tool_diameter(KH_tool_num, (
741 gcpy #        kh_max_depth))/2,90+kh_angle,180+kh_angle, KH_tool_num);
742 gcpy #    dxline(getxpos(),getypos(),tool_diameter(KH_tool_num, (
743 gcpy #        kh_max_depth))/2,kh_angle+asin((tool_diameter(KH_tool_num, (
744 gcpy #            kh_max_depth+4.36))/2)/(tool_diameter(KH_tool_num, (kh_max_depth
745 gcpy #                ))/2)),90+kh_angle, KH_tool_num);
746 gcpy #    dxline(getxpos(),getypos(),tool_diameter(KH_tool_num, (
747 gcpy #        kh_max_depth))/2,270+kh_angle,360+kh_angle-asin((tool_diameter(
748 gcpy #            KH_tool_num, (kh_max_depth+4.36))/2)/(tool_diameter(KH_tool_num,
749 gcpy #                (kh_max_depth))/2)), KH_tool_num);
750 gcpy #    dxline(getxpos()+((kh_distance*cos(kh_angle))),
751 gcpy #        getypos()+((kh_distance*sin(kh_angle))),tool_diameter(KH_tool_num,
752 gcpy #            (kh_max_depth+4.36))/2,0+kh_angle,90+kh_angle, KH_tool_num);
753 gcpy #    dxline(getxpos()+((kh_distance*cos(kh_angle))),getypos()+((
754 gcpy #        kh_distance*sin(kh_angle))),tool_diameter(KH_tool_num, (
755 gcpy #            kh_max_depth+4.36))/2,270+kh_angle,360+kh_angle, KH_tool_num);
756 gcpy #    dxline( getxpos()+tool_diameter(KH_tool_num, (kh_max_depth))/2*
757 gcpy #        cos(kh_angle+asin((tool_diameter(KH_tool_num, (kh_max_depth
758 gcpy #            +4.36))/2)/(tool_diameter(KH_tool_num, (kh_max_depth))/2))),
759 gcpy #        getypos()+tool_diameter(KH_tool_num, (kh_max_depth))/2*sin(
760 gcpy #            kh_angle+asin((tool_diameter(KH_tool_num, (kh_max_depth+4.36))
761 gcpy #                /2)/(tool_diameter(KH_tool_num, (kh_max_depth))/2))),
762 gcpy #        getxpos()+((kh_distance*cos(kh_angle)))-((tool_diameter(KH_tool_num
763 gcpy #            , (kh_max_depth+4.36))/2)*sin(kh_angle)),
764 gcpy #        getypos()+((kh_distance*sin(kh_angle)))+((tool_diameter(KH_tool_num
765 gcpy #            , (kh_max_depth+4.36))/2)*cos(kh_angle)), KH_tool_num);
766 gcpy #    //echo("a",tool_diameter(KH_tool_num, (kh_max_depth+4.36))/2);
767 gcpy #    //echo("c",tool_diameter(KH_tool_num, (kh_max_depth))/2);
768 gcpy #    echo("Angle",asin((tool_diameter(KH_tool_num, (kh_max_depth+4.36)
769 gcpy #        )/2)/(tool_diameter(KH_tool_num, (kh_max_depth))/2)));
770 gcpy #    //echo(kh_angle);
771 gcpy #    cutwithfeed(getxpos()+((kh_distance*cos(kh_angle))),getypos()+((
772 gcpy #        kh_distance*sin(kh_angle))),-kh_max_depth,feed);
773 gcpy #    toolpath = toolpath.union(self.cutline(self.xpos()+
774 gcpy #        kh_distance, self.ypos(), -kh_max_depth))
775 gcpy #    elif (kh_angle == 90):
776 gcpy #Lower left of entry hole
777 gcpy #        self.dxfarc(kh_tool_num, oXpos,oYpos,self.tool_radius(
778 gcpy #            kh_tool_num, 1),180,270)
779 gcpy #Lower right of entry hole
780 gcpy #        self.dxfarc(kh_tool_num, oXpos,oYpos,self.tool_radius(
781 gcpy #            kh_tool_num, 1),270,360)
782 gcpy #left slot
783 gcpy #        self.dxfline(kh_tool_num, oXpos-r, oYpos+ro, oXpos-r,
784 gcpy #            oYpos+kh_distance)
785 gcpy #right slot
786 gcpy #        self.dxfline(kh_tool_num, oXpos+r, oYpos+ro, oXpos+r,
787 gcpy #            oYpos+kh_distance)
788 gcpy #upper left of end of slot

```

```

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

```

```

823 gcpy #    dxfarc(getxpos(),getypos(),9.525/2,0,acos(tool_diameter(
KH_tool_num, 5)/tool_diameter(KH_tool_num, 1)), KH_tool_num);
824 gcpy #    //Upper left of entry hole
825 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)
;
826 gcpy #    //Actual line of cut
827 gcpy #    dxfline(getxpos(),getypos(),getxpos(),getypos()+kh_distance);
828 gcpy #    //upper right of slot
829 gcpy #    dxfarc(getxpos(),getypos()+kh_distance,tool_diameter(
KH_tool_num, (kh_max_depth+4.36))/2,0,90, KH_tool_num);
830 gcpy #    //upper left of slot
831 gcpy #    dxfarc(getxpos(),getypos()+kh_distance,tool_diameter(
KH_tool_num, (kh_max_depth+6.35))/2,90,180, KH_tool_num);
832 gcpy #    //right of slot
833 gcpy #    dxfline(
834 gcpy #        getxpos()+tool_diameter(KH_tool_num, (kh_max_depth))/2,
835 gcpy #        getypos()+(sqrt((tool_diameter(KH_tool_num,1)^2)-(
tool_diameter(KH_tool_num,5)^2))/2),//( (kh_max_depth-6.34))/2)
^2-(tool_diameter(KH_tool_num, (kh_max_depth-6.34))/2)^2,
        getxpos()+tool_diameter(KH_tool_num, (kh_max_depth))/2,
836 gcpy #    //end position at top of slot
837 gcpy #        getypos()+kh_distance,
838 gcpy #        KH_tool_num);
839 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);
841 gcpy #    hull(){
842 gcpy #        translate([xpos(), ypos(), zpos()]){
843 gcpy #            keyhole_shaft(6.35, 9.525);
844 gcpy #        }
845 gcpy #        translate([xpos(), ypos(), zpos()-kh_max_depth]){
846 gcpy #            keyhole_shaft(6.35, 9.525);
847 gcpy #        }
848 gcpy #    }
849 gcpy #    hull(){
850 gcpy #        translate([xpos(), ypos(), zpos()-kh_max_depth]){
851 gcpy #            keyhole_shaft(6.35, 9.525);
852 gcpy #        }
853 gcpy #        translate([xpos(), ypos()+kh_distance, zpos()-kh_max_depth])
{
854 gcpy #            keyhole_shaft(6.35, 9.525);
855 gcpy #        }
856 gcpy #    }
857 gcpy #    cutwithfeed(getxpos(),getypos(),-kh_max_depth,feed);
858 gcpy #    cutwithfeed(getxpos(),getypos()+kh_distance,-kh_max_depth,feed
);
859 gcpy #    setypos(getypos()-kh_distance);
860 gcpy # } else if (kh_angle == 180) {
861 gcpy #    //Lower right of entry hole
862 gcpy #    dxfarc(getxpos(),getypos(),9.525/2,270,360, KH_tool_num);
863 gcpy #    //Upper right of entry hole
864 gcpy #    dxfarc(getxpos(),getypos(),9.525/2,0,90, KH_tool_num);
865 gcpy #    //Upper left of entry hole
866 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);
867 gcpy #    //Lower left of entry hole
868 gcpy #    dxfarc(getxpos(),getypos(),9.525/2, 270-acos(tool_diameter(
KH_tool_num, 5)/tool_diameter(KH_tool_num, 1)), 270, KH_tool_num
);
869 gcpy #    //upper left of slot
870 gcpy #    dxfarc(getxpos()-kh_distance,getypos(),tool_diameter(
KH_tool_num, (kh_max_depth+6.35))/2,90,180, KH_tool_num);
871 gcpy #    //lower left of slot
872 gcpy #    dxfarc(getxpos()-kh_distance,getypos(),tool_diameter(
KH_tool_num, (kh_max_depth+6.35))/2,180,270, KH_tool_num);
873 gcpy #    //Actual line of cut
874 gcpy #    dxfline(getxpos(),getypos(),getxpos()-kh_distance,getypos());
875 gcpy #    //upper left slot
876 gcpy #    dxfline(
877 gcpy #        getxpos()-(sqrt((tool_diameter(KH_tool_num,1)^2)-(
tool_diameter(KH_tool_num,5)^2))/2),
878 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,
        getxpos()-kh_distance,
879 gcpy #

```

```

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

```



```

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

```

---

### 3.4.4 Difference of Stock, Rapids, and Toolpaths

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

```

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

```

---

```

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

```

---

## 3.5 Output files

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

3.5.1 G-code Overview

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

Command/Module	G-code
opengcodefile(s)(...); setupstock(...)	(export.nc) (stockMin: -109.5, -75mm, -8.35mm) (stockMax:109.5mm, 75mm, 0.00mm) (STOCK/BLOCK, 219, 150, 8.35, 109.5, 75, 8.35) G90 G21
movetosafez()	(Move to safe Z to avoid workholding) G53G0Z-5.000
toolchange(...);	(TOOL/MILL,3.17, 0.00, 0.00, 0.00) M6T102 M03S16000
cutoneaxis_setfeed(...);	(PREPOSITION FOR RAPID PLUNGE) G0X0Y0 Z0.25 G1Z0F100 G1 X109.5 Y75 Z-8.35F400 Z9
cutwithfeed(...);	
closegcodefile();	M05 M02

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

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

```

SECTION
2
ENTITIES
0
LWPOLYLINE
90
2
70
0
43
0
10
-31.375
20
-34.9152
10
-31.375
20
-18.75
0
ARC
10
-54.75
20
-37.5
40
4
50
0
51
90
0
ENDSEC
0
EOF

```

### 3.5.3 Python and OpenSCAD File Handling

The class `gcodepreview` will need additional commands for opening files. The original implementation in RapSCAD used a command `writeln` — fortunately, this command is easily re-created in Python, though it is made as a separate file for each sort of file which may be opened. Note that the `dxf` commands will be wrapped up with `if/elif` blocks which will write to additional file(s) based on tool number as set up above.

---

```

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

```

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

opendxffile

For writing to files it will be necessary to have commands for opening the files opengcodefile and opendxffile and setting the associated defaults. There is a separate function for each type of file, and for DXFs, there are multiple file instances, one for each combination of different type and size of tool which it is expected a project will work with. Each such file will be suffixed with the tool number.

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

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

opengcodefile

With matching OpenSCAD commands: opengcodefile for OpenSCAD:

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

and Python:

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

```

1060 gcpy                self.gcodefilename = basefilename + ".nc"
1061 gcpy                self.gc = open(self.gcodefilename, "w")
1062 gcpy
1063 gcpy                def opendxfile(self, basefilename = "export"):
1064 gcpy                    self.basefilename = basefilename
1065 gcpy                    global generatedxfs
1066 gcpy                    global dxfclosed
1067 gcpy                    self.dxfclosed = False
1068 gcpy                    if self.generatedxfs == True:
1069 gcpy                        self.generatedxfs = False
1070 gcpy                        self.dxffilename = basefilename + ".dxf"
1071 gcpy                        self.dxf = open(self.dxffilename, "w")
1072 gcpy                        self.dxfpreamble(-1)
1073 gcpy
1074 gcpy                def opendxfiles(self, basefilename = "export",
1075 gcpy                            large_square_tool_num = 0,
1076 gcpy                            small_square_tool_num = 0,
1077 gcpy                            large_ball_tool_num = 0,
1078 gcpy                            small_ball_tool_num = 0,
1079 gcpy                            large_V_tool_num = 0,
1080 gcpy                            small_V_tool_num = 0,
1081 gcpy                            DT_tool_num = 0,
1082 gcpy                            KH_tool_num = 0,
1083 gcpy                            Roundover_tool_num = 0,
1084 gcpy                            MISC_tool_num = 0):
1085 gcpy                    global generatedxfs
1086 gcpy                    self.basefilename = basefilename
1087 gcpy                    self.generatedxfs = True
1088 gcpy                    self.large_square_tool_num = large_square_tool_num
1089 gcpy                    self.small_square_tool_num = small_square_tool_num
1090 gcpy                    self.large_ball_tool_num = large_ball_tool_num
1091 gcpy                    self.small_ball_tool_num = small_ball_tool_num
1092 gcpy                    self.large_V_tool_num = large_V_tool_num
1093 gcpy                    self.small_V_tool_num = small_V_tool_num
1094 gcpy                    self.DT_tool_num = DT_tool_num
1095 gcpy                    self.KH_tool_num = KH_tool_num
1096 gcpy                    self.Roundover_tool_num = Roundover_tool_num
1097 gcpy                    self.MISC_tool_num = MISC_tool_num
1098 gcpy                    if self.generatedxfs == True:
1099 gcpy                        if (large_square_tool_num > 0):
1100 gcpy                            self.dxfllsqfilename = basefilename + str(
1101 gcpy                                large_square_tool_num) + ".dxf"
1102 gcpy                            print("Opening ", str(self.dxfllsqfilename))
1103 gcpy                            self.dxfllsq = open(self.dxfllsqfilename, "w")
1104 gcpy                        if (small_square_tool_num > 0):
1105 gcpy                            print("Opening small square")
1106 gcpy                            self.dxfllsqfilename = basefilename + str(
1107 gcpy                                small_square_tool_num) + ".dxf"
1108 gcpy                            self.dxfllsq = open(self.dxfllsqfilename, "w")
1109 gcpy                        if (large_ball_tool_num > 0):
1110 gcpy                            print("Opening large ball")
1111 gcpy                            self.dxfllblfilename = basefilename + str(
1112 gcpy                                large_ball_tool_num) + ".dxf"
1113 gcpy                            self.dxfllbl = open(self.dxfllblfilename, "w")
1114 gcpy                        if (small_ball_tool_num > 0):
1115 gcpy                            print("Opening small ball")
1116 gcpy                            self.dxfllblfilename = basefilename + str(
1117 gcpy                                small_ball_tool_num) + ".dxf"
1118 gcpy                            self.dxfllbl = open(self.dxfllblfilename, "w")
1119 gcpy                        if (large_V_tool_num > 0):
1120 gcpy                            print("Opening large V")
1121 gcpy                            self.dxfllVfilename = basefilename + str(
1122 gcpy                                large_V_tool_num) + ".dxf"
1123 gcpy                            self.dxfllV = open(self.dxfllVfilename, "w")
1124 gcpy                        if (small_V_tool_num > 0):
1125 gcpy                            print("Opening small V")
1126 gcpy                            self.dxfllmVfilename = basefilename + str(
1127 gcpy                                small_V_tool_num) + ".dxf"
1128 gcpy                            self.dxfllmV = open(self.dxfllmVfilename, "w")
1129 gcpy                        if (DT_tool_num > 0):
1130 gcpy                            print("Opening DT")
1131 gcpy                            self.dxfllDTfilename = basefilename + str(DT_tool_num
1132 gcpy                                ) + ".dxf"
1133 gcpy                            self.dxfllDT = open(self.dxfllDTfilename, "w")
1134 gcpy                        if (KH_tool_num > 0):
1135 gcpy                            print("Opening KH")
1136 gcpy                            self.dxfllKHfilename = basefilename + str(KH_tool_num
1137 gcpy                                ) + ".dxf"

```

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

```
1139 gcpy          if (large_square_tool_num > 0):
1140 gcpy              self.dxfpreamble(large_square_tool_num)
1141 gcpy          if (small_square_tool_num > 0):
1142 gcpy              self.dxfpreamble(small_square_tool_num)
1143 gcpy          if (large_ball_tool_num > 0):
1144 gcpy              self.dxfpreamble(large_ball_tool_num)
1145 gcpy          if (small_ball_tool_num > 0):
1146 gcpy              self.dxfpreamble(small_ball_tool_num)
1147 gcpy          if (large_V_tool_num > 0):
1148 gcpy              self.dxfpreamble(large_V_tool_num)
1149 gcpy          if (small_V_tool_num > 0):
1150 gcpy              self.dxfpreamble(small_V_tool_num)
1151 gcpy          if (DT_tool_num > 0):
1152 gcpy              self.dxfpreamble(DT_tool_num)
1153 gcpy          if (KH_tool_num > 0):
1154 gcpy              self.dxfpreamble(KH_tool_num)
1155 gcpy          if (Roundover_tool_num > 0):
1156 gcpy              self.dxfpreamble(Roundover_tool_num)
1157 gcpy          if (MISC_tool_num > 0):
1158 gcpy              self.dxfpreamble(MISC_tool_num)
```

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

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

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

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

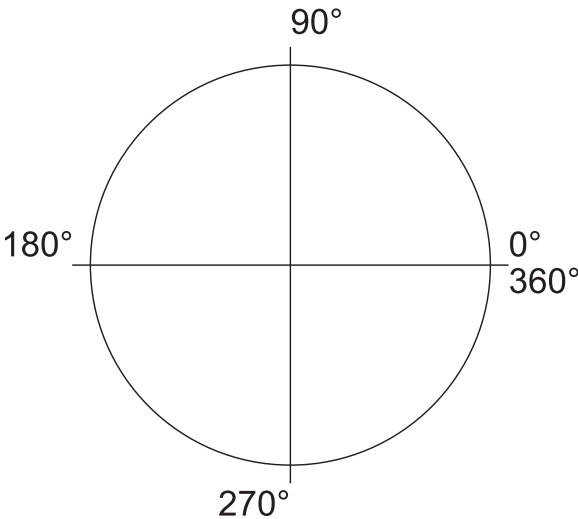
**dxfpreamble** This module requires that the tool number be passed in, and after writing out dxfpreamble, that value will be used to write out to the appropriate file with a series of if statements.

```
1160 gcpy          def dxfpreamble(self, tn):
1161 gcpy #              self.writedxf(tn,str(tn))
1162 gcpy              self.writedxf(tn,"0")
1163 gcpy              self.writedxf(tn,"SECTION")
1164 gcpy              self.writedxf(tn,"2")
1165 gcpy              self.writedxf(tn,"ENTITIES")
```

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

- dxfline
- dxfar
- a line dxfline
  - ARC — a notable option would be for the arc to close on itself, creating a circle: dxfar

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:

```
dxfar(10, 10, 5, 0, 90, small_square_tool_num);
dxfar(10, 10, 5, 90, 180, small_square_tool_num);
dxfar(10, 10, 5, 180, 270, small_square_tool_num);
dxfar(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.

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

```
1185 gcpy      def dxfar(self, tn, xcenter, ycenter, radius, anglebegin,
1186 gcpy          endangle):
1187 gcpy          if (self.generatedxf == True):
1188 gcpy              self.writedxf(tn, "0")
```

```

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

---

```

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

```



```

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

```

```
1323 gcpy #         toolpath = toolpath.translate([self.xpos(),self.ypos(),
1324 gcpy         self.zpos()])
1325 gcpy         self.dxfarc(self.currenttoolnumber(), xcenter,ycenter,
1326 gcpy         radius,180,270)
1327 gcpy         if (self.zpos == ez):
1328 gcpy             self.settzpos(0)
1329 gcpy         else:
1330 gcpy             self.settzpos((self.zpos()-ez)/90)
1331 gcpy         self.setxpos(ex)
1332 gcpy         self.setypos(ey)
1333 gcpy         self.setzpos(ez)
1334 gcpy         if self.generatepaths == True:
1335 gcpy             self.narcloop(269,180, xcenter, ycenter, radius)
1336 gcpy             self.toolpaths = self.toolpaths.union(toolpath)
1337 gcpy         else:
1338 gcpy             toolpath = self.narcloop(269,180, xcenter, ycenter,
1339 gcpy             radius)
1340 gcpy             return toolpath
1341 gcpy         def cutarcNWCWdxf(self, ex, ey, ez, xcenter, ycenter, radius):
1342 gcpy             global toolpath
1343 gcpy             toolpath = self.currenttool()
1344 gcpy             toolpath = toolpath.translate([self.xpos(),self.ypos(),
1345 gcpy             self.zpos()])
1346 gcpy             self.dxfarc(self.currenttoolnumber(), xcenter,ycenter,
1347 gcpy             radius,90,180)
1348 gcpy             if (self.zpos == ez):
1349 gcpy                 self.settzpos(0)
1350 gcpy             else:
1351 gcpy                 self.settzpos((self.zpos()-ez)/90)
1352 gcpy             self.setxpos(ex)
1353 gcpy             self.setypos(ey)
1354 gcpy             self.setzpos(ez)
1355 gcpy             if self.generatepaths == True:
1356 gcpy                 self.narcloop(179,90, xcenter, ycenter, radius)
1357 gcpy                 self.toolpaths = self.toolpaths.union(toolpath)
1358 gcpy             else:
1359 gcpy                 toolpath = self.narcloop(179,90, xcenter, ycenter,
1360 gcpy                 radius)
1361 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, stockYh
cutarcSWCCdxf(-(stockXwidth/4, stockYheight/4-stockYheight/16, -stockZthickness, -stockXwidth/4, stockYh
cutarcSECCdxf(-(stockXwidth/4-stockYheight/16), stockYheight/4, -stockZthickness, -stockXwidth/4, stockYh
```

```
1358 gcpy         def arcCCgc(self, ex, ey, ez, xcenter, ycenter, radius):
1359 gcpy             self.writegc("G03_X", str(ex), "Y", str(ey), "Z", str(ez)
1360 gcpy             , "R", str(radius))
1361 gcpy         def arcCWgc(self, ex, ey, ez, xcenter, ycenter, radius):
1362 gcpy             self.writegc("G02_X", str(ex), "Y", str(ey), "Z", str(ez)
1363 gcpy             , "R", str(radius))
```

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

```
1364 gcpy         def cutarcNECCdxfgc(self, ex, ey, ez, xcenter, ycenter, radius)
1365 gcpy             :
1366 gcpy             self.arcCCgc(ex, ey, ez, xcenter, ycenter, radius)
1367 gcpy             if self.generatepaths == True:
1368 gcpy                 self.cutarcNECCdxf(ex, ey, ez, xcenter, ycenter, radius
1369 gcpy                 )
1370 gcpy             else:
1371 gcpy                 return self.cutarcNECCdxf(ex, ey, ez, xcenter, ycenter,
1372 gcpy                 radius)
1373 gcpy         def cutarcNWCCdxfgc(self, ex, ey, ez, xcenter, ycenter, radius)
1374 gcpy             :
1375 gcpy             self.arcCCgc(ex, ey, ez, xcenter, ycenter, radius)
1376 gcpy             if self.generatepaths == False:
1377 gcpy                 return self.cutarcNWCCdxf(ex, ey, ez, xcenter, ycenter,
1378 gcpy                 radius)
1379 gcpy         def cutarcSWCCdxfgc(self, ex, ey, ez, xcenter, ycenter, radius)
1380 gcpy             :
```

```
1377 gcpy          self.arcCCgc(ex, ey, ez, xcenter, ycenter, radius)
1378 gcpy          if self.generatepaths == False:
1379 gcpy              return self.cutarcSWCCdxg(ex, ey, ez, xcenter, ycenter,
              radius)

1380 gcpy
1381 gcpy          def cutarcSECCdxfgc(self, ex, ey, ez, xcenter, ycenter, radius)
              :
1382 gcpy              self.arcCCgc(ex, ey, ez, xcenter, ycenter, radius)
1383 gcpy              if self.generatepaths == False:
1384 gcpy                  return self.cutarcSECCdxg(ex, ey, ez, xcenter, ycenter,
              radius)

1385 gcpy
1386 gcpy          def cutarcNECWdxfgc(self, ex, ey, ez, xcenter, ycenter, radius)
              :
1387 gcpy              self.arcCWgc(ex, ey, ez, xcenter, ycenter, radius)
1388 gcpy              if self.generatepaths == False:
1389 gcpy                  return self.cutarcNECWdxg(ex, ey, ez, xcenter, ycenter,
              radius)

1390 gcpy
1391 gcpy          def cutarcSECWdxfgc(self, ex, ey, ez, xcenter, ycenter, radius)
              :
1392 gcpy              self.arcCWgc(ex, ey, ez, xcenter, ycenter, radius)
1393 gcpy              if self.generatepaths == False:
1394 gcpy                  return self.cutarcSECWdxg(ex, ey, ez, xcenter, ycenter,
              radius)

1395 gcpy
1396 gcpy          def cutarcSWCWdxfgc(self, ex, ey, ez, xcenter, ycenter, radius)
              :
1397 gcpy              self.arcCWgc(ex, ey, ez, xcenter, ycenter, radius)
1398 gcpy              if self.generatepaths == False:
1399 gcpy                  return self.cutarcSWCWdxg(ex, ey, ez, xcenter, ycenter,
              radius)

1400 gcpy
1401 gcpy          def cutarcNWCWdxfgc(self, ex, ey, ez, xcenter, ycenter, radius)
              :
1402 gcpy              self.arcCWgc(ex, ey, ez, xcenter, ycenter, radius)
1403 gcpy              if self.generatepaths == False:
1404 gcpy                  return self.cutarcNWCWdxg(ex, ey, ez, xcenter, ycenter,
              radius)
```

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

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

```
1406 gcpy          def dxfgpostamble(self,tn):
1407 gcpy #              self.writedxg(tn,str(tn))
1408 gcpy              self.writedxg(tn,"0")
1409 gcpy              self.writedxg(tn,"ENDSEC")
1410 gcpy              self.writedxg(tn,"0")
1411 gcpy              self.writedxg(tn,"EOF")

1413 gcpy          def gcodepostamble(self):
1414 gcpy              self.writegc("Z12.700")
1415 gcpy              self.writegc("M05")
1416 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.

```
1418 gcpy      def closegcodefile(self):
1419 gcpy          self.gcodepreamble()
1420 gcpy          self.gc.close()
1421 gcpy
1422 gcpy      def closedxfile(self):
1423 gcpy          if self.generatedxf == True:
1424 gcpy              # global dxfclose
1425 gcpy              self.dxfpreamble(-1)
1426 gcpy              # self.dxfclosed = True
1427 gcpy              self.dxf.close()
1428 gcpy
1429 gcpy      def closedxfiles(self):
1430 gcpy          if self.generatedxfs == True:
1431 gcpy              if (self.large_square_tool_num > 0):
1432 gcpy                  self.dxfpreamble(self.large_square_tool_num)
1433 gcpy              if (self.small_square_tool_num > 0):
1434 gcpy                  self.dxfpreamble(self.small_square_tool_num)
1435 gcpy              if (self.large_ball_tool_num > 0):
1436 gcpy                  self.dxfpreamble(self.large_ball_tool_num)
1437 gcpy              if (self.small_ball_tool_num > 0):
1438 gcpy                  self.dxfpreamble(self.small_ball_tool_num)
1439 gcpy              if (self.large_V_tool_num > 0):
1440 gcpy                  self.dxfpreamble(self.large_V_tool_num)
1441 gcpy              if (self.small_V_tool_num > 0):
1442 gcpy                  self.dxfpreamble(self.small_V_tool_num)
1443 gcpy              if (self.DT_tool_num > 0):
1444 gcpy                  self.dxfpreamble(self.DT_tool_num)
1445 gcpy              if (self.KH_tool_num > 0):
1446 gcpy                  self.dxfpreamble(self.KH_tool_num)
1447 gcpy              if (self.Roundover_tool_num > 0):
1448 gcpy                  self.dxfpreamble(self.Roundover_tool_num)
1449 gcpy              if (self.MISC_tool_num > 0):
1450 gcpy                  self.dxfpreamble(self.MISC_tool_num)
1451 gcpy
1452 gcpy              if (self.large_square_tool_num > 0):
1453 gcpy                  self.dxfclsq.close()
1454 gcpy              if (self.small_square_tool_num > 0):
1455 gcpy                  self.dxfmsq.close()
1456 gcpy              if (self.large_ball_tool_num > 0):
1457 gcpy                  self.dxfclsq.close()
1458 gcpy              if (self.small_ball_tool_num > 0):
1459 gcpy                  self.dxfmsq.close()
1460 gcpy              if (self.large_V_tool_num > 0):
1461 gcpy                  self.dxfclsqV.close()
1462 gcpy              if (self.small_V_tool_num > 0):
1463 gcpy                  self.dxfmsqV.close()
1464 gcpy              if (self.DT_tool_num > 0):
1465 gcpy                  self.dxfDT.close()
1466 gcpy              if (self.KH_tool_num > 0):
1467 gcpy                  self.dxfKH.close()
1468 gcpy              if (self.Roundover_tool_num > 0):
1469 gcpy                  self.dxfRt.close()
1470 gcpy              if (self.MISC_tool_num > 0):
1471 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 gcpscad module closegcodefile(){
143 gcpscad     gcp.closegcodefile();
144 gcpscad }
145 gcpscad
146 gcpscad module closedxfiles(){
147 gcpscad     gcp.closedxfiles();
148 gcpscad }
149 gcpscad
150 gcpscad module closedxfile(){
151 gcpscad     gcp.closedxfile();
152 gcpscad }
```

## 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:

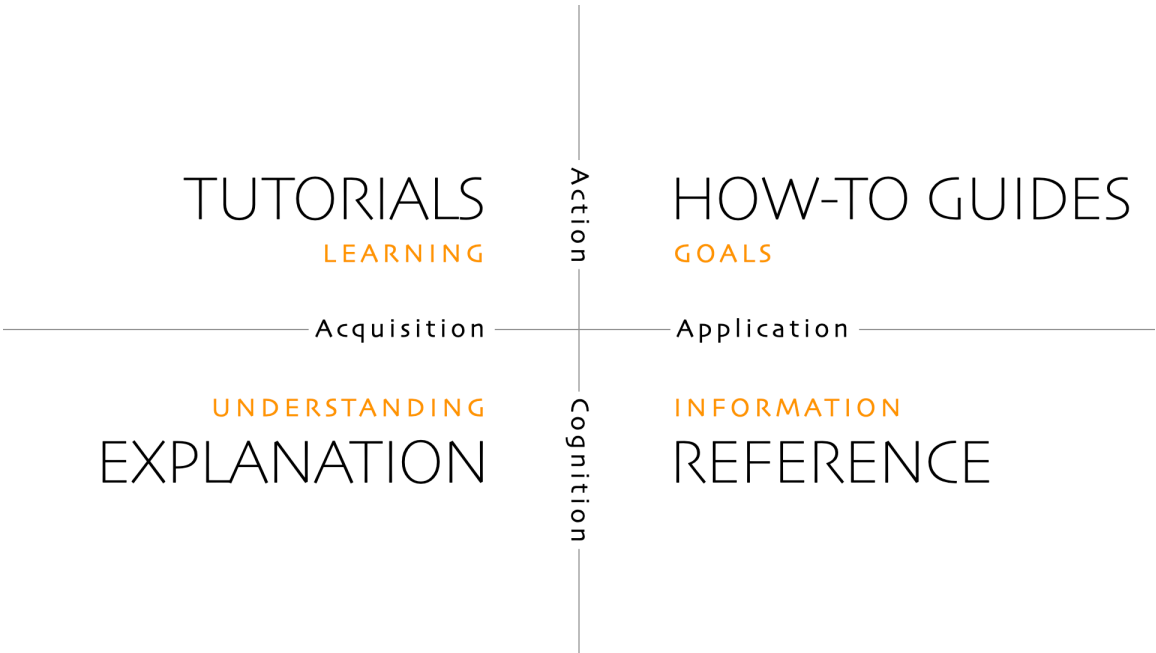
- Shallow Module
- Information Leakage
- Temporal Decomposition
- Overexposure
- Pass-Through Method
- Repetition
- Special-General Mixture
- Conjoined Methods
- Comment Repeats Code
- Implementation Documentation Contaminates Interface
- Vague Name
- Hard to Pick Name
- Hard to Describe
- Nonobvious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

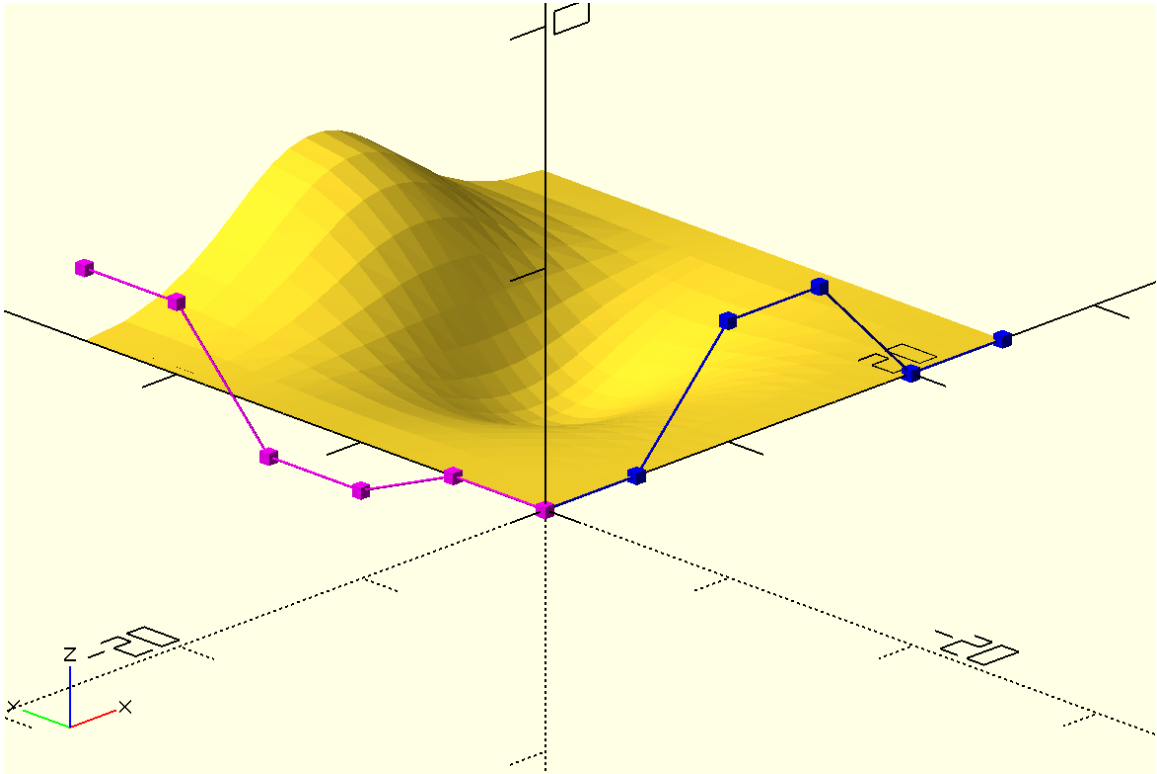
3 planes \* 3 Béziers \* (2 on-curve + 2 off-curve points) == 36 coordinate pairs

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/](https://old.reddit.com/r/OpenPythonSCAD/comments/1gjcz4z/pythonscad_will_get_a_new_spline_function/)

References

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

# Command Glossary

**settool** settool(102). 25

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



# Index

- ballnose, 26
- bowl tool, 26
- closedxfile, 59, 60
- closegcodefile, 59, 60
- currenttoolnum, 22
- currenttoolnumber, 25
- currenttoolshape, 28
- cut..., 32, 34
- cutarcCC, 36
- cutarcCW, 36
- cutkeyhole toolpath, 42
- cutKHgcdxf, 43
- cutline, 34
- dovetail, 28
- dxfarc, 55
- dxfline, 55
- dxfpreamble, 60
- dxfpreamble, 54
- dxfwrite, 54
- endmill square, 26
- endmill v, 26
- feed, 32
- gcodepreview, 21
  - writeln, 51
- gcp.setupstock, 23
- init, 21
- keyhole, 27
- mpx, 22
- mpy, 22
- mpz, 22
- opendxfile, 52
- opengcodefile, 52
- plunge, 32
- rapid..., 32
- rsc, 32
- settool, 25
- setupstock, 23
  - gcodepreview, 23
- setxpos, 23
- setypos, 23
- setzpos, 23
- speed, 32
- stepsizearc, 19
- stepsizeroundover, 19
- subroutine
  - gcodepreview, 23
  - writeln, 51
- threadmill, 27
- tool diameter, 30
- tool radius, 31
- toolchange, 28
- tpzinc, 22
- writedxDT, 54
- writedxKH, 54
- writedxflgbl, 54
- writedxflgsq, 54
- writedxflgV, 54
- writedxsmbl, 54
- writedxsmq, 54
- writedxsmV, 54
- xpos, 22
- ypos, 22
- zpos, 22

# Routines

- ballnose, 26
- bowl tool, 26
- closedxfile, 59, 60
- closegcodefile, 59, 60
- currenttoolnumber, 25
- cut..., 32, 34
- cutarcCC, 36
- cutarcCW, 36
- cutkeyhole toolpath, 42
- cutKHgcdxf, 43
- cutline, 34
- dovetail, 28
- dxfar, 55
- dxflin, 55
- dxfpreamble, 60
- dxfpreamble, 54
- dxfwrite, 54
- endmill square, 26
- endmill v, 26
- gcodepreview, 21, 23
- gcp.setupstock, 23
- init, 21
- keyhole, 27
- opendxfile, 52
- opengcodefile, 52
- rapid..., 32
- rcl, 32
- settool, 25
- setupstock, 23
- setxpos, 23
- setypos, 23
- setzpos, 23
- threadmill, 27
- tool diameter, 30
- tool radius, 31
- toolchange, 28
- writedxDT, 54
- writedxKH, 54
- writedxflbl, 54
- writedxflsq, 54
- writedxflV, 54
- writedxsmbl, 54
- writedxsmq, 54
- writedxsmV, 54
- writeln, 51
- xpos, 22
- ypos, 22
- zpos, 22

# Variables

currenttoolnum, 22	plunge, 32
currenttoolshape, 28	speed, 32
feed, 32	stepsizearc, 19
mpx, 22	stepsizeroundover, 19
mpy, 22	tpzinc, 22
mpz, 22	