

# The gcodepreview PythonSCAD library\*

Author: William F. Adams  
willadams at aol dot com

2025/07/4

### 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.

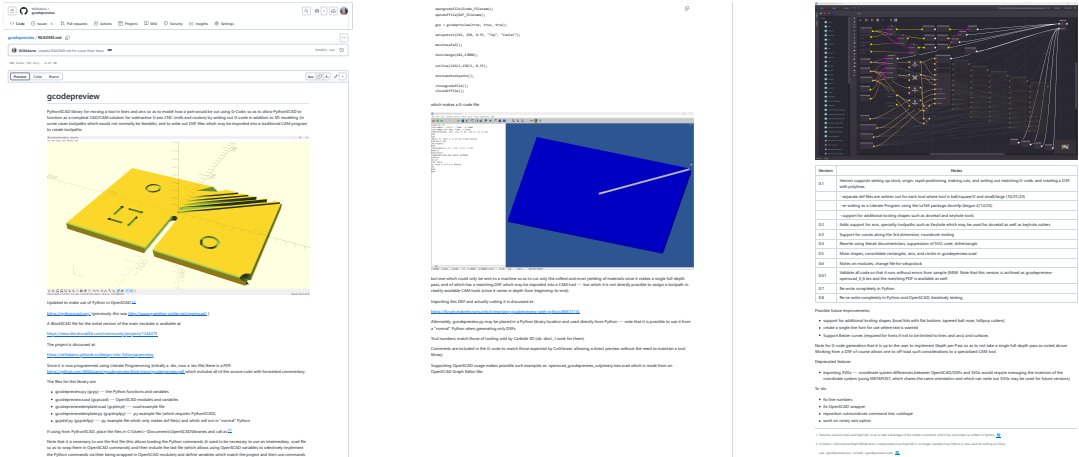
## Contents

<b>1</b>	<b>readme.md</b>	<b>3</b>
<b>2</b>	<b>Usage and Templates</b>	<b>7</b>
2.1	gcpdxf.py . . . . .	7
2.2	gcpcutdxf.py . . . . .	11
2.3	gcodepreviewtemplate.py . . . . .	13
2.4	gcodepreviewtemplate.scad . . . . .	18
<b>3</b>	<b>gcodepreview</b>	<b>22</b>
3.1	Module Naming Convention . . . . .	23
3.1.1	Parameters and Default Values . . . . .	25
3.2	Implementation files and gcodepreview class . . . . .	25
3.2.1	Position and Variables . . . . .	28
3.2.2	Initial Modules . . . . .	29
3.2.3	Adjustments and Additions . . . . .	32
3.3	Tools and Changes . . . . .	32
3.3.1	Numbering for Tools . . . . .	32
3.3.1.1	toolchange . . . . .	35
3.3.1.2	Square (including O-flute) . . . . .	36
3.3.1.3	Ball nose (including tapered ball nose) . . . . .	37
3.3.1.4	V . . . . .	38
3.3.1.5	Keyhole . . . . .	38
3.3.1.6	Bowl . . . . .	39
3.3.1.7	Tapered ball nose . . . . .	40
3.3.1.8	Roundover (corner rounding) . . . . .	40
3.3.1.9	Dovetails . . . . .	41
3.3.1.10	closing G-code . . . . .	42
3.3.2	Laser support . . . . .	42
3.4	Shapes and tool movement . . . . .	43
3.4.0.1	Tooling for Undercutting Toolpaths . . . . .	43
3.4.1	Generalized commands and cuts . . . . .	43
3.4.2	Movement and color . . . . .	44
3.4.2.1	toolmovement . . . . .	44
3.4.2.2	Normal Tooling/toolshapes . . . . .	45
3.4.2.3	Square (including O-flute) . . . . .	45
3.4.2.4	Ball nose (including tapered ball nose) . . . . .	46
3.4.2.5	bowl . . . . .	46
3.4.2.6	V . . . . .	46
3.4.2.7	Keyhole . . . . .	46
3.4.2.8	Tapered ball nose . . . . .	47
3.4.2.9	Dovetails . . . . .	47
3.4.2.10	Concave toolshapes . . . . .	47
3.4.2.11	Roundover tooling . . . . .	48
3.4.2.12	shaftmovement . . . . .	48
3.4.2.13	rapid and cut (lines) . . . . .	48
3.4.2.14	Arcs . . . . .	51
3.4.3	tooldiameter . . . . .	55
3.4.4	Feeds and Speeds . . . . .	57
3.5	Difference of Stock, Rapids, and Toolpaths . . . . .	57
3.6	Output files . . . . .	57
3.6.1	Python and OpenSCAD File Handling . . . . .	58

\*This file (gcodepreview) has version number vo.9, last revised 2025/07/4.

Contents	2
3.6.2 DXF Overview . . . . .	61
3.6.2.1 Writing to DXF files . . . . .	61
3.6.2.1.1 DXF Lines and Arcs . . . . .	62
3.6.3 G-code Overview . . . . .	68
3.6.3.1 Closings . . . . .	70
3.7 Cutting shapes and expansion . . . . .	71
3.7.0.1 Building blocks . . . . .	71
3.7.0.2 List of shapes . . . . .	71
3.7.0.2.1 circles . . . . .	73
3.7.0.2.2 rectangles . . . . .	73
3.7.0.2.3 Keyhole toolpath and undercut tooling . . . . .	76
3.7.0.2.4 Dovetail joinery and tooling . . . . .	83
3.7.0.2.5 Full-blind box joints . . . . .	85
3.8 (Reading) G-code Files . . . . .	90
4 Notes . . . . .	93
4.1 Other Resources . . . . .	93
4.1.1 Coding Style . . . . .	93
4.1.2 Coding References . . . . .	93
4.1.3 Documentation Style . . . . .	93
4.1.4 Holidays . . . . .	94
4.1.5 DXFs . . . . .	94
4.2 Future . . . . .	94
4.2.1 Images . . . . .	94
4.2.2 Bézier curves in 2 dimensions . . . . .	94
4.2.3 Bézier curves in 3 dimensions . . . . .	94
4.2.4 Mathematics . . . . .	95
Index . . . . .	98
Routines . . . . .	99
Variables . . . . .	100

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 or routers at this time, 4th-axis
  support may come in a future version) by writing out G-code in
  addition to 3D modeling (in certain cases toolpaths which would
  not normally be feasible), and to write out DXF files which may
  be imported into a traditional CAM program to create toolpaths.
4 rdme
5 rdme ![OpenSCAD gcodepreview Unit Tests](https://raw.githubusercontent.com/WillAdams/gcodepreview/main/gcodepreview_unittests.png?raw=
  true)
6 rdme
7 rdme Updated to make use of Python in OpenSCAD:[~rapcad]
8 rdme
9 rdme [~rapcad]: Previous versions had used RapCAD, so as to take
  advantage of the writeln command, which has since been re-
  written in Python.
10 rdme
11 rdme https://pythonscad.org/ (previously this was http://www.guenther-
  sohler.net/openscad/ )
12 rdme
13 rdme A BlockSCAD file for the initial version of the
14 rdme main modules is available at:
15 rdme
16 rdme https://www.blockscad3d.com/community/projects/1244473
17 rdme
18 rdme The project is discussed at:
19 rdme
20 rdme https://willadams.gitbook.io/design-into-3d/programming
21 rdme
22 rdme Since it is now programmed using Literate Programming (initially a
  .dtx, now a .tex file) there is a PDF: https://github.com/
  WillAdams/gcodepreview/blob/main/gcodepreview.pdf which includes
  all of the source code with formatted comments.
23 rdme
24 rdme The files for this library are:
25 rdme
26 rdme - gcodepreview.py (gcpy) --- the Python class/functions and
  variables
27 rdme - gcodepreview.scad (gcpscad) --- OpenSCAD modules and parameters
28 rdme
29 rdme And there several sample/template files which may be used as the
  starting point for a given project:
30 rdme
31 rdme - gcodepreviewtemplate.scad (gcptmpl) --- .scad example file
32 rdme - gcodepreviewtemplate.py (gcptmplpy) --- .py example file
33 rdme - gcpdxf.py (gcpdxfpy) --- .py example file which only makes dxf
  file(s) and which will run in "normal" Python in addition to
  PythonSCAD
34 rdme - gcpgc.py (gcpgc) --- .py example which loads a G-code file and
  generates a 3D preview showing how the G-code will cut
35 rdme
36 rdme If using from PythonSCAD, place the files in C:\Users\\~\Documents
  \OpenSCAD\libraries [~libraries] or, load them from Github using
  the command:
```

```

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

```

```

86 rdme
87 rdme | Version          | Notes          |
88 rdme | ----- | ----- |
89 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. |
90 rdme |              | - separate dxf files are written out for each
      |              | tool where tool is ball/square/V and small/large (10/31/23)
      |
91 rdme |              | - re-writing as a Literate Program using the
      |              | LaTeX package docmfp (begun 4/12/24)
      |
92 rdme |              | - support for additional tooling shapes such as
      |              | dovetail and keyhole tools
      |
93 rdme | 0.2          | Adds support for arcs, specialty toolpaths such
      |              | as Keyhole which may be used for dovetail as well as keyhole
      |              | cutters
      |
94 rdme | 0.3          | Support for curves along the 3rd dimension,
      |              | roundover tooling
      |
95 rdme | 0.4          | Rewrite using literati documentclass, suppression
      |              | of SVG code, dxfrextangle
      |
96 rdme | 0.5          | More shapes, consolidate rectangles, arcs, and
      |              | circles in gcodepreview.scad
      |
97 rdme | 0.6          | Notes on modules, change file for setupstock
      |
98 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)
99 rdme | 0.7          | Re-write completely in Python
      |
100 rdme | 0.8          | Re-re-write completely in Python and OpenSCAD,
      |              | iteratively testing
      |
101 rdme | 0.801        | Add support for bowl bits with flat bottom
      |
102 rdme | 0.802        | Add support for tapered ball-nose and V tools
      |              | with flat bottom
      |
103 rdme | 0.803        | Implement initial color support and joinery
      |              | modules (dovetail and full blind box joint modules)
      |
104 rdme | 0.9          | Re-write to use Python lists for 3D shapes for
      |              | toolpaths and rapids. |
105 rdme
106 rdme Possible future improvements:
107 rdme
108 rdme - support for post-processors
109 rdme - support for 4th-axis
110 rdme - support for two-sided machining (import an STL or other file to
      |              | use for stock, or possibly preserve the state after one cut and
      |              | then rotate the cut stock/part)
111 rdme - support for additional tooling shapes (lollipop cutters)
112 rdme - create a single line font for use where text is wanted
113 rdme - Support Bézier curves (required for fonts if not to be limited
      |              | to lines and arcs) and surfaces
114 rdme
115 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.

```

```
116 rdme
117 rdme To-do:
118 rdme
119 rdme - clock-wise arcs
120 rdme - add toolpath for cutting countersinks using ball-nose tool from
        inside working out
121 rdme - fix OpenSCAD wrapper and add any missing commands for Python
122 rdme - verify support for shaft on tooling
123 rdme - create a folder of template and sample files
124 rdme - clean up/comment out all mentions of previous versions/features/
        implementations/deprecated features
125 rdme
126 rdme Deprecated feature:
127 rdme
128 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)
```

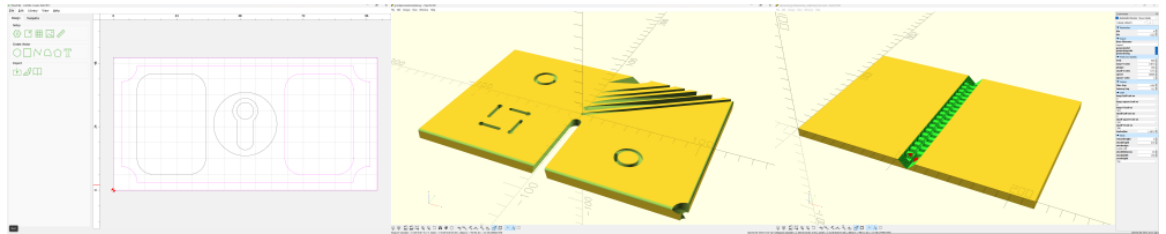
---

## 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(Python)SCAD, enabling the creation of 2D DXFs, G-code (which cuts a 2D or 3D part), 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 DXFs, and/or G-code with 3D modeling) for a preview. Providing them in a programmatic context allows making parts or design elements of parts (e.g., joinery) which would be tedious or difficult (or verging on impossible) 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 (so long as the dimensions and radii allows for reasonable tool (and toolpath) geometries).

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 if enabled, 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 collection of templates and a tutorial) are 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 and which may be used as a starting point for a new part/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 the requirements of subtractive machining.
- 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 avoid the requirement of implementing Depth per Pass (but note that this will lose the previewing of scalloped tool marks in places where they might appear otherwise)

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

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

### 2.1 `gcpdxf.py`

The most basic usage, with the fewest dependencies is to use “plain” Python to create `dxf` files. Note that this example includes an optional command `nimport(<URL>)` which if enabled/uncommented (and the following line commented out), will allow one to use OpenPythonSCAD to import the library from Github, sidestepping the need to download and install the library locally into an installation of OpenPythonSCAD. Usage in “normal” Python will require manually installing the `gcodepreview.py` file where Python can find it. A further consideration is where the file will be placed if the full path is not enumerated, the Desktop is the default destination for Microsoft Windows.

---

```

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, # generategcode
6 gcpdxfpy                               True   # generatedxf
7 gcpdxfpy                               )
8 gcpdxfpy
9 gcpdxfpy # [Stock] */
10 gcpdxfpy stockXwidth = 100
11 gcpdxfpy # [Stock] */
12 gcpdxfpy stockYheight = 50
13 gcpdxfpy
14 gcpdxfpy # [Export] */
15 gcpdxfpy Base_filename = "gcpdxf"
16 gcpdxfpy
17 gcpdxfpy
18 gcpdxfpy # [CAM] */
19 gcpdxfpy large_square_tool_num = 102
20 gcpdxfpy # [CAM] */
21 gcpdxfpy small_square_tool_num = 0
22 gcpdxfpy # [CAM] */
23 gcpdxfpy large_ball_tool_num = 0
24 gcpdxfpy # [CAM] */
25 gcpdxfpy small_ball_tool_num = 0
26 gcpdxfpy # [CAM] */
27 gcpdxfpy large_V_tool_num = 0
28 gcpdxfpy # [CAM] */
29 gcpdxfpy small_V_tool_num = 0
30 gcpdxfpy # [CAM] */
31 gcpdxfpy DT_tool_num = 374
32 gcpdxfpy # [CAM] */
33 gcpdxfpy KH_tool_num = 0
34 gcpdxfpy # [CAM] */
35 gcpdxfpy Roundover_tool_num = 0
36 gcpdxfpy # [CAM] */
37 gcpdxfpy MISC_tool_num = 0
38 gcpdxfpy
39 gcpdxfpy # [Design] */
40 gcpdxfpy inset = 3
41 gcpdxfpy # [Design] */
42 gcpdxfpy radius = 6
43 gcpdxfpy # [Design] */
44 gcpdxfpy cornerstyle = "Fillet" # "Chamfer", "Flipped Fillet"
45 gcpdxfpy
46 gcpdxfpy gcp.opendxf(file(Base_filename))
47 gcpdxfpy
48 gcpdxfpy gcp.dxfrectangle(large_square_tool_num, 0, 0, stockXwidth,
    stockYheight)
49 gcpdxfpy
50 gcpdxfpy gcp.setdxfcolor("Red")
51 gcpdxfpy
52 gcpdxfpy gcp.dxfarc(large_square_tool_num, inset, inset, radius, 0, 90)
53 gcpdxfpy gcp.dxfarc(large_square_tool_num, stockXwidth - inset, inset,
    radius, 90, 180)
54 gcpdxfpy gcp.dxfarc(large_square_tool_num, stockXwidth - inset, stockYheight
    - inset, radius, 180, 270)
55 gcpdxfpy gcp.dxfarc(large_square_tool_num, inset, stockYheight - inset,
    radius, 270, 360)
56 gcpdxfpy
57 gcpdxfpy gcp.dxfline(large_square_tool_num, inset, inset + radius, inset,
    stockYheight - (inset + radius))
58 gcpdxfpy gcp.dxfline(large_square_tool_num, inset + radius, inset,
    stockXwidth - (inset + radius), inset)
59 gcpdxfpy gcp.dxfline(large_square_tool_num, stockXwidth - inset, inset +
    radius, stockXwidth - inset, stockYheight - (inset + radius))
60 gcpdxfpy gcp.dxfline(large_square_tool_num, inset + radius, stockYheight -
    inset, stockXwidth - (inset + radius), stockYheight - inset)
61 gcpdxfpy
62 gcpdxfpy gcp.setdxfcolor("Blue")
63 gcpdxfpy
64 gcpdxfpy gcp.dxfrectangle(large_square_tool_num, radius +inset, radius,
    stockXwidth/2 - (radius * 4), stockYheight - (radius * 2),
    cornerstyle, radius)
65 gcpdxfpy gcp.dxfrectangle(large_square_tool_num, stockXwidth/2 + (radius *
    2) + inset, radius, stockXwidth/2 - (radius * 4), stockYheight -
    (radius * 2), cornerstyle, radius)

```



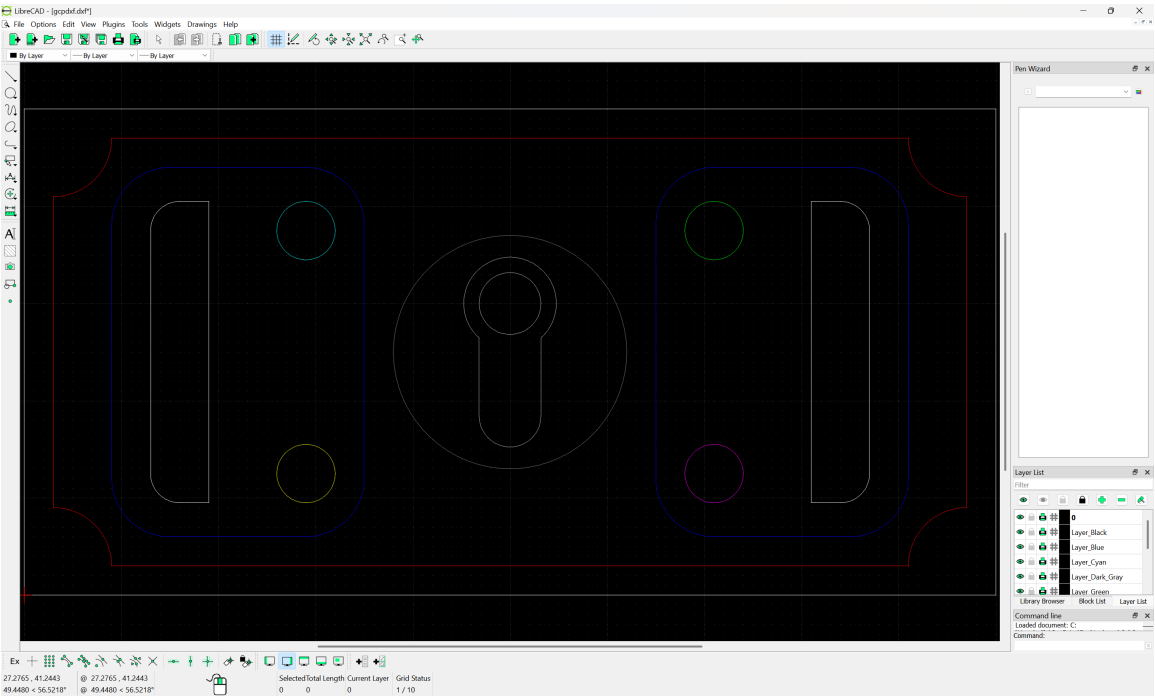
```

66 gcpdxfpyp
67 gcpdxfpyp gcp.setdxfc("Black")
68 gcpdxfpyp
69 gcpdxfpyp gcp.beginpolyline(large_square_tool_num)
70 gcpdxfpyp gcp.addvertex(large_square_tool_num, stockXwidth*0.75+radius*1.5,
    stockYheight/4-radius/2)
71 gcpdxfpyp gcp.addvertex(large_square_tool_num, stockXwidth*0.75+radius,
    stockYheight/4-radius/2)
72 gcpdxfpyp gcp.addvertex(large_square_tool_num, stockXwidth*0.75+radius,
    stockYheight*0.75+radius/2)
73 gcpdxfpyp gcp.addvertex(large_square_tool_num, stockXwidth*0.75+radius*1.5,
    stockYheight*0.75+radius/2)
74 gcpdxfpyp gcp.closepolyline(large_square_tool_num)
75 gcpdxfpyp
76 gcpdxfpyp gcp.dxfarc(large_square_tool_num, stockXwidth*0.75+radius*1.5,
    stockYheight*0.75, radius/2, 0, 90)
77 gcpdxfpyp
78 gcpdxfpyp gcp.beginpolyline(large_square_tool_num)
79 gcpdxfpyp gcp.addvertex(large_square_tool_num, stockXwidth*0.75+radius*2,
    stockYheight*0.75)
80 gcpdxfpyp gcp.addvertex(large_square_tool_num, stockXwidth*0.75+radius*2,
    stockYheight/4)
81 gcpdxfpyp gcp.closepolyline(large_square_tool_num)
82 gcpdxfpyp
83 gcpdxfpyp gcp.dxfarc(large_square_tool_num, stockXwidth*0.75+radius*1.5,
    stockYheight/4, radius/2, 270, 360)
84 gcpdxfpyp
85 gcpdxfpyp gcp.setdxfc("White")
86 gcpdxfpyp
87 gcpdxfpyp gcp.beginpolyline(large_square_tool_num)
88 gcpdxfpyp gcp.addvertex(large_square_tool_num, stockXwidth*0.25-radius*1.5,
    stockYheight/4-radius/2)
89 gcpdxfpyp gcp.addvertex(large_square_tool_num, stockXwidth*0.25-radius,
    stockYheight/4-radius/2)
90 gcpdxfpyp gcp.addvertex(large_square_tool_num, stockXwidth*0.25-radius,
    stockYheight*0.75+radius/2)
91 gcpdxfpyp gcp.addvertex(large_square_tool_num, stockXwidth*0.25-radius*1.5,
    stockYheight*0.75+radius/2)
92 gcpdxfpyp gcp.closepolyline(large_square_tool_num)
93 gcpdxfpyp
94 gcpdxfpyp gcp.dxfarc(large_square_tool_num, stockXwidth*0.25-radius*1.5,
    stockYheight*0.75, radius/2, 90, 180)
95 gcpdxfpyp
96 gcpdxfpyp gcp.beginpolyline(large_square_tool_num)
97 gcpdxfpyp gcp.addvertex(large_square_tool_num, stockXwidth*0.25-radius*2,
    stockYheight*0.75)
98 gcpdxfpyp gcp.addvertex(large_square_tool_num, stockXwidth*0.25-radius*2,
    stockYheight/4)
99 gcpdxfpyp gcp.closepolyline(large_square_tool_num)
100 gcpdxfpyp
101 gcpdxfpyp gcp.dxfarc(large_square_tool_num, stockXwidth*0.25-radius*1.5,
    stockYheight/4, radius/2, 180, 270)
102 gcpdxfpyp
103 gcpdxfpyp gcp.setdxfc("Yellow")
104 gcpdxfpyp gcp.dxfcircle(large_square_tool_num, stockXwidth/4+1+radius/2,
    stockYheight/4, radius/2)
105 gcpdxfpyp
106 gcpdxfpyp gcp.setdxfc("Green")
107 gcpdxfpyp gcp.dxfcircle(large_square_tool_num, stockXwidth*0.75-(1+radius/2),
    stockYheight*0.75, radius/2)
108 gcpdxfpyp
109 gcpdxfpyp gcp.setdxfc("Cyan")
110 gcpdxfpyp gcp.dxfcircle(large_square_tool_num, stockXwidth/4+1+radius/2,
    stockYheight*0.75, radius/2)
111 gcpdxfpyp
112 gcpdxfpyp gcp.setdxfc("Magenta")
113 gcpdxfpyp gcp.dxfcircle(large_square_tool_num, stockXwidth*0.75-(1+radius/2),
    stockYheight/4, radius/2)
114 gcpdxfpyp
115 gcpdxfpyp gcp.setdxfc("Dark_Gray")
116 gcpdxfpyp
117 gcpdxfpyp gcp.dxfcircle(large_square_tool_num, stockXwidth/2, stockYheight/2,
    radius * 2)
118 gcpdxfpyp
119 gcpdxfpyp gcp.setdxfc("Light_Gray")
120 gcpdxfpyp
121 gcpdxfpyp gcp.dxfKH(374, stockXwidth/2, stockYheight/5*3, 0, -7, 270,
    11.5875)

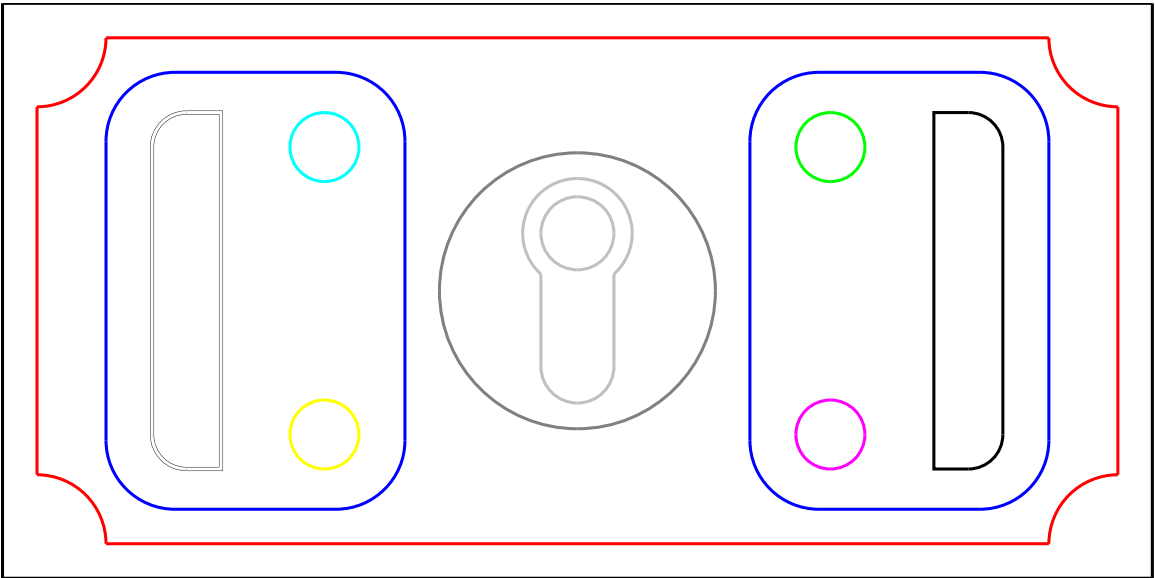
```

```
122 gcpdxfpyp
123 gcpdxfpyp gcp.closedxfile()
```

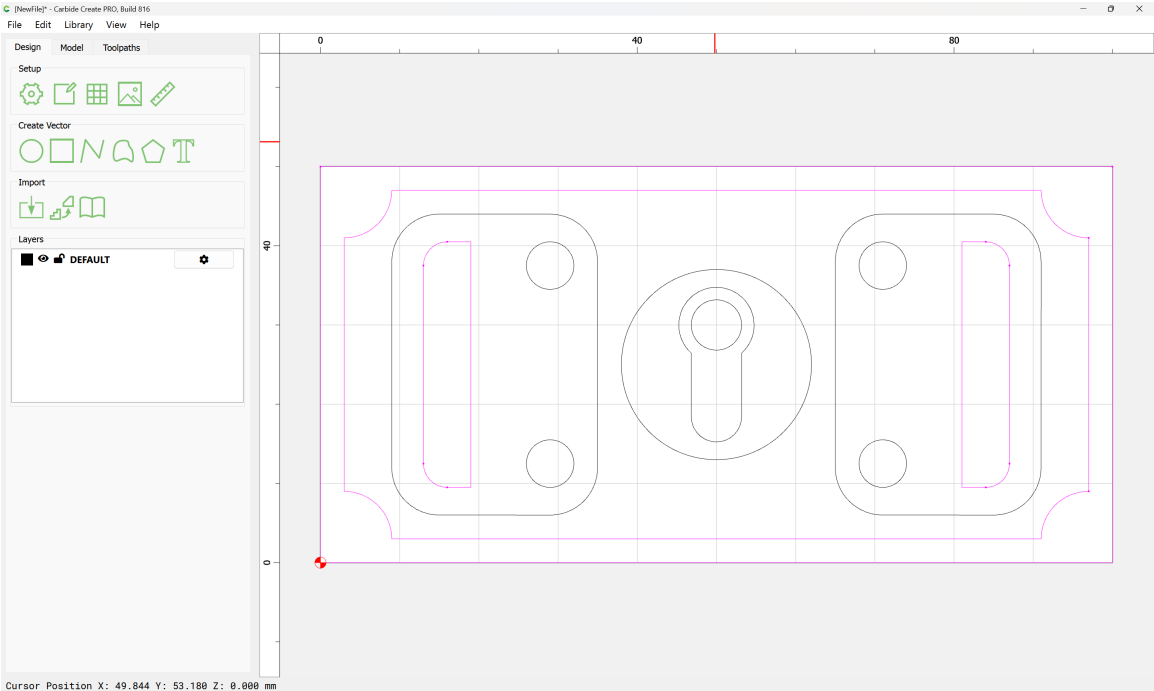
which creates a .dxf file which may be imported into any CAD program:



with the appearance (once converted into a .svg and then re-saved as a .pdf and edited so as to show the white elements):



and which may be imported into pretty much any CAD or CAM application, e.g., Carbide Create:



As shown/implied by the above code, the following commands/shapes are implemented:

- `dxfrectangle` (specify lower-left and upper-right corners)
  - `dxfrectangleround` (specified as “Fillet” and radius for the round option)
  - `dxfrectanglechamfer` (specified as “Chamfer” and radius for the round option)
  - `dxfrectangleflippedfillet` (specified as “Flipped Fillet” and radius for the option)
- `dxfcircle` (specifying their center and radius)
- `dxfline` (specifying begin/end points)
- `dxfarc` (specifying arc center, radius, and beginning/ending angles)
- `dxfKH` (specifying origin, depth, angle, distance)

2.2 gcpcutdxf.py

A notable limitation of the above is that there is no interactivity — the .dxf file is generated, then must be opened and the result of the run checked (if there is a DXF viewer/editor which will live-reload the file based on it being updated that would be obviated). Reworking the commands for the above design so as to show a 3D model is a straight-forward task:

```
1 gcpcutdxfpy from openscad import *
2 gcpcutdxfpy # nimport("https://raw.githubusercontent.com/WillAdams/gcodepreview
    /refs/heads/main/gcodepreview.py")
3 gcpcutdxfpy from gcodepreview import *
4 gcpcutdxfpy
5 gcpcutdxfpy fa = 2
6 gcpcutdxfpy fs = 0.125
7 gcpcutdxfpy
8 gcpcutdxfpy gcp = gcodepreview(False, # generategcode
9 gcpcutdxfpy                                True # generatedxf
10 gcpcutdxfpy                                )
11 gcpcutdxfpy
12 gcpcutdxfpy # [Stock] */
13 gcpcutdxfpy stockXwidth = 100
14 gcpcutdxfpy # [Stock] */
15 gcpcutdxfpy stockYheight = 50
16 gcpcutdxfpy # [Stock] */
17 gcpcutdxfpy stockZthickness = 3.175
18 gcpcutdxfpy # [Stock] */
19 gcpcutdxfpy zeroheight = "Top" # [Top, Bottom]
20 gcpcutdxfpy # [Stock] */
21 gcpcutdxfpy stockzero = "Lower-Left" # [Lower-Left, Center-Left, Top-Left,
    Center]
22 gcpcutdxfpy # [Stock] */
23 gcpcutdxfpy retractheight = 3.175
24 gcpcutdxfpy
25 gcpcutdxfpy # [Export] */
26 gcpcutdxfpy Base_filename = "gcpdxf"
27 gcpcutdxfpy
```

```

28 gcpcutdxfp
29 gcpcutdxfp # [CAM] */
30 gcpcutdxfp large_square_tool_num = 112
31 gcpcutdxfp # [CAM] */
32 gcpcutdxfp small_square_tool_num = 0
33 gcpcutdxfp # [CAM] */
34 gcpcutdxfp large_ball_tool_num = 111
35 gcpcutdxfp # [CAM] */
36 gcpcutdxfp small_ball_tool_num = 0
37 gcpcutdxfp # [CAM] */
38 gcpcutdxfp large_V_tool_num = 0
39 gcpcutdxfp # [CAM] */
40 gcpcutdxfp small_V_tool_num = 0
41 gcpcutdxfp # [CAM] */
42 gcpcutdxfp DT_tool_num = 374
43 gcpcutdxfp # [CAM] */
44 gcpcutdxfp KH_tool_num = 0
45 gcpcutdxfp # [CAM] */
46 gcpcutdxfp Roundover_tool_num = 0
47 gcpcutdxfp # [CAM] */
48 gcpcutdxfp MISC_tool_num = 0
49 gcpcutdxfp
50 gcpcutdxfp # [Design] */
51 gcpcutdxfp inset = 3
52 gcpcutdxfp # [Design] */
53 gcpcutdxfp radius = 6
54 gcpcutdxfp # [Design] */
55 gcpcutdxfp cornerstyle = "Fillet" # "Chamfer", "Flipped Fillet"
56 gcpcutdxfp
57 gcpcutdxfp gcp.opendxfile(Base_filename)
58 gcpcutdxfp
59 gcpcutdxfp gcp.setupstock(stockXwidth, stockYheight, stockZthickness,
    zeroheight, stockzero, retractheight)
60 gcpcutdxfp
61 gcpcutdxfp gcp.toolchange(large_square_tool_num)
62 gcpcutdxfp
63 gcpcutdxfp gcp.setdxfcolor("Red")
64 gcpcutdxfp
65 gcpcutdxfp gcp.cutrectanglidxf(large_square_tool_num, 0, 0, 0, stockXwidth,
    stockYheight, stockZthickness)
66 gcpcutdxfp
67 gcpcutdxfp gcp.toolchange(large_ball_tool_num)
68 gcpcutdxfp
69 gcpcutdxfp gcp.setdxfcolor("Gray")
70 gcpcutdxfp
71 gcpcutdxfp gcp.rapid(inset + radius, inset, 0, "laser")
72 gcpcutdxfp
73 gcpcutdxfp gcp.cutlinedxf(inset + radius, inset, -stockZthickness/2)
74 gcpcutdxfp gcp.cutquarterCCNEdxf(inset, inset + radius, -stockZthickness/2,
    radius)
75 gcpcutdxfp
76 gcpcutdxfp gcp.cutlinedxf(inset, stockYheight - (inset + radius), -
    stockZthickness/2)
77 gcpcutdxfp
78 gcpcutdxfp gcp.cutquarterCCSEdxf(inset + radius, stockYheight - inset, -
    stockZthickness/2, radius)
79 gcpcutdxfp
80 gcpcutdxfp gcp.cutlinedxf(stockXwidth - (inset + radius), stockYheight - inset
    , -stockZthickness/2)
81 gcpcutdxfp
82 gcpcutdxfp gcp.cutquarterCCSWdxf(stockXwidth - inset, stockYheight - (inset +
    radius), -stockZthickness/2, radius)
83 gcpcutdxfp
84 gcpcutdxfp gcp.cutlinedxf(stockXwidth - (inset), (inset + radius), -
    stockZthickness/2)
85 gcpcutdxfp
86 gcpcutdxfp gcp.cutquarterCCNWdxf(stockXwidth - (inset + radius), inset, -
    stockZthickness/2, radius)
87 gcpcutdxfp
88 gcpcutdxfp gcp.cutlinedxf((inset + radius), inset, -stockZthickness/2)
89 gcpcutdxfp
90 gcpcutdxfp gcp.setdxfcolor("Blue")
91 gcpcutdxfp
92 gcpcutdxfp gcp.rapid(radius + inset + radius, radius, 0, "laser")
93 gcpcutdxfp
94 gcpcutdxfp gcp.cutrectanglerounddxf(large_square_tool_num, radius +inset,
    radius, 0, stockXwidth/2 - (radius * 4), stockYheight - (radius
    * 2), -stockZthickness/4, radius)

```

```

95 gpcutdxfp
96 gpcutdxfp gcp.rapid(stockXwidth/2 + (radius * 2) + inset + radius, radius, 0,
    "laser")

97 gpcutdxfp
98 gpcutdxfp gcp.cutrectanglerounddx(f(large_square_tool_num, stockXwidth/2 + (
    radius * 2) + inset, radius, 0, stockXwidth/2 - (radius * 4),
    stockYheight - (radius * 2), -stockZthickness/4, radius)

99 gpcutdxfp
100 gpcutdxfp gcp.setdxfc("Red")
101 gpcutdxfp
102 gpcutdxfp gcp.rapid(stockXwidth/2 + radius, stockYheight/2, 0, "laser")
103 gpcutdxfp
104 gpcutdxfp gcp.toolchange(large_square_tool_num)
105 gpcutdxfp
106 gpcutdxfp gcp.cutcircleCC(stockXwidth/2, stockYheight/2, 0, -stockZthickness,
    radius)

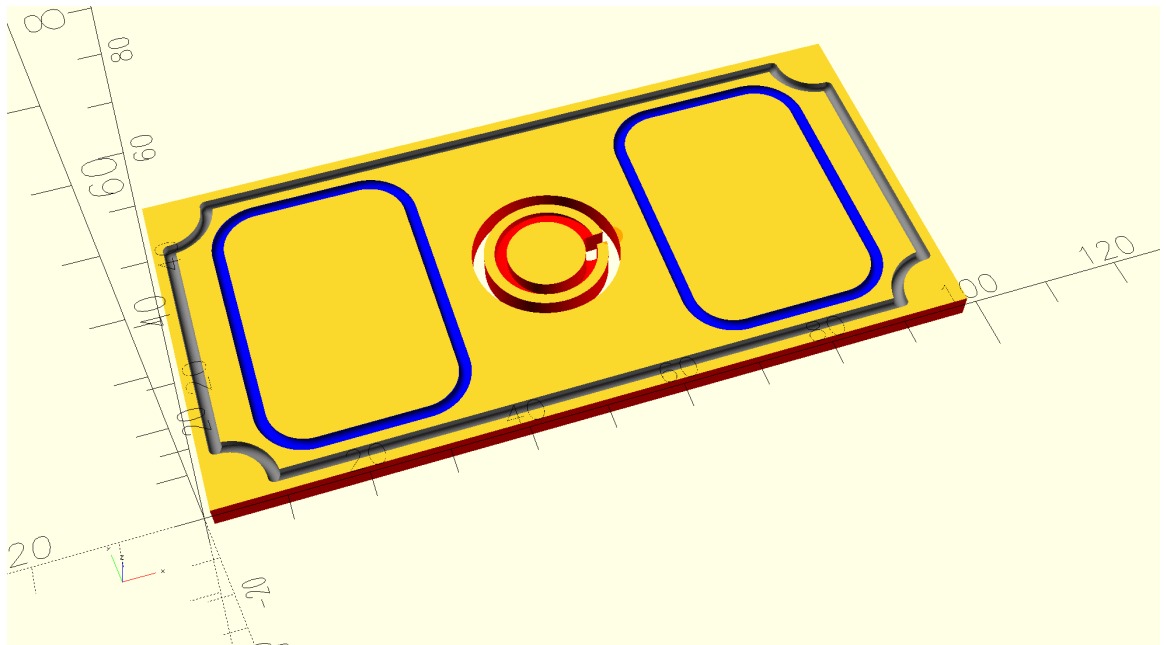
107 gpcutdxfp
108 gpcutdxfp gcp.cutcircleCC(stockXwidth/2, stockYheight/2, -stockZthickness, -
    stockZthickness, radius*1.5)

109 gpcutdxfp
110 gpcutdxfp gcp.closedxfile()
111 gpcutdxfp
112 gpcutdxfp gcp.stockandtoolpaths()

```

---

which creates the design:



and which allows an interactive usage in working up a design such as for lasercutting.

### 2.3 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 generatedxf = True
20 gcptmplpy # [Export] */
21 gcptmplpy generategcode = True

```

```

22 gcptmplpy
23 gcptmplpy # [Stock] */
24 gcptmplpy stockXwidth = 220
25 gcptmplpy # [Stock] */
26 gcptmplpy stockYheight = 150
27 gcptmplpy # [Stock] */
28 gcptmplpy stockZthickness = 8.35
29 gcptmplpy # [Stock] */
30 gcptmplpy zeroheight = "Top" # [Top, Bottom]
31 gcptmplpy # [Stock] */
32 gcptmplpy stockzero = "Center" # [Lower-Left, Center-Left, Top-Left, Center]
33 gcptmplpy # [Stock] */
34 gcptmplpy retractheight = 9
35 gcptmplpy
36 gcptmplpy # [CAM] */
37 gcptmplpy toolradius = 1.5875
38 gcptmplpy # [CAM] */
39 gcptmplpy large_square_tool_num = 201 # [0:0, 112:112, 102:102, 201:201]
40 gcptmplpy # [CAM] */
41 gcptmplpy small_square_tool_num = 102 # [0:0, 122:122, 112:112, 102:102]
42 gcptmplpy # [CAM] */
43 gcptmplpy large_ball_tool_num = 202 # [0:0, 111:111, 101:101, 202:202]
44 gcptmplpy # [CAM] */
45 gcptmplpy small_ball_tool_num = 101 # [0:0, 121:121, 111:111, 101:101]
46 gcptmplpy # [CAM] */
47 gcptmplpy large_V_tool_num = 301 # [0:0, 301:301, 690:690]
48 gcptmplpy # [CAM] */
49 gcptmplpy small_V_tool_num = 390 # [0:0, 390:390, 301:301]
50 gcptmplpy # [CAM] */
51 gcptmplpy DT_tool_num = 814 # [0:0, 814:814, 808079:808079]
52 gcptmplpy # [CAM] */
53 gcptmplpy KH_tool_num = 374 # [0:0, 374:374, 375:375, 376:376, 378:378]
54 gcptmplpy # [CAM] */
55 gcptmplpy Roundover_tool_num = 56142 # [56142:56142, 56125:56125, 1570:1570]
56 gcptmplpy # [CAM] */
57 gcptmplpy MISC_tool_num = 0 # [501:501, 502:502, 45982:45982]
58 gcptmplpy #501 https://shop.carbide3d.com/collections/cutters/products/501-
    engraving-bit
59 gcptmplpy #502 https://shop.carbide3d.com/collections/cutters/products/502-
    engraving-bit
60 gcptmplpy #204 tapered ball nose 0.0625", 0.2500", 1.50", 3.6ř
61 gcptmplpy #304 tapered ball nose 0.1250", 0.2500", 1.50", 2.4ř
62 gcptmplpy #648 threadmill_shaft(2.4, 0.75, 18)
63 gcptmplpy #45982 Carbide Tipped Bowl & Tray 1/4 Radius x 3/4 Dia x 5/8 x 1/4
    Inch Shank
64 gcptmplpy #13921 https://www.amazon.com/Yonico-Groove-Bottom-Router-Degree/dp
    /B0CPJPTMPP
65 gcptmplpy
66 gcptmplpy # [Feeds and Speeds] */
67 gcptmplpy plunge = 100
68 gcptmplpy # [Feeds and Speeds] */
69 gcptmplpy feed = 400
70 gcptmplpy # [Feeds and Speeds] */
71 gcptmplpy speed = 16000
72 gcptmplpy # [Feeds and Speeds] */
73 gcptmplpy small_square_ratio = 0.75 # [0.25:2]
74 gcptmplpy # [Feeds and Speeds] */
75 gcptmplpy large_ball_ratio = 1.0 # [0.25:2]
76 gcptmplpy # [Feeds and Speeds] */
77 gcptmplpy small_ball_ratio = 0.75 # [0.25:2]
78 gcptmplpy # [Feeds and Speeds] */
79 gcptmplpy large_V_ratio = 0.875 # [0.25:2]
80 gcptmplpy # [Feeds and Speeds] */
81 gcptmplpy small_V_ratio = 0.625 # [0.25:2]
82 gcptmplpy # [Feeds and Speeds] */
83 gcptmplpy DT_ratio = 0.75 # [0.25:2]
84 gcptmplpy # [Feeds and Speeds] */
85 gcptmplpy KH_ratio = 0.75 # [0.25:2]
86 gcptmplpy # [Feeds and Speeds] */
87 gcptmplpy RO_ratio = 0.5 # [0.25:2]
88 gcptmplpy # [Feeds and Speeds] */
89 gcptmplpy MISC_ratio = 0.5 # [0.25:2]
90 gcptmplpy
91 gcptmplpy gcp = gcodepreview(generategcode,
92 gcptmplpy generatedxf,
93 gcptmplpy )
94 gcptmplpy
95 gcptmplpy gcp.opengcodefile(Base_filename)

```

```

96 gcptmplpy gcp.opendxfile(Base_filename)
97 gcptmplpy gcp.opendxfiles(Base_filename,
98 gcptmplpy         large_square_tool_num,
99 gcptmplpy         small_square_tool_num,
100 gcptmplpy         large_ball_tool_num,
101 gcptmplpy         small_ball_tool_num,
102 gcptmplpy         large_V_tool_num,
103 gcptmplpy         small_V_tool_num,
104 gcptmplpy         DT_tool_num,
105 gcptmplpy         KH_tool_num,
106 gcptmplpy         Roundover_tool_num,
107 gcptmplpy         MISC_tool_num)
108 gcptmplpy gcp.setupstock(stockXwidth, stockYheight, stockZthickness,
        zeroheight, stockzero, retractheight)
109 gcptmplpy
110 gcptmplpy gcp.movetosafeZ()
111 gcptmplpy
112 gcptmplpy gcp.toolchange(102, 10000)
113 gcptmplpy
114 gcptmplpy gcp.rapidZ(0)
115 gcptmplpy
116 gcptmplpy gcp.cutlinedxfgc(stockXwidth/2, stockYheight/2, -stockZthickness)
117 gcptmplpy
118 gcptmplpy gcp.rapidZ(retractheight)
119 gcptmplpy gcp.toolchange(201, 10000)
120 gcptmplpy gcp.rapidXY(0, stockYheight/16)
121 gcptmplpy gcp.rapidZ(0)
122 gcptmplpy gcp.cutlinedxfgc(stockXwidth/16*7, stockYheight/2, -stockZthickness
        )
123 gcptmplpy
124 gcptmplpy gcp.rapidZ(retractheight)
125 gcptmplpy gcp.toolchange(202, 10000)
126 gcptmplpy gcp.rapidXY(0, stockYheight/8)
127 gcptmplpy gcp.rapidZ(0)
128 gcptmplpy gcp.cutlinedxfgc(stockXwidth/16*6, stockYheight/2, -stockZthickness
        )
129 gcptmplpy
130 gcptmplpy gcp.rapidZ(retractheight)
131 gcptmplpy gcp.toolchange(101, 10000)
132 gcptmplpy gcp.rapidXY(0, stockYheight/16*3)
133 gcptmplpy gcp.rapidZ(0)
134 gcptmplpy gcp.cutlinedxfgc(stockXwidth/16*5, stockYheight/2, -stockZthickness
        )
135 gcptmplpy
136 gcptmplpy gcp.setzpos(retractheight)
137 gcptmplpy gcp.toolchange(390, 10000)
138 gcptmplpy gcp.rapidXY(0, stockYheight/16*4)
139 gcptmplpy gcp.rapidZ(0)
140 gcptmplpy gcp.cutlinedxfgc(stockXwidth/16*4, stockYheight/2, -stockZthickness
        )
141 gcptmplpy gcp.rapidZ(retractheight)
142 gcptmplpy
143 gcptmplpy gcp.toolchange(301, 10000)
144 gcptmplpy gcp.rapidXY(0, stockYheight/16*6)
145 gcptmplpy gcp.rapidZ(0)
146 gcptmplpy gcp.cutlinedxfgc(stockXwidth/16*2, stockYheight/2, -stockZthickness
        )
147 gcptmplpy
148 gcptmplpy rapids = gcp.rapid(gcp.xpos(), gcp.ypos(), retractheight)
149 gcptmplpy gcp.toolchange(102, 10000)
150 gcptmplpy
151 gcptmplpy gcp.rapid(-stockXwidth/4+stockYheight/16, +stockYheight/4, 0)
152 gcptmplpy
153 gcptmplpy #gcp.cutarcCC(0, 90, gcp.xpos()-stockYheight/16, gcp.ypos(),
        stockYheight/16, -stockZthickness/4)
154 gcptmplpy #gcp.cutarcCC(90, 180, gcp.xpos(), gcp.ypos()-stockYheight/16,
        stockYheight/16, -stockZthickness/4)
155 gcptmplpy #gcp.cutarcCC(180, 270, gcp.xpos()+stockYheight/16, gcp.ypos(),
        stockYheight/16, -stockZthickness/4)
156 gcptmplpy #gcp.cutarcCC(270, 360, gcp.xpos(), gcp.ypos()+stockYheight/16,
        stockYheight/16, -stockZthickness/4)
157 gcptmplpy gcp.cutquarterCCNEdxf(gcp.xpos() - stockYheight/8, gcp.ypos() +
        stockYheight/8, -stockZthickness/4, stockYheight/8)
158 gcptmplpy gcp.cutquarterCCNWdxf(gcp.xpos() - stockYheight/8, gcp.ypos() -
        stockYheight/8, -stockZthickness/2, stockYheight/8)
159 gcptmplpy gcp.cutquarterCCSWdxf(gcp.xpos() + stockYheight/8, gcp.ypos() -
        stockYheight/8, -stockZthickness * 0.75, stockYheight/8)
160 gcptmplpy gcp.cutquarterCCSEdxf(gcp.xpos() + stockYheight/8, gcp.ypos() +

```

```

        stockYheight/8, -stockZthickness, stockYheight/8)
161 gcptmplpy
162 gcptmplpy gcp.movetosafeZ()
163 gcptmplpy gcp.rapidXY(stockXwidth/4-stockYheight/16, -stockYheight/4)
164 gcptmplpy gcp.rapidZ(0)
165 gcptmplpy
166 gcptmplpy
167 gcptmplpy #gcp.cutarcCW(180, 90, gcp.xpos()+stockYheight/16, gcp.ypos(),
        stockYheight/16, -stockZthickness/4)
168 gcptmplpy #gcp.cutarcCW(90, 0, gcp.xpos(), gcp.ypos()-stockYheight/16,
        stockYheight/16, -stockZthickness/4)
169 gcptmplpy #gcp.cutarcCW(360, 270, gcp.xpos()-stockYheight/16, gcp.ypos(),
        stockYheight/16, -stockZthickness/4)
170 gcptmplpy #gcp.cutarcCW(270, 180, gcp.xpos(), gcp.ypos()+stockYheight/16,
        stockYheight/16, -stockZthickness/4)
171 gcptmplpy
172 gcptmplpy #gcp.movetosafeZ()
173 gcptmplpy #gcp.toolchange(201, 10000)
174 gcptmplpy #gcp.rapidXY(stockXwidth/2, -stockYheight/2)
175 gcptmplpy #gcp.rapidZ(0)
176 gcptmplpy
177 gcptmplpy #gcp.cutlinedxfgc(gcp.xpos(), gcp.ypos(), -stockZthickness)
178 gcptmplpy #test = gcp.cutlinedxfgc(gcp.xpos(), gcp.ypos(), -stockZthickness)
179 gcptmplpy
180 gcptmplpy #gcp.movetosafeZ()
181 gcptmplpy #gcp.rapidXY(stockXwidth/2-6.34, -stockYheight/2)
182 gcptmplpy #gcp.rapidZ(0)
183 gcptmplpy
184 gcptmplpy #gcp.cutarcCW(180, 90, stockXwidth/2, -stockYheight/2, 6.34, -
        stockZthickness)
185 gcptmplpy
186 gcptmplpy
187 gcptmplpy gcp.movetosafeZ()
188 gcptmplpy gcp.toolchange(814, 10000)
189 gcptmplpy gcp.rapidXY(0, -(stockYheight/2+12.7))
190 gcptmplpy gcp.rapidZ(0)
191 gcptmplpy
192 gcptmplpy gcp.cutlinedxfgc(gcp.xpos(), gcp.ypos(), -stockZthickness)
193 gcptmplpy gcp.cutlinedxfgc(gcp.xpos(), -12.7, -stockZthickness)
194 gcptmplpy
195 gcptmplpy gcp.rapidXY(0, -(stockYheight/2+12.7))
196 gcptmplpy gcp.movetosafeZ()
197 gcptmplpy gcp.toolchange(374, 10000)
198 gcptmplpy gcp.rapidXY(stockXwidth/4-stockXwidth/16, -(stockYheight/4+
        stockYheight/16))
199 gcptmplpy gcp.rapidZ(0)
200 gcptmplpy
201 gcptmplpy gcp.rapidZ(retractheight)
202 gcptmplpy gcp.toolchange(374, 10000)
203 gcptmplpy gcp.rapidXY(-stockXwidth/4-stockXwidth/16, -(stockYheight/4+
        stockYheight/16))
204 gcptmplpy gcp.rapidZ(0)
205 gcptmplpy
206 gcptmplpy gcp.cutline(gcp.xpos(), gcp.ypos(), -stockZthickness/2)
207 gcptmplpy gcp.cutlinedxfgc(gcp.xpos()+stockYheight/9, gcp.ypos(), gcp.zpos())
208 gcptmplpy
209 gcptmplpy gcp.cutline(gcp.xpos()-stockYheight/9, gcp.ypos(), gcp.zpos())
210 gcptmplpy gcp.cutline(gcp.xpos(), gcp.ypos(), 0)
211 gcptmplpy
212 gcptmplpy #key = gcp.cutkeyholegcdxf(KH_tool_num, 0, stockZthickness*0.75, "E
        ", stockYheight/9)
213 gcptmplpy #key = gcp.cutKHgcdxf(374, 0, stockZthickness*0.75, 90,
        stockYheight/9)
214 gcptmplpy #toolpaths = toolpaths.union(key)
215 gcptmplpy
216 gcptmplpy gcp.rapidZ(retractheight)
217 gcptmplpy gcp.rapidXY(-stockXwidth/4+stockXwidth/16, -(stockYheight/4+
        stockYheight/16))
218 gcptmplpy gcp.rapidZ(0)
219 gcptmplpy gcp.cutline(gcp.xpos(), gcp.ypos(), -stockZthickness/2)
220 gcptmplpy gcp.cutlinedxfgc(gcp.xpos(), gcp.ypos()+stockYheight/9, gcp.zpos())
221 gcptmplpy
222 gcptmplpy gcp.cutline(gcp.xpos(), gcp.ypos()-stockYheight/9, gcp.zpos())
223 gcptmplpy gcp.cutline(gcp.xpos(), gcp.ypos(), 0)
224 gcptmplpy
225 gcptmplpy gcp.rapidZ(retractheight)
226 gcptmplpy gcp.rapidXY(-stockXwidth/4+stockXwidth/16, -(stockYheight/4-
        stockYheight/8))

```



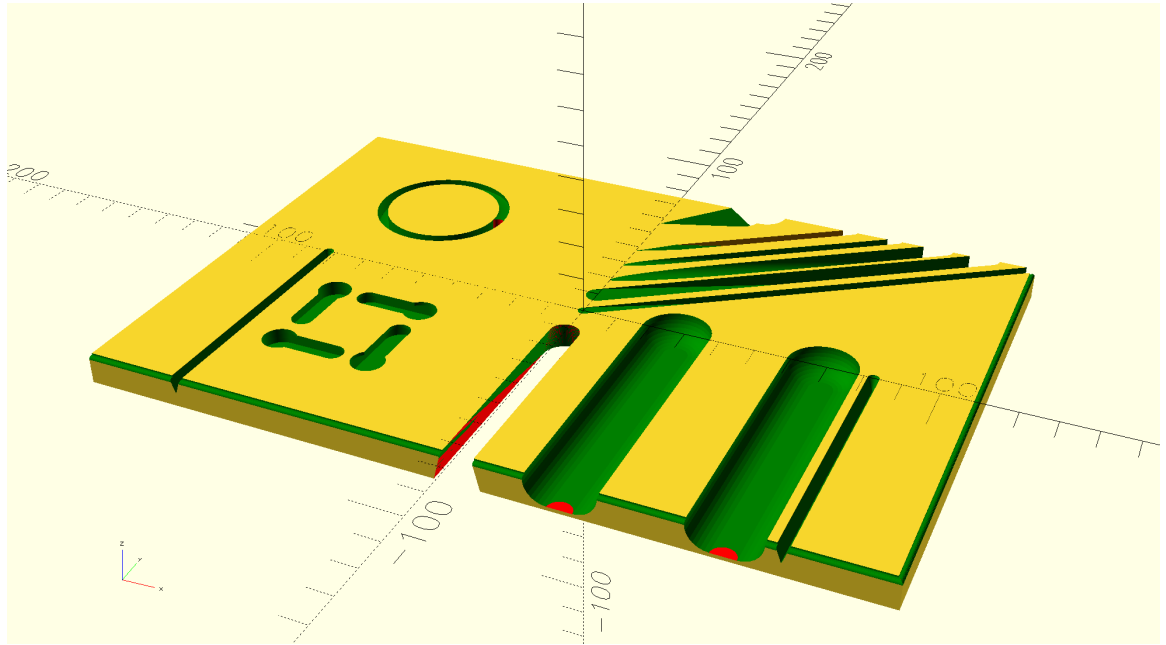
```

227 gcptmplpy gcp.rapidZ(0)
228 gcptmplpy
229 gcptmplpy gcp.cutline(gcp.xpos(), gcp.ypos(), -stockZthickness/2)
230 gcptmplpy gcp.cutlinedxfgc(gcp.xpos()-stockYheight/9, gcp.ypos(), gcp.zpos())
231 gcptmplpy
232 gcptmplpy gcp.cutline(gcp.xpos()+stockYheight/9, gcp.ypos(), gcp.zpos())
233 gcptmplpy gcp.cutline(gcp.xpos(), gcp.ypos(), 0)
234 gcptmplpy
235 gcptmplpy gcp.rapidZ(retractheight)
236 gcptmplpy gcp.rapidXY(-stockXwidth/4-stockXwidth/16, -(stockYheight/4-
    stockYheight/8))
237 gcptmplpy gcp.rapidZ(0)
238 gcptmplpy gcp.cutline(gcp.xpos(), gcp.ypos(), -stockZthickness/2)
239 gcptmplpy gcp.cutlinedxfgc(gcp.xpos(), gcp.ypos()-stockYheight/9, gcp.zpos())
240 gcptmplpy gcp.cutline(gcp.xpos(), gcp.ypos()+stockYheight/9, gcp.zpos())
241 gcptmplpy gcp.cutline(gcp.xpos(), gcp.ypos(), 0)
242 gcptmplpy
243 gcptmplpy gcp.rapidZ(retractheight)
244 gcptmplpy gcp.toolchange(56142, 10000)
245 gcptmplpy gcp.rapidXY(-stockXwidth/2, -(stockYheight/2+0.508/2))
246 gcptmplpy gcp.cutline(gcp.xpos(), gcp.ypos(), -1.531)
247 gcptmplpy gcp.cutlinedxfgc(stockXwidth/2+0.508/2, -(stockYheight/2+0.508/2),
    -1.531)
248 gcptmplpy
249 gcptmplpy gcp.rapidZ(retractheight)
250 gcptmplpy
251 gcptmplpy gcp.cutline(gcp.xpos(), gcp.ypos(), -1.531)
252 gcptmplpy gcp.cutlinedxfgc(stockXwidth/2+0.508/2, (stockYheight/2+0.508/2),
    -1.531)
253 gcptmplpy
254 gcptmplpy gcp.rapidZ(retractheight)
255 gcptmplpy gcp.toolchange(45982, 10000)
256 gcptmplpy gcp.rapidXY(stockXwidth/8, 0)
257 gcptmplpy gcp.cutline(gcp.xpos(), gcp.ypos(), -(stockZthickness*7/8))
258 gcptmplpy gcp.cutlinedxfgc(gcp.xpos(), -stockYheight/2, -(stockZthickness
    *7/8))
259 gcptmplpy
260 gcptmplpy gcp.rapidZ(retractheight)
261 gcptmplpy gcp.toolchange(204, 10000)
262 gcptmplpy gcp.rapidXY(stockXwidth*0.3125, 0)
263 gcptmplpy gcp.cutline(gcp.xpos(), gcp.ypos(), -(stockZthickness*7/8))
264 gcptmplpy gcp.cutlinedxfgc(gcp.xpos(), -stockYheight/2, -(stockZthickness
    *7/8))
265 gcptmplpy
266 gcptmplpy gcp.rapidZ(retractheight)
267 gcptmplpy gcp.toolchange(502, 10000)
268 gcptmplpy gcp.rapidXY(stockXwidth*0.375, 0)
269 gcptmplpy gcp.cutline(gcp.xpos(), gcp.ypos(), -4.24)
270 gcptmplpy gcp.cutlinedxfgc(gcp.xpos(), -stockYheight/2, -4.24)
271 gcptmplpy
272 gcptmplpy gcp.rapidZ(retractheight)
273 gcptmplpy gcp.toolchange(13921, 10000)
274 gcptmplpy gcp.rapidXY(-stockXwidth*0.375, 0)
275 gcptmplpy gcp.cutline(gcp.xpos(), gcp.ypos(), -stockZthickness/2)
276 gcptmplpy gcp.cutlinedxfgc(gcp.xpos(), -stockYheight/2, -stockZthickness/2)
277 gcptmplpy
278 gcptmplpy gcp.rapidZ(retractheight)
279 gcptmplpy
280 gcptmplpy gcp.stockandtoolpaths()
281 gcptmplpy
282 gcptmplpy gcp.closegcodefile()
283 gcptmplpy gcp.closedxfiles()
284 gcptmplpy gcp.closedxfile()

```

---

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



## 2.4 gcodepreviewtemplate.scad

Since the project began in OpenSCAD, having an implementation in that language has always been a goal. This is quite straight-forward since the Python code when imported into OpenSCAD may be accessed by quite simple modules which are for the most part, a series of decorators/de-descriptors which wrap up the Python definitions as OpenSCAD modules. Moreover, such an implementation will facilitate usage by tools intended for this application such as OpenSCAD Graph Editor: <https://github.com/derkork/openscad-graph-editor>.

---

```

1 gcptmpl #!/OpenSCAD
2 gcptmpl
3 gcptmpl use <gcodepreview.py>
4 gcptmpl include <gcodepreview.scad>
5 gcptmpl
6 gcptmpl $fa = 2;
7 gcptmpl $fs = 0.125;
8 gcptmpl fa = 2;
9 gcptmpl fs = 0.125;
10 gcptmpl
11 gcptmpl /* [Stock] */
12 gcptmpl stockXwidth = 219;
13 gcptmpl /* [Stock] */
14 gcptmpl stockYheight = 150;
15 gcptmpl /* [Stock] */
16 gcptmpl stockZthickness = 8.35;
17 gcptmpl /* [Stock] */
18 gcptmpl zeroheight = "Top"; // [Top, Bottom]
19 gcptmpl /* [Stock] */
20 gcptmpl stockzero = "Center"; // [Lower-Left, Center-Left, Top-Left, Center
    ]
21 gcptmpl /* [Stock] */
22 gcptmpl retractheight = 9;
23 gcptmpl
24 gcptmpl /* [Export] */
25 gcptmpl Base_filename = "export";
26 gcptmpl /* [Export] */
27 gcptmpl generatedxf = true;
28 gcptmpl /* [Export] */
29 gcptmpl generategcode = true;
30 gcptmpl
31 gcptmpl /* [CAM] */
32 gcptmpl toolradius = 1.5875;
33 gcptmpl /* [CAM] */
34 gcptmpl large_square_tool_num = 0; // [0:0, 112:112, 102:102, 201:201]
35 gcptmpl /* [CAM] */
36 gcptmpl small_square_tool_num = 102; // [0:0, 122:122, 112:112, 102:102]
37 gcptmpl /* [CAM] */
38 gcptmpl large_ball_tool_num = 0; // [0:0, 111:111, 101:101, 202:202]
39 gcptmpl /* [CAM] */
40 gcptmpl small_ball_tool_num = 0; // [0:0, 121:121, 111:111, 101:101]
41 gcptmpl /* [CAM] */
42 gcptmpl large_V_tool_num = 0; // [0:0, 301:301, 690:690]
43 gcptmpl /* [CAM] */

```

```

44 gcptmpl small_V_tool_num = 0; // [0:0, 390:390, 301:301]
45 gcptmpl /* [CAM] */
46 gcptmpl DT_tool_num = 0; // [0:0, 814:814, 808079:808079]
47 gcptmpl /* [CAM] */
48 gcptmpl KH_tool_num = 0; // [0:0, 374:374, 375:375, 376:376, 378:378]
49 gcptmpl /* [CAM] */
50 gcptmpl Roundover_tool_num = 0; // [56142:56142, 56125:56125, 1570:1570]
51 gcptmpl /* [CAM] */
52 gcptmpl MISC_tool_num = 0; // [648:648, 45982:45982]
53 gcptmpl //648 threadmill_shaft(2.4, 0.75, 18)
54 gcptmpl //45982 Carbide Tipped Bowl & Tray 1/4 Radius x 3/4 Dia x 5/8 x 1/4
      Inch Shank
55 gcptmpl
56 gcptmpl /* [Feeds and Speeds] */
57 gcptmpl plunge = 100;
58 gcptmpl /* [Feeds and Speeds] */
59 gcptmpl feed = 400;
60 gcptmpl /* [Feeds and Speeds] */
61 gcptmpl speed = 16000;
62 gcptmpl /* [Feeds and Speeds] */
63 gcptmpl small_square_ratio = 0.75; // [0.25:2]
64 gcptmpl /* [Feeds and Speeds] */
65 gcptmpl large_ball_ratio = 1.0; // [0.25:2]
66 gcptmpl /* [Feeds and Speeds] */
67 gcptmpl small_ball_ratio = 0.75; // [0.25:2]
68 gcptmpl /* [Feeds and Speeds] */
69 gcptmpl large_V_ratio = 0.875; // [0.25:2]
70 gcptmpl /* [Feeds and Speeds] */
71 gcptmpl small_V_ratio = 0.625; // [0.25:2]
72 gcptmpl /* [Feeds and Speeds] */
73 gcptmpl DT_ratio = 0.75; // [0.25:2]
74 gcptmpl /* [Feeds and Speeds] */
75 gcptmpl KH_ratio = 0.75; // [0.25:2]
76 gcptmpl /* [Feeds and Speeds] */
77 gcptmpl RO_ratio = 0.5; // [0.25:2]
78 gcptmpl /* [Feeds and Speeds] */
79 gcptmpl MISC_ratio = 0.5; // [0.25:2]
80 gcptmpl
81 gcptmpl thegeneratedxf = generatedxf == true ? 1 : 0;
82 gcptmpl thegenerategcode = generategcode == true ? 1 : 0;
83 gcptmpl
84 gcptmpl gcp = gcodepreview(thegenerategcode,
85 gcptmpl                      thegeneratedxf,
86 gcptmpl                      );
87 gcptmpl
88 gcptmpl.opengcodefile(Base_filename);
89 gcptmpl.opendxxfile(Base_filename);
90 gcptmpl.opendxxfiles(Base_filename,
91 gcptmpl                      large_square_tool_num,
92 gcptmpl                      small_square_tool_num,
93 gcptmpl                      large_ball_tool_num,
94 gcptmpl                      small_ball_tool_num,
95 gcptmpl                      large_V_tool_num,
96 gcptmpl                      small_V_tool_num,
97 gcptmpl                      DT_tool_num,
98 gcptmpl                      KH_tool_num,
99 gcptmpl                      Roundover_tool_num,
100 gcptmpl                      MISC_tool_num);
101 gcptmpl
102 gcptmpl.setupstock(stockXwidth, stockYheight, stockZthickness, zeroheight,
      stockzero);
103 gcptmpl
104 gcptmpl //echo(gcp);
105 gcptmpl //gcpversion();
106 gcptmpl
107 gcptmpl //c = myfunc(4);
108 gcptmpl //echo(c);
109 gcptmpl
110 gcptmpl //echo(getvv());
111 gcptmpl
112 gcptmpl.outline(stockXwidth/2, stockYheight/2, -stockZthickness);
113 gcptmpl
114 gcptmpl.rapidZ(retractheight);
115 gcptmpl.toolchange(201, 10000);
116 gcptmpl.rapidXY(0, stockYheight/16);
117 gcptmpl.rapidZ(0);
118 gcptmpl.cutlinedxfgc(stockXwidth/16*7, stockYheight/2, -stockZthickness);
119 gcptmpl

```

```

120 gcptmpl
121 gcptmpl rapidZ(retractheight);
122 gcptmpl toolchange(202, 10000);
123 gcptmpl rapidXY(0, stockYheight/8);
124 gcptmpl rapidZ(0);
125 gcptmpl cutlinedxfgc(stockXwidth/16*6, stockYheight/2, -stockZthickness);
126 gcptmpl
127 gcptmpl rapidZ(retractheight);
128 gcptmpl toolchange(101, 10000);
129 gcptmpl rapidXY(0, stockYheight/16*3);
130 gcptmpl rapidZ(0);
131 gcptmpl cutlinedxfgc(stockXwidth/16*5, stockYheight/2, -stockZthickness);
132 gcptmpl
133 gcptmpl rapidZ(retractheight);
134 gcptmpl toolchange(390, 10000);
135 gcptmpl rapidXY(0, stockYheight/16*4);
136 gcptmpl rapidZ(0);
137 gcptmpl
138 gcptmpl cutlinedxfgc(stockXwidth/16*4, stockYheight/2, -stockZthickness);
139 gcptmpl rapidZ(retractheight);
140 gcptmpl
141 gcptmpl toolchange(301, 10000);
142 gcptmpl rapidXY(0, stockYheight/16*6);
143 gcptmpl rapidZ(0);
144 gcptmpl
145 gcptmpl cutlinedxfgc(stockXwidth/16*2, stockYheight/2, -stockZthickness);
146 gcptmpl
147 gcptmpl
148 gcptmpl movetosafeZ();
149 gcptmpl rapid(gcp.xpos(), gcp.ypos(), retractheight);
150 gcptmpl toolchange(102, 10000);
151 gcptmpl
152 gcptmpl //rapidXY(stockXwidth/4+stockYheight/8+stockYheight/16, +
           stockYheight/8);
153 gcptmpl rapidXY(-stockXwidth/4+stockXwidth/16, (stockYheight/4));//+
           stockYheight/16
154 gcptmpl rapidZ(0);
155 gcptmpl
156 gcptmpl //cutarcCW(360, 270, gcp.xpos()-stockYheight/16, gcp.ypos(),
           stockYheight/16, -stockZthickness);
157 gcptmpl //gcp.cutarcCW(270, 180, gcp.xpos(), gcp.ypos()+stockYheight/16,
           stockYheight/16))
158 gcptmpl cutarcCC(0, 90, gcp.xpos()-stockYheight/16, gcp.ypos(),
           stockYheight/16, -stockZthickness/4);
159 gcptmpl cutarcCC(90, 180, gcp.xpos(), gcp.ypos()-stockYheight/16,
           stockYheight/16, -stockZthickness/4);
160 gcptmpl cutarcCC(180, 270, gcp.xpos()+stockYheight/16, gcp.ypos(),
           stockYheight/16, -stockZthickness/4);
161 gcptmpl cutarcCC(270, 360, gcp.xpos(), gcp.ypos()+stockYheight/16,
           stockYheight/16, -stockZthickness/4);
162 gcptmpl
163 gcptmpl movetosafeZ();
164 gcptmpl //rapidXY(stockXwidth/4+stockYheight/8-stockYheight/16, -
           stockYheight/8);
165 gcptmpl rapidXY(stockXwidth/4-stockYheight/16, -(stockYheight/4));
166 gcptmpl rapidZ(0);
167 gcptmpl
168 gcptmpl cutarcCW(180, 90, gcp.xpos()+stockYheight/16, gcp.ypos(),
           stockYheight/16, -stockZthickness/4);
169 gcptmpl cutarcCW(90, 0, gcp.xpos(), gcp.ypos()-stockYheight/16,
           stockYheight/16, -stockZthickness/4);
170 gcptmpl cutarcCW(360, 270, gcp.xpos()-stockYheight/16, gcp.ypos(),
           stockYheight/16, -stockZthickness/4);
171 gcptmpl cutarcCW(270, 180, gcp.xpos(), gcp.ypos()+stockYheight/16,
           stockYheight/16, -stockZthickness/4);
172 gcptmpl
173 gcptmpl movetosafeZ();
174 gcptmpl toolchange(201, 10000);
175 gcptmpl rapidXY(stockXwidth /2 -6.34, - stockYheight /2);
176 gcptmpl rapidZ(0);
177 gcptmpl cutarcCW(180, 90, stockXwidth /2, -stockYheight/2, 6.34, -
           stockZthickness);
178 gcptmpl
179 gcptmpl movetosafeZ();
180 gcptmpl rapidXY(stockXwidth/2, -stockYheight/2);
181 gcptmpl rapidZ(0);
182 gcptmpl
183 gcptmpl gcp.cutlinedxfgc(gcp.xpos(), gcp.ypos(), -stockZthickness);

```

```

184 gcptmpl
185 gcptmpl movetosafeZ();
186 gcptmpl toolchange(814, 10000);
187 gcptmpl rapidXY(0, -(stockYheight/2+12.7));
188 gcptmpl rapidZ(0);
189 gcptmpl
190 gcptmpl cutlinedxfgc(xpos(), ypos(), -stockZthickness);
191 gcptmpl cutlinedxfgc(xpos(), -12.7, -stockZthickness);
192 gcptmpl rapidXY(0, -(stockYheight/2+12.7));
193 gcptmpl
194 gcptmpl //rapidXY(stockXwidth/2-6.34, -stockYheight/2);
195 gcptmpl //rapidZ(0);
196 gcptmpl
197 gcptmpl //movetosafeZ();
198 gcptmpl //toolchange(374, 10000);
199 gcptmpl //rapidXY(-(stockXwidth/4 - stockXwidth /16), -(stockYheight/4 +
      stockYheight/16))

200 gcptmpl
201 gcptmpl //cutline(xpos(), ypos(), (stockZthickness/2) * -1);
202 gcptmpl //cutlinedxfgc(xpos() + stockYheight /9, ypos(), zpos());
203 gcptmpl //cutline(xpos() - stockYheight /9, ypos(), zpos());
204 gcptmpl //cutline(xpos(), ypos(), 0);
205 gcptmpl
206 gcptmpl movetosafeZ();
207 gcptmpl
208 gcptmpl toolchange(374, 10000);
209 gcptmpl rapidXY(-stockXwidth/4-stockXwidth/16, -(stockYheight/4+
      stockYheight/16))
210 gcptmpl //rapidXY(-(stockXwidth/4 - stockXwidth /16), -(stockYheight/4 +
      stockYheight/16))
211 gcptmpl rapidZ(0);
212 gcptmpl
213 gcptmpl cutline(xpos(), ypos(), (stockZthickness/2) * -1);
214 gcptmpl cutlinedxfgc(xpos() + stockYheight /9, ypos(), zpos());
215 gcptmpl cutline(xpos() - stockYheight /9, ypos(), zpos());
216 gcptmpl cutline(xpos(), ypos(), 0);
217 gcptmpl
218 gcptmpl rapidZ(retractheight);
219 gcptmpl rapidXY(-stockXwidth/4+stockXwidth/16, -(stockYheight/4+
      stockYheight/16));
220 gcptmpl rapidZ(0);
221 gcptmpl cutline(gcp.xpos(), gcp.ypos(), -stockZthickness/2);
222 gcptmpl cutlinedxfgc(gcp.xpos(), gcp.ypos()+stockYheight/9, gcp.zpos());
223 gcptmpl cutline(gcp.xpos(), gcp.ypos()-stockYheight/9, gcp.zpos());
224 gcptmpl cutline(gcp.xpos(), gcp.ypos(), 0);
225 gcptmpl
226 gcptmpl rapidZ(retractheight);
227 gcptmpl rapidXY(-stockXwidth/4+stockXwidth/16, -(stockYheight/4-
      stockYheight/8));
228 gcptmpl rapidZ(0);
229 gcptmpl cutline(gcp.xpos(), gcp.ypos(), -stockZthickness/2);
230 gcptmpl cutlinedxfgc(gcp.xpos()-stockYheight/9, gcp.ypos(), gcp.zpos());
231 gcptmpl cutline(gcp.xpos()+stockYheight/9, gcp.ypos(), gcp.zpos());
232 gcptmpl cutline(gcp.xpos(), gcp.ypos(), 0);
233 gcptmpl
234 gcptmpl rapidZ(retractheight);
235 gcptmpl rapidXY(-stockXwidth/4-stockXwidth/16, -(stockYheight/4-
      stockYheight/8));
236 gcptmpl rapidZ(0);
237 gcptmpl cutline(gcp.xpos(), gcp.ypos(), -stockZthickness/2);
238 gcptmpl cutlinedxfgc(gcp.xpos(), gcp.ypos()-stockYheight/9, gcp.zpos());
239 gcptmpl cutline(gcp.xpos(), gcp.ypos()+stockYheight/9, gcp.zpos());
240 gcptmpl cutline(gcp.xpos(), gcp.ypos(), 0);
241 gcptmpl
242 gcptmpl
243 gcptmpl
244 gcptmpl rapidZ(retractheight);
245 gcptmpl gcp.toolchange(56142, 10000);
246 gcptmpl gcp.rapidXY(-stockXwidth/2, -(stockYheight/2+0.508/2));
247 gcptmpl cutlineZgcfeed(-1.531, plunge);
248 gcptmpl //cutline(gcp.xpos(), gcp.ypos(), -1.531);
249 gcptmpl cutlinedxfgc(stockXwidth/2+0.508/2, -(stockYheight/2+0.508/2),
      -1.531);

250 gcptmpl
251 gcptmpl rapidZ(retractheight);
252 gcptmpl //gcp.toolchange(56125, 10000)
253 gcptmpl cutlineZgcfeed(-1.531, plunge);
254 gcptmpl //toolpaths.append(gcp.cutline(gcp.xpos(), gcp.ypos(), -1.531))

```

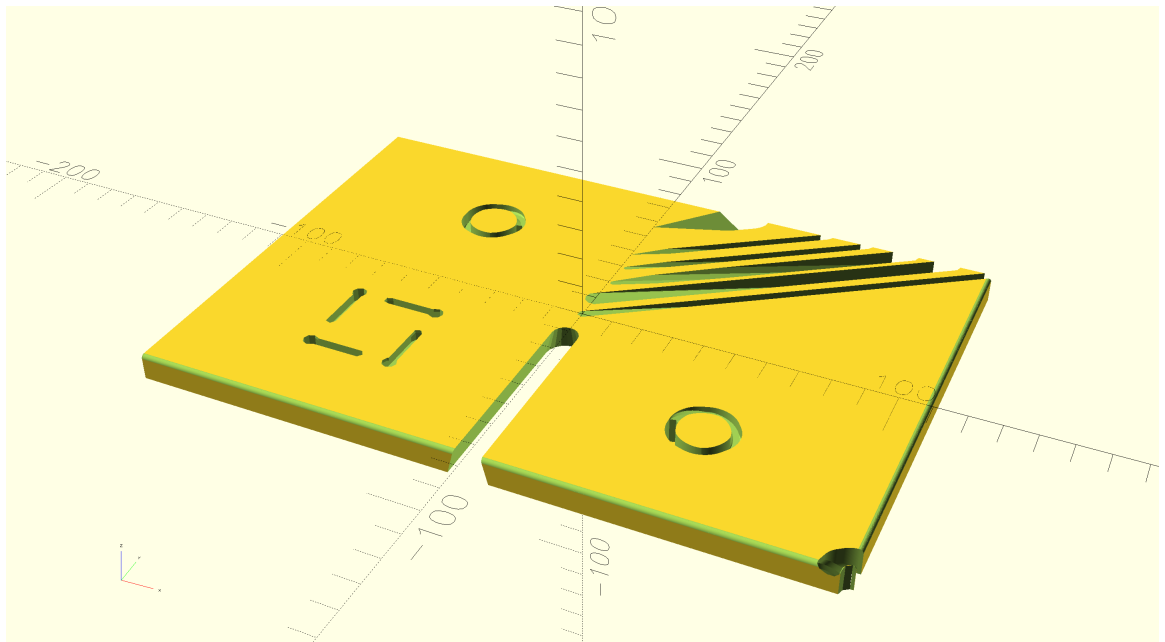
```

255 gcptmpl cutlinedxfgc(stockXwidth/2+0.508/2, (stockYheight/2+0.508/2),
    -1.531);
256 gcptmpl
257 gcptmpl stockandtoolpaths();
258 gcptmpl //stockwotoolpaths();
259 gcptmpl //outputtoolpaths();
260 gcptmpl
261 gcptmpl //makecube(3, 2, 1);
262 gcptmpl
263 gcptmpl //instantiatecube();
264 gcptmpl
265 gcptmpl closegcodefile();
266 gcptmpl closedxfiles();
267 gcptmpl closedxfile();

```

---

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



### 3 *gcodepreview*

This library for PythonSCAD works by using Python code as a back-end so as to persistently store and access variables, and to write out files while both modeling the motion of a 3-axis CNC machine (note that at least a 4<sup>th</sup> additional axis may be worked up as a future option and supporting the work-around of two-sided (flip) machining by using an imported file as the Stock or preserving state and affording a second operation seems promising) and if desired, writing out DXF and/or G-code files (as opposed to the normal technique of rendering to a 3D model and writing out an STL or STEP or other model format and using a traditional CAM application). There are multiple modes for this, doing so may require at least two files:

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

```
import gcodepreview_standalone as gcp
```

or used in an OpenSCAD file:  

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

with an additional OpenSCAD module which allows accessing it and that there is an option for loading directly from the Github repository implemented in PythonSCAD
- An OpenSCAD file: *gcodepreview.scad* (*gcpsc*) — 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.

In earlier versions there were several possible ways to work with the 3D models of the cuts, either directly displaying the returned 3D model when explicitly called for after storing it in a variable or calling it up as a calculation (Python command `ouput(<foo>)` or OpenSCAD returning a model, or calling an appropriate OpenSCAD command), however as-of v0.9 the tool movements

are modeled as lists of `hull()` operations which must be processed as such and are differenced from the stock. The templates set up these options as noted, and ensure that `True == true`.

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

### 3.1 Module Naming Convention

The original implementation required three files and used a convention for prefacing commands with `o` or `p`, but this requirement was obviated in the full Python re-write. The current implementation depends upon the class being instantiated as `gcp` as a sufficient differentiation between the Python and the OpenSCAD versions of commands which will otherwise share the same name.

Number will be abbreviated as `num` rather than `no`, and the short form will be used internally for variable names, while the complete word will be used in commands.

In some instances, `the` will be used as a prefix.

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 `currenttoolnumber()` may be used to access that number, and is arguably the preferred mechanism. An exception is when there are multiple tool `#s` as when opening a file — collecting them all at the end is a more straight-forward approach.

In natural languages such as English, there is an order to various parts of speech such as adjectives — since various prefixes and suffixes will be used for module names, having a consistent ordering/usage will help in consistency and make expression clearer. The ordering should be: sequence (if necessary), action, function, parameter, filetype, and where possible a hierarchy of large/general to small/specific should be maintained.

- Both prefix and suffix
  - `dxf` (action (write out to `DXF` file), filetype)
- Prefixes
  - `generate` (Boolean) — used to identify which types of actions will be done (note that in the interest of brevity the check for this will be deferred until the last possible moment, see below)
  - `write` (action) — used to write to files, will include a check for the matching `generate` command, which being `true` will cause the write to the file to actually transpire
  - `cut` (action — create tool movement removing volume from 3D object)
  - `rapid` (action — create tool movement of 3D object so as to show any collision or rubbing)
  - `open` (action (file))
  - `close` (action (file))
  - `set` (action/function) — note that the matching `get` is implicit in functions which return variables, e.g., `xpos()`
  - `current`
- Nouns (shapes)
  - `arc`
  - `line`
  - `rectangle`
  - `circle`
- Suffixes
  - `feed` (parameter)
  - `gcode/gc` (filetype)
  - `pos` — position
  - `tool`
  - `loop`
  - `CC/CW`
  - `number/num` — note that `num` is used internally for variable names, while `number` will be used for module/function names, making it straight-forward to ensure that functions and variables have different names for purposes of scope

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

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

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

Note that certain commands (`dxlinegc`, `dxarcgc`, `linegc`, `arcgc`) are either redundant or unlikely to be needed, and will most likely not be implemented (it seems contradictory that one would write out a move command to a G-code file without making that cut in the 3D preview). Note that there may be additional versions as required for the convenience of notation or cutting, in particular, a set of `cutarc<quadrant><direction>gc` commands was warranted during the initial development of arc-related commands.

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

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

Principles for naming modules (and variables):

- minimize use of underscores (for convenience sake, underscores are not used for index entries)
- identify which aspect of the project structure is being worked with (`cut(ting)`, `dx`, `gcode`, `tool`, etc.) note the `gcodepreview` class which will normally be imported as `gcp` so that module `<foo>` will be called as `gcp.<foo>` from Python and by the same `<foo>` in OpenSCAD

The following commands for various shapes either have been implemented (monospace) or have not yet been implemented, but likely will need to be (regular type):

- rectangle

cutrectangle

cutrectangleround

Another consideration is that all commands which write files will check to see if a given filetype is enabled or no, since that check is deferred to the last as noted above for the sake of conciseness.

There are multiple modes for programming PythonSCAD:

- Python — in `gcodepreview` this allows writing out `dx` files
- OpenSCAD — see: <https://openscad.org/documentation.html>
- Programming in OpenSCAD with variables and calling Python — this requires 3 files and was originally used in the project as written up at: [https://github.com/WillAdams/gcodepreview/blob/main/gcodepreview-openscad\\_0\\_6.pdf](https://github.com/WillAdams/gcodepreview/blob/main/gcodepreview-openscad_0_6.pdf) (for further details see below, notably various commented out lines in the source `.tex` file)
- Programming in OpenSCAD and calling Python where all variables as variables are held in Python classes (this is the technique used as of v0.8)
- Programming in Python and calling OpenSCAD — [https://old.reddit.com/r/OpenPythonSCAD/comments/1heczmi/finally\\_using\\_scad\\_modules/](https://old.reddit.com/r/OpenPythonSCAD/comments/1heczmi/finally_using_scad_modules/)

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

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

\lstset{firstnumber=\thegcpscad}
```



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

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

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

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

```
\end{writecode}
\addtocounter{pyscads}{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{gcpsy}{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 for versions after v0.6 (see below) and that this technique was extended to class nested within another class.

### 3.1.1 Parameters and Default Values

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

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

In short, rather than hard-code numbers, for example in loops, they will be assigned as default values, and thus afford the user/programmer the option of changing them when the module is called.

## 3.2 Implementation files and *gcodepreview* class

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

---

```
1 gcpy #!/usr/bin/env python
2 gcpy #icon "C:\Program Files\PythonSCAD\bin\openscad.exe" --trust-python
3 gcpy #Currently tested with https://www.pythonscad.org/downloads/
    PythonSCAD_nolibfive-2025.06.04-x86-64-Installer.exe and Python
    3.11
4 gcpy #gcodepreview 0.9, for use with PythonSCAD,
5 gcpy #if using from PythonSCAD using OpenSCAD code, see gcodepreview.
    scad
6 gcpy
7 gcpy import sys
8 gcpy
9 gcpy # add math functions (sqrt)
10 gcpy import math
11 gcpy
12 gcpy # getting openscad functions into namespace
13 gcpy #https://github.com/gsohler/openscad/issues/39
14 gcpy try:
15 gcpy     from openscad import *
16 gcpy except ModuleNotFoundError as e:
17 gcpy     print("OpenSCAD module not loaded.")
18 gcpy
19 gcpy def pygcpversion():
20 gcpy     thegcpversion = 0.9
21 gcpy     return thegcpversion
```

---

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

---

```

1 gcpscad #!/OpenSCAD
2 gcpscad
3 gcpscad //gcodepreview version 0.8
4 gcpscad //
5 gcpscad //used via include <gcodepreview.scad>;
6 gcpscad //
7 gcpscad
8 gcpscad use <gcodepreview.py>
9 gcpscad
10 gcpscad module gcpversion(){
11 gcpscad echo(pygcpversion());
12 gcpscad }
13 gcpscad
14 gcpscad //function myfunc(var) = gcp.myfunc(var);
15 gcpscad //
16 gcpscad //function getvv() = gcp.getvv();
17 gcpscad //
18 gcpscad //module makecube(xdim, ydim, zdim){
19 gcpscad //gcp.makecube(xdim, ydim, zdim);
20 gcpscad //}
21 gcpscad //
22 gcpscad //module placecube(){
23 gcpscad //gcp.placecube();
24 gcpscad //}
25 gcpscad //
26 gcpscad //module instantiatecube(){
27 gcpscad //gcp.instantiatecube();
28 gcpscad //}
29 gcpscad //

```

---

If all functions are to be handled within Python, then they will need to be gathered into a class which contains them and which is initialized so as to define shared variables and initial program state, and then there will need to be objects/commands for each aspect of the program, each of which will utilise needed variables and will contain appropriate functionality. Note that they will be divided between mandatory and optional functions/variables/objects:

- Mandatory

- stocksetup:
  - \* stockXwidth, stockYheight, stockZthickness, zeroheight, stockzero, retractheight
- gcpfiles:
  - \* basefilename, generatedxf, generategcode
- largesquaretool:
  - \* large\_square\_tool\_num, toolradius, plunge, feed, speed
- currenttoolnum
  - \* endmilltype
  - \* diameter
  - \* flute
  - \* shaftdiameter
  - \* shaftheight
  - \* shaftlength
  - \* toolnumber
  - \* cutcolor
  - \* rapidcolor
  - \* shaftcolor

- Optional

- smallsquaretool:
  - \* small\_square\_tool\_num, small\_square\_ratio
- largeballtool:
  - \* large\_ball\_tool\_num, large\_ball\_ratio
- largeVtool:
  - \* large\_V\_tool\_num, large\_V\_ratio
- smallballtool:
  - \* small\_ball\_tool\_num, small\_ball\_ratio
- smallVtool:
  - \* small\_V\_tool\_num, small\_V\_ratio

- DTtool:
  - \* DT\_tool\_num, DT\_ratio
- KHtool:
  - \* KH\_tool\_num, KH\_ratio
- Roundovertool:
  - \* Roundover\_tool\_num, RO\_ratio
- misctool:
  - \* MISC\_tool\_num, MISC\_ratio

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

---

```

23 gcpy class gcodepreview:
24 gcpy
25 gcpy     def __init__(self,
26 gcpy                 generategcode = False,
27 gcpy                 generatedxf = False,
28 gcpy                 gcpfa = 2,
29 gcpy                 gcpfs = 0.125,
30 gcpy                 steps = 10
31 gcpy                 ):
32 gcpy         """
33 gcpy         Initialize gcodepreview object.
34 gcpy
35 gcpy         Parameters
36 gcpy         -----
37 gcpy         generategcode : boolean
38 gcpy                        Enables writing out G-code.
39 gcpy         generatedxf   : boolean
40 gcpy                        Enables writing out DXF file(s).
41 gcpy
42 gcpy         Returns
43 gcpy         -----
44 gcpy         object
45 gcpy         The initialized gcodepreview object.
46 gcpy         """
47 gcpy         if generategcode == 1:
48 gcpy             self.generategcode = True
49 gcpy         elif generategcode == 0:
50 gcpy             self.generategcode = False
51 gcpy         else:
52 gcpy             self.generategcode = generategcode
53 gcpy         if generatedxf == 1:
54 gcpy             self.generatedxf = True
55 gcpy         elif generatedxf == 0:
56 gcpy             self.generatedxf = False
57 gcpy         else:
58 gcpy             self.generatedxf = generatedxf
59 gcpy # unless multiple dxfs are enabled, the check for them is of course
        False
60 gcpy         self.generatedxfs = False
61 gcpy # set up 3D previewing parameters
62 gcpy         fa = gcpfa
63 gcpy         fs = gcpfs
64 gcpy         self.steps = steps
65 gcpy # initialize the machine state
66 gcpy         self.mc = "Initialized"
67 gcpy         self.mpx = float(0)
68 gcpy         self.mpy = float(0)
69 gcpy         self.mpz = float(0)
70 gcpy         self.tpz = float(0)
71 gcpy # initialize the toolpath state
72 gcpy         self.retractheight = 5
73 gcpy # initialize the DEFAULT tool
74 gcpy         self.currenttoolnum = 102
75 gcpy         self.endmilltype = "square"
76 gcpy         self.diameter = 3.175
77 gcpy         self.flute = 12.7
78 gcpy         self.shaftdiameter = 3.175
79 gcpy         self.shaftheight = 12.7
80 gcpy         self.shaftlength = 19.5
81 gcpy         self.toolnumber = "100036"
82 gcpy         self.cutcolor = "green"
83 gcpy         self.rapidcolor = "orange"

```

```
84 gcpy          self.shaftcolor = "red"
85 gcpy # the variables for holding 3D models must be initialized as empty
      lists so as to ensure that only append or extend commands are
      used with them
86 gcpy          self.rapids = []
87 gcpy          self.toolpaths = []
88 gcpy
89 gcpy #      def myfunc(self, var):
90 gcpy #          self.vv = var * var
91 gcpy #          return self.vv
92 gcpy #
93 gcpy #      def getvv(self):
94 gcpy #          return self.vv
95 gcpy #
96 gcpy #      def checkint(self):
97 gcpy #          return self.mc
98 gcpy #
99 gcpy #      def makecube(self, xdim, ydim, zdim):
100 gcpy #          self.c=cube([xdim, ydim, zdim])
101 gcpy #
102 gcpy #      def placecube(self):
103 gcpy #          show(self.c)
104 gcpy #
105 gcpy #      def instantiatecube(self):
106 gcpy #          return self.c
```

3.2.1 Position and Variables

In modeling the machine motion and G-code it will be necessary to have the machine track several variables for machine position, the current tool and its parameters, 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.

In early versions, a 3D model of the tool was available as currenttool itself and used where appropriate, but in v0.9, this was changed to using lists for concatenating the hulled shapes of tool movements, so the module, toolmovement which given begin/end position returns the appropriate shape(s) as a list.

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

```
116 gcpy      def xpos(self):
117 gcpy          return self.mpx
118 gcpy
119 gcpy      def ypos(self):
120 gcpy          return self.mpy
121 gcpy
122 gcpy      def zpos(self):
123 gcpy          return self.mpz
```

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

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

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

setxpos

setypos

setzpos

```
132 gcpy      def setxpos(self, newxpos):
133 gcpy          self.mpx = newxpos
134 gcpy
135 gcpy      def setypos(self, newypos):
136 gcpy          self.mpy = newypos
137 gcpy
138 gcpy      def setzpos(self, newzpos):
139 gcpy          self.mpz = newzpos
```

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

Initializing the machine state requires zeroing out the three machine position variables:

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

Rather than a specific command for this, the code will be in-lined where appropriate (note that if machine initialization becomes sufficiently complex to warrant it, then a suitable command will need to be coded). Note that the variables are declared in the `__init__` of the class.

toolmovementendmilltypediameterfluterarapids toolpathsgcodepreview setupstockgcp.setupstock

The toolmovement class requires that the tool be defined in terms of endmilltype, diameter, flute (length), ra (radius or angle depending on context), and tip, and in turn defines the tool number as described below. An interface which calls this routine based on tool number will allow a return to the previous style of usage.

There will be two variables to record toolmovement, rapids and toolpaths. Initialized as empty lists, toolmovements will be extended to the lists.

The first such setup subroutine is gcodepreview setupstock which is appropriately enough, to set up the stock, and perform other initializations — initially, the only thing done in Python was to set the value of the persistent (Python) variables (see `initializemachinestate()` above), but the rewritten standalone version handles all necessary actions.

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.

```
148 gcpy      def setupstock(self, stockXwidth,
149 gcpy          stockYheight,
150 gcpy          stockZthickness,
151 gcpy          zeroheight,
152 gcpy          stockzero,
153 gcpy          retractheight):
154 gcpy          """
155 gcpy          Set up blank/stock for material and position/zero.
156 gcpy
157 gcpy          Parameters
158 gcpy          -----
159 gcpy          stockXwidth : float
160 gcpy                      X extent/dimension
161 gcpy          stockYheight : float
162 gcpy                      Y extent/dimension
163 gcpy          stockZthickness : boolean
164 gcpy                      Z extent/dimension
165 gcpy          zeroheight : string
166 gcpy                      Top or Bottom, determines if Z extent will
167 gcpy                      be positive or negative
168 gcpy          stockzero : string
169 gcpy                      Lower-Left, Center-Left, Top-Left, Center,
170 gcpy                      determines XY position of stock
171 gcpy          retractheight : float
172 gcpy                      Distance which tool retracts above surface
173 gcpy                      of stock.
174 gcpy
175 gcpy          Returns
176 gcpy          -----
177 gcpy          none
178 gcpy          """
179 gcpy          self.stockXwidth = stockXwidth
180 gcpy          self.stockYheight = stockYheight
181 gcpy          self.stockZthickness = stockZthickness
182 gcpy          self.zeroheight = zeroheight
183 gcpy          self.stockzero = stockzero
```

```
181 gcpy          self.retractheight = retractheight
182 gcpy          self.stock = cube([stockXwidth, stockYheight,
                                   stockZthickness])
```

zeroheight      A series of if statements parse the zeroheight (Z-axis) and stockzero (X- and Y-axes) param-  
stockzero      eters so as to place the stock in place and suitable G-code comments are added for CutViewer.

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

```

        stockXwidth), ",_", str(self.stockYheight),
        ",_ ", str(self.stockZthickness), ",_0.00,_
        0.00,_0.00)")
220 gcpy      if self.stockzero == "Center-Left":
221 gcpy      self.stock = self.stock.translate([0, -self.
        stockYheight / 2, 0])
222 gcpy      if self.generategcode == True:
223 gcpy      self.writegc("(stockMin:0.00mm,_", str(self.
        stockYheight/2), "mm,_0.00mm)")
224 gcpy      self.writegc("(stockMax:", str(self.stockXwidth
        ), "mm,_", str(self.stockYheight/2), "mm,_-
        , str(self.stockZthickness), "mm)")
225 gcpy      self.writegc("(STOCK/BLOCK,_", str(self.
        stockXwidth), ",_ ", str(self.stockYheight),
        ",_ ", str(self.stockZthickness), ",_0.00,_",
        str(self.stockYheight/2), ",_0.00mm)");
226 gcpy      if self.stockzero == "Top-Left":
227 gcpy      self.stock = self.stock.translate([0, -self.
        stockYheight, 0])
228 gcpy      if self.generategcode == True:
229 gcpy      self.writegc("(stockMin:0.00mm,_", str(self.
        stockYheight), "mm,_0.00mm)")
230 gcpy      self.writegc("(stockMax:", str(self.stockXwidth
        ), "mm,_0.00mm,_", str(self.stockZthickness)
        , "mm)")
231 gcpy      self.writegc("(STOCK/BLOCK,_", str(self.
        stockXwidth), ",_ ", str(self.stockYheight),
        ",_ ", str(self.stockZthickness), ",_0.00,_",
        str(self.stockYheight), ",_0.00)")
232 gcpy      if self.stockzero == "Center":
233 gcpy      self.stock = self.stock.translate([-self.
        stockXwidth / 2, -self.stockYheight / 2, 0])
234 gcpy      if self.generategcode == True:
235 gcpy      self.writegc("(stockMin:_", str(self.
        stockXwidth/2), ",_ ", str(self.stockYheight
        /2), "mm,_0.00mm)")
236 gcpy      self.writegc("(stockMax:", str(self.stockXwidth
        /2), "mm,_", str(self.stockYheight/2), "mm,_
        , str(self.stockZthickness), "mm)")
237 gcpy      self.writegc("(STOCK/BLOCK,_", str(self.
        stockXwidth), ",_ ", str(self.stockYheight),
        ",_ ", str(self.stockZthickness), ",_ ", str(
        self.stockXwidth/2), ",_ ", str(self.
        stockYheight/2), ",_0.00)")
238 gcpy      if self.generategcode == True:
239 gcpy      self.writegc("G90");
240 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 **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 OpenPython-SCAD 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 OpenSCAD version is simply a descriptor:

```

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

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

```

242 gcpy      def setupcuttingarea(self, sizeX, sizeY, sizeZ, extentleft,
        extentfb, extentd):
243 gcpy #      self.initializemachinestate()
244 gcpy      c=cube([sizeX,sizeY,sizeZ])
245 gcpy      c = c.translate([extentleft,extentfb,extentd])
246 gcpy      self.stock = c
247 gcpy      self.toolpaths = []
248 gcpy      return c
```

### 3.2.3 Adjustments and Additions

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

Shifting the stock is simple:

```
250 gcpy      def shiftstock(self, shiftX, shiftY, shiftZ):
251 gcpy          self.stock = self.stock.translate([shiftX, shiftY, shiftZ
                ])

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

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

```
253 gcpy      def addtostock(self, stockXwidth, stockYheight, stockZthickness
                ,
254 gcpy          shiftX = 0,
255 gcpy          shiftY = 0,
256 gcpy          shiftZ = 0):
257 gcpy          addedpart = cube([stockXwidth, stockYheight,
                stockZthickness])
258 gcpy          addedpart = addedpart.translate([shiftX, shiftY, shiftZ])
259 gcpy          self.stock = self.stock.union(addedpart)
```

the OpenSCAD module is a descriptor as expected:

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

## 3.3 Tools and Changes

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

Due to stack limits in OpenSCAD for the CSG tree, instead, the shapes will be stored in two variables as lists processed/created using a command `toolmovement` which will subsume all tool related functionality. As other routines need access to information about the current tool, appropriate routines will allow its variables will be queried.

The base/entry functionality has the instance being defined in terms of a basic set of variables (one of which is overloaded to serve multiple purposes, depending on the type of endmill).

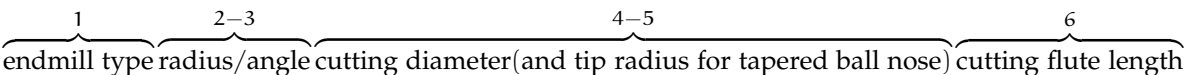
Note that it will also be necessary to write out a tool description compatible with the program CutViewer as a G-code comment so that it may be used as a 3D previewer for the G-code for tool changes in G-code. Several forms are available as described below.

### 3.3.1 Numbering for Tools

Currently, the numbering scheme used is that of the various manufacturers of the tools, or descriptive short-hand numbers created for tools which lack such a designation (with a disclosure that the author is a Carbide 3D employee).

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

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



- 1st digit: endmill type:
  - 0 - "O"-flute
  - 1 - square
  - 2 - ball
  - 3 - V



- 4 - bowl
  - 5 - tapered ball
  - 6 - roundover
  - 7 - thread-cutting
  - 8 - dovetail
  - 9 - other (e.g., keyhole, lollipop, or manufacturer number — if manufacturer number is used, then the 9 and any padding zeroes will be removed from the G-code or DXF when writing out file(s))
- 2nd and 3rd digits shape radius (ball/roundover) or angle (V), 2nd and 3rd digit together 10–99 indicate measurement in tenth of a millimeter. 2nd digit:
  - 0 - Imperial (00 indicates n/a or square)
  - any other value for both the 2nd and 3rd digits together indicate a metric measurement or an angle in degrees
- 3rd digit (if 2nd is 0 indicating Imperial)
  - 1 - 1/32<sup>nd</sup>
  - 2 - 1/16
  - 3 - 1/8
  - 4 - 1/4
  - 5 - 5/16
  - 6 - 3/8
  - 7 - 1/2
  - 8 - 3/4
  - 9 - >1" or other
- 4th and 5th digits cutting diameter as 2nd and 3rd above except 4th digit indicates tip radius for tapered ball nose and such tooling is only represented in Imperial measure:
- 4th digit (tapered ball nose)
  - 1 - 0.01 in (this is the 0.254mm of the #501 and 502)
  - 2 - 0.015625 in (1/64th)
  - 3 - 0.0295
  - 4 - 0.03125 in (1/32nd)
  - 5 - 0.0335
  - 6 - 0.0354
  - 7 - 0.0625 in (1/16th)
  - 8 - 0.125 in (1/8th)
  - 9 - 0.25 in (1/4)
- 6th digit cutting flute length:
  - 0 - other
  - 1 - calculate based on V angle
  - 2 - 1/16
  - 3 - 1/8
  - 4 - 1/4
  - 5 - 5/16
  - 6 - 1/2
  - 7 - 3/4
  - 8 - "long reach" or greater than 3/4"
  - 9 - calculate based on radius
- or 6th digit tip diameter for roundover tooling (added to cutting diameter to arrive at actual cutting diameter — note that these values are the same as for the tip radius of the #501 and 502)
  - 1 - 0.01 in
  - 2 - 0.015625 in (1/64th)
  - 3 - 0.0295
  - 4 - 0.03125 in (1/32nd)
  - 5 - 0.0335

- 6 - 0.0354
- 7 - 0.0625 in (1/16th)
- 8 - 0.125 in (1/8th)
- 9 - 0.25 in (1/4)

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

- Square
  - #122 == 100012
  - #112 == 100024
  - #102 == 100036 (also #326 (Amana 46200-K))
  - #201 == 100047 (also #251 and #322 (Amana 46202-K))
  - #205 == 100048
  - #324 == 100048 (Amana 46170-K)
- Ball
  - #121 == 201012
  - #111 == 202024
  - #101 == 203036
  - #202 == 204047
  - #325 == 204048 (Amana 46376-K)
- V
  - #301 == 390074
  - #302 == 360071
  - #327 == 360098 (Amana RC-1148)
- Single (O) flute
  - #282 == 000204
  - #274 == 000036
  - #278 == 000047
- Tapered Ball Nose
  - #501 == 530131
  - #502 == 540131

(note that some dimensions were rounded off/approximated)

Extending that to the non-Carbide 3D tooling thus implemented:

- Dovetail
  - 814 == 814071
  - 45828 == 808071
- Keyhole Tool
  - 374 == 906043
  - 375 == 906053
  - 376 == 907040
  - 378 == 907050
- Roundover Tool
  - 56142 == 602032
  - 56125 == 603042
  - 1568 == 603032
  - 1570
  - 1572 == 604042
  - 1574
- Threadmill
  - 648 == 7
- Bowl bit
  - 45981
  - 45982
  - 1370
  - 1372

Tools which do not have calculated numbers filled in are not supported by the system as currently defined in an unambiguous fashion (instead filling in the manufacturer’s tool number padded with zeros is hard-coded). Notable limitations:

- No way to indicate flute geometry beyond O-flute
- Lack of precision for metric tooling/limited support for Imperial sizes, notably, the dimensions used are scaled for smaller tooling and are not suited to larger scale tooling such as bowl bits
- No way to indicate several fairly common shapes including keyhole, lollipop, and flat-bottomed V/chamfer tools (except of course for using 9#####)

A further consideration is that it is not possible to represent tools unambiguously, so that given a tool definition it is possible to derive the manufacturer’s tool number, *e.g.*,

```
self.currenttoolshape = self.toolshapes("square", 3.175, 12.7)
```

representing three different tools (Carbide 3D #201 (upcut), #251 (downcut), and #322 (Amana 46202-K)). Affording some sort of hinting to the user may be warranted, or a mechanism to allow specifying a given manufacturer tool as part of setting up a job.

A more likely scheme is that manufacturer tool numbers will be used to identify tooling, the generated number will be used internally, then the saved manufacturer number will be exported to the G-code file, or used when generating a DXF filename for a given set of tool movements.

```
261 gcpy      def currenttoolnumber(self):
262 gcpy      return(self.currenttoolnum)
```

toolchange     The toolchange command will need to set several variables.  
Mandatory variables include:

- endmilltype
  - O-flute
  - square
  - ball
  - V
  - keyhole
  - dovetail
  - roundover
  - tapered ball
- diameter
- flute

and depending on the tool geometry, several additional variables will be necessary (usually derived from `self.ra`):

- radius
- angle

an optional setting of a `toolnumber` may be useful in the future.

tool number 3.3.1.1 **toolchange** This command accepts a `tool number` and assigns its characteristics as pa-  
toolchange rameters. It then applies the appropriate commands for a `toolchange`. Note that it is expected that this code will be updated as needed when new tooling is introduced as additional modules which require specific tooling are added.

Note that the comments written out in G-code correspond to those used by the G-code pre-viewing tool CutViewer (which is unfortunately, no longer readily available). Similarly, the G-code previewing functionality in this library expects that such comments will be in place so as to model the stock.

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

Note that there are many varieties of tooling and not all will be directly supported, and that at need, additional tool shape support may be added under `misc`.

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 and then applying a `union`. This approach will not work within Python, so it will be necessary to instead assign and select the tool as part of the `toolmovement` command.

```
264 gcpy      def toolchange(self, tool_number, speed = 10000):
265 gcpy          self.currenttoolnum = tool_number
266 gcpy
267 gcpy          if (self.generategcode == True):
268 gcpy              self.writegc("(Toolpath)")
269 gcpy              self.writegc("M05")
```

toolchange     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. A simple if-then structure, the variables necessary for defining the toolshape are (re)defined each time the toolmovement command is called so that they may be used by the command toolmovement for actually modeling the shapes and the path and the resultant material removal.

3.3.1.2 Square (including O-flute)     The CutViewer values include:

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

```
287 gcpy      if (tool_number == 201): #201/251/322 (Amana 46202-K) ==
288 gcpy          100047
289 gcpy          self.writegc("(TOOL/MILL,□6.35,□0.00,□0.00,□0.00)")
290 gcpy          self.endmilltype = "square"
291 gcpy          self.diameter = 6.35
292 gcpy          self.flute = 19.05
293 gcpy          self.shaftdiameter = 6.35
294 gcpy          self.shaftheight = 19.05
295 gcpy          self.shaftlength = 20.0
296 gcpy          self.toolnumber = "100047"
297 gcpy      elif (tool_number == 102): #102/326 == 100036
298 gcpy          self.writegc("(TOOL/MILL,□3.175,□0.00,□0.00,□0.00)")
299 gcpy          self.endmilltype = "square"
300 gcpy          self.diameter = 3.175
301 gcpy          self.flute = 12.7
302 gcpy          self.shaftdiameter = 3.175
303 gcpy          self.shaftheight = 12.7
304 gcpy          self.shaftlength = 20.0
305 gcpy          self.toolnumber = 100036
306 gcpy      elif (tool_number == 112): #112 == 100024
307 gcpy          self.writegc("(TOOL/MILL,□1.5875,□0.00,□0.00,□0.00)")
308 gcpy          self.endmilltype = "square"
309 gcpy          self.diameter = 1.5875
310 gcpy          self.flute = 6.35
311 gcpy          self.shaftdiameter = 3.175
312 gcpy          self.shaftheight = 6.35
313 gcpy          self.shaftlength = 12.0
314 gcpy          self.toolnumber = "100024"
315 gcpy      elif (tool_number == 122): #122 == 100012
316 gcpy          self.writegc("(TOOL/MILL,□0.79375,□0.00,□0.00,□0.00)")
317 gcpy          self.endmilltype = "square"
318 gcpy          self.diameter = 0.79375
319 gcpy          self.flute = 1.5875
320 gcpy          self.shaftdiameter = 3.175
321 gcpy          self.shaftheight = 1.5875
322 gcpy          self.shaftlength = 12.0
323 gcpy          self.toolnumber = "100012"
324 gcpy      elif (tool_number == 324): #324 (Amana 46170-K) == 100048
325 gcpy          self.writegc("(TOOL/MILL,□6.35,□0.00,□0.00,□0.00)")
326 gcpy          self.endmilltype = "square"
327 gcpy          self.diameter = 6.35
328 gcpy          self.flute = 22.225
329 gcpy          self.shaftdiameter = 6.35
330 gcpy          self.shaftheight = 22.225
331 gcpy          self.shaftlength = 20.0
332 gcpy          self.toolnumber = "100048"
333 gcpy      elif (tool_number == 205): #205 == 100048
334 gcpy          self.writegc("(TOOL/MILL,□6.35,□0.00,□0.00,□0.00)")
335 gcpy          self.endmilltype = "square"
336 gcpy          self.diameter = 6.35
337 gcpy          self.flute = 25.4
338 gcpy          self.shaftdiameter = 6.35
339 gcpy          self.shaftheight = 25.4
340 gcpy          self.shaftlength = 20.0
341 gcpy          self.toolnumber = "100048"
342 gcpy      #
```

Making a distinction betwixt Square and O-flute tooling may be removed from a future version.

---

```

310 gcpy          elif (tool_number == 282): #282 == 000204
311 gcpy              self.writegc("(T00L/MILL,␣2.0,␣0.00,␣0.00,␣0.00)")
312 gcpy              self.endmilltype = "0-flute"
313 gcpy              self.diameter = 2.0
314 gcpy              self.flute = 6.35
315 gcpy              self.shaftdiameter = 6.35
316 gcpy              self.shaftheight = 6.35
317 gcpy              self.shaftlength = 12.0
318 gcpy              self.toolnumber = "000204"
319 gcpy          elif (tool_number == 274): #274 == 000036
320 gcpy              self.writegc("(T00L/MILL,␣3.175,␣0.00,␣0.00,␣0.00)")
321 gcpy              self.endmilltype = "0-flute"
322 gcpy              self.diameter = 3.175
323 gcpy              self.flute = 12.7
324 gcpy              self.shaftdiameter = 3.175
325 gcpy              self.shaftheight = 12.7
326 gcpy              self.shaftlength = 20.0
327 gcpy              self.toolnumber = "000036"
328 gcpy          elif (tool_number == 278): #278 == 000047
329 gcpy              self.writegc("(T00L/MILL,␣6.35,␣0.00,␣0.00,␣0.00)")
330 gcpy              self.endmilltype = "0-flute"
331 gcpy              self.diameter = 6.35
332 gcpy              self.flute = 19.05
333 gcpy              self.shaftdiameter = 3.175
334 gcpy              self.shaftheight = 19.05
335 gcpy              self.shaftlength = 20.0
336 gcpy              self.toolnumber = "000047"
337 gcpy          #

```

---

### 3.3.1.3 Ball nose (including tapered ball nose) Additional shapes continue the elifs...

---

```

333 gcpy          elif (tool_number == 202): #202 == 204047
334 gcpy              self.writegc("(T00L/MILL,␣6.35,␣3.175,␣0.00,␣0.00)")
335 gcpy              self.endmilltype = "ball"
336 gcpy              self.diameter = 6.35
337 gcpy              self.flute = 19.05
338 gcpy              self.shaftdiameter = 6.35
339 gcpy              self.shaftheight = 19.05
340 gcpy              self.shaftlength = 20.0
341 gcpy              self.toolnumber = "204047"
342 gcpy          elif (tool_number == 101): #101 == 203036
343 gcpy              self.writegc("(T00L/MILL,␣3.175,␣1.5875,␣0.00,␣0.00)")
344 gcpy              self.endmilltype = "ball"
345 gcpy              self.diameter = 3.175
346 gcpy              self.flute = 12.7
347 gcpy              self.shaftdiameter = 3.175
348 gcpy              self.shaftheight = 12.7
349 gcpy              self.shaftlength = 20.0
350 gcpy              self.toolnumber = "203036"
351 gcpy          elif (tool_number == 111): #111 == 202024
352 gcpy              self.writegc("(T00L/MILL,␣1.5875,␣0.79375,␣0.00,␣0.00)"
353 gcpy                  )
354 gcpy              self.endmilltype = "ball"
355 gcpy              self.diameter = 1.5875
356 gcpy              self.flute = 6.35
357 gcpy              self.shaftdiameter = 3.175
358 gcpy              self.shaftheight = 6.35
359 gcpy              self.shaftlength = 20.0
360 gcpy              self.toolnumber = "202024"
361 gcpy          elif (tool_number == 121): #121 == 201012
362 gcpy              self.writegc("(T00L/MILL,␣3.175,␣0.79375,␣0.00,␣0.00)")
363 gcpy              self.endmilltype = "ball"
364 gcpy              self.diameter = 0.79375
365 gcpy              self.flute = 1.5875
366 gcpy              self.shaftdiameter = 3.175
367 gcpy              self.shaftheight = 1.5875
368 gcpy              self.shaftlength = 20.0
369 gcpy              self.toolnumber = "201012"
370 gcpy          elif (tool_number == 325): #325 (Amana 46376-K) == 204048
371 gcpy              self.writegc("(T00L/MILL,␣6.35,␣3.175,␣0.00,␣0.00)")
372 gcpy              self.endmilltype = "ball"
373 gcpy              self.diameter = 6.35
374 gcpy              self.flute = 25.4
375 gcpy              self.shaftdiameter = 6.35
376 gcpy              self.shaftheight = 25.4
377 gcpy              self.shaftlength = 20.0

```

```
377 gcpy                self.toolnumber = "204048"
378 gcpy #
```

---

3.3.1.4 V Note that one V tool is described as an Engraver in Carbide Create.

---

```
356 gcpy                elif (tool_number == 301): #301 == 390074
357 gcpy                    self.writegc("(T00L/MILL,␣0.10,␣0.05,␣6.35,␣45.00)")
358 gcpy                    self.endmilltype = "V"
359 gcpy                    self.diameter = 12.7
360 gcpy                    self.flute = 6.35
361 gcpy                    self.angle = 90
362 gcpy                    self.shaftdiameter = 6.35
363 gcpy                    self.shaftheight = 6.35
364 gcpy                    self.shaftlength = 20.0
365 gcpy                    self.toolnumber = "390074"
366 gcpy                elif (tool_number == 302): #302 == 360071
367 gcpy                    self.writegc("(T00L/MILL,␣0.10,␣0.05,␣6.35,␣30.00)")
368 gcpy                    self.endmilltype = "V"
369 gcpy                    self.diameter = 12.7
370 gcpy                    self.flute = 11.067
371 gcpy                    self.angle = 60
372 gcpy                    self.shaftdiameter = 6.35
373 gcpy                    self.shaftheight = 11.067
374 gcpy                    self.shaftlength = 20.0
375 gcpy                    self.toolnumber = "360071"
376 gcpy                elif (tool_number == 390): #390 == 390032
377 gcpy                    self.writegc("(T00L/MILL,␣0.03,␣0.00,␣1.5875,␣45.00)")
378 gcpy                    self.endmilltype = "V"
379 gcpy                    self.diameter = 3.175
380 gcpy                    self.flute = 1.5875
381 gcpy                    self.angle = 90
382 gcpy                    self.shaftdiameter = 3.175
383 gcpy                    self.shaftheight = 1.5875
384 gcpy                    self.shaftlength = 20.0
385 gcpy                    self.toolnumber = "390032"
386 gcpy                elif (tool_number == 327): #327 (Amana RC-1148) == 360098
387 gcpy                    self.writegc("(T00L/MILL,␣0.03,␣0.00,␣13.4874,␣30.00)")
388 gcpy                    self.endmilltype = "V"
389 gcpy                    self.diameter = 25.4
390 gcpy                    self.flute = 22.134
391 gcpy                    self.angle = 60
392 gcpy                    self.shaftdiameter = 6.35
393 gcpy                    self.shaftheight = 22.134
394 gcpy                    self.shaftlength = 20.0
395 gcpy                    self.toolnumber = "360098"
396 gcpy                elif (tool_number == 323): #323 == 330041 30 degree V Amana
397 gcpy                    , 45771-K
398 gcpy                    self.writegc("(T00L/MILL,␣0.10,␣0.05,␣11.18,␣15.00)")
399 gcpy                    self.endmilltype = "V"
400 gcpy                    self.diameter = 6.35
401 gcpy                    self.flute = 11.849
402 gcpy                    self.angle = 30
403 gcpy                    self.shaftdiameter = 6.35
404 gcpy                    self.shaftheight = 11.849
405 gcpy                    self.shaftlength = 20.0
406 gcpy                    self.toolnumber = "330041"
406 gcpy #
```

---

3.3.1.5 Keyhole Keyhole tooling will primarily be used with a dedicated toolpath.

---

```
379 gcpy                elif (tool_number == 374): #374 == 906043
380 gcpy                    self.writegc("(T00L/MILL,␣9.53,␣0.00,␣3.17,␣0.00)")
381 gcpy                    self.endmilltype = "keyhole"
382 gcpy                    self.diameter = 9.525
383 gcpy                    self.flute = 3.175
384 gcpy                    self.radius = 6.35
385 gcpy                    self.shaftdiameter = 6.35
386 gcpy                    self.shaftheight = 3.175
387 gcpy                    self.shaftlength = 20.0
388 gcpy                    self.toolnumber = "906043"
389 gcpy                elif (tool_number == 375): #375 == 906053
390 gcpy                    self.writegc("(T00L/MILL,␣9.53,␣0.00,␣3.17,␣0.00)")
391 gcpy                    self.endmilltype = "keyhole"
392 gcpy                    self.diameter = 9.525
393 gcpy                    self.flute = 3.175
```

```

394 gcpy          self.radius = 8
395 gcpy          self.shaftdiameter = 6.35
396 gcpy          self.shaftheight = 3.175
397 gcpy          self.shaftlength = 20.0
398 gcpy          self.toolnumber = "906053"
399 gcpy          elif (tool_number == 376): #376 == 907040
400 gcpy          self.writetc("T00L/MILL,␣12.7,␣0.00,␣4.77,␣0.00)")
401 gcpy          self.endmilltype = "keyhole"
402 gcpy          self.diameter = 12.7
403 gcpy          self.flute = 4.7625
404 gcpy          self.radius = 6.35
405 gcpy          self.shaftdiameter = 6.35
406 gcpy          self.shaftheight = 4.7625
407 gcpy          self.shaftlength = 20.0
408 gcpy          self.toolnumber = "907040"
409 gcpy          elif (tool_number == 378): #378 == 907050
410 gcpy          self.writetc("T00L/MILL,␣12.7,␣0.00,␣4.77,␣0.00)")
411 gcpy          self.endmilltype = "keyhole"
412 gcpy          self.diameter = 12.7
413 gcpy          self.flute = 4.7625
414 gcpy          self.radius = 8
415 gcpy          self.shaftdiameter = 6.35
416 gcpy          self.shaftheight = 4.7625
417 gcpy          self.shaftlength = 20.0
418 gcpy          self.toolnumber = "907050"
419 gcpy #

```

---

### 3.3.1.6 Bowl This geometry is also useful for square endmills with a radius.

---

```

402 gcpy          elif (tool_number == 45981): #45981 == 445981
403 gcpy #Amana Carbide Tipped Bowl & Tray 1/8 Radius x 1/2 Dia x 1/2 x 1/4
      Inch Shank
404 gcpy          self.writetc("T00L/MILL,0.03,␣0.00,␣10.00,␣30.00)")
405 gcpy          self.writetc("T00L/MILL,␣15.875,␣6.35,␣19.05,␣0.00)")
406 gcpy          self.endmilltype = "bowl"
407 gcpy          self.diameter = 12.7
408 gcpy          self.flute = 12.7
409 gcpy          self.radius = 3.175
410 gcpy          self.shaftdiameter = 6.35
411 gcpy          self.shaftheight = 12.7
412 gcpy          self.shaftlength = 20.0
413 gcpy          self.toolnumber = "445981"
414 gcpy          elif (tool_number == 45982):#0.507/2, 4.509
415 gcpy          self.writetc("T00L/MILL,␣15.875,␣6.35,␣19.05,␣0.00)")
416 gcpy          self.endmilltype = "bowl"
417 gcpy          self.diameter = 19.05
418 gcpy          self.flute = 15.875
419 gcpy          self.radius = 6.35
420 gcpy          self.shaftdiameter = 6.35
421 gcpy          self.shaftheight = 15.875
422 gcpy          self.shaftlength = 20.0
423 gcpy          self.toolnumber = "445982"
424 gcpy #          elif (tool_number == 1370): #1370 == 401370
425 gcpy #Whiteside Bowl & Tray Bit 1/4"SH, 1/8"R, 7/16"CD (5/16" cutting
      flute length)
426 gcpy          self.writetc("T00L/MILL,␣11.1125,␣8,␣3.175,␣0.00)")
427 gcpy          self.endmilltype = "bowl"
428 gcpy          self.diameter = 11.1125
429 gcpy          self.flute = 8
430 gcpy          self.radius = 3.175
431 gcpy          self.shaftdiameter = 6.35
432 gcpy          self.shaftheight = 8
433 gcpy          self.shaftlength = 20.0
434 gcpy          self.toolnumber = "401370"
435 gcpy #          elif (tool_number == 1372): #1372/45982 == 401372
436 gcpy #Whiteside Bowl & Tray Bit 1/4"SH, 1/4"R, 3/4"CD (5/8" cutting
      flute length)
437 gcpy #Amana Carbide Tipped Bowl & Tray 1/4 Radius x 3/4 Dia x 5/8 x 1/4
      Inch Shank
438 gcpy          self.writetc("T00L/MILL,␣19.5,␣15.875,␣6.35,␣0.00)")
439 gcpy          self.endmilltype = "bowl"
440 gcpy          self.diameter = 19.5
441 gcpy          self.flute = 15.875
442 gcpy          self.radius = 6.35
443 gcpy          self.shaftdiameter = 6.35
444 gcpy          self.shaftheight = 15.875
445 gcpy          self.shaftlength = 20.0

```

```
446 gcpy                self.toolnumber = "401372"
447 gcpy #
```

---

**3.3.1.7 Tapered ball nose** One vendor which provides such tooling is Precise Bits: <https://www.precisebits.com/products/carbidebits/taperedcarve250b2f.asp&filter=7>, but unfortunately, their tool numbering is ambiguous, the version of each major number (204 and 304) for their 1/4" shank tooling which is sufficiently popular to also be offered in a ZRN coating will be used. Similarly, the #501 and #502 PCB engravers from Carbide 3D are also supported.

---

```
425 gcpy                elif (tool_number == 501): #501 == 530131
426 gcpy                self.writegc("(TOOL/MILL,0.03,▯0.00,▯10.00,▯30.00)")
427 gcpy #                self.currenttoolshape = self.toolshapes("tapered ball
", 3.175, 5.561, 30, 0.254)
428 gcpy                self.endmilltype = "tapered▯ball"
429 gcpy                self.diameter = 3.175
430 gcpy                self.flute = 5.561
431 gcpy                self.angle = 30
432 gcpy                self.tip = 0.254
433 gcpy                self.shaftdiameter = 3.175
434 gcpy                self.shaftheight = 5.561
435 gcpy                self.shaftlength = 10.0
436 gcpy                self.toolnumber = "530131"
437 gcpy                elif (tool_number == 502): #502 == 540131
438 gcpy                self.writegc("(TOOL/MILL,0.03,▯0.00,▯10.00,▯20.00)")
439 gcpy #                self.currenttoolshape = self.toolshapes("tapered ball
", 3.175, 4.117, 40, 0.254)
440 gcpy                self.endmilltype = "tapered▯ball"
441 gcpy                self.diameter = 3.175
442 gcpy                self.flute = 4.117
443 gcpy                self.angle = 40
444 gcpy                self.tip = 0.254
445 gcpy                self.shaftdiameter = 3.175
446 gcpy                self.shaftheight = 4.117
447 gcpy                self.shaftlength = 10.0
448 gcpy                self.toolnumber = "540131"
449 gcpy #                elif (tool_number == 204):#
450 gcpy #                self.writegc("(")
451 gcpy #                self.currenttoolshape = self.tapered_ball(1.5875,
6.35, 38.1, 3.6)
452 gcpy #                elif (tool_number == 304):#
453 gcpy #                self.writegc("(")
454 gcpy #                self.currenttoolshape = self.tapered_ball(3.175, 6.35,
38.1, 2.4)
455 gcpy #
```

---

**3.3.1.8 Roundover (corner rounding)** Note that the parameters will need to incorporate the tip diameter into the overall diameter

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

---

```
448 gcpy                elif (tool_number == 56125):#0.508/2, 1.531 56125 == 603042
449 gcpy                self.writegc("(TOOL/CRMILL,▯0.508,▯6.35,▯3.175,▯7.9375,
▯3.175)")
450 gcpy                self.endmilltype = "roundover"
451 gcpy                self.tip = 0.508
452 gcpy                self.diameter = 6.35 - self.tip
453 gcpy                self.flute = 8 - self.tip
454 gcpy                self.radius = 3.175 - self.tip
455 gcpy                self.shaftdiameter = 6.35
456 gcpy                self.shaftheight = 8
457 gcpy                self.shaftlength = 10.0
458 gcpy                self.toolnumber = "603042"
459 gcpy                elif (tool_number == 56142):#0.508/2, 2.921 56142 == 602032
460 gcpy                self.writegc("(TOOL/CRMILL,▯0.508,▯3.571875,▯1.5875,▯
5.55625,▯1.5875)")
461 gcpy                self.endmilltype = "roundover"
462 gcpy                self.tip = 0.508
463 gcpy                self.diameter = 3.175 - self.tip
464 gcpy                self.flute = 4.7625 - self.tip
465 gcpy                self.radius = 1.5875 - self.tip
466 gcpy                self.shaftdiameter = 3.175
467 gcpy                self.shaftheight = 4.7625
468 gcpy                self.shaftlength = 10.0
469 gcpy                self.toolnumber = "602032"
```



```

470 gcpy #         elif (tool_number == 312):#1.524/2, 3.175
471 gcpy #             self.writegc("(TOOL/CRMILL, Diameter1, Diameter2,
Radius, Height, Length)")
472 gcpy #         elif (tool_number == 1568):#0.507/2, 4.509 1568 == 603032
473 gcpy ##FIX             self.writegc("(TOOL/CRMILL, 0.17018, 9.525,
4.7625, 12.7, 4.7625)")
474 gcpy ##             self.currenttoolshape = self.toolshapes("roundover",
3.175, 6.35, 3.175, 0.396875)
475 gcpy #             self.endmilltype = "roundover"
476 gcpy #             self.diameter = 3.175
477 gcpy #             self.flute = 6.35
478 gcpy #             self.radius = 3.175
479 gcpy #             self.tip = 0.396875
480 gcpy #             self.toolnumber = "603032"
481 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
482 gcpy #             elif (tool_number == 1570):#0.507/2, 4.509 1570 == 600002
?!!?
483 gcpy #             self.writegc("(TOOL/CRMILL, 0.17018, 9.525, 4.7625,
12.7, 4.7625)")
484 gcpy ##             self.currenttoolshape = self.toolshapes("roundover",
4.7625, 9.525, 4.7625, 0.396875)
485 gcpy #             self.endmilltype = "roundover"
486 gcpy #             self.diameter = 4.7625
487 gcpy #             self.flute = 9.525
488 gcpy #             self.radius = 4.7625
489 gcpy #             self.tip = 0.396875
490 gcpy #             self.toolnumber = "600002"
491 gcpy #             elif (tool_number == 1572): #1572 = 604042
492 gcpy ##FIX             self.writegc("(TOOL/CRMILL, 0.17018, 9.525,
4.7625, 12.7, 4.7625)")
493 gcpy ##             self.currenttoolshape = self.toolshapes("roundover",
6.35, 12.7, 6.35, 0.396875)
494 gcpy #             self.endmilltype = "roundover"
495 gcpy #             self.diameter = 6.35
496 gcpy #             self.flute = 12.7
497 gcpy #             self.radius = 6.35
498 gcpy #             self.tip = 0.396875
499 gcpy #             self.toolnumber = "604042"
500 gcpy #             elif (tool_number == 1574): #1574 == 600062
501 gcpy ##FIX             self.writegc("(TOOL/CRMILL, 0.17018, 9.525,
4.7625, 12.7, 4.7625)")
502 gcpy ##             self.currenttoolshape = self.toolshapes("roundover",
9.525, 19.5, 9.515, 0.396875)
503 gcpy #             self.endmilltype = "roundover"
504 gcpy #             self.diameter = 9.525
505 gcpy #             self.flute = 19.5
506 gcpy #             self.radius = 9.515
507 gcpy #             self.tip = 0.396875
508 gcpy #             self.toolnumber = "600062"
509 gcpy #

```

---

**3.3.1.9 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).

```

471 gcpy             elif (tool_number == 814): #814 == 814071
472 gcpy #Item 18J1607, 1/2" 14ř Dovetail Bit, 8mm shank
473 gcpy             self.writegc("(TOOL/MILL, 12.7, 6.367, 12.7, 0.00)")
474 gcpy             # dt_bottomdiameter, dt_topdiameter, dt_height, dt_angle
)
475 gcpy             # https://www.leevalley.com/en-us/shop/tools/power-tool-
accessories/router-bits/30172-dovetail-bits?item=18J1607
476 gcpy #             self.currenttoolshape = self.toolshapes("dovetail",
12.7, 12.7, 14)
477 gcpy             self.endmilltype = "dovetail"
478 gcpy             self.diameter = 12.7
479 gcpy             self.flute = 12.7
480 gcpy             self.angle = 14
481 gcpy             self.toolnumber = "814071"
482 gcpy             elif (tool_number == 808079): #45828 == 808071
483 gcpy             self.writegc("(TOOL/MILL, 12.7, 6.816, 20.95, 0.00)")
484 gcpy             # http://www.amanatool.com/45828-carbide-tipped-dovetail
-8-deg-x-1-2-dia-x-825-x-1-4-inch-shank.html
485 gcpy #             self.currenttoolshape = self.toolshapes("dovetail",
12.7, 20.955, 8)

```

```
486 gcpy          self.endmilltype = "dovetail"
487 gcpy          self.diameter = 12.7
488 gcpy          self.flute = 20.955
489 gcpy          self.angle = 8
490 gcpy          self.toolnumber = "808071"
491 gcpy #
```

---

Each tool must be modeled in 3D using OpenSCAD commands, but it will also be necessary to have a consistent structure for managing the various shapes and aspects of shapes.

While tool shapes were initially handled as geometric shapes stored in Python variables, processing them as such after the fashion of OpenSCAD required the use of union() commands and assigning a small initial object (usually a primitive placed at the origin) so that the union could take place. This has the result of creating a nested union structure in the CSG tree which can quickly become so deeply nested that it exceeds the limits set in PythonSCAD.

As was discussed in the PythonSCAD Google Group (<https://groups.google.com/g/pythonscad/c/rTiYa38W8tY>), if a list is used instead, then the contents of the list are added all at once at a single level when processed.

An example file which shows this concept:

```
from openscad import *
fn=200

box = cube([40,40,40])

features = []

features.append(cube([36,36,40]) + [2,2,2])
features.append(cylinder(d=20,h=5) + [20,20,-1])
features.append(cylinder(d=3,h=10) ^ [[5,35],[5,35], -1])

part = difference(box, features)

show(part)
```

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

```
49 gpcpscad module toolchange(tool_number , speed){
50 gpcpscad     gcp.toolchange(tool_number , speed);
51 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.)

**3.3.1.10 closing G-code** 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.

```
494 gcpy #          self.writegc("M6T", str(tool_number))
495 gcpy #          self.writegc("M03S", str(speed))
```

---

3.3.2 Laser support

Two possible options for supporting a laser present themselves: color-coded DXFs or direct G-code support. An example file for the latter:

<https://lasergrbl.com/test-file-and-samples/depth-of-focus-test/>

```
M3 S0
S0
G0X0Y16
S1000
G1X100F1200
S0
M5 S0
M3 S0
S0
G0X0Y12
S1000
G1X100F1000
S0
```

```

M5 S0
M3 S0
S0
GOX0Y8
S1000
G1X100F800
S0
M5 S0
M3 S0
S0
GOX0Y4
S1000
G1X100F600
S0
M5 S0
M3 S0
S0
GOX0Y0
S1000
G1X100F400
S0
M5 S0

```

### 3.4 Shapes and tool movement

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

The majority of commands will be more general, focusing on tooling which is generally supported by this library, moving in lines and arcs so as to describe shapes which lend themselves to representation with those tools and which match up with both toolpaths and supported geometry in Carbide Create, and the usage requirements of the typical user.

This structure has the notable advantage that if a tool shape is represented as a list and always handled thus, then representing complex shapes which need to be represented in discrete elements/parts becomes a natural thing to do and the program architecture is simpler since all possible shapes may be handled by the same code/logic with no need to identify different shapes and handle them differently.

Note that it will be preferable to use `extend` if the variable to be added contains a list rather than `append` since the former will flatten out the list and add the individual elements, so that a list remains a list of elements rather than becoming a list of lists and elements, except that there will be at least two elements to each tool model list:

- cutting *tool* shape (note that this may be either a single model, or a list of discrete slices of the tool shape)
- *shaft*

and when a cut is made by hulling each element from the cut begin position to its end position, this will be done using different colors so that the shaft rubbing may be identified on the 3D surface of the preview of the cut.

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

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

#### 3.4.1 Generalized commands and cuts

The first consideration is a naming convention which will allow a generalized set of associated commands to be defined. The initial version will only create OpenSCAD commands for 3D modeling and write out matching DXF files. At a later time this will be extended with G-code support.

There are three different movements in G-code which will need to be handled. Rapid commands will be used for `G0` movements and will not appear in DXFs but will appear in G-code files, while straight line cut (`G1`) and arc (`G2/G3`) commands may appear in both G-code and DXF files, depending on the specific command invoked.

3.4.2 Movement and color

toolmovement

shaftmovement

The first command which must be defined is `toolmovement` which is used as the core of the other commands, affording a 3D model of the tool moving in a straight line. A matching `shaftmovement` command will allow modeling collision of the shaft with the stock should it occur. This differentiation raises the matter of color representation. Using a different color for the shape of the endmill when cutting and for rapid movements will similarly allow identifying instances of the tool crashing through stock at rapid speed.

```
497 gcpy      def setcolor(self,
498 gcpy              cutcolor = "green",
499 gcpy              rapidcolor = "orange",
500 gcpy              shaftcolor = "red"):
501 gcpy          self.cutcolor = cutcolor
502 gcpy          self.rapidcolor = rapidcolor
503 gcpy          self.shaftcolor = shaftcolor
```

The possible colors are those of Web colors ([https://en.wikipedia.org/wiki/Web\\_colors](https://en.wikipedia.org/wiki/Web_colors)), while `dxf` has its own set of colors based on numbers (see table) and applying a Venn diagram and removing problematic extremes we arrive at the third column above as black and white are potentially inconsistent/confusing since at least one CAD program toggles them based on light/dark mode being applied to its interface.

Table 1: Colors in OpenSCAD and dxf

Web Colors (OpenSCAD)	DXF	Both
Black	"Black" (0)	
Red	"Red" (1)	Red
Yellow	"Yellow" (2)	Yellow
Green	"Green" (3)	Green
	"Cyan" (4)	
Blue	"Blue" (5)	Blue
	"Magenta" (6)	
White	"White" (7)	
Gray	"Dark Gray" (8)	(Dark) Gray
	"Light Gray" (9)	
Silver		
Maroon		
Olive		
Lime		
Aqua		
Teal		
Navy		
Fuchsia		
Purple		

(note that the names are not case-sensitive)

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

Note that the variables `self.rapids` and `self.toolpaths` are used to hold the list of accumulated 3D models of the rapid motions and cuts as elements in lists so that they may be differenced from the stock.

**3.4.2.1 toolmovement** The `toolmovement` command incorporates the color variables to indicate cutting and differentiate rapid movements and the tool shaft.

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):



```
550 gcpy          tslist.append(hull(ts.translate([bx, by, bz]), ts.  
                    translate([ex, ey, ez])))  
551 gcpy          return tslist
```

---

ballnose **3.4.2.4 Ball nose (including tapered ball nose)** The ballnose is modeled as a hemisphere joined with a cylinder:

```
566 gcpy          if self.endmilltype == "ball":  
567 gcpy              b = sphere(r=(self.diameter / 2))  
568 gcpy              s = cylinder(r1=(self.diameter / 2), r2=(self.diameter  
                    / 2), h=self.flute, center=False)  
569 gcpy              bs = union(b, s)  
570 gcpy              bs = bs.translate([0, 0, (self.diameter / 2)])  
571 gcpy              tslist.append(hull(bs.translate([bx, by, bz]), bs.  
                    translate([ex, ey, ez])))  
572 gcpy              return tslist  
573 gcpy #
```

---

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

```
589 gcpy          if self.endmilltype == "bowl":  
590 gcpy              inner = cylinder(r1 = self.diameter/2 - self.radius, r2  
                    = self.diameter/2 - self.radius, h = self.flute)  
591 gcpy              outer = cylinder(r1 = self.diameter/2, r2 = self.  
                    diameter/2, h = self.flute - self.radius)  
592 gcpy              outer = outer.translate([0,0, self.radius])  
593 gcpy              slices = hull(outer, inner)  
594 gcpy #          slices = cylinder(r1 = 0.0001, r2 = 0.0001, h = 0.0001, center  
                    =False)  
595 gcpy              for i in range(1, 90 - self.steps, self.steps):  
596 gcpy                  slice = cylinder(r1 = self.diameter / 2 - self.  
                    radius + self.radius * Sin(i), r2 = self.  
                    diameter / 2 - self.radius + self.radius * Sin(i  
                    +self.steps), h = self.radius/90, center=False)  
597 gcpy                  slices = hull(slices, slice.translate([0, 0, self.  
                    radius - self.radius * Cos(i+self.steps)]))  
598 gcpy                  tslist.append(hull(slices.translate([bx, by, bz]),  
                    slices.translate([ex, ey, ez])))  
599 gcpy              return tslist  
600 gcpy #
```

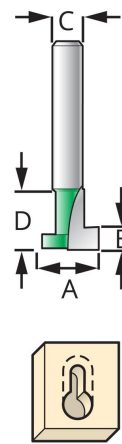
---

endmill v **3.4.2.6 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):

```
612 gcpy          if self.endmilltype == "V":  
613 gcpy              v = cylinder(r1=0, r2=(self.diameter / 2), h=((self.  
                    diameter / 2) / Tan((self.angle / 2))), center=False  
                    )  
614 gcpy #          s = cylinder(r1=(self.diameter / 2), r2=(self.  
                    diameter / 2), h=self.flute, center=False)  
615 gcpy #          sh = s.translate([0, 0, ((self.diameter / 2) / Tan  
                    ((self.angle / 2)))]])  
616 gcpy              tslist.append(hull(v.translate([bx, by, bz]), v.  
                    translate([ex, ey, ez])))  
617 gcpy              return tslist
```

---

**3.4.2.7 Keyhole** Keyhole toolpaths (see: subsection 3.7.0.2.3 are intended for use with tooling which projects beyond 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

```
635 gcpy         if self.endmilltype == "keyhole":
636 gcpy             kh = cylinder(r1=(self.diameter / 2), r2=(self.diameter
                        / 2), h=self.flute, center=False)
637 gcpy             sh = (cylinder(r1=(self.radius / 2), r2=(self.radius /
                        2), h=self.flute*2, center=False))
638 gcpy             tslist.append(hull(kh.translate([bx, by, bz]), kh.
                        translate([ex, ey, ez])))
639 gcpy             tslist.append(hull(sh.translate([bx, by, bz]), sh.
                        translate([ex, ey, ez])))
640 gcpy             return tslist
```

**3.4.2.8 Tapered ball nose** The tapered ball nose tool is modeled as a sphere at the tip and a pair of cylinders, where one (a cone) describes the taper, while the other represents the shaft.

```
658 gcpy         if self.endmilltype == "tapered_ball":
659 gcpy             b = sphere(r=(self.tip / 2))
660 gcpy             s = cylinder(r1=(self.tip / 2), r2=(self.diameter / 2),
                        h=self.flute, center=False)
661 gcpy             bshape = union(b, s)
662 gcpy             tslist.append(hull(bshape.translate([bx, by, bz]),
                        bshape.translate([ex, ey, ez])))
663 gcpy             return tslist
```

dovetail **3.4.2.9 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)

```
681 gcpy         if self.endmilltype == "dovetail":
682 gcpy             dt = cylinder(r1=(self.diameter / 2), r2=(self.diameter
                        / 2) - self.flute * Tan(self.angle), h= self.flute,
                        center=False)
683 gcpy             tslist.append(hull(dt.translate([bx, by, bz]), dt.
                        translate([ex, ey, ez])))
684 gcpy             return tslist
685 gcpy         if self.endmilltype == "other":
686 gcpy             tslist = []
687 gcpy #         def dovetail(self, dt_bottomdiameter, dt_topdiameter,
                        dt_height, dt_angle):
688 gcpy #             return cylinder(r1=(dt_bottomdiameter / 2), r2=(
                        dt_topdiameter / 2), h= dt_height, center=False)
```

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

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

**3.4.2.11 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 3.4.2.10 — roundover tooling will need to generate a list of slices of the tool shape hulled together.

```
684 gcpy          if self.endmilltype == "roundover":
685 gcpy              shaft = cylinder(self.steps, self.tip/2, self.tip/2)
686 gcpy              toolpath = hull(shaft.translate([bx, by, bz]), shaft.
                                translate([ex, ey, ez]))
687 gcpy              shaft = cylinder(self.flute, self.diameter/2 + self.tip
                                /2, self.diameter/2 + self.tip/2)
688 gcpy              toolpath = toolpath.union(hull(shaft.translate([bx, by,
                                bz + self.radius]), shaft.translate([ex, ey, ez +
                                self.radius])))
689 gcpy              tslist = [toolpath]
690 gcpy              slice = cylinder(0.0001, 0.0001, 0.0001)
691 gcpy              slices = slice
692 gcpy              for i in range(1, 90 - self.steps, self.steps):
693 gcpy                  dx = self.radius*cos(i)
694 gcpy                  dxx = self.radius*cos(i + self.steps)
695 gcpy                  dzz = self.radius*sin(i)
696 gcpy                  dz = self.radius*sin(i + self.steps)
697 gcpy                  dh = dz - dzz
698 gcpy                  slice = cylinder(r1 = self.tip/2+self.radius-dx, r2
                                = self.tip/2+self.radius-dxx, h = dh)
699 gcpy                  slices = slices.union(hull(slice.translate([bx, by,
                                bz+dz]), slice.translate([ex, ey, ez+dz])))
700 gcpy              tslist.append(slices)
701 gcpy              return tslist
702 gcpy #
```

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

shaftmovement A similar routine will be used to handle the shaftmovement.

shaftmovement **3.4.2.12 shaftmovement** The shaftmovement command uses variables defined as part of the tool definition to determine the Z-axis position of the cylinder used to represent the shaft and its diameter and height:

```
755 gcpy          def shaftmovement(self, bx, by, bz, ex, ey, ez):
756 gcpy              tslist = []
757 gcpy              ts = cylinder(r1=(self.shaftdiameter / 2), r2=(self.
                                shaftdiameter / 2), h=self.shaftlength, center = False)
758 gcpy              ts = ts.translate([0, 0, self.shaftheight])
759 gcpy              tslist.append(hull(ts.translate([bx, by, bz]), ts.translate
                                ([ex, ey, ez])))
760 gcpy              return tslist
```

rapid **3.4.2.13 rapid and cut (lines)** A matching pair of commands is made for these, and rapid is used as the basis for a series of commands which match typical usages of G0.

Note the addition of a Laser mode which simulates the tool having been turned off — likely further changes will be required.

```
778 gcpy          def rapid(self, ex, ey, ez, laser = 0):
779 gcpy #              print(self.rapidcolor)
780 gcpy              if laser == 0:
781 gcpy                  tm = self.toolmovement(self.xpos(), self.ypos(), self.
                                zpos(), ex, ey, ez)
782 gcpy                  tm = color(tm, self.shaftcolor)
783 gcpy                  ts = self.shaftmovement(self.xpos(), self.ypos(), self.
                                zpos(), ex, ey, ez)
784 gcpy                  ts = color(ts, self.rapidcolor)
785 gcpy                  self.toolpaths.extend([tm, ts])
786 gcpy              self.setxpos(ex)
787 gcpy              self.setypos(ey)
788 gcpy              self.setzpos(ez)
789 gcpy
790 gcpy          def cutline(self, ex, ey, ez):
791 gcpy #              print(self.cutcolor)
792 gcpy #              print(ex, ey, ez)
793 gcpy              tm = self.toolmovement(self.xpos(), self.ypos(), self.zpos
                                (), ex, ey, ez)
```



---

```

794 gcpy          tm = color(tm, self.cutcolor)
795 gcpy          ts = self.shaftmovement(self.xpos(), self.ypos(), self.zpos
              (), ex, ey, ez)
796 gcpy          ts = color(ts, self.rapidcolor)
797 gcpy          self.setxpos(ex)
798 gcpy          self.setypos(ey)
799 gcpy          self.setzpos(ez)
800 gcpy          self.toolpaths.extend([tm, ts])

```

---

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:

---

```

801 gcpy          def movetosafeZ(self):
802 gcpy              rapid = self.rapid(self.xpos(), self.ypos(), self.
                  retractheight)
803 gcpy #          if self.generatepaths == True:
804 gcpy #              rapid = self.rapid(self.xpos(), self.ypos(), self.
                  retractheight)
805 gcpy #              self.rapids = self.rapids.union(rapid)
806 gcpy #          else:
807 gcpy # if (generategcode == true) {
808 gcpy # //      writecomment("PREPOSITION FOR RAPID PLUNGE");Z25.650
809 gcpy # //G1Z24.663F381.0, "F", str(plunge)
810 gcpy #          if self.generatepaths == False:
811 gcpy #              return rapid
812 gcpy #          else:
813 gcpy #              return cube([0.001, 0.001, 0.001])
814 gcpy          return rapid
815 gcpy
816 gcpy          def rapidXYZ(self, ex, ey, ez):
817 gcpy              rapid = self.rapid(ex, ey, ez)
818 gcpy #          if self.generatepaths == False:
819 gcpy              return rapid
820 gcpy
821 gcpy          def rapidXY(self, ex, ey):
822 gcpy              rapid = self.rapid(ex, ey, self.zpos())
823 gcpy #          if self.generatepaths == True:
824 gcpy #              self.rapids = self.rapids.union(rapid)
825 gcpy #          else:
826 gcpy #          if self.generatepaths == False:
827 gcpy              return rapid
828 gcpy
829 gcpy          def rapidXZ(self, ex, ez):
830 gcpy              rapid = self.rapid(ex, self.ypos(), ez)
831 gcpy #          if self.generatepaths == False:
832 gcpy              return rapid
833 gcpy
834 gcpy          def rapidYZ(self, ey, ez):
835 gcpy              rapid = self.rapid(self.xpos(), ey, ez)
836 gcpy #          if self.generatepaths == False:
837 gcpy              return rapid
838 gcpy
839 gcpy          def rapidX(self, ex):
840 gcpy              rapid = self.rapid(ex, self.ypos(), self.zpos())
841 gcpy #          if self.generatepaths == False:
842 gcpy              return rapid
843 gcpy
844 gcpy          def rapidY(self, ey):
845 gcpy              rapid = self.rapid(self.xpos(), ey, self.zpos())
846 gcpy #          if self.generatepaths == False:
847 gcpy              return rapid
848 gcpy
849 gcpy          def rapidZ(self, ez):
850 gcpy              rapid = [self.rapid(self.xpos(), self.ypos(), ez)]
851 gcpy #          if self.generatepaths == True:
852 gcpy #              self.rapids = self.rapids.union(rapid)
853 gcpy #          else:
854 gcpy #          if self.generatepaths == False:
855 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):

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

Similarly, there is a series of cutline... commands as predicted above.

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

```
856 gcpy      def cutlinedxf(self, ex, ey, ez):
857 gcpy          self.dxfline(self.currenttoolnumber(), self.xpos(), self.
                    ypos(), ex, ey)
858 gcpy          self.cutline(ex, ey, ez)
859 gcpy
860 gcpy      def cutlinedxfgc(self, ex, ey, ez):
861 gcpy          self.dxfline(self.currenttoolnumber(), self.xpos(), self.
                    ypos(), ex, ey)
862 gcpy          self.writegc("G01_X", str(ex), "Y", str(ey), "Z", str(ez)
                    )
863 gcpy          self.cutline(ex, ey, ez)
864 gcpy
865 gcpy      def cutvertexdxf(self, ex, ey, ez):
866 gcpy          self.addvertex(self.currenttoolnumber(), ex, ey)
867 gcpy          self.writegc("G01_X", str(ex), "Y", str(ey), "Z", str(ez)
                    )
868 gcpy          self.cutline(ex, ey, ez)
869 gcpy
870 gcpy      def cutlineXYZwithfeed(self, ex, ey, ez, feed):
871 gcpy          return self.cutline(ex, ey, ez)
872 gcpy
873 gcpy      def cutlineXYZ(self, ex, ey, ez):
874 gcpy          return self.cutline(ex, ey, ez)
875 gcpy
876 gcpy      def cutlineXYwithfeed(self, ex, ey, feed):
877 gcpy          return self.cutline(ex, ey, self.zpos())
878 gcpy
879 gcpy      def cutlineXY(self, ex, ey):
880 gcpy          return self.cutline(ex, ey, self.zpos())
881 gcpy
882 gcpy      def cutlineXZwithfeed(self, ex, ez, feed):
883 gcpy          return self.cutline(ex, self.ypos(), ez)
884 gcpy
885 gcpy      def cutlineXZ(self, ex, ez):
886 gcpy          return self.cutline(ex, self.ypos(), ez)
887 gcpy
888 gcpy      def cutlineXwithfeed(self, ex, feed):
889 gcpy          return self.cutline(ex, self.ypos(), self.zpos())
890 gcpy
891 gcpy      def cutlineX(self, ex):
892 gcpy          return self.cutline(ex, self.ypos(), self.zpos())
893 gcpy
894 gcpy      def cutlineYZ(self, ey, ez):
895 gcpy          return self.cutline(self.xpos(), ey, ez)
896 gcpy
897 gcpy      def cutlineYwithfeed(self, ey, feed):
898 gcpy          return self.cutline(self.xpos(), ey, self.zpos())
899 gcpy
900 gcpy      def cutlineY(self, ey):
901 gcpy          return self.cutline(self.xpos(), ey, self.zpos())
```

```
902 gcpy
903 gcpy      def cutlineZgcfeed(self, ez, feed):
904 gcpy          self.writegc("G01┘Z", str(ez), "F", str(feed))
905 gcpy          return self.cutline(self.xpos(), self.ypos(), ez)
906 gcpy
907 gcpy      def cutlineZwithfeed(self, ez, feed):
908 gcpy          return self.cutline(self.xpos(), self.ypos(), ez)
909 gcpy
910 gcpy      def cutlineZ(self, ez):
911 gcpy          return self.cutline(self.xpos(), self.ypos(), ez)
```

The matching OpenSCAD command is a descriptor:

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

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

- cutarcCW — cut a partial arc described in a clock-wise direction
- cutarcCC — counter-clock-wise
- cutarcNWCW — cut the upper-left quadrant of a circle moving clockwise
- cutarcNWCC — upper-left quadrant counter-clockwise
- cutarcNECW
- cutarcNECC
- cutarcSECW
- cutarcSECC
- cutarcNECW
- cutarcNECC
- cutcircleCC — while it won't matter for generating a DXF, when G-code is implemented direction of cut will be a consideration for that
- cutcircleCW
- cutcircleCCdx
- cutcircleCWdx

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

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

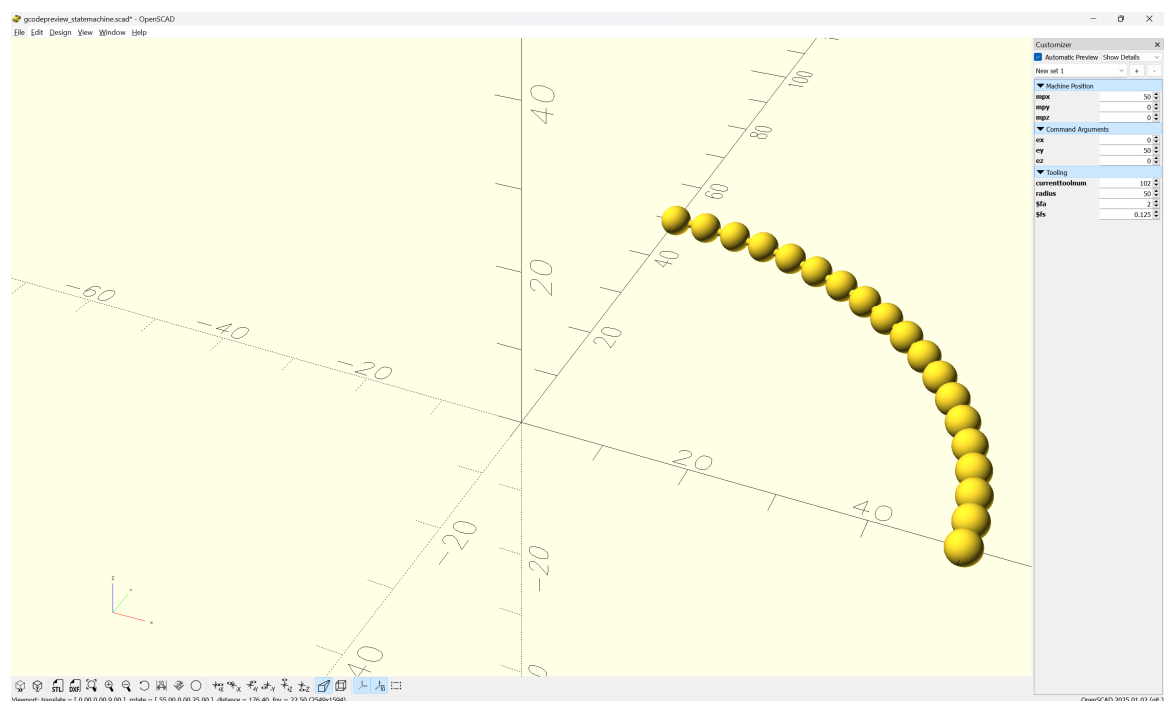
Cutting the quadrant arcs greatly simplifies the calculation and interface for the modules. A full set of 8 will be necessary, then circles will have a pair of modules (one for each cut direction) made for them.

Parameters which will need to be passed in are:

- `ex` — note that the matching origins (`bx`, `by`, `bz`) as well as the (current) toolnumber are accessed using the appropriate commands for machine position
- `ey`
- `ez` — allowing a different Z position will make possible threading and similar helical tool-paths
- `xcenter` — the center position will be specified as an absolute position which will require calculating the offset when it is used for G-code's IJ, for which `xctr/yctr` are suggested
- `ycenter`
- `radius` — while this could be calculated, passing it in as a parameter is both convenient and (potentially) could be used as a check on the other parameters
- `tpzreldim` — the relative depth (or increase in height) of the current cutting motion

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

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



The code for which makes this obvious:

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

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

/* [Tooling] */
currenttoolnum = 102;

machine_extents();

radius = 50;
```

```

$fa = 2;
$fs = 0.125;

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

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

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

```

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.)

---

```

963 gcpy      def cutarcCC(self, barc, earc, xcenter, ycenter, radius,
964 gcpy      tpzreldim, stepsizearc=1):
965 gcpy      tpzinc = tpzreldim / (earc - barc)
966 gcpy      i = barc
967 gcpy      while i < earc:
968 gcpy          self.cutline(xcenter + radius * Cos(math.radians(i)),
969 gcpy          ycenter + radius * Sin(math.radians(i)), self.zpos()
970 gcpy          +tpzinc)
971 gcpy          i += stepsizearc
972 gcpy      self.setxpos(xcenter + radius * Cos(math.radians(earc)))
973 gcpy      self.setypos(ycenter + radius * Sin(math.radians(earc)))
974 gcpy
975 gcpy      def cutarcCW(self, barc, earc, xcenter, ycenter, radius,
976 gcpy      tpzreldim, stepsizearc=1):
977 gcpy          print(str(self.zpos()))
978 gcpy          print(str(ez))
979 gcpy          print(str(barc - earc))
980 gcpy          tpzinc = ez - self.zpos() / (barc - earc)
981 gcpy          print(str(tpzinc))
982 gcpy          global toolpath
983 gcpy          print("Entering n toolpath")
984 gcpy          tpzinc = tpzreldim / (barc - earc)
985 gcpy          cts = self.currenttoolshape
986 gcpy          toolpath = cts
987 gcpy          toolpath = toolpath.translate([self.xpos(), self.ypos(),
988 gcpy          self.zpos()])
989 gcpy          toolpath = []
990 gcpy          i = barc
991 gcpy          while i > earc:
992 gcpy              self.cutline(xcenter + radius * Cos(math.radians(i)),

```

```

        ycenter + radius * Sin(math.radians(i)), self.zpos()
        +tpzinc)
988 gcpy #         self.setxpos(xcenter + radius * Cos(math.radians(i)))
989 gcpy #         self.setypos(ycenter + radius * Sin(math.radians(i)))
990 gcpy #         print(str(self.xpos()), str(self.ypos()), str(self.zpos
        ())))
991 gcpy #         self.setzpos(self.zpos()+tpzinc)
992 gcpy         i += abs(stepsizearc) * -1
993 gcpy #         self.dxfarc(self.currenttoolnumber(), xcenter, ycenter,
        radius, barc, earc)
994 gcpy #         if self.generatepaths == True:
995 gcpy #             print("Unioning n toolpath")
996 gcpy #             self.toolpaths = self.toolpaths.union(toolpath)
997 gcpy #         else:
998 gcpy         self.setxpos(xcenter + radius * Cos(math.radians(earc)))
999 gcpy         self.setypos(ycenter + radius * Sin(math.radians(earc)))
1000 gcpy #         self.toolpaths.extend(toolpath)
1001 gcpy #         if self.generatepaths == False:
1002 gcpy #             return toolpath
1003 gcpy #         else:
1004 gcpy #             return cube([0.01, 0.01, 0.01])
```

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

```

1012 gcpy         def cutarcCWdxf(self, barc, earc, xcenter, ycenter, radius,
        tpzreldim, stepsizearc=1):
1013 gcpy         self.cutarcCW(barc, earc, xcenter, ycenter, radius,
        tpzreldim, stepsizearc=1)
1014 gcpy         self.dxfarc(self.currenttoolnumber(), xcenter, ycenter,
        radius, earc, barc)
1015 gcpy #         if self.generatepaths == False:
1016 gcpy #             return toolpath
1017 gcpy #         else:
1018 gcpy #             return cube([0.01, 0.01, 0.01])
1019 gcpy
1020 gcpy         def cutarcCCdxf(self, barc, earc, xcenter, ycenter, radius,
        tpzreldim, stepsizearc=1):
1021 gcpy         self.cutarcCC(barc, earc, xcenter, ycenter, radius,
        tpzreldim, stepsizearc=1)
1022 gcpy         self.dxfarc(self.currenttoolnumber(), xcenter, ycenter,
        radius, barc, earc)
```

Matching OpenSCAD modules are easily made:

```

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

An alternate interface which matches how G2/G3 arcs are programmed in G-code is a useful option:

```

1020 gcpy         def cutquarterCCNE(self, ex, ey, ez, radius):
1021 gcpy         if self.zpos() == ez:
1022 gcpy             tpzinc = 0
1023 gcpy         else:
1024 gcpy             tpzinc = (ez - self.zpos()) / 90
1025 gcpy #             print("tpzinc ", tpzinc)
1026 gcpy         i = 1
1027 gcpy         while i < 91:
1028 gcpy             self.cutline(ex + radius * Cos(i), ey - radius + radius
        * Sin(i), self.zpos()+tpzinc)
1029 gcpy             i += 1
1030 gcpy
1031 gcpy         def cutquarterCCNW(self, ex, ey, ez, radius):
1032 gcpy         if self.zpos() == ez:
1033 gcpy             tpzinc = 0
1034 gcpy         else:
1035 gcpy             tpzinc = (ez - self.zpos()) / 90
1036 gcpy #             tpzinc = (self.zpos() + ez) / 90
1037 gcpy         print("tpzinc", tpzinc)
1038 gcpy         i = 91
1039 gcpy         while i < 181:
1040 gcpy             self.cutline(ex + radius + radius * Cos(i), ey + radius
```

```

        * Sin(i), self.zpos()+tpzinc)
1041 gcpy        i += 1
1042 gcpy
1043 gcpy    def cutquarterCCSW(self, ex, ey, ez, radius):
1044 gcpy        if self.zpos() == ez:
1045 gcpy            tpzinc = 0
1046 gcpy        else:
1047 gcpy            tpzinc = (ez - self.zpos()) / 90
1048 gcpy #            tpzinc = (self.zpos() + ez) / 90
1049 gcpy        print("tpzinc_", tpzinc)
1050 gcpy        i = 181
1051 gcpy        while i < 271:
1052 gcpy            self.cutline(ex + radius * Cos(i), ey + radius + radius
                * Sin(i), self.zpos()+tpzinc)
1053 gcpy            i += 1
1054 gcpy
1055 gcpy    def cutquarterCCSE(self, ex, ey, ez, radius):
1056 gcpy        if self.zpos() == ez:
1057 gcpy            tpzinc = 0
1058 gcpy        else:
1059 gcpy            tpzinc = (ez - self.zpos()) / 90
1060 gcpy #            tpzinc = (self.zpos() + ez) / 90
1061 gcpy        print("tpzinc_", tpzinc)
1062 gcpy        i = 271
1063 gcpy        while i < 361:
1064 gcpy            self.cutline(ex - radius + radius * Cos(i), ey + radius
                * Sin(i), self.zpos()+tpzinc)
1065 gcpy            i += 1
1066 gcpy
1067 gcpy    def cutquarterCCNEdx(self, ex, ey, ez, radius):
1068 gcpy        self.cutquarterCCNE(ex, ey, ez, radius)
1069 gcpy        self.dxfarc(self.currenttoolnumber(), ex, ey - radius,
                radius, 0, 90)
1070 gcpy
1071 gcpy    def cutquarterCCNWdx(self, ex, ey, ez, radius):
1072 gcpy        self.cutquarterCCNW(ex, ey, ez, radius)
1073 gcpy        self.dxfarc(self.currenttoolnumber(), ex + radius, ey,
                radius, 90, 180)
1074 gcpy
1075 gcpy    def cutquarterCCSWdx(self, ex, ey, ez, radius):
1076 gcpy        self.cutquarterCCSW(ex, ey, ez, radius)
1077 gcpy        self.dxfarc(self.currenttoolnumber(), ex, ey + radius,
                radius, 180, 270)
1078 gcpy
1079 gcpy    def cutquarterCCSEdx(self, ex, ey, ez, radius):
1080 gcpy        self.cutquarterCCSE(ex, ey, ez, radius)
1081 gcpy        self.dxfarc(self.currenttoolnumber(), ex - radius, ey,
                radius, 270, 360)
```

3.4.3 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.

tool diameter

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

```

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

tool diameter

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

```

1069 gcpy    def tool_diameter(self, ptd_tool, ptd_depth):
1070 gcpy # Square 122, 112, 102, 201
1071 gcpy        if ptd_tool == 122:
1072 gcpy            return 0.79375
1073 gcpy        if ptd_tool == 112:
1074 gcpy            return 1.5875
1075 gcpy        if ptd_tool == 102:
1076 gcpy            return 3.175
1077 gcpy        if ptd_tool == 201:
1078 gcpy            return 6.35
```

```

1079 gcpy # Ball 121, 111, 101, 202
1080 gcpy         if ptd_tool == 122:
1081 gcpy             if ptd_depth > 0.396875:
1082 gcpy                 return 0.79375
1083 gcpy             else:
1084 gcpy                 return ptd_tool
1085 gcpy         if ptd_tool == 112:
1086 gcpy             if ptd_depth > 0.79375:
1087 gcpy                 return 1.5875
1088 gcpy             else:
1089 gcpy                 return ptd_tool
1090 gcpy         if ptd_tool == 101:
1091 gcpy             if ptd_depth > 1.5875:
1092 gcpy                 return 3.175
1093 gcpy             else:
1094 gcpy                 return ptd_tool
1095 gcpy         if ptd_tool == 202:
1096 gcpy             if ptd_depth > 3.175:
1097 gcpy                 return 6.35
1098 gcpy             else:
1099 gcpy                 return ptd_tool
1100 gcpy # V 301, 302, 390
1101 gcpy         if ptd_tool == 301:
1102 gcpy             return ptd_tool
1103 gcpy         if ptd_tool == 302:
1104 gcpy             return ptd_tool
1105 gcpy         if ptd_tool == 390:
1106 gcpy             return ptd_tool
1107 gcpy # Keyhole
1108 gcpy         if ptd_tool == 374:
1109 gcpy             if ptd_depth < 3.175:
1110 gcpy                 return 9.525
1111 gcpy             else:
1112 gcpy                 return 6.35
1113 gcpy         if ptd_tool == 375:
1114 gcpy             if ptd_depth < 3.175:
1115 gcpy                 return 9.525
1116 gcpy             else:
1117 gcpy                 return 8
1118 gcpy         if ptd_tool == 376:
1119 gcpy             if ptd_depth < 4.7625:
1120 gcpy                 return 12.7
1121 gcpy             else:
1122 gcpy                 return 6.35
1123 gcpy         if ptd_tool == 378:
1124 gcpy             if ptd_depth < 4.7625:
1125 gcpy                 return 12.7
1126 gcpy             else:
1127 gcpy                 return 8
1128 gcpy # Dovetail
1129 gcpy         if ptd_tool == 814:
1130 gcpy             if ptd_depth > 12.7:
1131 gcpy                 return 6.35
1132 gcpy             else:
1133 gcpy                 return ptd_tool
1134 gcpy         if ptd_tool == 808079:
1135 gcpy             if ptd_depth > 20.95:
1136 gcpy                 return 6.816
1137 gcpy             else:
1138 gcpy                 return ptd_tool
1139 gcpy # Bowl Bit
1140 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
1141 gcpy         if ptd_tool == 45982:
1142 gcpy             if ptd_depth > 6.35:
1143 gcpy                 return 15.875
1144 gcpy             else:
1145 gcpy                 return ptd_tool
1146 gcpy # Tapered Ball Nose
1147 gcpy         if ptd_tool == 204:
1148 gcpy             if ptd_depth > 6.35:
1149 gcpy                 return ptd_tool
1150 gcpy         if ptd_tool == 304:
1151 gcpy             if ptd_depth > 6.35:
1152 gcpy                 return ptd_tool
1153 gcpy             else:
1154 gcpy                 return ptd_tool

```

---



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

```
1156 gcpy      def tool_radius(self, ptd_tool, ptd_depth):
1157 gcpy          tr = self.tool_diameter(ptd_tool, ptd_depth)/2
1158 gcpy          return tr
```

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

3.4.4 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

3.5 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.

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

```
1160 gcpy      def stockandtoolpaths(self, option = "stockandtoolpaths"):
1161 gcpy          if option == "stock":
1162 gcpy              show(self.stock)
1163 gcpy          elif option == "toolpaths":
1164 gcpy              show(self.toolpaths)
1165 gcpy          elif option == "rapids":
1166 gcpy              show(self.rapids)
1167 gcpy          else:
1168 gcpy              part = self.stock.difference(self.rapids)
1169 gcpy              part = self.stock.difference(self.toolpaths)
1170 gcpy              show(part)
```

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

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

3.6 Output files

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

3.6.1 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.

```
1183 gcpy      def writegc(self, *arguments):
1184 gcpy          if self.generategcode == True:
1185 gcpy              line_to_write = ""
1186 gcpy              for element in arguments:
1187 gcpy                  line_to_write += element
1188 gcpy              self.gc.write(line_to_write)
1189 gcpy              self.gc.write("\n")
1190 gcpy
1191 gcpy      def writedxf(self, toolnumber, *arguments):
1192 gcpy #          global dxfclosed
1193 gcpy          line_to_write = ""
1194 gcpy          for element in arguments:
1195 gcpy              line_to_write += element
1196 gcpy          if self.generatedxif == True:
1197 gcpy              if self.dxfclosed == False:
1198 gcpy                  self.dxf.write(line_to_write)
1199 gcpy                  self.dxf.write("\n")
1200 gcpy          if self.generatedxifs == True:
1201 gcpy              self.writedxfs(toolnumber, line_to_write)
1202 gcpy
1203 gcpy      def writedxfs(self, toolnumber, line_to_write):
1204 gcpy #          print("Processing writing toolnumber", toolnumber)
1205 gcpy #          line_to_write = ""
1206 gcpy #          for element in arguments:
1207 gcpy #              line_to_write += element
1208 gcpy          if (toolnumber == 0):
1209 gcpy              return
1210 gcpy          elif self.generatedxifs == True:
1211 gcpy              if (self.large_square_tool_num == toolnumber):
1212 gcpy                  self.dxflds.write(line_to_write)
1213 gcpy                  self.dxflds.write("\n")
1214 gcpy              if (self.small_square_tool_num == toolnumber):
1215 gcpy                  self.dxflds.write(line_to_write)
1216 gcpy                  self.dxflds.write("\n")
1217 gcpy              if (self.large_ball_tool_num == toolnumber):
1218 gcpy                  self.dxfldbl.write(line_to_write)
1219 gcpy                  self.dxfldbl.write("\n")
1220 gcpy              if (self.small_ball_tool_num == toolnumber):
1221 gcpy                  self.dxfldbl.write(line_to_write)
1222 gcpy                  self.dxfldbl.write("\n")
1223 gcpy              if (self.large_V_tool_num == toolnumber):
1224 gcpy                  self.dxfldV.write(line_to_write)
1225 gcpy                  self.dxfldV.write("\n")
1226 gcpy              if (self.small_V_tool_num == toolnumber):
1227 gcpy                  self.dxfldV.write(line_to_write)
1228 gcpy                  self.dxfldV.write("\n")
1229 gcpy              if (self.DT_tool_num == toolnumber):
1230 gcpy                  self.dxfDT.write(line_to_write)
1231 gcpy                  self.dxfDT.write("\n")
1232 gcpy              if (self.KH_tool_num == toolnumber):
1233 gcpy                  self.dxfKH.write(line_to_write)
1234 gcpy                  self.dxfKH.write("\n")
1235 gcpy              if (self.Roundover_tool_num == toolnumber):
1236 gcpy                  self.dxfRt.write(line_to_write)
1237 gcpy                  self.dxfRt.write("\n")
1238 gcpy              if (self.MISC_tool_num == toolnumber):
1239 gcpy                  self.dxfMt.write(line_to_write)
1240 gcpy                  self.dxfMt.write("\n")
```

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

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

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

```
111 gpcscad module opendxfile(basefilename){
112 gpcscad     gcp.opendxfile(basefilename);
113 gpcscad }
114 gpcscad
115 gpcscad module opendxfiles(Base_filename, large_square_tool_num,
    small_square_tool_num, large_ball_tool_num, small_ball_tool_num,
    large_V_tool_num, small_V_tool_num, DT_tool_num, KH_tool_num,
    Roundover_tool_num, MISC_tool_num) {
116 gpcscad     gcp.opendxfiles(Base_filename, large_square_tool_num,
    small_square_tool_num, large_ball_tool_num,
    small_ball_tool_num, large_V_tool_num, small_V_tool_num,
    DT_tool_num, KH_tool_num, Roundover_tool_num, MISC_tool_num)
    ;
117 gpcscad }
```

opengcodefile      With matching OpenSCAD commands: opengcodefile for OpenSCAD:

```
119 gpcscad module opengcodefile(basefilename, currenttoolnum, toolradius,
    plunge, feed, speed) {
120 gpcscad     gcp.opengcodefile(basefilename, currenttoolnum, toolradius,
    plunge, feed, speed);
121 gpcscad }
```

and Python:

```
1242 gcpy      def opengcodefile(self, basefilename = "export",
1243 gcpy          currenttoolnum = 102,
1244 gcpy          toolradius = 3.175,
1245 gcpy          plunge = 400,
1246 gcpy          feed = 1600,
1247 gcpy          speed = 10000
1248 gcpy      ):
1249 gcpy          self.basefilename = basefilename
1250 gcpy          self.currenttoolnum = currenttoolnum
1251 gcpy          self.toolradius = toolradius
1252 gcpy          self.plunge = plunge
1253 gcpy          self.feed = feed
1254 gcpy          self.speed = speed
1255 gcpy          if self.generategcode == True:
1256 gcpy              self.gcodefilename = basefilename + ".nc"
1257 gcpy              self.gc = open(self.gcodefilename, "w")
1258 gcpy              self.writegc("(Design␣File:␣" + self.basefilename + ")")
1259 gcpy
1260 gcpy      def opendxfile(self, basefilename = "export"):
1261 gcpy          self.basefilename = basefilename
1262 gcpy          # global generatedxfs
1263 gcpy          # global dxfclosed
1264 gcpy          self.dxfclosed = False
1265 gcpy          self.dxfcolor = "Black"
1266 gcpy          if self.generatedxfs == True:
1267 gcpy              self.generatedxfs = False
1268 gcpy              self.dxffilename = basefilename + ".dxf"
1269 gcpy              self.dxf = open(self.dxffilename, "w")
1270 gcpy              self.dxfpreamble(-1)
1271 gcpy
1272 gcpy      def opendxfiles(self, basefilename = "export",
1273 gcpy          large_square_tool_num = 0,
1274 gcpy          small_square_tool_num = 0,
1275 gcpy          large_ball_tool_num = 0,
1276 gcpy          small_ball_tool_num = 0,
1277 gcpy          large_V_tool_num = 0,
1278 gcpy          small_V_tool_num = 0,
1279 gcpy          DT_tool_num = 0,
1280 gcpy          KH_tool_num = 0,
1281 gcpy          Roundover_tool_num = 0,
1282 gcpy          MISC_tool_num = 0):
1283 gcpy          # global generatedxfs
1284 gcpy          self.basefilename = basefilename
1285 gcpy          self.generatedxfs = True
1286 gcpy          self.large_square_tool_num = large_square_tool_num
1287 gcpy          self.small_square_tool_num = small_square_tool_num
1288 gcpy          self.large_ball_tool_num = large_ball_tool_num
1289 gcpy          self.small_ball_tool_num = small_ball_tool_num
1290 gcpy          self.large_V_tool_num = large_V_tool_num
```

```
1291 gcpy self.small_V_tool_num = small_V_tool_num
1292 gcpy self.DT_tool_num = DT_tool_num
1293 gcpy self.KH_tool_num = KH_tool_num
1294 gcpy self.Roundover_tool_num = Roundover_tool_num
1295 gcpy self.MISC_tool_num = MISC_tool_num
1296 gcpy if self.generatedxf == True:
1297 gcpy     if (large_square_tool_num > 0):
1298 gcpy         self.dxf_lgsqfilename = basefilename + str(
1299 gcpy             large_square_tool_num) + ".dxf"
1300 gcpy         print("Opening ", str(self.dxf_lgsqfilename))
1301 gcpy         self.dxf_lgsq = open(self.dxf_lgsqfilename, "w")
1302 gcpy     if (small_square_tool_num > 0):
1303 gcpy         print("Opening small square")
1304 gcpy         self.dxf_smsqfilename = basefilename + str(
1305 gcpy             small_square_tool_num) + ".dxf"
1306 gcpy         self.dxf_smsq = open(self.dxf_smsqfilename, "w")
1307 gcpy     if (large_ball_tool_num > 0):
1308 gcpy         print("Opening large ball")
1309 gcpy         self.dxf_lgblfilename = basefilename + str(
1310 gcpy             large_ball_tool_num) + ".dxf"
1311 gcpy         self.dxf_lgbl = open(self.dxf_lgblfilename, "w")
1312 gcpy     if (small_ball_tool_num > 0):
1313 gcpy         print("Opening small ball")
1314 gcpy         self.dxf_smbfilename = basefilename + str(
1315 gcpy             small_ball_tool_num) + ".dxf"
1316 gcpy         self.dxf_smb = open(self.dxf_smbfilename, "w")
1317 gcpy     if (large_V_tool_num > 0):
1318 gcpy         print("Opening large V")
1319 gcpy         self.dxf_lgVfilename = basefilename + str(
1320 gcpy             large_V_tool_num) + ".dxf"
1321 gcpy         self.dxf_lgV = open(self.dxf_lgVfilename, "w")
1322 gcpy     if (small_V_tool_num > 0):
1323 gcpy         print("Opening small V")
1324 gcpy         self.dxf_smVfilename = basefilename + str(
1325 gcpy             small_V_tool_num) + ".dxf"
1326 gcpy         self.dxf_smV = open(self.dxf_smVfilename, "w")
1327 gcpy     if (DT_tool_num > 0):
1328 gcpy         print("Opening DT")
1329 gcpy         self.dxf_DTfilename = basefilename + str(DT_tool_num
1330 gcpy             ) + ".dxf"
1331 gcpy         self.dxf_DT = open(self.dxf_DTfilename, "w")
1332 gcpy     if (KH_tool_num > 0):
1333 gcpy         print("Opening KH")
1334 gcpy         self.dxf_KHfilename = basefilename + str(KH_tool_num
1335 gcpy             ) + ".dxf"
1336 gcpy         self.dxf_KH = open(self.dxf_KHfilename, "w")
1337 gcpy     if (Roundover_tool_num > 0):
1338 gcpy         print("Opening Rt")
1339 gcpy         self.dxf_Rtfilename = basefilename + str(
1340 gcpy             Roundover_tool_num) + ".dxf"
1341 gcpy         self.dxf_Rt = open(self.dxf_Rtfilename, "w")
1342 gcpy     if (MISC_tool_num > 0):
1343 gcpy         print("Opening Mt")
1344 gcpy         self.dxf_Mtfilename = basefilename + str(
1345 gcpy             MISC_tool_num) + ".dxf"
1346 gcpy         self.dxf_Mt = open(self.dxf_Mtfilename, "w")
```

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

```
1336 gcpy if (large_square_tool_num > 0):
1337 gcpy     self.dxfpreamble(large_square_tool_num)
1338 gcpy if (small_square_tool_num > 0):
1339 gcpy     self.dxfpreamble(small_square_tool_num)
1340 gcpy if (large_ball_tool_num > 0):
1341 gcpy     self.dxfpreamble(large_ball_tool_num)
1342 gcpy if (small_ball_tool_num > 0):
1343 gcpy     self.dxfpreamble(small_ball_tool_num)
1344 gcpy if (large_V_tool_num > 0):
1345 gcpy     self.dxfpreamble(large_V_tool_num)
1346 gcpy if (small_V_tool_num > 0):
1347 gcpy     self.dxfpreamble(small_V_tool_num)
1348 gcpy if (DT_tool_num > 0):
1349 gcpy     self.dxfpreamble(DT_tool_num)
1350 gcpy if (KH_tool_num > 0):
1351 gcpy     self.dxfpreamble(KH_tool_num)
1352 gcpy if (Roundover_tool_num > 0):
1353 gcpy     self.dxfpreamble(Roundover_tool_num)
```

```
1354 gcpy          if (MISC_tool_num > 0):
1355 gcpy          self.dxfpreamble(MISC_tool_num)
```

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

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

3.6.2 DXF Overview

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

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

**3.6.2.1 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.

```
1357 gcpy      def dxfpreamble(self, tn):
1358 gcpy #          self.writedxf(tn, str(tn))
1359 gcpy          self.writedxf(tn, "0")
1360 gcpy          self.writedxf(tn, "SECTION")
1361 gcpy          self.writedxf(tn, "2")
1362 gcpy          self.writedxf(tn, "ENTITIES")
```

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

dxfline • a line dxfline

beginpolyline • connected lines beginpolyline/addvertex/closepolyline

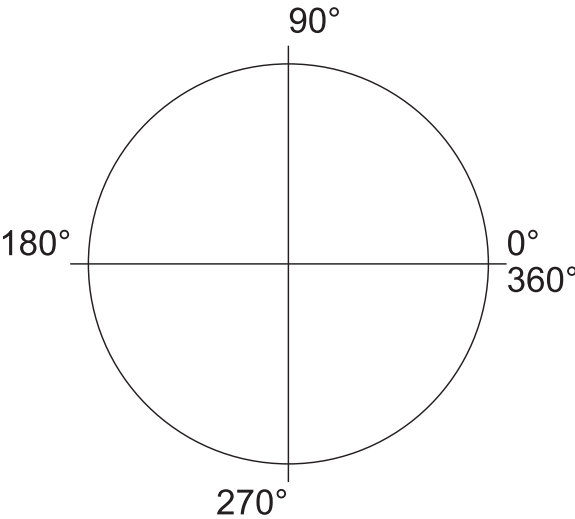
addvertex • arc dxfarc

closepolyline • circle — a notable option would be for the arc to close on itself, creating a circle dxfcircle

dxfarc

dxfcircle

DXF orders arcs counter-clockwise:



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

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

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

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

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

```
1364 gcpy      def setdxfcOLOR(self, color):
1365 gcpy          self.dxfcolor = color
1366 gcpy          self.cutcolor = color
1367 gcpy
1368 gcpy      def writedxfcOLOR(self, tn):
1369 gcpy          self.writedxf(tn, "8")
1370 gcpy          if (self.dxfcolor == "Black"):
1371 gcpy              self.writedxf(tn, "Layer_Black")
1372 gcpy          if (self.dxfcolor == "Red"):
1373 gcpy              self.writedxf(tn, "Layer_Red")
1374 gcpy          if (self.dxfcolor == "Yellow"):
1375 gcpy              self.writedxf(tn, "Layer_Yellow")
1376 gcpy          if (self.dxfcolor == "Green"):
1377 gcpy              self.writedxf(tn, "Layer_Green")
1378 gcpy          if (self.dxfcolor == "Cyan"):
1379 gcpy              self.writedxf(tn, "Layer_Cyan")
1380 gcpy          if (self.dxfcolor == "Blue"):
1381 gcpy              self.writedxf(tn, "Layer_Blue")
1382 gcpy          if (self.dxfcolor == "Magenta"):
1383 gcpy              self.writedxf(tn, "Layer_Magenta")
1384 gcpy          if (self.dxfcolor == "White"):
1385 gcpy              self.writedxf(tn, "Layer_White")
1386 gcpy          if (self.dxfcolor == "Dark_Gray"):
1387 gcpy              self.writedxf(tn, "Layer_Dark_Gray")
1388 gcpy          if (self.dxfcolor == "Light_Gray"):
1389 gcpy              self.writedxf(tn, "Layer_Light_Gray")
1390 gcpy
1391 gcpy          self.writedxf(tn, "62")
1392 gcpy          if (self.dxfcolor == "Black"):
1393 gcpy              self.writedxf(tn, "0")
1394 gcpy          if (self.dxfcolor == "Red"):
1395 gcpy              self.writedxf(tn, "1")
1396 gcpy          if (self.dxfcolor == "Yellow"):
1397 gcpy              self.writedxf(tn, "2")
1398 gcpy          if (self.dxfcolor == "Green"):
1399 gcpy              self.writedxf(tn, "3")
1400 gcpy          if (self.dxfcolor == "Cyan"):
1401 gcpy              self.writedxf(tn, "4")
1402 gcpy          if (self.dxfcolor == "Blue"):
1403 gcpy              self.writedxf(tn, "5")
1404 gcpy          if (self.dxfcolor == "Magenta"):
1405 gcpy              self.writedxf(tn, "6")
1406 gcpy          if (self.dxfcolor == "White"):
1407 gcpy              self.writedxf(tn, "7")
1408 gcpy          if (self.dxfcolor == "Dark_Gray"):
1409 gcpy              self.writedxf(tn, "8")
1410 gcpy          if (self.dxfcolor == "Light_Gray"):
1411 gcpy              self.writedxf(tn, "9")
1412 gcpy
1413 gcpy
1414 gcpy
1415 gcpy #
1416 gcpy      self.writedxfcOLOR(tn)
1417 gcpy #
1418 gcpy      self.writedxf(tn, "10")
```

---

```
123 gcpSCAD module setdxfcOLOR(color){
124 gcpSCAD     gcp.setdxfcOLOR(color);
125 gcpSCAD }
```

---

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.

```
1412 gcpy      def dxfline(self, tn, xbegin, ybegin, xend, yend):
1413 gcpy          self.writedxf(tn, "0")
1414 gcpy          self.writedxf(tn, "LINE")
1415 gcpy #
1416 gcpy          self.writedxfcOLOR(tn)
1417 gcpy #
1418 gcpy          self.writedxf(tn, "10")
```

```
1419 gcpy          self.writedxf(tn, str(xbegin))
1420 gcpy          self.writedxf(tn, "20")
1421 gcpy          self.writedxf(tn, str(ybegin))
1422 gcpy          self.writedxf(tn, "30")
1423 gcpy          self.writedxf(tn, "0.0")
1424 gcpy          self.writedxf(tn, "11")
1425 gcpy          self.writedxf(tn, str(xend))
1426 gcpy          self.writedxf(tn, "21")
1427 gcpy          self.writedxf(tn, str(yend))
1428 gcpy          self.writedxf(tn, "31")
1429 gcpy          self.writedxf(tn, "0.0")
```

---

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

First, begin the polyline:

```
1431 gcpy          def beginpolyline(self, tn):#, xbegin, ybegin
1432 gcpy          self.writedxf(tn, "0")
1433 gcpy          self.writedxf(tn, "POLYLINE")
1434 gcpy          self.writedxf(tn, "8")
1435 gcpy          self.writedxf(tn, "default")
1436 gcpy          self.writedxf(tn, "66")
1437 gcpy          self.writedxf(tn, "1")
1438 gcpy          #
1439 gcpy          self.writedxfcolor(tn)
1440 gcpy          #
1441 gcpy          #          self.writedxf(tn, "10")
1442 gcpy          #          self.writedxf(tn, str(xbegin))
1443 gcpy          #          self.writedxf(tn, "20")
1444 gcpy          #          self.writedxf(tn, str(ybegin))
1445 gcpy          #          self.writedxf(tn, "30")
1446 gcpy          #          self.writedxf(tn, "0.0")
1447 gcpy          self.writedxf(tn, "70")
1448 gcpy          self.writedxf(tn, "0")
```

---

then add as many vertexes as are wanted:

```
1449 gcpy          def addvertex(self, tn, xend, yend):
1450 gcpy          self.writedxf(tn, "0")
1451 gcpy          self.writedxf(tn, "VERTEX")
1452 gcpy          self.writedxf(tn, "8")
1453 gcpy          self.writedxf(tn, "default")
1454 gcpy          self.writedxf(tn, "70")
1455 gcpy          self.writedxf(tn, "32")
1456 gcpy          self.writedxf(tn, "10")
1457 gcpy          self.writedxf(tn, str(xend))
1458 gcpy          self.writedxf(tn, "20")
1459 gcpy          self.writedxf(tn, str(yend))
1460 gcpy          self.writedxf(tn, "30")
1461 gcpy          self.writedxf(tn, "0.0")
```

---

then end the sequence:

```
1463 gcpy          def closepolyline(self, tn):
1464 gcpy          self.writedxf(tn, "0")
1465 gcpy          self.writedxf(tn, "SEQEND")
```

---

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

```
1467 gcpy          def dxfarc(self, tn, xcenter, ycenter, radius, anglebegin,
1468 gcpy          endangle):
1469 gcpy          if (self.generatedxf == True):
1470 gcpy          self.writedxf(tn, "0")
1471 gcpy          self.writedxf(tn, "ARC")
1472 gcpy          #
1473 gcpy          self.writedxfcolor(tn)
1474 gcpy          #
1475 gcpy          self.writedxf(tn, "10")
1476 gcpy          self.writedxf(tn, str(xcenter))
1477 gcpy          self.writedxf(tn, "20")
1478 gcpy          self.writedxf(tn, str(ycenter))
1479 gcpy          self.writedxf(tn, "40")
1480 gcpy          self.writedxf(tn, str(radius))
```

---



```

1480 gcpy                self.writedxf(tn, "50")
1481 gcpy                self.writedxf(tn, str(anglebegin))
1482 gcpy                self.writedxf(tn, "51")
1483 gcpy                self.writedxf(tn, str(endangle))
1484 gcpy
1485 gcpy                def gcodearc(self, tn, xcenter, ycenter, radius, anglebegin,
                                endangle):
1486 gcpy                    if (self.generategcode == True):
1487 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.

---

```

1489 gcpy                def cutarcNECCdxf(self, ex, ey, ez, xcenter, ycenter, radius):
1490 gcpy                #            global toolpath
1491 gcpy                #            toolpath = self.currenttool()
1492 gcpy                #            toolpath = toolpath.translate([self.xpos(), self.ypos(),
self.zpos()])
1493 gcpy                self.dxfarc(self.currenttoolnumber(), xcenter, ycenter,
                                radius, 0, 90)
1494 gcpy                if (self.zpos == ez):
1495 gcpy                    self.settzpos(0)
1496 gcpy                else:
1497 gcpy                    self.settzpos((self.zpos()-ez)/90)
1498 gcpy                #            self.setxpos(ex)
1499 gcpy                #            self.setypos(ey)
1500 gcpy                #            self.setzpos(ez)
1501 gcpy                #            if self.generatepaths == True:
1502 gcpy                #                print("Unioning cutarcNECCdxf toolpath")
1503 gcpy                self.arcloop(1, 90, xcenter, ycenter, radius)
1504 gcpy                #            self.toolpaths = self.toolpaths.union(toolpath)
1505 gcpy                #            else:
1506 gcpy                #                toolpath = self.arcloop(1, 90, xcenter, ycenter,
radius)
1507 gcpy                #                print("Returning cutarcNECCdxf toolpath")
1508 gcpy                return toolpath
1509 gcpy
1510 gcpy                def cutarcNWCCdxf(self, ex, ey, ez, xcenter, ycenter, radius):
1511 gcpy                #            global toolpath
1512 gcpy                #            toolpath = self.currenttool()
1513 gcpy                #            toolpath = toolpath.translate([self.xpos(), self.ypos(),
self.zpos()])
1514 gcpy                self.dxfarc(self.currenttoolnumber(), xcenter, ycenter,
                                radius, 90, 180)
1515 gcpy                if (self.zpos == ez):
1516 gcpy                    self.settzpos(0)
1517 gcpy                else:
1518 gcpy                    self.settzpos((self.zpos()-ez)/90)
1519 gcpy                #            self.setxpos(ex)
1520 gcpy                #            self.setypos(ey)
1521 gcpy                #            self.setzpos(ez)
1522 gcpy                #            if self.generatepaths == True:
1523 gcpy                #                self.arcloop(91, 180, xcenter, ycenter, radius)
1524 gcpy                #                self.toolpaths = self.toolpaths.union(toolpath)
1525 gcpy                #            else:
1526 gcpy                toolpath = self.arcloop(91, 180, xcenter, ycenter, radius)
1527 gcpy                return toolpath
1528 gcpy
1529 gcpy                def cutarcSWCCdxf(self, ex, ey, ez, xcenter, ycenter, radius):
1530 gcpy                #            global toolpath
1531 gcpy                #            toolpath = self.currenttool()
1532 gcpy                #            toolpath = toolpath.translate([self.xpos(), self.ypos(),
self.zpos()])
1533 gcpy                self.dxfarc(self.currenttoolnumber(), xcenter, ycenter,
                                radius, 180, 270)
1534 gcpy                if (self.zpos == ez):
1535 gcpy                    self.settzpos(0)
1536 gcpy                else:
1537 gcpy                    self.settzpos((self.zpos()-ez)/90)
1538 gcpy                #            self.setxpos(ex)
1539 gcpy                #            self.setypos(ey)
1540 gcpy                #            self.setzpos(ez)
1541 gcpy                if self.generatepaths == True:
1542 gcpy                    self.arcloop(181, 270, xcenter, ycenter, radius)
1543 gcpy                #            self.toolpaths = self.toolpaths.union(toolpath)
1544 gcpy                else:
1545 gcpy                    toolpath = self.arcloop(181, 270, xcenter, ycenter,
radius)

```

```

1546 gcpy                return toolpath
1547 gcpy
1548 gcpy    def cutarcSECCdxf(self, ex, ey, ez, xcenter, ycenter, radius):
1549 gcpy #        global toolpath
1550 gcpy #        toolpath = self.currenttool()
1551 gcpy #        toolpath = toolpath.translate([self.xpos(), self.ypos(),
self.zpos()])
1552 gcpy        self.dxfarc(self.currenttoolnumber(), xcenter, ycenter,
radius, 270, 360)
1553 gcpy        if (self.zpos == ez):
1554 gcpy            self.settzpos(0)
1555 gcpy        else:
1556 gcpy            self.settzpos((self.zpos()-ez)/90)
1557 gcpy #        self.setxpos(ex)
1558 gcpy #        self.setypos(ey)
1559 gcpy #        self.setzpos(ez)
1560 gcpy        if self.generatepaths == True:
1561 gcpy            self.arcloop(271, 360, xcenter, ycenter, radius)
1562 gcpy #        self.toolpaths = self.toolpaths.union(toolpath)
1563 gcpy        else:
1564 gcpy            toolpath = self.arcloop(271, 360, xcenter, ycenter,
radius)
1565 gcpy            return toolpath
1566 gcpy
1567 gcpy    def cutarcNECWdxf(self, ex, ey, ez, xcenter, ycenter, radius):
1568 gcpy #        global toolpath
1569 gcpy #        toolpath = self.currenttool()
1570 gcpy #        toolpath = toolpath.translate([self.xpos(), self.ypos(),
self.zpos()])
1571 gcpy        self.dxfarc(self.currenttoolnumber(), xcenter, ycenter,
radius, 0, 90)
1572 gcpy        if (self.zpos == ez):
1573 gcpy            self.settzpos(0)
1574 gcpy        else:
1575 gcpy            self.settzpos((self.zpos()-ez)/90)
1576 gcpy #        self.setxpos(ex)
1577 gcpy #        self.setypos(ey)
1578 gcpy #        self.setzpos(ez)
1579 gcpy        if self.generatepaths == True:
1580 gcpy            self.narcloop(89, 0, xcenter, ycenter, radius)
1581 gcpy #        self.toolpaths = self.toolpaths.union(toolpath)
1582 gcpy        else:
1583 gcpy            toolpath = self.narcloop(89, 0, xcenter, ycenter,
radius)
1584 gcpy            return toolpath
1585 gcpy
1586 gcpy    def cutarcSECWdxf(self, ex, ey, ez, xcenter, ycenter, radius):
1587 gcpy #        global toolpath
1588 gcpy #        toolpath = self.currenttool()
1589 gcpy #        toolpath = toolpath.translate([self.xpos(), self.ypos(),
self.zpos()])
1590 gcpy        self.dxfarc(self.currenttoolnumber(), xcenter, ycenter,
radius, 270, 360)
1591 gcpy        if (self.zpos == ez):
1592 gcpy            self.settzpos(0)
1593 gcpy        else:
1594 gcpy            self.settzpos((self.zpos()-ez)/90)
1595 gcpy #        self.setxpos(ex)
1596 gcpy #        self.setypos(ey)
1597 gcpy #        self.setzpos(ez)
1598 gcpy        if self.generatepaths == True:
1599 gcpy            self.narcloop(359, 270, xcenter, ycenter, radius)
1600 gcpy #        self.toolpaths = self.toolpaths.union(toolpath)
1601 gcpy        else:
1602 gcpy            toolpath = self.narcloop(359, 270, xcenter, ycenter,
radius)
1603 gcpy            return toolpath
1604 gcpy
1605 gcpy    def cutarcSWCWdxf(self, ex, ey, ez, xcenter, ycenter, radius):
1606 gcpy #        global toolpath
1607 gcpy #        toolpath = self.currenttool()
1608 gcpy #        toolpath = toolpath.translate([self.xpos(), self.ypos(),
self.zpos()])
1609 gcpy        self.dxfarc(self.currenttoolnumber(), xcenter, ycenter,
radius, 180, 270)
1610 gcpy        if (self.zpos == ez):
1611 gcpy            self.settzpos(0)
1612 gcpy        else:

```

```
1613 gcpy          self.settzpos((self.zpos()-ez)/90)
1614 gcpy #        self.setxpos(ex)
1615 gcpy #        self.setypos(ey)
1616 gcpy #        self.setzpos(ez)
1617 gcpy          if self.generatepaths == True:
1618 gcpy            self.narcloop(269, 180, xcenter, ycenter, radius)
1619 gcpy #          self.toolpaths = self.toolpaths.union(toolpath)
1620 gcpy          else:
1621 gcpy            toolpath = self.narcloop(269, 180, xcenter, ycenter,
              radius)
1622 gcpy            return toolpath
1623 gcpy
1624 gcpy          def cutarcNWCWdxf(self, ex, ey, ez, xcenter, ycenter, radius):
1625 gcpy #            global toolpath
1626 gcpy #            toolpath = self.currentttool()
1627 gcpy #            toolpath = toolpath.translate([self.xpos(), self.ypos(),
self.zpos()])
1628 gcpy            self.dxfarc(self.currenttoolnumber(), xcenter, ycenter,
              radius, 90, 180)
1629 gcpy          if (self.zpos == ez):
1630 gcpy            self.settzpos(0)
1631 gcpy          else:
1632 gcpy            self.settzpos((self.zpos()-ez)/90)
1633 gcpy #          self.setxpos(ex)
1634 gcpy #          self.setypos(ey)
1635 gcpy #          self.setzpos(ez)
1636 gcpy          if self.generatepaths == True:
1637 gcpy            self.narcloop(179, 90, xcenter, ycenter, radius)
1638 gcpy #          self.toolpaths = self.toolpaths.union(toolpath)
1639 gcpy          else:
1640 gcpy            toolpath = self.narcloop(179, 90, xcenter, ycenter,
              radius)
1641 gcpy            return toolpath
```

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

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

```
1643 gcpy          def arcCCgc(self, ex, ey, ez, xcenter, ycenter, radius):
1644 gcpy            self.writegc("G03_X", str(ex), "Y", str(ey), "Z", str(ez)
              , "R", str(radius))
1645 gcpy
1646 gcpy          def arcCWgc(self, ex, ey, ez, xcenter, ycenter, radius):
1647 gcpy            self.writegc("G02_X", str(ex), "Y", str(ey), "Z", str(ez)
              , "R", str(radius))
```

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

```
1649 gcpy          def cutarcNECCdxfgc(self, ex, ey, ez, xcenter, ycenter, radius)
:
1650 gcpy            self.arcCCgc(ex, ey, ez, xcenter, ycenter, radius)
1651 gcpy          if self.generatepaths == True:
1652 gcpy            self.cutarcNECCdxf(ex, ey, ez, xcenter, ycenter, radius
              )
1653 gcpy          else:
1654 gcpy            return self.cutarcNECCdxf(ex, ey, ez, xcenter, ycenter,
              radius)
1655 gcpy
1656 gcpy          def cutarcNWCCdxfgc(self, ex, ey, ez, xcenter, ycenter, radius)
:
1657 gcpy            self.arcCCgc(ex, ey, ez, xcenter, ycenter, radius)
1658 gcpy          if self.generatepaths == False:
1659 gcpy            return self.cutarcNWCCdxf(ex, ey, ez, xcenter, ycenter,
              radius)
1660 gcpy
1661 gcpy          def cutarcSWCCdxfgc(self, ex, ey, ez, xcenter, ycenter, radius)
:
1662 gcpy            self.arcCCgc(ex, ey, ez, xcenter, ycenter, radius)
1663 gcpy          if self.generatepaths == False:
1664 gcpy            return self.cutarcSWCCdxf(ex, ey, ez, xcenter, ycenter,
              radius)
1665 gcpy
1666 gcpy          def cutarcSECCdxfgc(self, ex, ey, ez, xcenter, ycenter, radius)
:
```

```
1667 gcpy          self.arcCCgc(ex, ey, ez, xcenter, ycenter, radius)
1668 gcpy          if self.generatepaths == False:
1669 gcpy              return self.cutarcSECCdxfc(ex, ey, ez, xcenter, ycenter,
              radius)

1670 gcpy
1671 gcpy          def cutarcNECWdxfgc(self, ex, ey, ez, xcenter, ycenter, radius)
              :
1672 gcpy              self.arcCWgc(ex, ey, ez, xcenter, ycenter, radius)
1673 gcpy              if self.generatepaths == False:
1674 gcpy                  return self.cutarcNECWdxfc(ex, ey, ez, xcenter, ycenter,
              radius)

1675 gcpy
1676 gcpy          def cutarcSECWdxfgc(self, ex, ey, ez, xcenter, ycenter, radius)
              :
1677 gcpy              self.arcCWgc(ex, ey, ez, xcenter, ycenter, radius)
1678 gcpy              if self.generatepaths == False:
1679 gcpy                  return self.cutarcSECWdxfc(ex, ey, ez, xcenter, ycenter,
              radius)

1680 gcpy
1681 gcpy          def cutarcSWCWdxfgc(self, ex, ey, ez, xcenter, ycenter, radius)
              :
1682 gcpy              self.arcCWgc(ex, ey, ez, xcenter, ycenter, radius)
1683 gcpy              if self.generatepaths == False:
1684 gcpy                  return self.cutarcSWCWdxfc(ex, ey, ez, xcenter, ycenter,
              radius)

1685 gcpy
1686 gcpy          def cutarcNWCWdxfgc(self, ex, ey, ez, xcenter, ycenter, radius)
              :
1687 gcpy              self.arcCWgc(ex, ey, ez, xcenter, ycenter, radius)
1688 gcpy              if self.generatepaths == False:
1689 gcpy                  return self.cutarcNWCWdxfc(ex, ey, ez, xcenter, ycenter,
              radius)

```

---

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

---

3.6.3 G-code Overview

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

Command/Module	G-code
opengcodefile(s)(...); setupstock(...)	(export.nc) (stockMin: -109.5, -75mm, -8.35mm) (stockMax:109.5mm, 75mm, 0.00mm) (STOCK/BLOCK, 219, 150, 8.35, 109.5, 75, 8.35) G90 G21
movetosafez()	(Move to safe Z to avoid workholding) G53G0Z-5.000
toolchange(...);	(TOOL/MILL, 3.17, 0.00, 0.00, 0.00) M6T102 M03S16000
cutoneaxis_setfeed(...);	(PREPOSITION FOR RAPID PLUNGE) GOX0Y0 Z0.25 G1Z0F100 G1 X109.5 Y75 Z-8.35F400 Z9
cutwithfeed(...);	
closegcodefile();	M05 M02

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

G-code	Command/Module
(Design File: ) (stockMin:0.00mm, -152.40mm, -34.92mm) (stockMax:109.50mm, -77.40mm, 0.00mm) (STOCK/BLOCK, 109.50, 75.00, 34.92, 0.00, 152.40, 34.92) G90 G21	opengcodefile(s)(...); setupstock(...);
(Move to safe Z to avoid workholding) G53G0Z-5.000	movetosafez()
(Toolpath: Contour Toolpath 1) M05 (TOOL/MILL, 3.17, 0.00, 0.00, 0.00) M6T102 M03S10000	toolchange(...);
(PREPOSITION FOR RAPID PLUNGE) GOX0.000Y-152.400 Z0.250	writecomment(...) rapid(...) rapid(...)
G1Z-1.000F203.2 X109.500Y-77.400F508.0 X57.918Y16.302Z-0.726 Y22.023Z-1.023 X61.190Z-0.681 Y21.643 X57.681 Z12.700	cutwithfeed(...); cutwithfeed(...);
M05 M02	closegcodefile();

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

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

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

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

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

```
1691 gcpy      def dxfpostamble(self, tn):
1692 gcpy #          self.writedxf(tn, str(tn))
1693 gcpy          self.writedxf(tn, "0")
1694 gcpy          self.writedxf(tn, "ENDSEC")
1695 gcpy          self.writedxf(tn, "0")
1696 gcpy          self.writedxf(tn, "EOF")

1698 gcpy      def gcodepostamble(self):
1699 gcpy          self.writegc("Z12.700")
1700 gcpy          self.writegc("M05")
1701 gcpy          self.writegc("M02")
```

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

```
1703 gcpy      def closegcodefile(self):
1704 gcpy          if self.generategcode == True:
1705 gcpy              self.gcodepostamble()
1706 gcpy              self.gc.close()

1708 gcpy      def closedxfile(self):
1709 gcpy          if self.generatedxfile == True:
1710 gcpy #              global dxfclose
1711 gcpy              self.dxfpostamble(-1)
1712 gcpy #              self.dxfclosed = True
1713 gcpy              self.dxf.close()

1715 gcpy      def closedxfiles(self):
1716 gcpy          if self.generatedxfiles == True:
1717 gcpy              if (self.large_square_tool_num > 0):
1718 gcpy                  self.dxfpostamble(self.large_square_tool_num)
1719 gcpy              if (self.small_square_tool_num > 0):
1720 gcpy                  self.dxfpostamble(self.small_square_tool_num)
1721 gcpy              if (self.large_ball_tool_num > 0):
1722 gcpy                  self.dxfpostamble(self.large_ball_tool_num)
1723 gcpy              if (self.small_ball_tool_num > 0):
1724 gcpy                  self.dxfpostamble(self.small_ball_tool_num)
1725 gcpy              if (self.large_V_tool_num > 0):
1726 gcpy                  self.dxfpostamble(self.large_V_tool_num)
1727 gcpy              if (self.small_V_tool_num > 0):
1728 gcpy                  self.dxfpostamble(self.small_V_tool_num)
1729 gcpy              if (self.DT_tool_num > 0):
1730 gcpy                  self.dxfpostamble(self.DT_tool_num)
1731 gcpy              if (self.KH_tool_num > 0):
1732 gcpy                  self.dxfpostamble(self.KH_tool_num)
1733 gcpy              if (self.Roundover_tool_num > 0):
1734 gcpy                  self.dxfpostamble(self.Roundover_tool_num)
1735 gcpy              if (self.MISC_tool_num > 0):
1736 gcpy                  self.dxfpostamble(self.MISC_tool_num)
1737 gcpy
1738 gcpy              if (self.large_square_tool_num > 0):
1739 gcpy                  self.dxfllsq.close()
1740 gcpy              if (self.small_square_tool_num > 0):
1741 gcpy                  self.dxfllsq.close()
1742 gcpy              if (self.large_ball_tool_num > 0):
1743 gcpy                  self.dxfllbl.close()
1744 gcpy              if (self.small_ball_tool_num > 0):
1745 gcpy                  self.dxfllbl.close()
1746 gcpy              if (self.large_V_tool_num > 0):
1747 gcpy                  self.dxfllV.close()
1748 gcpy              if (self.small_V_tool_num > 0):
1749 gcpy                  self.dxfllV.close()
1750 gcpy              if (self.DT_tool_num > 0):
1751 gcpy                  self.dxfDT.close()
1752 gcpy              if (self.KH_tool_num > 0):
1753 gcpy                  self.dxfKH.close()
1754 gcpy              if (self.Roundover_tool_num > 0):
1755 gcpy                  self.dxfRt.close()
1756 gcpy              if (self.MISC_tool_num > 0):
1757 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.

---

```

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

```

---

### 3.7 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 3.3.1.1) 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.

**3.7.0.1 Building blocks**     The outlines of shapes will be defined using:

- lines — `dxflines`
- arcs — `dxfarcs`

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

**3.7.0.2 List of shapes**     In the TUG presentation/paper: <http://tug.org/TUGboat/tb40-2/tb125adams-3d.pdf> a list of 2D shapes was put forward — which of these will need to be created, or if some more general solution will be put forward is uncertain. For the time being, shapes will be implemented on an as-needed basis, as modified by the interaction with the requirements of toolpaths. Shapes for which code exists (or is trivially coded) are indicated by **Forest Green** — for those which have sub-classes, if all are feasible only the higher level is so called out.

- 0
  - **circle** — `dxfcircle`
  - ellipse (oval) (requires some sort of non-arc curve)
    - \* egg-shaped
  - **annulus** (one circle within another, forming a ring) — handled by nested circles
  - superellipse (see astroid below)
- 1
  - **cone with rounded end (arc)**—see also “sector” under 3 below
- 2
  - **semicircle/circular/half-circle segment** (arc and a straight line); see also sector below
  - arch—curve possibly smoothly joining a pair of straight lines with a flat bottom
  - lens/vesica piscis (two convex curves)
  - lune/crescent (one convex, one concave curve)
  - heart (two curves)
  - tomoe (comma shape)—non-arc curves
- 3
  - **triangle**
    - \* equilateral
    - \* isosceles
    - \* right triangle

- \* scalene
  - (circular) sector (two straight edges, one convex arc)
    - \* quadrant (90°)
    - \* sextants (60°)
    - \* octants (45°)
  - deltoid curve (three concave arcs)
  - Reuleaux triangle (three convex arcs)
  - arbelos (one convex, two concave arcs)
  - two straight edges, one concave arc—an example is the hyperbolic sector<sup>1</sup>
  - two convex, one concave arc
- 4
  - rectangle (including square) — `dxfrectangle`, `dxfrectangleround`
  - parallelogram
  - rhombus
  - trapezoid/trapezium
  - kite
  - ring/annulus segment (straight line, concave arc, straight line, convex arc)
  - astroid (four concave arcs)
  - salinon (four semicircles)
  - three straight lines and one concave arc

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

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

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

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

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

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

Some further considerations:

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

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



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

```
1759 gcpy      def dxfcircle(self, tool_num, xcenter, ycenter, radius):
1760 gcpy          self.dxfarc(tool_num, xcenter, ycenter, radius, 0, 90)
1761 gcpy          self.dxfarc(tool_num, xcenter, ycenter, radius, 90, 180)
1762 gcpy          self.dxfarc(tool_num, xcenter, ycenter, radius, 180, 270)
1763 gcpy          self.dxfarc(tool_num, xcenter, ycenter, radius, 270, 360)
```

Actually cutting the circle is much the same, with the added consideration of entry point if Z height is not above the surface of the stock/already removed material, directionality (counter-clockwise vs. clockwise), and depth (beginning and end depths must be specified which should allow usage of this for thread-cutting and similar purposes).  
Center is specified, but the actual entry point is the right-most edge.

```
1765 gcpy      def cutcircleCC(self, xcenter, ycenter, bz, ez, radius):
1766 gcpy          self.setzpos(bz)
1767 gcpy          self.cutquarterCCNE(xcenter, ycenter + radius, self.zpos()
+ ez/4, radius)
1768 gcpy          self.cutquarterCCNW(xcenter - radius, ycenter, self.zpos()
+ ez/4, radius)
1769 gcpy          self.cutquarterCCSW(xcenter, ycenter - radius, self.zpos()
+ ez/4, radius)
1770 gcpy          self.cutquarterCCSE(xcenter + radius, ycenter, self.zpos()
+ ez/4, radius)
1771 gcpy
1772 gcpy      def cutcircleCCdxf(self, xcenter, ycenter, bz, ez, radius):
1773 gcpy          self.cutcircleCC(self, xcenter, ycenter, bz, ez, radius)
1774 gcpy          self.dxfcircle(self, tool_num, xcenter, ycenter, radius)
```

A Drill toolpath is a simple plunge operation which will have a matching circle to define it.

3.7.0.2.2 rectangles There are two obvious forms for rectangles, square cornered and rounded:

```
1814 gcpy      def dxfrectangle(self, tool_num, xorigin, yorigin, xwidth,
yheight, corners = "Square", radius = 6):
1815 gcpy          if corners == "Square":
1816 gcpy              self.dxfline(tool_num, xorigin, yorigin, xorigin +
xwidth, yorigin)
1817 gcpy              self.dxfline(tool_num, xorigin + xwidth, yorigin,
xorigin + xwidth, yorigin + yheight)
1818 gcpy              self.dxfline(tool_num, xorigin + xwidth, yorigin +
yheight, xorigin, yorigin + yheight)
1819 gcpy              self.dxfline(tool_num, xorigin, yorigin + yheight,
xorigin, yorigin)
1820 gcpy          elif corners == "Fillet":
1821 gcpy              self.dxfrectangleround(tool_num, xorigin, yorigin,
xwidth, yheight, radius)
1822 gcpy          elif corners == "Chamfer":
1823 gcpy              self.dxfrectanglechamfer(tool_num, xorigin, yorigin,
xwidth, yheight, radius)
1824 gcpy          elif corners == "Flipped_Fillet":
1825 gcpy              self.dxfrectangleflippedfillet(tool_num, xorigin,
yorigin, xwidth, yheight, radius)
```

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

```
1827 gcpy      def dxfrectangleround(self, tool_num, xorigin, yorigin, xwidth,
yheight, radius):
1828 gcpy      # begin section
1829 gcpy          self.writedxf(tool_num, "0")
1830 gcpy          self.writedxf(tool_num, "SECTION")
1831 gcpy          self.writedxf(tool_num, "2")
1832 gcpy          self.writedxf(tool_num, "ENTITIES")
1833 gcpy          self.writedxf(tool_num, "0")
1834 gcpy          self.writedxf(tool_num, "LWPOLYLINE")
1835 gcpy          self.writedxf(tool_num, "5")
1836 gcpy          self.writedxf(tool_num, "4E")
1837 gcpy          self.writedxf(tool_num, "100")
1838 gcpy          self.writedxf(tool_num, "AcDbEntity")
1839 gcpy          self.writedxf(tool_num, "8")
1840 gcpy          self.writedxf(tool_num, "0")
1841 gcpy          self.writedxf(tool_num, "6")
1842 gcpy          self.writedxf(tool_num, "ByLayer")
1843 gcpy      #
```

```

1844 gcpy          self.writedxfcolor(tool_num)
1845 gcpy #
1846 gcpy          self.writedxf(tool_num, "370")
1847 gcpy          self.writedxf(tool_num, "-1")
1848 gcpy          self.writedxf(tool_num, "100")
1849 gcpy          self.writedxf(tool_num, "AcDbPolyline")
1850 gcpy          self.writedxf(tool_num, "90")
1851 gcpy          self.writedxf(tool_num, "8")
1852 gcpy          self.writedxf(tool_num, "70")
1853 gcpy          self.writedxf(tool_num, "1")
1854 gcpy          self.writedxf(tool_num, "43")
1855 gcpy          self.writedxf(tool_num, "0")
1856 gcpy #1 upper right corner before arc (counter-clockwise)
1857 gcpy          self.writedxf(tool_num, "10")
1858 gcpy          self.writedxf(tool_num, str(xorigin + xwidth))
1859 gcpy          self.writedxf(tool_num, "20")
1860 gcpy          self.writedxf(tool_num, str(yorigin + yheight - radius))
1861 gcpy          self.writedxf(tool_num, "42")
1862 gcpy          self.writedxf(tool_num, "0.414213562373095")
1863 gcpy #2 upper right corner after arc
1864 gcpy          self.writedxf(tool_num, "10")
1865 gcpy          self.writedxf(tool_num, str(xorigin + xwidth - radius))
1866 gcpy          self.writedxf(tool_num, "20")
1867 gcpy          self.writedxf(tool_num, str(yorigin + yheight))
1868 gcpy #3 upper left corner before arc (counter-clockwise)
1869 gcpy          self.writedxf(tool_num, "10")
1870 gcpy          self.writedxf(tool_num, str(xorigin + radius))
1871 gcpy          self.writedxf(tool_num, "20")
1872 gcpy          self.writedxf(tool_num, str(yorigin + yheight))
1873 gcpy          self.writedxf(tool_num, "42")
1874 gcpy          self.writedxf(tool_num, "0.414213562373095")
1875 gcpy #4 upper left corner after arc
1876 gcpy          self.writedxf(tool_num, "10")
1877 gcpy          self.writedxf(tool_num, str(xorigin))
1878 gcpy          self.writedxf(tool_num, "20")
1879 gcpy          self.writedxf(tool_num, str(yorigin + yheight - radius))
1880 gcpy #5 lower left corner before arc (counter-clockwise)
1881 gcpy          self.writedxf(tool_num, "10")
1882 gcpy          self.writedxf(tool_num, str(xorigin))
1883 gcpy          self.writedxf(tool_num, "20")
1884 gcpy          self.writedxf(tool_num, str(yorigin + radius))
1885 gcpy          self.writedxf(tool_num, "42")
1886 gcpy          self.writedxf(tool_num, "0.414213562373095")
1887 gcpy #6 lower left corner after arc
1888 gcpy          self.writedxf(tool_num, "10")
1889 gcpy          self.writedxf(tool_num, str(xorigin + radius))
1890 gcpy          self.writedxf(tool_num, "20")
1891 gcpy          self.writedxf(tool_num, str(yorigin))
1892 gcpy #7 lower right corner before arc (counter-clockwise)
1893 gcpy          self.writedxf(tool_num, "10")
1894 gcpy          self.writedxf(tool_num, str(xorigin + xwidth - radius))
1895 gcpy          self.writedxf(tool_num, "20")
1896 gcpy          self.writedxf(tool_num, str(yorigin))
1897 gcpy          self.writedxf(tool_num, "42")
1898 gcpy          self.writedxf(tool_num, "0.414213562373095")
1899 gcpy #8 lower right corner after arc
1900 gcpy          self.writedxf(tool_num, "10")
1901 gcpy          self.writedxf(tool_num, str(xorigin + xwidth))
1902 gcpy          self.writedxf(tool_num, "20")
1903 gcpy          self.writedxf(tool_num, str(yorigin + radius))
1904 gcpy # end current section
1905 gcpy          self.writedxf(tool_num, "0")
1906 gcpy          self.writedxf(tool_num, "SEQEND")

```

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

```

1919 gcpy      def dxfrectanglechamfer(self, tool_num, xorigin, yorigin,
1920 gcpy          xwidth, yheight, radius):
1921 gcpy          self.dxflines(tool_num, xorigin + radius, yorigin, xorigin,
1922 gcpy          yorigin + radius)
1923 gcpy          self.dxflines(tool_num, xorigin, yorigin + yheight - radius,
1924 gcpy          xorigin + radius, yorigin + yheight)
1925 gcpy          self.dxflines(tool_num, xorigin + xwidth - radius, yorigin +
1926 gcpy          yheight, xorigin + xwidth, yorigin + yheight - radius)
1927 gcpy          self.dxflines(tool_num, xorigin + xwidth - radius, yorigin,
1928 gcpy          xorigin + xwidth, yorigin + radius)

```

```
1925 gcpy      self.dxfline(tool_num, xorigin + radius, yorigin, xorigin +
                xwidth - radius, yorigin)
1926 gcpy      self.dxfline(tool_num, xorigin + xwidth, yorigin + radius,
                xorigin + xwidth, yorigin + yheight - radius)
1927 gcpy      self.dxfline(tool_num, xorigin + xwidth - radius, yorigin +
                yheight, xorigin + radius, yorigin + yheight)
1928 gcpy      self.dxfline(tool_num, xorigin, yorigin + yheight - radius,
                xorigin, yorigin + radius)
```

Flipped Fillet:

```
1930 gcpy      def dxfrectangleflippedfillet(self, tool_num, xorigin, yorigin,
                xwidth, yheight, radius):
1931 gcpy      self.dxfarc(tool_num, xorigin, yorigin, radius, 0, 90)
1932 gcpy      self.dxfarc(tool_num, xorigin + xwidth, yorigin, radius,
                90, 180)
1933 gcpy      self.dxfarc(tool_num, xorigin + xwidth, yorigin + yheight,
                radius, 180, 270)
1934 gcpy      self.dxfarc(tool_num, xorigin, yorigin + yheight, radius,
                270, 360)
1935 gcpy
1936 gcpy      self.dxfline(tool_num, xorigin + radius, yorigin, xorigin +
                xwidth - radius, yorigin)
1937 gcpy      self.dxfline(tool_num, xorigin + xwidth, yorigin + radius,
                xorigin + xwidth, yorigin + yheight - radius)
1938 gcpy      self.dxfline(tool_num, xorigin + xwidth - radius, yorigin +
                yheight, xorigin + radius, yorigin + yheight)
1939 gcpy      self.dxfline(tool_num, xorigin, yorigin + yheight - radius,
                xorigin, yorigin + radius)
```

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

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

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

```
1941 gcpy      def cutrectangle(self, tool_num, bx, by, bz, xwidth, yheight,
                zdepth):
1942 gcpy      self.cutline(bx, by, bz)
1943 gcpy      self.cutline(bx, by, bz - zdepth)
1944 gcpy      self.cutline(bx + xwidth, by, bz - zdepth)
1945 gcpy      self.cutline(bx + xwidth, by + yheight, bz - zdepth)
1946 gcpy      self.cutline(bx, by + yheight, bz - zdepth)
1947 gcpy      self.cutline(bx, by, bz - zdepth)
1948 gcpy
1949 gcpy      def cutrectangledxf(self, tool_num, bx, by, bz, xwidth, yheight
                , zdepth):
1950 gcpy      self.cutrectangle(tool_num, bx, by, bz, xwidth, yheight,
                zdepth)
1951 gcpy      self.dxfrectangle(tool_num, bx, by, xwidth, yheight, "
                Square")
```

The rounded forms instantiate a radius:

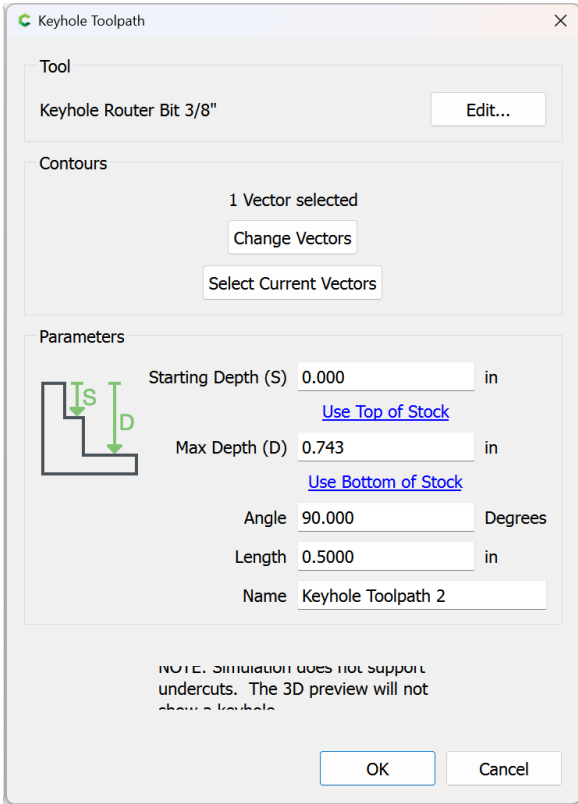
```
1956 gcpy      def cutrectangleround(self, tool_num, bx, by, bz, xwidth,
                yheight, zdepth, radius):
1957 gcpy #      self.rapid(bx + radius, by, bz)
1958 gcpy      self.cutline(bx + radius, by, bz + zdepth)
1959 gcpy      self.cutline(bx + xwidth - radius, by, bz + zdepth)
1960 gcpy      self.cutquarterCCSE(bx + xwidth, by + radius, bz + zdepth,
                radius)
1961 gcpy      self.cutline(bx + xwidth, by + yheight - radius, bz +
                zdepth)
1962 gcpy      self.cutquarterCCNE(bx + xwidth - radius, by + yheight, bz
                + zdepth, radius)
1963 gcpy      self.cutline(bx + radius, by + yheight, bz + zdepth)
1964 gcpy      self.cutquarterCCNW(bx, by + yheight - radius, bz + zdepth,
                radius)
1965 gcpy      self.cutline(bx, by + radius, bz + zdepth)
1966 gcpy      self.cutquarterCCSW(bx + radius, by, bz + zdepth, radius)
```

```
1967 gcpy
1968 gcpy      def cutrectanglerounddx(self, tool_num, bx, by, bz, xwidth,
1969 gcpy          yheight, zdepth, radius):
1970 gcpy          self.cutrectangleround(tool_num, bx, by, bz, xwidth,
          yheight, zdepth, radius)
          self.dxfrectangleround(tool_num, bx, by, xwidth, yheight,
          radius)
```

**3.7.0.2.3 Keyhole toolpath and undercut tooling** The first topologically unusual toolpath is cutkeyhole toolpath cutkeyhole toolpath — where other toolpaths have a direct correspondence between the associated geometry and the area cut, that Keyhole toolpaths may be used with tooling which undercuts and which will result in the creation of two different physical physical regions: the visible surface matching the union of the tool perimeter at the entry point and the linear movement of the shaft and the larger region of the tool perimeter at the depth which the tool is plunged to and moved along.

Tooling for such toolpaths is defined at paragraph 3.4.0.1

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:

```
1971 gcpy      def cutkeyholegcdxf(self, kh_tool_num, kh_start_depth,
1972 gcpy          kh_max_depth, kht_direction, kh_distance):
1973 gcpy          toolpath = self.cutKHgcdxf(kh_tool_num, kh_start_depth,
1974 gcpy              kh_max_depth, 90, kh_distance)
1975 gcpy          elif (kht_direction == "S"):
1976 gcpy              toolpath = self.cutKHgcdxf(kh_tool_num, kh_start_depth,
1977 gcpy                  kh_max_depth, 270, kh_distance)
1978 gcpy          elif (kht_direction == "E"):
1979 gcpy              toolpath = self.cutKHgcdxf(kh_tool_num, kh_start_depth,
1980 gcpy                  kh_max_depth, 0, kh_distance)
1981 gcpy          elif (kht_direction == "W"):
1982 gcpy              toolpath = self.cutKHgcdxf(kh_tool_num, kh_start_depth,
1983 gcpy                  kh_max_depth, 180, kh_distance)
1984 gcpy          if self.generatepaths == True:
1985 gcpy              self.toolpaths = union([self.toolpaths, toolpath])
1986 gcpy          return toolpath
1987 gcpy          else:
1988 gcpy              return cube([0.01, 0.01, 0.01])

155 gcpscad module cutkeyholegcdxf(kh_tool_num, kh_start_depth, kh_max_depth,
156 gcpscad     kht_direction, kh_distance){
157 gcpscad     gcp.cutkeyholegcdxf(kh_tool_num, kh_start_depth, kh_max_depth,
158 gcpscad         kht_direction, kh_distance);
159 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:

```
1986 gcpy      def cutKHgcdxf(self, kh_tool_num, kh_start_depth, kh_max_depth,
1987 gcpy          kh_angle, kh_distance):
1988 gcpy          oXpos = self.xpos()
1989 gcpy          oYpos = self.ypos()
1990 gcpy          self.dxfKH(kh_tool_num, self.xpos(), self.ypos(),
1991 gcpy              kh_start_depth, kh_max_depth, kh_angle, kh_distance)
1992 gcpy          toolpath = self.cutline(self.xpos(), self.ypos(), -
1993 gcpy              kh_max_depth)
1994 gcpy          self.setxpos(oXpos)
1995 gcpy          self.setypos(oYpos)
1996 gcpy          if self.generatepaths == False:
1997 gcpy              return toolpath
1998 gcpy          else:
1999 gcpy              return cube([0.001, 0.001, 0.001])

1998 gcpy      def dxfKH(self, kh_tool_num, oXpos, oYpos, kh_start_depth,
1999 gcpy          kh_max_depth, kh_angle, kh_distance):
2000 gcpy          # oXpos = self.xpos()
2001 gcpy          # oYpos = self.ypos()
2002 gcpy          #Circle at entry hole
2003 gcpy          self.dxfarc(kh_tool_num, oXpos, oYpos, self.tool_radius(
2004 gcpy              kh_tool_num, 7), 0, 90)
2005 gcpy          self.dxfarc(kh_tool_num, oXpos, oYpos, self.tool_radius(
2006 gcpy              kh_tool_num, 7), 90, 180)
2007 gcpy          self.dxfarc(kh_tool_num, oXpos, oYpos, self.tool_radius(
2008 gcpy              kh_tool_num, 7), 180, 270)
2009 gcpy          self.dxfarc(kh_tool_num, oXpos, oYpos, self.tool_radius(
2010 gcpy              kh_tool_num, 7), 270, 360)
```

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

```
2006 gcpy #pre-calculate needed values
2007 gcpy     r = self.tool_radius(kh_tool_num, 7)
2008 gcpy     # print(r)
2009 gcpy     rt = self.tool_radius(kh_tool_num, 1)
2010 gcpy     # print(rt)
2011 gcpy     ro = math.sqrt((self.tool_radius(kh_tool_num, 1))**2-(self.
2012 gcpy         tool_radius(kh_tool_num, 7))**2)
2013 gcpy     # print(ro)
2014 gcpy     angle = math.degrees(math.acos(ro/rt))
2015 gcpy     #Outlines of entry hole and slot
```

```

2015 gcpy          if (kh_angle == 0):
2016 gcpy #Lower left of entry hole
2017 gcpy          self.dxfarc(kh_tool_num, self.xpos(), self.ypos(), self
                           .tool_radius(kh_tool_num, 1), 180, 270)
2018 gcpy #Upper left of entry hole
2019 gcpy          self.dxfarc(kh_tool_num, self.xpos(), self.ypos(), self
                           .tool_radius(kh_tool_num, 1), 90, 180)
2020 gcpy #Upper right of entry hole
2021 gcpy #          self.dxfarc(kh_tool_num, self.xpos(), self.ypos(), rt,
                           41.810, 90)
2022 gcpy          self.dxfarc(kh_tool_num, self.xpos(), self.ypos(), rt,
                           angle, 90)
2023 gcpy #Lower right of entry hole
2024 gcpy          self.dxfarc(kh_tool_num, self.xpos(), self.ypos(), rt,
                           270, 360-angle)
2025 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))))
2026 gcpy #Actual line of cut
2027 gcpy #          self.dxfline(kh_tool_num, self.xpos(), self.ypos(),
                           self.xpos()+kh_distance, self.ypos())
2028 gcpy #upper right of end of slot (kh_max_depth+4.36))/2
2029 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)
2030 gcpy #lower right of end of slot
2031 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)
2032 gcpy #upper right slot
2033 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))
2034 gcpy #          self.dxfline(kh_tool_num, self.xpos()+(math.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)
2035 gcpy #end position at top of slot
2036 gcpy #lower right slot
2037 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))
2038 gcpy #          dxline(kh_tool_num, self.xpos()+(math.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)
2039 gcpy #end position at top of slot
2040 gcpy #          hull(){
2041 gcpy #              translate([xpos(), ypos(), zpos()]){
2042 gcpy #                  keyhole_shaft(6.35, 9.525);
2043 gcpy #              }
2044 gcpy #              translate([xpos(), ypos(), zpos()-kh_max_depth]){
2045 gcpy #                  keyhole_shaft(6.35, 9.525);
2046 gcpy #              }
2047 gcpy #          }
2048 gcpy #          hull(){
2049 gcpy #              translate([xpos(), ypos(), zpos()-kh_max_depth]){
2050 gcpy #                  keyhole_shaft(6.35, 9.525);
2051 gcpy #              }
2052 gcpy #              translate([xpos()+kh_distance, ypos(), zpos()-kh_max_depth])
{
2053 gcpy #                  keyhole_shaft(6.35, 9.525);
2054 gcpy #              }
2055 gcpy #          }
2056 gcpy #          cutwithfeed(getxpos(), getypos(), -kh_max_depth, feed);
2057 gcpy #          cutwithfeed(getxpos()+kh_distance, getypos(), -kh_max_depth,
feed);
2058 gcpy #          setxpos(getxpos()-kh_distance);
2059 gcpy #      } else if (kh_angle > 0 && kh_angle < 90) {

```

```

2060 gcpy #//echo(kh_angle);
2061 gcpy #   dxfarc(getxpos(), getypos(), tool_diameter(KH_tool_num, (
        kh_max_depth))/2, 90+kh_angle, 180+kh_angle, KH_tool_num);
2062 gcpy #   dxfarc(getxpos(), getypos(), tool_diameter(KH_tool_num, (
        kh_max_depth))/2, 180+kh_angle, 270+kh_angle, KH_tool_num);
2063 gcpy #dxfarc(getxpos(), getypos(), tool_diameter(KH_tool_num, (
        kh_max_depth))/2, kh_angle+asin((tool_diameter(KH_tool_num, (
        kh_max_depth+4.36))/2)/(tool_diameter(KH_tool_num, (kh_max_depth
        ))/2)), 90+kh_angle, KH_tool_num);
2064 gcpy #dxfarc(getxpos(), getypos(), tool_diameter(KH_tool_num, (
        kh_max_depth))/2, 270+kh_angle, 360+kh_angle-asin((tool_diameter
        (KH_tool_num, (kh_max_depth+4.36))/2)/(tool_diameter(KH_tool_num
        , (kh_max_depth))/2)), KH_tool_num);
2065 gcpy #dxfarc(getxpos()+(kh_distance*cos(kh_angle)),
2066 gcpy #   getypos()+(kh_distance*sin(kh_angle)), tool_diameter(KH_tool_num
        , (kh_max_depth+4.36))/2, 0+kh_angle, 90+kh_angle, KH_tool_num);
2067 gcpy #dxfarc(getxpos()+(kh_distance*cos(kh_angle)), getypos()+(
        kh_distance*sin(kh_angle)), tool_diameter(KH_tool_num, (
        kh_max_depth+4.36))/2, 270+kh_angle, 360+kh_angle, KH_tool_num);
2068 gcpy #dxfline( getxpos()+tool_diameter(KH_tool_num, (kh_max_depth))/2*
        cos(kh_angle+asin((tool_diameter(KH_tool_num, (kh_max_depth
        +4.36))/2)/(tool_diameter(KH_tool_num, (kh_max_depth))/2))),
2069 gcpy #   getypos()+tool_diameter(KH_tool_num, (kh_max_depth))/2*sin(
        kh_angle+asin((tool_diameter(KH_tool_num, (kh_max_depth+4.36))
        /2)/(tool_diameter(KH_tool_num, (kh_max_depth))/2))),
2070 gcpy #   getxpos()+(kh_distance*cos(kh_angle))-((tool_diameter(KH_tool_num
        , (kh_max_depth+4.36))/2)*sin(kh_angle)),
2071 gcpy #   getypos()+(kh_distance*sin(kh_angle))+((tool_diameter(KH_tool_num
        , (kh_max_depth+4.36))/2)*cos(kh_angle)), KH_tool_num);
2072 gcpy #//echo("a", tool_diameter(KH_tool_num, (kh_max_depth+4.36))/2);
2073 gcpy #//echo("c", tool_diameter(KH_tool_num, (kh_max_depth))/2);
2074 gcpy #echo("Aangle", asin((tool_diameter(KH_tool_num, (kh_max_depth
        +4.36))/2)/(tool_diameter(KH_tool_num, (kh_max_depth))/2)));
2075 gcpy #//echo(kh_angle);
2076 gcpy #   cutwithfeed(getxpos()+(kh_distance*cos(kh_angle)), getypos()+(
        kh_distance*sin(kh_angle)), -kh_max_depth, feed);
2077 gcpy #       toolpath = toolpath.union(self.cutline(self.xpos()+
        kh_distance, self.ypos(), -kh_max_depth))
2078 gcpy         elif (kh_angle == 90):
2079 gcpy #Lower left of entry hole
2080 gcpy         self.dxfarc(kh_tool_num, oXpos, oYpos, self.tool_radius
        (kh_tool_num, 1), 180, 270)
2081 gcpy #Lower right of entry hole
2082 gcpy         self.dxfarc(kh_tool_num, oXpos, oYpos, self.tool_radius
        (kh_tool_num, 1), 270, 360)
2083 gcpy #left slot
2084 gcpy         self.dxfline(kh_tool_num, oXpos-r, oYpos+ro, oXpos-r,
        oYpos+kh_distance)
2085 gcpy #right slot
2086 gcpy         self.dxfline(kh_tool_num, oXpos+r, oYpos+ro, oXpos+r,
        oYpos+kh_distance)
2087 gcpy #upper left of end of slot
2088 gcpy         self.dxfarc(kh_tool_num, oXpos, oYpos+kh_distance, r,
        90, 180)
2089 gcpy #upper right of end of slot
2090 gcpy         self.dxfarc(kh_tool_num, oXpos, oYpos+kh_distance, r,
        0, 90)
2091 gcpy #Upper right of entry hole
2092 gcpy         self.dxfarc(kh_tool_num, oXpos, oYpos, rt, 0, 90-angle)
2093 gcpy #Upper left of entry hole
2094 gcpy         self.dxfarc(kh_tool_num, oXpos, oYpos, rt, 90+angle,
        180)
2095 gcpy #       toolpath = toolpath.union(self.cutline(oXpos, oYpos+
        kh_distance, -kh_max_depth))
2096 gcpy         elif (kh_angle == 180):
2097 gcpy #Lower right of entry hole
2098 gcpy         self.dxfarc(kh_tool_num, oXpos, oYpos, self.tool_radius
        (kh_tool_num, 1), 270, 360)
2099 gcpy #Upper right of entry hole
2100 gcpy         self.dxfarc(kh_tool_num, oXpos, oYpos, self.tool_radius
        (kh_tool_num, 1), 0, 90)
2101 gcpy #Upper left of entry hole
2102 gcpy         self.dxfarc(kh_tool_num, oXpos, oYpos, rt, 90, 180-
        angle)
2103 gcpy #Lower left of entry hole
2104 gcpy         self.dxfarc(kh_tool_num, oXpos, oYpos, rt, 180+angle,
        270)
2105 gcpy #upper slot

```

```

2106 gcpy                self.dxfline(kh_tool_num, oXpos-ro, oYpos-r, oXpos-
                           kh_distance, oYpos-r)
2107 gcpy #lower slot
2108 gcpy                self.dxfline(kh_tool_num, oXpos-ro, oYpos+r, oXpos-
                           kh_distance, oYpos+r)
2109 gcpy #upper left of end of slot
2110 gcpy                self.dxfarc(kh_tool_num, oXpos-kh_distance, oYpos, r,
                                   90, 180)
2111 gcpy #lower left of end of slot
2112 gcpy                self.dxfarc(kh_tool_num, oXpos-kh_distance, oYpos, r,
                                   180, 270)
2113 gcpy #                toolpath = toolpath.union(self.cutline(oXpos-
                           kh_distance, oYpos, -kh_max_depth))
2114 gcpy                elif (kh_angle == 270):
2115 gcpy #Upper left of entry hole
2116 gcpy                self.dxfarc(kh_tool_num, oXpos, oYpos, self.tool_radius
                                   (kh_tool_num, 1), 90, 180)
2117 gcpy #Upper right of entry hole
2118 gcpy                self.dxfarc(kh_tool_num, oXpos, oYpos, self.tool_radius
                                   (kh_tool_num, 1), 0, 90)
2119 gcpy #left slot
2120 gcpy                self.dxfline(kh_tool_num, oXpos-r, oYpos-ro, oXpos-r,
                                   oYpos-kh_distance)
2121 gcpy #right slot
2122 gcpy                self.dxfline(kh_tool_num, oXpos+r, oYpos-ro, oXpos+r,
                                   oYpos-kh_distance)
2123 gcpy #lower left of end of slot
2124 gcpy                self.dxfarc(kh_tool_num, oXpos, oYpos-kh_distance, r,
                                   180, 270)
2125 gcpy #lower right of end of slot
2126 gcpy                self.dxfarc(kh_tool_num, oXpos, oYpos-kh_distance, r,
                                   270, 360)
2127 gcpy #lower right of entry hole
2128 gcpy                self.dxfarc(kh_tool_num, oXpos, oYpos, rt, 180, 270-
                                   angle)
2129 gcpy #lower left of entry hole
2130 gcpy                self.dxfarc(kh_tool_num, oXpos, oYpos, rt, 270+angle,
                                   360)
2131 gcpy #                toolpath = toolpath.union(self.cutline(oXpos, oYpos-
                           kh_distance, -kh_max_depth))
2132 gcpy #                print(self.zpos())
2133 gcpy #                self.setxpos(oXpos)
2134 gcpy #                self.setypos(oYpos)
2135 gcpy #                if self.generatepaths == False:
2136 gcpy #                    return toolpath
2137 gcpy
2138 gcpy # } else if (kh_angle == 90) {
2139 gcpy # //Lower left of entry hole
2140 gcpy # dxfarc(getxpos(), getypos(), 9.525/2, 180, 270, KH_tool_num);
2141 gcpy # //Lower right of entry hole
2142 gcpy # dxfarc(getxpos(), getypos(), 9.525/2, 270, 360, KH_tool_num);
2143 gcpy # //Upper right of entry hole
2144 gcpy # dxfarc(getxpos(), getypos(), 9.525/2, 0, acos(tool_diameter(
KH_tool_num, 5)/tool_diameter(KH_tool_num, 1)), KH_tool_num);
2145 gcpy # //Upper left of entry hole
2146 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
);
2147 gcpy # //Actual line of cut
2148 gcpy # dxfline(getxpos(), getypos(), getxpos(), getypos()+kh_distance
);
2149 gcpy # //upper right of slot
2150 gcpy # dxfarc(getxpos(), getypos()+kh_distance, tool_diameter(
KH_tool_num, (kh_max_depth+4.36))/2, 0, 90, KH_tool_num);
2151 gcpy # //upper left of slot
2152 gcpy # dxfarc(getxpos(), getypos()+kh_distance, tool_diameter(
KH_tool_num, (kh_max_depth+6.35))/2, 90, 180, KH_tool_num);
2153 gcpy # //right of slot
2154 gcpy # dxfline(
2155 gcpy #     getxpos()+tool_diameter(KH_tool_num, (kh_max_depth))/2,
2156 gcpy #     getypos()+(math.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,
2157 gcpy #     getxpos()+tool_diameter(KH_tool_num, (kh_max_depth))/2,
2158 gcpy # //end position at top of slot
2159 gcpy #     getypos()+kh_distance,
2160 gcpy #     KH_tool_num);
2161 gcpy # dxfline(getxpos()-tool_diameter(KH_tool_num, (kh_max_depth))

```



```

/2, getypos()+(math.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);
2162 gcpy #   hull(){
2163 gcpy #       translate([xpos(), ypos(), zpos()]){
2164 gcpy #           keyhole_shaft(6.35, 9.525);
2165 gcpy #       }
2166 gcpy #       translate([xpos(), ypos(), zpos()-kh_max_depth]){
2167 gcpy #           keyhole_shaft(6.35, 9.525);
2168 gcpy #       }
2169 gcpy #   }
2170 gcpy #   hull(){
2171 gcpy #       translate([xpos(), ypos(), zpos()-kh_max_depth]){
2172 gcpy #           keyhole_shaft(6.35, 9.525);
2173 gcpy #       }
2174 gcpy #       translate([xpos(), ypos()+kh_distance, zpos()-kh_max_depth])
{
2175 gcpy #           keyhole_shaft(6.35, 9.525);
2176 gcpy #       }
2177 gcpy #   }
2178 gcpy #   cutwithfeed(getxpos(), getypos(), -kh_max_depth, feed);
2179 gcpy #   cutwithfeed(getxpos(), getypos()+kh_distance, -kh_max_depth,
feed);
2180 gcpy #   setypos(getypos()-kh_distance);
2181 gcpy # } else if (kh_angle == 180) {
2182 gcpy #     //Lower right of entry hole
2183 gcpy #     dxfarc(getxpos(), getypos(), 9.525/2, 270, 360, KH_tool_num);
2184 gcpy #     //Upper right of entry hole
2185 gcpy #     dxfarc(getxpos(), getypos(), 9.525/2, 0, 90, KH_tool_num);
2186 gcpy #     //Upper left of entry hole
2187 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);
2188 gcpy #     //Lower left of entry hole
2189 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
);
2190 gcpy #     //upper left of slot
2191 gcpy #     dxfarc(getxpos()-kh_distance, getypos(), tool_diameter(
KH_tool_num, (kh_max_depth+6.35))/2, 90, 180, KH_tool_num);
2192 gcpy #     //lower left of slot
2193 gcpy #     dxfarc(getxpos()-kh_distance, getypos(), tool_diameter(
KH_tool_num, (kh_max_depth+6.35))/2, 180, 270, KH_tool_num);
2194 gcpy #     //Actual line of cut
2195 gcpy #     dxfline(getxpos(), getypos(), getxpos()-kh_distance, getypos()
);
2196 gcpy #     //upper left slot
2197 gcpy #     dxfline(
2198 gcpy #         getxpos()-(math.sqrt((tool_diameter(KH_tool_num, 1)^2)-(
tool_diameter(KH_tool_num, 5)^2))/2),
2199 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,
2200 gcpy #         getxpos()-kh_distance,
2201 gcpy #         //end position at top of slot
2202 gcpy #         getypos()+tool_diameter(KH_tool_num, (kh_max_depth))/2,
2203 gcpy #         KH_tool_num);
2204 gcpy #     //lower right slot
2205 gcpy #     dxfline(
2206 gcpy #         getxpos()-(math.sqrt((tool_diameter(KH_tool_num, 1)^2)-(
tool_diameter(KH_tool_num, 5)^2))/2),
2207 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,
2208 gcpy #         getxpos()-kh_distance,
2209 gcpy #         //end position at top of slot
2210 gcpy #         getypos()-tool_diameter(KH_tool_num, (kh_max_depth))/2,
2211 gcpy #         KH_tool_num);
2212 gcpy #     hull(){
2213 gcpy #         translate([xpos(), ypos(), zpos()]){
2214 gcpy #             keyhole_shaft(6.35, 9.525);
2215 gcpy #         }
2216 gcpy #         translate([xpos(), ypos(), zpos()-kh_max_depth]){
2217 gcpy #             keyhole_shaft(6.35, 9.525);
2218 gcpy #         }
2219 gcpy #     }
2220 gcpy #     hull(){

```

```

2221 gcpy #         translate([xpos(), ypos(), zpos()-kh_max_depth]){
2222 gcpy #             keyhole_shaft(6.35, 9.525);
2223 gcpy #         }
2224 gcpy #         translate([xpos()-kh_distance, ypos(), zpos()-kh_max_depth])
2225 gcpy #     {
2226 gcpy #         keyhole_shaft(6.35, 9.525);
2227 gcpy #     }
2228 gcpy #     cutwithfeed(getxpos(), getypos(), -kh_max_depth, feed);
2229 gcpy #     cutwithfeed(getxpos()-kh_distance, getypos(), -kh_max_depth,
2230 gcpy #         feed);
2231 gcpy #     setxpos(getxpos()+kh_distance);
2232 gcpy # } else if (kh_angle == 270) {
2233 gcpy #     //Upper right of entry hole
2234 gcpy #     dxfarc(getxpos(), getypos(), 9.525/2, 0, 90, KH_tool_num);
2235 gcpy #     //Upper left of entry hole
2236 gcpy #     dxfarc(getxpos(), getypos(), 9.525/2, 90, 180, KH_tool_num);
2237 gcpy #     //lower right of slot
2238 gcpy #     dxfarc(getxpos(), getypos()-kh_distance, tool_diameter(
2239 gcpy #         KH_tool_num, (kh_max_depth+4.36))/2, 270, 360, KH_tool_num);
2240 gcpy #     //lower left of slot
2241 gcpy #     dxfarc(getxpos(), getypos()-kh_distance, tool_diameter(
2242 gcpy #         KH_tool_num, (kh_max_depth+4.36))/2, 180, 270, KH_tool_num);
2243 gcpy #     //Actual line of cut
2244 gcpy #     dxfline(getxpos(), getypos(), getxpos(), getypos()-kh_distance
2245 gcpy #         );
2246 gcpy #     //right of slot
2247 gcpy #     dxfline(
2248 gcpy #         getxpos()+tool_diameter(KH_tool_num, (kh_max_depth))/2,
2249 gcpy #         getypos()-(math.sqrt((tool_diameter(KH_tool_num, 1)^2)-(
2250 gcpy #             tool_diameter(KH_tool_num, 5)^2))/2), //( (kh_max_depth-6.34))
2251 gcpy #             /2)^2-(tool_diameter(KH_tool_num, (kh_max_depth-6.34))/2)^2,
2252 gcpy #             getxpos()+tool_diameter(KH_tool_num, (kh_max_depth))/2,
2253 gcpy #             //end position at top of slot
2254 gcpy #             getypos()-kh_distance,
2255 gcpy #             KH_tool_num);
2256 gcpy #     //left of slot
2257 gcpy #     dxfline(
2258 gcpy #         getxpos()-tool_diameter(KH_tool_num, (kh_max_depth))/2,
2259 gcpy #         getypos()-(math.sqrt((tool_diameter(KH_tool_num, 1)^2)-(
2260 gcpy #             tool_diameter(KH_tool_num, 5)^2))/2), //( (kh_max_depth-6.34))
2261 gcpy #             /2)^2-(tool_diameter(KH_tool_num, (kh_max_depth-6.34))/2)^2,
2262 gcpy #             getxpos()-tool_diameter(KH_tool_num, (kh_max_depth))/2,
2263 gcpy #             //end position at top of slot
2264 gcpy #             getypos()-kh_distance,
2265 gcpy #             KH_tool_num);
2266 gcpy #     //Lower right of entry hole
2267 gcpy #     dxfarc(getxpos(), getypos(), 9.525/2, 360-acos(tool_diameter(
2268 gcpy #         KH_tool_num, 5)/tool_diameter(KH_tool_num, 1)), 360, KH_tool_num
2269 gcpy #         );
2270 gcpy #     //Lower left of entry hole
2271 gcpy #     dxfarc(getxpos(), getypos(), 9.525/2, 180, 180+acos(
2272 gcpy #         tool_diameter(KH_tool_num, 5)/tool_diameter(KH_tool_num, 1)),
2273 gcpy #         KH_tool_num);
2274 gcpy #     hull(){
2275 gcpy #         translate([xpos(), ypos(), zpos()]){
2276 gcpy #             keyhole_shaft(6.35, 9.525);
2277 gcpy #         }
2278 gcpy #         translate([xpos(), ypos(), zpos()-kh_max_depth]){
2279 gcpy #             keyhole_shaft(6.35, 9.525);
2280 gcpy #         }
2281 gcpy #     }
2282 gcpy #     hull(){
2283 gcpy #         translate([xpos(), ypos(), zpos()-kh_max_depth]){
2284 gcpy #             keyhole_shaft(6.35, 9.525);
2285 gcpy #         }
2286 gcpy #         translate([xpos()-kh_distance, ypos(), zpos()-kh_max_depth])
2287 gcpy #     {
2288 gcpy #         keyhole_shaft(6.35, 9.525);
2289 gcpy #     }
2290 gcpy #     cutwithfeed(getxpos(), getypos(), -kh_max_depth, feed);
2291 gcpy #     cutwithfeed(getxpos()-kh_distance, getypos(), -kh_max_depth,
2292 gcpy #         feed);
2293 gcpy #     setypos(getypos()+kh_distance);
2294 gcpy # }
2295 gcpy # }

```

---

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

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

Making such cuts will require dovetail tooling such as:

- 808079 <https://www.amanatool.com/45828-carbide-tipped-dovetail-8-deg-x-1-2-dia-x-825-x-1.html>
- 814 <https://www.leevalley.com/en-us/shop/tools/power-tool-accessories/router-bits/30172-dovetail-bits?item=18J1607>

Two commands are required:

```
2284 gcpy      def cut_pins(self, Joint_Width, stockZthickness,
2285 gcpy          Number_of_Dovetails, Spacing, Proportion, DTT_diameter,
                DTT_angle):
2286 gcpy          DT0 = Tan(math.radians(DTT_angle)) * (stockZthickness *
2287 gcpy              Proportion)
2288 gcpy          DTR = DTT_diameter/2 - DT0
2289 gcpy          cpr = self.rapidXY(0, stockZthickness + Spacing/2)
2290 gcpy          ctp = self.cutlinedxfgc(self.xpos(), self.ypos(), -
                stockZthickness * Proportion)
2291 gcpy          #      ctp = ctp.union(self.cutlinedxfgc(Joint_Width / (
                Number_of_Dovetails * 2), self.ypos(), -stockZthickness *
                Proportion))
2292 gcpy          i = 1
2293 gcpy          while i < Number_of_Dovetails * 2:
2294 gcpy              print(i)
2295 gcpy              ctp = ctp.union(self.cutlinedxfgc(i * (Joint_Width / (
                Number_of_Dovetails * 2)), self.ypos(), -
                stockZthickness * Proportion))
2296 gcpy              ctp = ctp.union(self.cutlinedxfgc(i * (Joint_Width / (
                Number_of_Dovetails * 2)), (stockZthickness +
                Spacing) + (stockZthickness * Proportion) - (
                DTT_diameter/2), -(stockZthickness * Proportion)))
2297 gcpy              ctp = ctp.union(self.cutlinedxfgc(i * (Joint_Width / (
                Number_of_Dovetails * 2)), stockZthickness + Spacing
                /2, -(stockZthickness * Proportion)))
2298 gcpy              ctp = ctp.union(self.cutlinedxfgc((i + 1) * (
                Joint_Width / (Number_of_Dovetails * 2)),
                stockZthickness + Spacing/2, -(stockZthickness *
                Proportion)))
2299 gcpy              self.dxfrectangleround(self.currenttoolnumber(),
2300 gcpy                  i * (Joint_Width / (Number_of_Dovetails * 2))-DTR,
2301 gcpy                  stockZthickness + (Spacing/2) - DTR,
2302 gcpy                  DTR * 2,
2303 gcpy                  (stockZthickness * Proportion) + Spacing/2 + DTR *
2304 gcpy                      2 - (DTT_diameter/2),
2305 gcpy                      DTR)
2306 gcpy              i += 2
2307 gcpy              self.rapidZ(0)
2308 gcpy              return ctp
```

and

```
2307 gcpy      def cut_tails(self, Joint_Width, stockZthickness,
2308 gcpy          Number_of_Dovetails, Spacing, Proportion, DTT_diameter,
                DTT_angle):
2309 gcpy          DT0 = Tan(math.radians(DTT_angle)) * (stockZthickness *
2310 gcpy              Proportion)
2311 gcpy          DTR = DTT_diameter/2 - DT0
2312 gcpy          cpr = self.rapidXY(0, 0)
2313 gcpy          ctp = self.cutlinedxfgc(self.xpos(), self.ypos(), -
                stockZthickness * Proportion)
2314 gcpy          ctp = ctp.union(self.cutlinedxfgc(
                Joint_Width / (Number_of_Dovetails * 2) - (DTT_diameter
                - DT0),
2315 gcpy              self.ypos(),
2316 gcpy              -stockZthickness * Proportion))
2317 gcpy          i = 1
2318 gcpy          while i < Number_of_Dovetails * 2:
2319 gcpy              ctp = ctp.union(self.cutlinedxfgc(
2320 gcpy                  i * (Joint_Width / (Number_of_Dovetails * 2)) - (
                DTT_diameter - DT0),
                stockZthickness * Proportion - DTT_diameter / 2,
```

```

2321 gcpy          -(stockZthickness * Proportion)))
2322 gcpy          ctp = ctp.union(self.cutarcCWdxf(180, 90,
2323 gcpy            i * (Joint_Width / (Number_of_Dovetails * 2)),
2324 gcpy            stockZthickness * Proportion - DTT_diameter / 2,
2325 gcpy #          self.ypos(),
2326 gcpy            DTT_diameter - DT0, 0, 1))
2327 gcpy          ctp = ctp.union(self.cutarcCWdxf(90, 0,
2328 gcpy            i * (Joint_Width / (Number_of_Dovetails * 2)),
2329 gcpy            stockZthickness * Proportion - DTT_diameter / 2,
2330 gcpy            DTT_diameter - DT0, 0, 1))
2331 gcpy          ctp = ctp.union(self.cutlinedxfgc(
2332 gcpy            i * (Joint_Width / (Number_of_Dovetails * 2)) + (
2333 gcpy              DTT_diameter - DT0),
2334 gcpy            0,
2335 gcpy            -(stockZthickness * Proportion)))
2336 gcpy          ctp = ctp.union(self.cutlinedxfgc(
2337 gcpy            (i + 2) * (Joint_Width / (Number_of_Dovetails * 2))
2338 gcpy            - (DTT_diameter - DT0),
2339 gcpy            0,
2340 gcpy            -(stockZthickness * Proportion)))
2341 gcpy          i += 2
2342 gcpy          self.rapidZ(0)
2343 gcpy          self.rapidXY(0, 0)
2344 gcpy          ctp = ctp.union(self.cutlinedxfgc(self.xpos(), self.ypos(),
2345 gcpy            -stockZthickness * Proportion))
2346 gcpy          self.dxfarc(self.currenttoolnumber(), 0, 0, DTR, 180, 270)
2347 gcpy          self.dxfline(self.currenttoolnumber(), -DTR, 0, -DTR,
2348 gcpy            stockZthickness + DTR)
2349 gcpy          self.dxfarc(self.currenttoolnumber(), 0, stockZthickness +
2350 gcpy            DTR, DTR, 90, 180)
2351 gcpy          self.dxfline(self.currenttoolnumber(), 0, stockZthickness +
2352 gcpy            DTR * 2, Joint_Width, stockZthickness + DTR * 2)
2353 gcpy          i = 0
2354 gcpy          while i < Number_of_Dovetails * 2:
2355 gcpy            ctp = ctp.union(self.cutline(i * (Joint_Width / (
2356 gcpy              Number_of_Dovetails * 2)), stockZthickness + DT0, -(
2357 gcpy                stockZthickness * Proportion)))
2358 gcpy            ctp = ctp.union(self.cutline((i+2) * (Joint_Width / (
2359 gcpy              Number_of_Dovetails * 2)), stockZthickness + DT0, -(
2360 gcpy                stockZthickness * Proportion)))
2361 gcpy            ctp = ctp.union(self.cutline((i+2) * (Joint_Width / (
2362 gcpy              Number_of_Dovetails * 2)), 0, -(stockZthickness *
2363 gcpy                Proportion)))
2364 gcpy            self.dxfarc(self.currenttoolnumber(), i * (Joint_Width
2365 gcpy              / (Number_of_Dovetails * 2)), 0, DTR, 270, 360)
2366 gcpy            self.dxfline(self.currenttoolnumber(),
2367 gcpy              i * (Joint_Width / (Number_of_Dovetails * 2)) + DTR
2368 gcpy              ,
2369 gcpy              0,
2370 gcpy              i * (Joint_Width / (Number_of_Dovetails * 2)) + DTR
2371 gcpy              , stockZthickness * Proportion - DTT_diameter /
2372 gcpy              2)
2373 gcpy            self.dxfarc(self.currenttoolnumber(), (i + 1) * (
2374 gcpy              Joint_Width / (Number_of_Dovetails * 2)),
2375 gcpy              stockZthickness * Proportion - DTT_diameter / 2, (
2376 gcpy              Joint_Width / (Number_of_Dovetails * 2)) - DTR, 90,
2377 gcpy              180)
2378 gcpy            self.dxfarc(self.currenttoolnumber(), (i + 1) * (
2379 gcpy              Joint_Width / (Number_of_Dovetails * 2)),
2380 gcpy              stockZthickness * Proportion - DTT_diameter / 2, (
2381 gcpy              Joint_Width / (Number_of_Dovetails * 2)) - DTR, 0,
2382 gcpy              90)
2383 gcpy            self.dxfline(self.currenttoolnumber(),
2384 gcpy              (i + 2) * (Joint_Width / (Number_of_Dovetails * 2))
2385 gcpy              - DTR,
2386 gcpy              0,
2387 gcpy              (i + 2) * (Joint_Width / (Number_of_Dovetails * 2))
2388 gcpy              - DTR, stockZthickness * Proportion -
2389 gcpy              DTT_diameter / 2)
2390 gcpy            self.dxfarc(self.currenttoolnumber(), (i + 2) * (
2391 gcpy              Joint_Width / (Number_of_Dovetails * 2)), 0, DTR,
2392 gcpy              180, 270)
2393 gcpy            i += 2
2394 gcpy            self.dxfarc(self.currenttoolnumber(), Joint_Width,
2395 gcpy              stockZthickness + DTR, DTR, 0, 90)
2396 gcpy            self.dxfline(self.currenttoolnumber(), Joint_Width + DTR,
2397 gcpy              stockZthickness + DTR, Joint_Width + DTR, 0)
2398 gcpy            self.dxfarc(self.currenttoolnumber(), Joint_Width, 0, DTR,

```

```
270, 360)
2368 gcpy      return ctp
```

which are used as:

```
toolpaths = gcp.cut_pins(stockXwidth, stockZthickness, Number_of_Dovetails, Spacing, Proportion, DTT_di
toolpaths.append(gcp.cut_tails(stockXwidth, stockZthickness, Number_of_Dovetails, Spacing, Proportion, I
```

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

**3.7.0.2.5 Full-blind box joints** BlocksCAD project page at: <https://www.blocks cad3d.com/community/projects/1943966> and discussion at: <https://community.carbide3d.com/t/full-blind-box-joints-in-carbide-create/53329>

Full-blind box joints will require 3 separate tools:

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

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

- horizontal
- vertical

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

```
2370 gcpy      def Full_Blind_Finger_Joint_square(self, bx, by, orientation,
2371 gcpy          side, width, thickness, Number_of_Pins, largeVdiameter,
2372 gcpy          smallDiameter, normalormirror = "Default"):
2373 gcpy          # Joint_Orientation = "Horizontal" "Even" == "Lower", "Odd" ==
2374 gcpy          # "Upper"
2375 gcpy          # Joint_Orientation = "Vertical" "Even" == "Left", "Odd" == "
2376 gcpy          # "Right"
2377 gcpy          if (orientation == "Vertical"):
2378 gcpy              if (normalormirror == "Default" and side != "Both"):
2379 gcpy                  if (side == "Left"):
2380 gcpy                      normalormirror = "Even"
2381 gcpy                  if (side == "Right"):
2382 gcpy                      normalormirror = "Odd"
2383 gcpy          if (orientation == "Horizontal"):
2384 gcpy              if (normalormirror == "Default" and side != "Both"):
2385 gcpy                  if (side == "Lower"):
2386 gcpy                      normalormirror = "Even"
2387 gcpy                  if (side == "Upper"):
2388 gcpy                      normalormirror = "Odd"
2389 gcpy          Finger_Width = ((Number_of_Pins * 2) - 1) * smallDiameter *
2390 gcpy          1.1
2391 gcpy          Finger_Origin = width/2 - Finger_Width/2
2392 gcpy          rapid = self.rapidZ(0)
2393 gcpy          self.setdxcolor("Cyan")
2394 gcpy          rapid = rapid.union(self.rapidXY(bx, by))
2395 gcpy          toolpath = (self.Finger_Joint_square(bx, by, orientation,
2396 gcpy              side, width, thickness, Number_of_Pins, Finger_Origin,
2397 gcpy              smallDiameter))
2398 gcpy          if (orientation == "Vertical"):
2399 gcpy              if (side == "Both"):
2400 gcpy                  toolpath = self.cutrectangleroundxf(self.
2401 gcpy                      currenttoolnum, bx - (thickness - smallDiameter
2402 gcpy                      /2), by-smallDiameter/2, 0, (thickness * 2) -
2403 gcpy                      smallDiameter, width+smallDiameter, (
2404 gcpy                      smallDiameter / 2) / Tan(math.radians(45)),
2405 gcpy                      smallDiameter/2)
2406 gcpy              if (side == "Left"):
2407 gcpy                  toolpath = self.cutrectangleroundxf(self.
2408 gcpy                      currenttoolnum, bx - (smallDiameter/2), by-
2409 gcpy                      smallDiameter/2, 0, thickness, width+
2410 gcpy                      smallDiameter, ((smallDiameter / 2) / Tan(math.
2411 gcpy                      radians(45))), smallDiameter/2)
2412 gcpy              if (side == "Right"):
2413 gcpy                  toolpath = self.cutrectangleroundxf(self.
2414 gcpy                      currenttoolnum, bx - (thickness - smallDiameter
2415 gcpy                      /2), by-smallDiameter/2, 0, thickness, width+
2416 gcpy                      smallDiameter, ((smallDiameter / 2) / Tan(math.
2417 gcpy                      radians(45))), smallDiameter/2)
```

```

2398 gcpy          toolpath = toolpath.union(self.Finger_Joint_square(bx, by,
2399 gcpy              orientation, side, width, thickness, Number_of_Pins,
2400 gcpy              Finger-Origin, smallDiameter))
2401 gcpy          if (orientation == "Horizontal"):
2402 gcpy              if (side == "Both"):
2403 gcpy                  toolpath = self.cutrectanglerounddxf(
2404 gcpy                      self.currenttoolnum,
2405 gcpy                      bx-smallDiameter/2,
2406 gcpy                      by - (thickness - smallDiameter/2),
2407 gcpy                      0,
2408 gcpy                      width+smallDiameter,
2409 gcpy                      (thickness * 2) - smallDiameter,
2410 gcpy                      (smallDiameter / 2) / Tan(math.radians(45)),
2411 gcpy                      smallDiameter/2)
2412 gcpy              if (side == "Lower"):
2413 gcpy                  toolpath = self.cutrectanglerounddxf(
2414 gcpy                      self.currenttoolnum,
2415 gcpy                      bx - (smallDiameter/2),
2416 gcpy                      by - smallDiameter/2,
2417 gcpy                      0,
2418 gcpy                      width+smallDiameter,
2419 gcpy                      thickness,
2420 gcpy                      ((smallDiameter / 2) / Tan(math.radians(45))),
2421 gcpy                      smallDiameter/2)
2422 gcpy              if (side == "Upper"):
2423 gcpy                  toolpath = self.cutrectanglerounddxf(
2424 gcpy                      self.currenttoolnum,
2425 gcpy                      bx - smallDiameter/2,
2426 gcpy                      by - (thickness - smallDiameter/2),
2427 gcpy                      0,
2428 gcpy                      width+smallDiameter,
2429 gcpy                      thickness,
2430 gcpy                      ((smallDiameter / 2) / Tan(math.radians(45))),
2431 gcpy                      smallDiameter/2)
2432 gcpy          toolpath = toolpath.union(self.Finger_Joint_square(bx, by,
2433 gcpy              orientation, side, width, thickness, Number_of_Pins,
2434 gcpy              Finger-Origin, smallDiameter))
2435 gcpy          return toolpath
2436 gcpy
2437 gcpy          def Finger_Joint_square(self, bx, by, orientation, side, width,
2438 gcpy              thickness, Number_of_Pins, Finger-Origin, smallDiameter,
2439 gcpy              normalormirror = "Default"):
2440 gcpy              jointdepth = -(thickness - (smallDiameter / 2) / Tan(math.
2441 gcpy                  radians(45)))
2442 gcpy              # Joint_Orientation = "Horizontal" "Even" == "Lower", "Odd" ==
2443 gcpy                  "Upper"
2444 gcpy              # Joint_Orientation = "Vertical" "Even" == "Left", "Odd" == "
2445 gcpy                  Right"
2446 gcpy              if (orientation == "Vertical"):
2447 gcpy                  if (normalormirror == "Default" and side != "Both"):
2448 gcpy                      if (side == "Left"):
2449 gcpy                          normalormirror = "Even"
2450 gcpy                      if (side == "Right"):
2451 gcpy                          normalormirror = "Odd"
2452 gcpy              if (orientation == "Horizontal"):
2453 gcpy                  if (normalormirror == "Default" and side != "Both"):
2454 gcpy                      if (side == "Lower"):
2455 gcpy                          normalormirror = "Even"
2456 gcpy                      if (side == "Upper"):
2457 gcpy                          normalormirror = "Odd"
2458 gcpy              radius = smallDiameter/2
2459 gcpy              jointwidth = thickness - smallDiameter
2460 gcpy              toolpath = self.currenttool()
2461 gcpy              rapid = self.rapidZ(0)
2462 gcpy              self.setdxfcolor("Blue")
2463 gcpy              toolpath = toolpath.union(self.cutlineZgcfeed(jointdepth
2464 gcpy                  ,1000))
2465 gcpy              self.beginpolyline(self.currenttool())
2466 gcpy              if (orientation == "Vertical"):
2467 gcpy                  rapid = rapid.union(self.rapidXY(bx, by + Finger-Origin
2468 gcpy                      ))
2469 gcpy              self.addvertex(self.currenttoolnumber(), self.xpos(),
2470 gcpy                  self.ypos())
2471 gcpy              toolpath = toolpath.union(self.cutlineZgcfeed(
2472 gcpy                  jointdepth,1000))
2473 gcpy              i = 0
2474 gcpy              while i <= Number_of_Pins - 1:
2475 gcpy                  if (side == "Right"):

```

```

2463 gcpy                toolpath = toolpath.union(self.cutvertexdxf(
                        self.xpos(), self.ypos() + smallDiameter +
                        radius/5, jointdepth))
2464 gcpy                if (side == "Left" or side == "Both"):
2465 gcpy                    toolpath = toolpath.union(self.cutvertexdxf(
                        self.xpos(), self.ypos() + radius,
                        jointdepth))
2466 gcpy                toolpath = toolpath.union(self.cutvertexdxf(
                        self.xpos() + jointwidth, self.ypos(),
                        jointdepth))
2467 gcpy                toolpath = toolpath.union(self.cutvertexdxf(
                        self.xpos(), self.ypos() + radius/5,
                        jointdepth))
2468 gcpy                toolpath = toolpath.union(self.cutvertexdxf(
                        self.xpos() - jointwidth, self.ypos(),
                        jointdepth))
2469 gcpy                toolpath = toolpath.union(self.cutvertexdxf(
                        self.xpos(), self.ypos() + radius,
                        jointdepth))
2470 gcpy                if (side == "Left"):
2471 gcpy                    toolpath = toolpath.union(self.cutvertexdxf(
                        self.xpos(), self.ypos() + smallDiameter +
                        radius/5, jointdepth))
2472 gcpy                if (side == "Right" or side == "Both"):
2473 gcpy                    if (i < (Number_of_Pins - 1)):
2474 gcpy                        # print(i)
2475 gcpy                        toolpath = toolpath.union(self.cutvertexdxf(
                            self.xpos(), self.ypos() + radius,
                            jointdepth))
2476 gcpy                        toolpath = toolpath.union(self.cutvertexdxf(
                            self.xpos() - jointwidth, self.ypos(),
                            jointdepth))
2477 gcpy                        toolpath = toolpath.union(self.cutvertexdxf(
                            self.xpos(), self.ypos() + radius/5,
                            jointdepth))
2478 gcpy                        toolpath = toolpath.union(self.cutvertexdxf(
                            self.xpos() + jointwidth, self.ypos(),
                            jointdepth))
2479 gcpy                        toolpath = toolpath.union(self.cutvertexdxf(
                            self.xpos(), self.ypos() + radius,
                            jointdepth))
2480 gcpy                    i += 1
2481 gcpy                    # Joint_Orientation = "Horizontal" "Even" == "Lower", "Odd" ==
                        "Upper"
2482 gcpy                    if (orientation == "Horizontal"):
2483 gcpy                        rapid = rapid.union(self.rapidXY(bx + Finger_Origin, by
                            ))
2484 gcpy                        self.addvertex(self.currenttoolnumber(), self.xpos(),
                            self.ypos())
2485 gcpy                        toolpath = toolpath.union(self.cutlineZgcfeed(
                            jointdepth,1000))
2486 gcpy                    i = 0
2487 gcpy                    while i <= Number_of_Pins - 1:
2488 gcpy                        if (side == "Upper"):
2489 gcpy                            toolpath = toolpath.union(self.cutvertexdxf(
                                self.xpos() + smallDiameter + radius/5, self
                                .ypos(), jointdepth))
2490 gcpy                        if (side == "Lower" or side == "Both"):
2491 gcpy                            toolpath = toolpath.union(self.cutvertexdxf(
                                self.xpos() + radius, self.ypos(),
                                jointdepth))
2492 gcpy                            toolpath = toolpath.union(self.cutvertexdxf(
                                self.xpos(), self.ypos() + jointwidth,
                                jointdepth))
2493 gcpy                            toolpath = toolpath.union(self.cutvertexdxf(
                                self.xpos() + radius/5, self.ypos(),
                                jointdepth))
2494 gcpy                            toolpath = toolpath.union(self.cutvertexdxf(
                                self.xpos(), self.ypos() - jointwidth,
                                jointdepth))
2495 gcpy                            toolpath = toolpath.union(self.cutvertexdxf(
                                self.xpos() + radius, self.ypos(),
                                jointdepth))
2496 gcpy                        if (side == "Lower"):
2497 gcpy                            toolpath = toolpath.union(self.cutvertexdxf(
                                self.xpos() + smallDiameter + radius/5, self
                                .ypos(), jointdepth))
2498 gcpy                        if (side == "Upper" or side == "Both"):

```

```

2499 gcpy                                if (i < (Number_of_Pins - 1)):
2500 gcpy                                #         print(i)
2501 gcpy                                toolpath = toolpath.union(self.cutvertexdxf
                                     (self.xpos() + radius, self.ypos(),
                                     jointdepth))
2502 gcpy                                toolpath = toolpath.union(self.cutvertexdxf
                                     (self.xpos(), self.ypos() - jointwidth,
                                     jointdepth))
2503 gcpy                                toolpath = toolpath.union(self.cutvertexdxf
                                     (self.xpos() + radius/5, self.ypos(),
                                     jointdepth))
2504 gcpy                                toolpath = toolpath.union(self.cutvertexdxf
                                     (self.xpos(), self.ypos() + jointwidth,
                                     jointdepth))
2505 gcpy                                toolpath = toolpath.union(self.cutvertexdxf
                                     (self.xpos() + radius, self.ypos(),
                                     jointdepth))
2506 gcpy                                i += 1
2507 gcpy                                self.closepolyline(self.currenttoolnumber())
2508 gcpy                                return toolpath
2509 gcpy
2510 gcpy
2511 gcpy    def Full_Blind_Finger_Joint_smallV(self, bx, by, orientation,
        side, width, thickness, Number_of_Pins, largeVdiameter,
        smallDiameter):
2512 gcpy        rapid = self.rapidZ(0)
2513 gcpy        #         rapid = rapid.union(self.rapidXY(bx, by))
2514 gcpy        self.setdxfcolor("Red")
2515 gcpy        if (orientation == "Vertical"):
2516 gcpy            rapid = rapid.union(self.rapidXY(bx, by - smallDiameter
                /6))
2517 gcpy            toolpath = self.cutlineZgcfeed(-thickness,1000)
2518 gcpy            toolpath = self.cutlinedxfgc(bx, by + width +
                smallDiameter/6, - thickness)
2519 gcpy        if (orientation == "Horizontal"):
2520 gcpy            rapid = rapid.union(self.rapidXY(bx - smallDiameter/6,
                by))
2521 gcpy            toolpath = self.cutlineZgcfeed(-thickness,1000)
2522 gcpy            toolpath = self.cutlinedxfgc(bx + width + smallDiameter
                /6, by, -thickness)
2523 gcpy        #         rapid = self.rapidZ(0)
2524 gcpy
2525 gcpy        return toolpath
2526 gcpy
2527 gcpy    def Full_Blind_Finger_Joint_largeV(self, bx, by, orientation,
        side, width, thickness, Number_of_Pins, largeVdiameter,
        smallDiameter):
2528 gcpy        radius = smallDiameter/2
2529 gcpy        rapid = self.rapidZ(0)
2530 gcpy        Finger_Width = ((Number_of_Pins * 2) - 1) * smallDiameter *
        1.1
2531 gcpy        Finger-Origin = width/2 - Finger_Width/2
2532 gcpy        #         rapid = rapid.union(self.rapidXY(bx, by))
2533 gcpy        # Joint_Orientation = "Horizontal" "Even" == "Lower", "Odd" ==
        "Upper"
2534 gcpy        # Joint_Orientation = "Vertical" "Even" == "Left", "Odd" == "
        Right"
2535 gcpy        if (orientation == "Vertical"):
2536 gcpy            rapid = rapid.union(self.rapidXY(bx, by))
2537 gcpy            toolpath = self.cutlineZgcfeed(-thickness,1000)
2538 gcpy            toolpath = toolpath.union(self.cutlinedxfgc(bx, by +
                Finger-Origin, -thickness))
2539 gcpy            rapid = self.rapidZ(0)
2540 gcpy            rapid = rapid.union(self.rapidXY(bx, by + width -
                Finger-Origin))
2541 gcpy            self.setdxfcolor("Blue")
2542 gcpy            toolpath = toolpath.union(self.cutlineZgcfeed(-
                thickness,1000))
2543 gcpy            toolpath = toolpath.union(self.cutlinedxfgc(bx, by +
                width, -thickness))
2544 gcpy            if (side == "Left" or side == "Both"):
2545 gcpy                rapid = self.rapidZ(0)
2546 gcpy                self.setdxfcolor("Dark_Gray")
2547 gcpy                rapid = rapid.union(self.rapidXY(bx+thickness-(
                smallDiameter / 2) / Tan(math.radians(45)), by -
                radius/2))
2548 gcpy            toolpath = toolpath.union(self.cutlineZgcfeed(-(
                smallDiameter / 2) / Tan(math.radians(45))

```



```

,10000))
2549 gcpy      toolpath = toolpath.union(self.cutlinedxfgc(bx+
                thickness-(smallDiameter / 2) / Tan(math.radians
                (45)), by + width + radius/2, -(smallDiameter /
                2) / Tan(math.radians(45))))
2550 gcpy      rapid = self.rapidZ(0)
2551 gcpy      self.setdxfc("Green")
2552 gcpy      rapid = rapid.union(self.rapidXY(bx+thickness/2, by
                +width))
2553 gcpy      toolpath = toolpath.union(self.cutlineZgcfeed(-
                thickness/2,1000))
2554 gcpy      toolpath = toolpath.union(self.cutlinedxfgc(bx+
                thickness/2, by + width -thickness, -thickness
                /2))
2555 gcpy      rapid = self.rapidZ(0)
2556 gcpy      rapid = rapid.union(self.rapidXY(bx+thickness/2, by
                ))
2557 gcpy      toolpath = toolpath.union(self.cutlineZgcfeed(-
                thickness/2,1000))
2558 gcpy      toolpath = toolpath.union(self.cutlinedxfgc(bx+
                thickness/2, by +thickness, -thickness/2))
2559 gcpy      if (side == "Right" or side == "Both"):
2560 gcpy          rapid = self.rapidZ(0)
2561 gcpy          self.setdxfc("DarkGray")
2562 gcpy          rapid = rapid.union(self.rapidXY(bx-(thickness-(
                smallDiameter / 2) / Tan(math.radians(45))), by
                - radius/2))
2563 gcpy      toolpath = toolpath.union(self.cutlineZgcfeed(-(
                smallDiameter / 2) / Tan(math.radians(45))
                ,10000))
2564 gcpy      toolpath = toolpath.union(self.cutlinedxfgc(bx-(
                thickness-(smallDiameter / 2) / Tan(math.radians
                (45))), by + width + radius/2, -(smallDiameter /
                2) / Tan(math.radians(45))))
2565 gcpy      rapid = self.rapidZ(0)
2566 gcpy      self.setdxfc("Green")
2567 gcpy      rapid = rapid.union(self.rapidXY(bx-thickness/2, by
                +width))
2568 gcpy      toolpath = toolpath.union(self.cutlineZgcfeed(-
                thickness/2,1000))
2569 gcpy      toolpath = toolpath.union(self.cutlinedxfgc(bx-
                thickness/2, by + width -thickness, -thickness
                /2))
2570 gcpy      rapid = self.rapidZ(0)
2571 gcpy      rapid = rapid.union(self.rapidXY(bx-thickness/2, by
                ))
2572 gcpy      toolpath = toolpath.union(self.cutlineZgcfeed(-
                thickness/2,1000))
2573 gcpy      toolpath = toolpath.union(self.cutlinedxfgc(bx-
                thickness/2, by +thickness, -thickness/2))
2574 gcpy      # Joint_Orientation = "Horizontal" "Even" == "Lower", "Odd" ==
                "Upper"
2575 gcpy      if (orientation == "Horizontal"):
2576 gcpy          rapid = rapid.union(self.rapidXY(bx, by))
2577 gcpy          self.setdxfc("Blue")
2578 gcpy          toolpath = self.cutlineZgcfeed(-thickness,1000)
2579 gcpy          toolpath = toolpath.union(self.cutlinedxfgc(bx +
                Finger_Origin, by, -thickness))
2580 gcpy          rapid = rapid.union(self.rapidZ(0))
2581 gcpy          rapid = rapid.union(self.rapidXY(bx + width -
                Finger_Origin, by))
2582 gcpy          toolpath = toolpath.union(self.cutlineZgcfeed(-
                thickness,1000))
2583 gcpy          toolpath = toolpath.union(self.cutlinedxfgc(bx + width,
                by, -thickness))
2584 gcpy          if (side == "Lower" or side == "Both"):
2585 gcpy              rapid = self.rapidZ(0)
2586 gcpy              self.setdxfc("DarkGray")
2587 gcpy              rapid = rapid.union(self.rapidXY(bx - radius, by+
                thickness-(smallDiameter / 2) / Tan(math.radians
                (45))))
2588 gcpy          toolpath = toolpath.union(self.cutlineZgcfeed(-(
                smallDiameter / 2) / Tan(math.radians(45))
                ,10000))
2589 gcpy          toolpath = toolpath.union(self.cutlinedxfgc(bx +
                width + radius, by+thickness-(smallDiameter / 2)
                / Tan(math.radians(45)), -(smallDiameter / 2) /
                Tan(math.radians(45))))

```

```

2590 gcpy          rapid = self.rapidZ(0)
2591 gcpy          self.setdxfcolor("Green")
2592 gcpy          rapid = rapid.union(self.rapidXY(bx+width, by+
                thickness/2))
2593 gcpy          toolpath = toolpath.union(self.cutlineZgcfeed(-
                thickness/2,1000))
2594 gcpy          toolpath = toolpath.union(self.cutlinedxfgc(bx +
                width -thickness, by+thickness/2, -thickness/2))
2595 gcpy          rapid = self.rapidZ(0)
2596 gcpy          rapid = rapid.union(self.rapidXY(bx, by+thickness
                /2))
2597 gcpy          toolpath = toolpath.union(self.cutlineZgcfeed(-
                thickness/2,1000))
2598 gcpy          toolpath = toolpath.union(self.cutlinedxfgc(bx +
                thickness, by+thickness/2, -thickness/2))
2599 gcpy          if (side == "Upper" or side == "Both"):
2600 gcpy          rapid = self.rapidZ(0)
2601 gcpy          self.setdxfcolor("DarkGray")
2602 gcpy          rapid = rapid.union(self.rapidXY(bx - radius, by-(
                thickness-(smallDiameter / 2) / Tan(math.radians
                (45))))))
2603 gcpy          toolpath = toolpath.union(self.cutlineZgcfeed(-(
                smallDiameter / 2) / Tan(math.radians(45))
                ,10000))
2604 gcpy          toolpath = toolpath.union(self.cutlinedxfgc(bx +
                width + radius, by-(thickness-(smallDiameter /
                2) / Tan(math.radians(45))), -(smallDiameter /
                2) / Tan(math.radians(45))))
2605 gcpy          rapid = self.rapidZ(0)
2606 gcpy          self.setdxfcolor("Green")
2607 gcpy          rapid = rapid.union(self.rapidXY(bx+width, by-
                thickness/2))
2608 gcpy          toolpath = toolpath.union(self.cutlineZgcfeed(-
                thickness/2,1000))
2609 gcpy          toolpath = toolpath.union(self.cutlinedxfgc(bx +
                width -thickness, by-thickness/2, -thickness/2))
2610 gcpy          rapid = self.rapidZ(0)
2611 gcpy          rapid = rapid.union(self.rapidXY(bx, by-thickness
                /2))
2612 gcpy          toolpath = toolpath.union(self.cutlineZgcfeed(-
                thickness/2,1000))
2613 gcpy          toolpath = toolpath.union(self.cutlinedxfgc(bx +
                thickness, by-thickness/2, -thickness/2))
2614 gcpy          rapid = self.rapidZ(0)
2615 gcpy          return toolpath
2616 gcpy
2617 gcpy          def Full_Blind_Finger_Joint(self, bx, by, orientation, side,
                width, thickness, largeVdiameter, smallDiameter,
                normalormirror = "Default", squaretool = 102, smallV = 390,
                largeV = 301):
2618 gcpy          Number_of_Pins = int(((width - thickness * 2) / (
                smallDiameter * 2.2) / 2) + 0.0) * 2 + 1
2619 gcpy          # print("Number of Pins: ",Number_of_Pins)
2620 gcpy          self.movetosafeZ()
2621 gcpy          self.toolchange(squaretool, 17000)
2622 gcpy          toolpath = self.Full_Blind_Finger_Joint_square(bx, by,
                orientation, side, width, thickness, Number_of_Pins,
                largeVdiameter, smallDiameter)
2623 gcpy          self.movetosafeZ()
2624 gcpy          self.toolchange(smallV, 17000)
2625 gcpy          toolpath = toolpath.union(self.
                Full_Blind_Finger_Joint_smallV(bx, by, orientation, side
                , width, thickness, Number_of_Pins, largeVdiameter,
                smallDiameter))
2626 gcpy          self.toolchange(largeV, 17000)
2627 gcpy          toolpath = toolpath.union(self.
                Full_Blind_Finger_Joint_largeV(bx, by, orientation, side
                , width, thickness, Number_of_Pins, largeVdiameter,
                smallDiameter))
2628 gcpy          return toolpath

```

---

### 3.8 (Reading) G-code Files

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

First, a template file will be necessary:

---

```

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

```

---

previewgcodefile Which simply needs to call the previewgcodefile command:

---

```

2549 gcpy      def previewgcodefile(self, gc_file):
2550 gcpy          gc_file = open(gc_file, 'r')
2551 gcpy          gcfilecontents = []
2552 gcpy          with gc_file as file:
2553 gcpy              for line in file:
2554 gcpy                  command = line
2555 gcpy                  gcfilecontents.append(line)
2556 gcpy
2557 gcpy          numlinesfound = 0
2558 gcpy          for line in gcfilecontents:
2559 gcpy              # print(line)
2560 gcpy              if line[:10] == "(stockMin:":
2561 gcpy                  subdivisions = line.split()
2562 gcpy                  extentleft = float(subdivisions[0][10:-3])
2563 gcpy                  extentfb = float(subdivisions[1][:-3])
2564 gcpy                  extentd = float(subdivisions[2][:-3])
2565 gcpy                  numlinesfound = numlinesfound + 1
2566 gcpy              if line[:13] == "(STOCK/BLOCK,":
2567 gcpy                  subdivisions = line.split()
2568 gcpy                  sizeX = float(subdivisions[0][13:-1])
2569 gcpy                  sizeY = float(subdivisions[1][:-1])
2570 gcpy                  sizeZ = float(subdivisions[4][:-1])
2571 gcpy                  numlinesfound = numlinesfound + 1
2572 gcpy              if line[:3] == "G21":
2573 gcpy                  units = "mm"
2574 gcpy                  numlinesfound = numlinesfound + 1
2575 gcpy              if numlinesfound >=3:
2576 gcpy                  break
2577 gcpy              # print(numlinesfound)
2578 gcpy
2579 gcpy          self.setupcuttingarea(sizeX, sizeY, sizeZ, extentleft,
                extentfb, extentd)
2580 gcpy
2581 gcpy          commands = []
2582 gcpy          for line in gcfilecontents:
2583 gcpy              Xc = 0
2584 gcpy              Yc = 0
2585 gcpy              Zc = 0
2586 gcpy              Fc = 0
2587 gcpy              Xp = 0.0
2588 gcpy              Yp = 0.0
2589 gcpy              Zp = 0.0
2590 gcpy              if line == "G53G0Z-5.000\n":
2591 gcpy                  self.movetosafeZ()
2592 gcpy              if line[:3] == "M6T":
2593 gcpy                  tool = int(line[3:])
2594 gcpy                  self.toolchange(tool)
2595 gcpy              if line[:2] == "G0":
2596 gcpy                  machinestate = "rapid"
2597 gcpy              if line[:2] == "G1":
2598 gcpy                  machinestate = "cutline"
2599 gcpy              if line[:2] == "G0" or line[:2] == "G1" or line[:1] ==
                "X" or line[:1] == "Y" or line[:1] == "Z":
2600 gcpy                  if "F" in line:
2601 gcpy                      Fplus = line.split("F")
2602 gcpy                      Fc = 1
2603 gcpy                      fr = float(Fplus[1])
2604 gcpy                      line = Fplus[0]
2605 gcpy                  if "Z" in line:
2606 gcpy                      Zplus = line.split("Z")
2607 gcpy                      Zc = 1
2608 gcpy                      Zp = float(Zplus[1])
2609 gcpy                      line = Zplus[0]

```

```

2610 gcpy          if "Y" in line:
2611 gcpy              Yplus = line.split("Y")
2612 gcpy              Yc = 1
2613 gcpy              Yp = float(Yplus[1])
2614 gcpy              line = Yplus[0]
2615 gcpy          if "X" in line:
2616 gcpy              Xplus = line.split("X")
2617 gcpy              Xc = 1
2618 gcpy              Xp = float(Xplus[1])
2619 gcpy          if Zc == 1:
2620 gcpy              if Yc == 1:
2621 gcpy                  if Xc == 1:
2622 gcpy                      if machinestate == "rapid":
2623 gcpy                          command = "rapidXYZ(" + str(Xp) + "
                                ,␣" + str(Yp) + ",␣" + str(Zp) +
                                ")"
                                self.rapidXYZ(Xp, Yp, Zp)
2624 gcpy                      else:
2625 gcpy                          command = "cutlineXYZ(" + str(Xp) +
                                ",␣" + str(Yp) + ",␣" + str(Zp) +
                                + ")"
                                self.cutlineXYZ(Xp, Yp, Zp)
2626 gcpy
2627 gcpy          else:
2628 gcpy              if machinestate == "rapid":
2629 gcpy                  command = "rapidYZ(" + str(Yp) + ",
2630 gcpy                      ␣" + str(Zp) + ")"
2631 gcpy                  self.rapidYZ(Yp, Zp)
2632 gcpy              else:
2633 gcpy                  command = "cutlineYZ(" + str(Yp) +
                                ",␣" + str(Zp) + ")"
2634 gcpy                  self.cutlineYZ(Yp, Zp)
2635 gcpy          else:
2636 gcpy              if Xc == 1:
2637 gcpy                  if machinestate == "rapid":
2638 gcpy                      command = "rapidXZ(" + str(Xp) + ",
                                ␣" + str(Zp) + ")"
2639 gcpy                      self.rapidXZ(Xp, Zp)
2640 gcpy                  else:
2641 gcpy                      command = "cutlineXZ(" + str(Xp) +
                                ",␣" + str(Zp) + ")"
2642 gcpy                      self.cutlineXZ(Xp, Zp)
2643 gcpy              else:
2644 gcpy                  if machinestate == "rapid":
2645 gcpy                      command = "rapidZ(" + str(Zp) + ")"
2646 gcpy                      self.rapidZ(Zp)
2647 gcpy                  else:
2648 gcpy                      command = "cutlineZ(" + str(Zp) + "
                                )"
2649 gcpy                      self.cutlineZ(Zp)
2650 gcpy          else:
2651 gcpy              if Yc == 1:
2652 gcpy                  if Xc == 1:
2653 gcpy                      if machinestate == "rapid":
2654 gcpy                          command = "rapidXY(" + str(Xp) + ",
                                ␣" + str(Yp) + ")"
2655 gcpy                          self.rapidXY(Xp, Yp)
2656 gcpy                      else:
2657 gcpy                          command = "cutlineXY(" + str(Xp) +
                                ",␣" + str(Yp) + ")"
2658 gcpy                          self.cutlineXY(Xp, Yp)
2659 gcpy          else:
2660 gcpy              if machinestate == "rapid":
2661 gcpy                  command = "rapidY(" + str(Yp) + ")"
2662 gcpy                  self.rapidY(Yp)
2663 gcpy              else:
2664 gcpy                  command = "cutlineY(" + str(Yp) + "
                                )"
2665 gcpy                  self.cutlineY(Yp)
2666 gcpy          else:
2667 gcpy              if Xc == 1:
2668 gcpy                  if machinestate == "rapid":
2669 gcpy                      command = "rapidX(" + str(Xp) + ")"
2670 gcpy                      self.rapidX(Xp)
2671 gcpy                  else:
2672 gcpy                      command = "cutlineX(" + str(Xp) + "
                                )"
2673 gcpy                      self.cutlineX(Xp)
2674 gcpy          commands.append(command)

```

```
2675 gcpy #                print(line)
2676 gcpy #                print(command)
2677 gcpy #                print(machinestate, Xc, Yc, Zc)
2678 gcpy #                print(Xp, Yp, Zp)
2679 gcpy #                print("/n")
2680 gcpy
2681 gcpy #                for command in commands:
2682 gcpy #                    print(command)
2683 gcpy
2684 gcpy                show(self.stockandtoolpaths())
```

---

## 4 Notes

### 4.1 Other Resources

#### 4.1.1 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

#### 4.1.2 Coding References

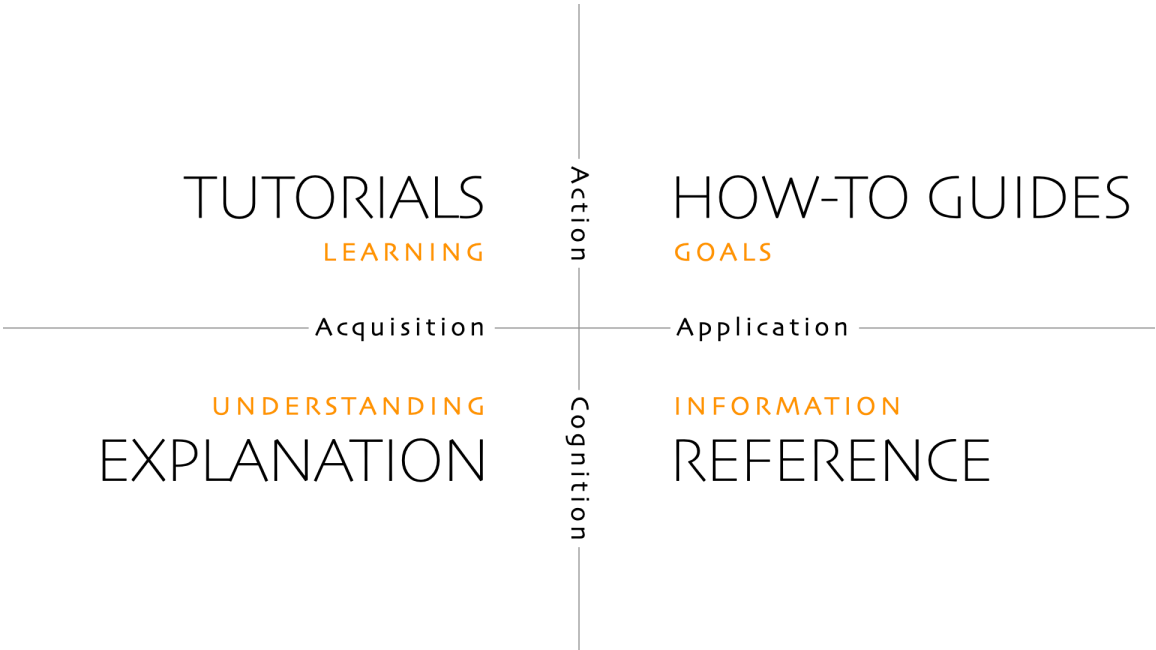
<https://thewhitetulip.gitbook.io/py/06-file-handling>

#### 4.1.3 Documentation Style

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

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

resulting in a matrix of:



where:

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

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

4.1.4 Holidays

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

4.1.5 DXFs

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

4.2 Future

4.2.1 Images

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

4.2.2 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>

4.2.3 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:

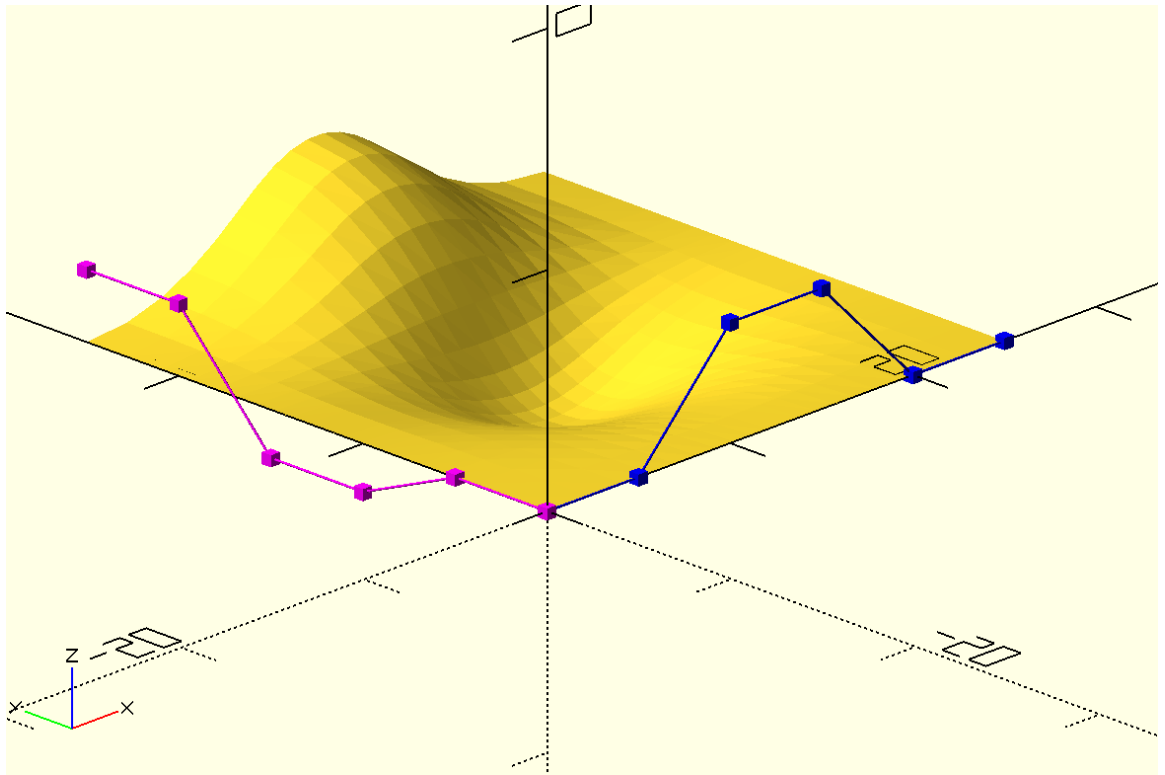
3 planes \* 3 Béziers \* (2 on-curve + 2 off-curve points) == 36 coordinate pairs

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/)

#### 4.2.4 Mathematics

<https://elementsofprogramming.com/>

References

[ConstGeom] Walmsley, Brian. *Construction Geometry*. 2d ed., Centennial College Press, 1981.

[MkCalc] Horvath, Joan, and Rich Cameron. *Make: Calculus: Build models to learn, visualize, and explore*. First edition., Make: Community LLC, 2022.

[MkGeom] Horvath, Joan, and Rich Cameron. *Make: Geometry: Learn by 3D Printing, Coding and Exploring*. First edition., Make: Community LLC, 2021.

[MkTrig] Horvath, Joan, and Rich Cameron. *Make: Trigonometry: Build your way from triangles to analytic geometry*. First edition., Make: Community LLC, 2023.

[PractShopMath] Begnal, Tom. *Practical Shop Math: Simple Solutions to Workshop Fractions, Formulas + Geometric Shapes*. Updated edition, Spring House Press, 2018.

[RS274] Thomas R. Kramer, Frederick M. Proctor, Elena R. Messina.  
[https://tsapps.nist.gov/publication/get\\_pdf.cfm?pub\\_id=823374](https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=823374)  
<https://www.nist.gov/publications/nist-rs274ngc-interpreter-version-3>

[SoftwareDesign] Ousterhout, John K. *A Philosophy of Software Design*. First Edition., Yaknyam Press, Palo Alto, Ca., 2018



# Command Glossary

. 25

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

# Index

- addvertex, 62
- ballnose, 46
- beginpolyline, 62
- closedxfile, 70, 71
- closegcodefile, 70, 71
- closepolyline, 62
- currenttoolnum, 28
- cut..., 43, 50
- cutarcCC, 52
- cutarcCW, 52
- cutkeyhole toolpath, 76
- cutKHgcdxf, 77
- cutline, 50
- cutrectangle, 75
- diameter, 29
- dovetail, 47
- dxfarc, 62
- dxfcircle, 62
- dxfline, 62
- dxfpreamble, 70, 71
- dxfpreamble, 62
- dxfwrite, 61
- endmill square, 45
- endmill v, 46
- endmilltype, 29
- feed, 57
- flute, 29
- gcodepreview, 27
  - writeln, 58
- gcp.setupstock, 29
- init, 27
- mpx, 28
- mpy, 28
- mpz, 28
- opendxfile, 58
- opengcodefile, 58, 59
- plunge, 57
- previewgcodefile, 91
- ra, 29
- rapid, 48
- rapid..., 43
- rapids, 29
- roundover, 48
- setupstock, 29
  - gcodepreview, 29
- setxpos, 28
- setypos, 28
- setzpos, 28
- shaftmovement, 44, 48
- speed, 57
- stockzero, 30
- subroutine
  - gcodepreview, 29
  - writeln, 58
- tip, 29
- tool diameter, 55
- tool number, 35
- tool radius, 57
- toolchange, 35, 36
- toolmovement, 28, 29, 32, 36, 44
- toolpaths, 29
- tpzinc, 28
- writedxfileDT, 62
- writedxfileKH, 62
- writedxfilegbl, 61
- writedxfilegsq, 61
- writedxfilegV, 61
- writedxfilesmbl, 61
- writedxfilemsq, 61
- writedxfilemV, 62
- xpos, 28
- ypos, 28
- zeroheight, 30
- zpos, 28

# Routines

addvertex, 62	previewgcodefile, 91
ballnose, 46	rapid, 48
beginpolyline, 62	rapid..., 43
	roundover, 48
closedxfile, 70, 71	
closegcodefile, 70, 71	setupstock, 29
closepolyline, 62	setxpos, 28
cut..., 43, 50	setypos, 28
cutarcCC, 52	setzpos, 28
cutarcCW, 52	shaftmovement, 44, 48
cutkeyhole toolpath, 76	
cutKHgcdxf, 77	tool diameter, 55
cutline, 50	tool radius, 57
cutrectangle, 75	toolchange, 35, 36
	toolmovement, 28, 29, 32, 36, 44
dovetail, 47	
dxfarc, 62	writedxfileDT, 62
dxfcircle, 62	writedxfileKH, 62
dxfline, 62	writedxfilegbl, 61
dxfpreamble, 70, 71	writedxfilegsq, 61
dxfpreamble, 62	writedxfilegV, 61
dxfwrite, 61	writedxfilesmbl, 61
	writedxfilesmsq, 61
endmill square, 45	writedxfilesmV, 62
endmill v, 46	writeln, 58
gcodepreview, 27, 29	xpos, 28
gcp.setupstock, 29	
	ypos, 28
init, 27	
	zpos, 28
opendxfile, 58	
opengcodefile, 58, 59	

# Variables

currenttoolnum, 28	ra, 29
diameter, 29	rapids, 29
endmilltype, 29	speed, 57
feed, 57	stockzero, 30
flute, 29	tip, 29
mpx, 28	tool number, 35
mpy, 28	toolpaths, 29
mpz, 28	tpzinc, 28
plunge, 57	zeroheight, 30