

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.

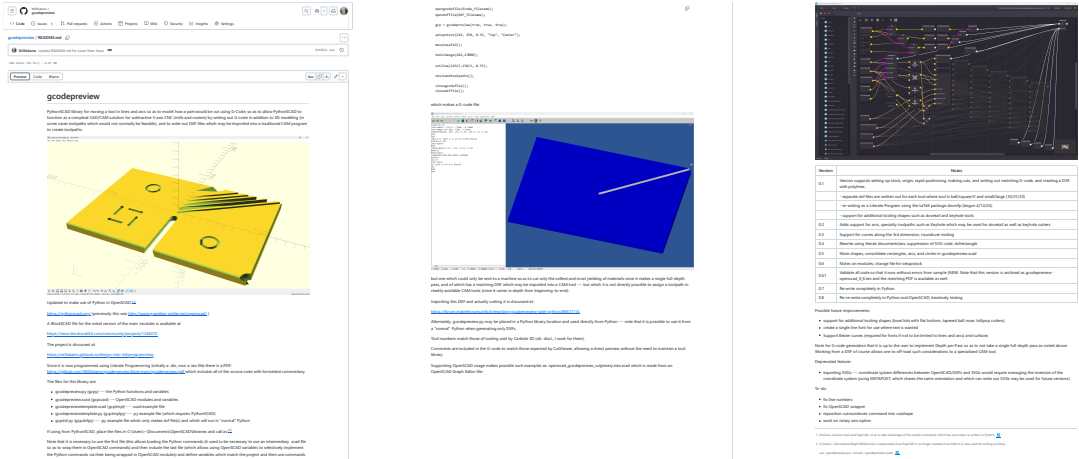
Contents

1	readme.md	3
2	Usage and Templates	7
2.1	gcpdxf.py	7
2.2	gcpcutdxf.py	11
2.3	gcodepreviewtemplate.py	13
2.4	gcodepreviewtemplate.scad	18
3	gcodepreview	23
3.1	Cutviewer	23
3.1.1	Stock size and placement	23
3.1.2	Tool Shapes	23
3.1.2.1	Tool/Mill (Square, radiused, ball-nose, and tapered-ball)	24
3.1.2.2	Corner Rounding, (roundover)	24
3.1.2.3	V shaped tooling (and variations)	24
3.2	Module Naming Convention	24
3.2.1	Parameters and Default Values	27
3.3	Implementation files and gcodepreview class	27
3.3.1	Position and Variables	29
3.3.2	Initial Modules	30
3.3.2.1	setupstock	31
3.3.3	Adjustments and Additions	33
3.4	Tools and Shapes and Changes	34
3.4.1	Numbering for Tools	34
3.4.1.1	toolchange	37
3.4.1.2	Square (including O-flute)	38
3.4.1.3	Ball-nose (including tapered-ball)	39
3.4.1.4	V	40
3.4.1.5	Keyhole	41
3.4.1.6	Bowl	41
3.4.1.7	Tapered ball nose	42
3.4.1.8	Roundover (corner rounding)	43
3.4.1.9	Dovetails	44
3.4.1.10	closing G-code	45
3.4.2	Laser support	45
3.5	Shapes and tool movement	45
3.5.0.1	Tooling for Undercutting Toolpaths	46
3.5.1	Generalized commands and cuts	46
3.5.2	Movement and color	46
3.5.2.1	toolmovement	47
3.5.2.2	Normal Tooling/toolshapes	48
3.5.2.3	Square (including O-flute)	48
3.5.2.4	Ball nose (including tapered ball nose)	48
3.5.2.5	bowl	48
3.5.2.6	V	49
3.5.2.7	Keyhole	49
3.5.2.8	Tapered ball nose	49
3.5.2.9	Dovetails	50
3.5.2.10	Concave toolshapes	50
3.5.2.11	Roundover tooling	50
3.5.2.12	shaftmovement	50

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

3.5.2.13	rapid and cut (lines)	51
3.5.2.14	Arcs	53
3.5.3	tooldiameter	58
3.5.4	Feeds and Speeds	60
3.6	Difference of Stock, Rapids, and Toolpaths	60
3.7	Output files	60
3.7.1	Python and OpenSCAD File Handling	61
3.7.2	DXF Overview	64
3.7.2.1	Writing to DXF files	64
3.7.2.1.1	DXF Lines and Arcs	65
3.7.3	G-code Overview	71
3.7.3.1	Closings	73
3.8	Cutting shapes and expansion	74
3.8.0.1	Building blocks	74
3.8.0.2	List of shapes	74
3.8.0.2.1	circles	76
3.8.0.2.2	rectangles	76
3.8.0.2.3	Keyhole toolpath and undercut tooling	79
3.8.0.2.4	Dovetail joinery and tooling	86
3.8.0.2.5	Full-blind box joints	88
3.9	(Reading) G-code Files	93
4	Notes	96
4.1	Other Resources	96
4.1.1	Coding Style	96
4.1.2	Coding References	97
4.1.3	Documentation Style	97
4.1.4	Holidays	97
4.1.5	DXFs	97
4.2	Future	98
4.2.1	Images	98
4.2.2	Bézier curves in 2 dimensions	98
4.2.3	Bézier curves in 3 dimensions	98
4.2.4	Mathematics	99
	Index	102
	Routines	103
	Variables	104

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/gcodepreviewtemplate.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 gcpdxf.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, dxfrectangle

           |
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.

```

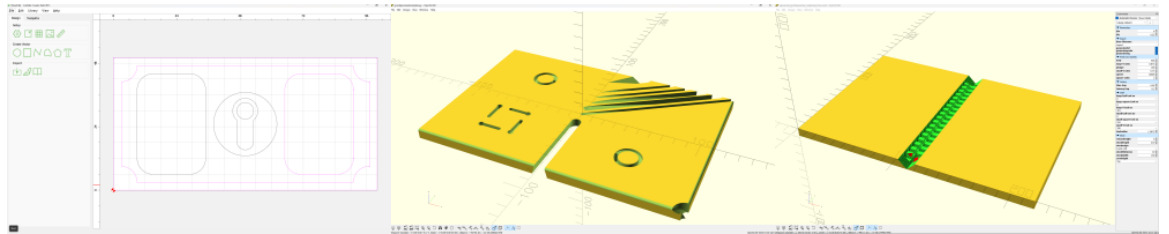
```
116 rdme
117 rdme To-do:
118 rdme
119 rdme - determine why one quadrant of arc command doesn't work in
        OpenSCAD
120 rdme - clock-wise arcs
121 rdme - add toolpath for cutting countersinks using ball-nose tool from
        inside working out
122 rdme - verify OpenSCAD wrapper and add any missing commands for Python
123 rdme - verify support for shaft on tooling
124 rdme - create a folder of template and sample files
125 rdme - clean up/comment out all mentions of previous versions/features/
        implementations/deprecated features
126 rdme - fully implement/verify describing/saving/loading tools using
        CutViewer comments
127 rdme
128 rdme Deprecated feature:
129 rdme
130 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)

```



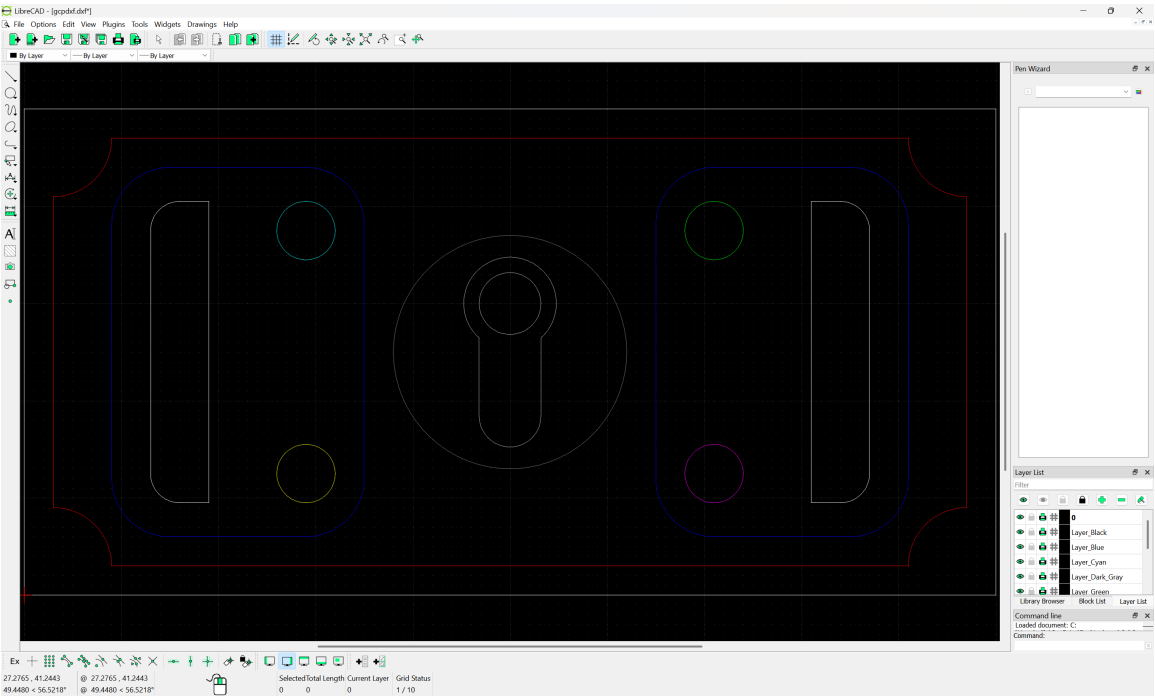
```

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)

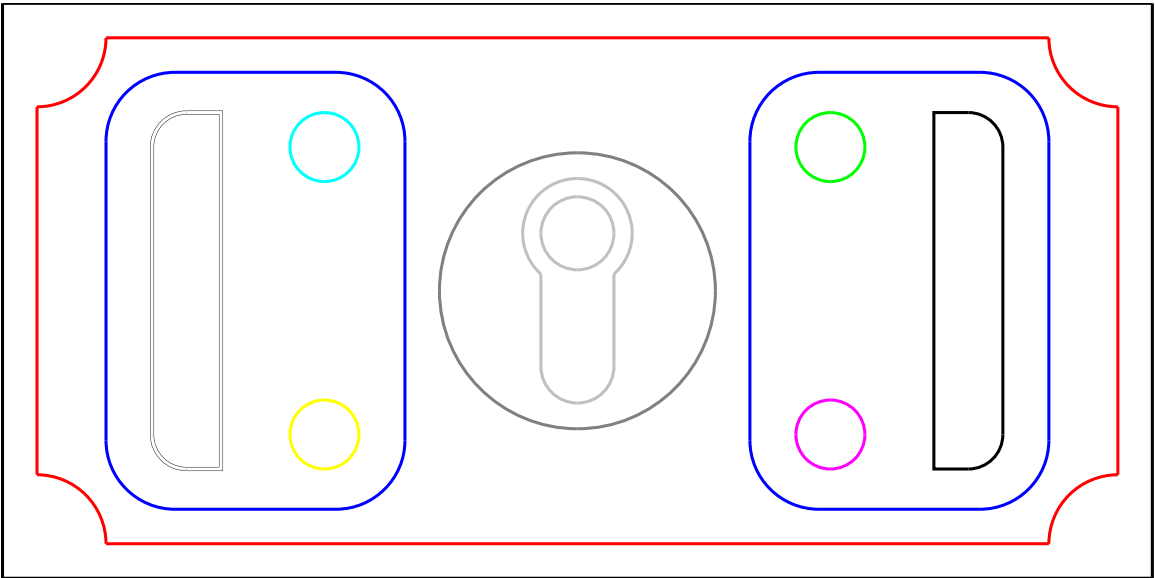
```

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

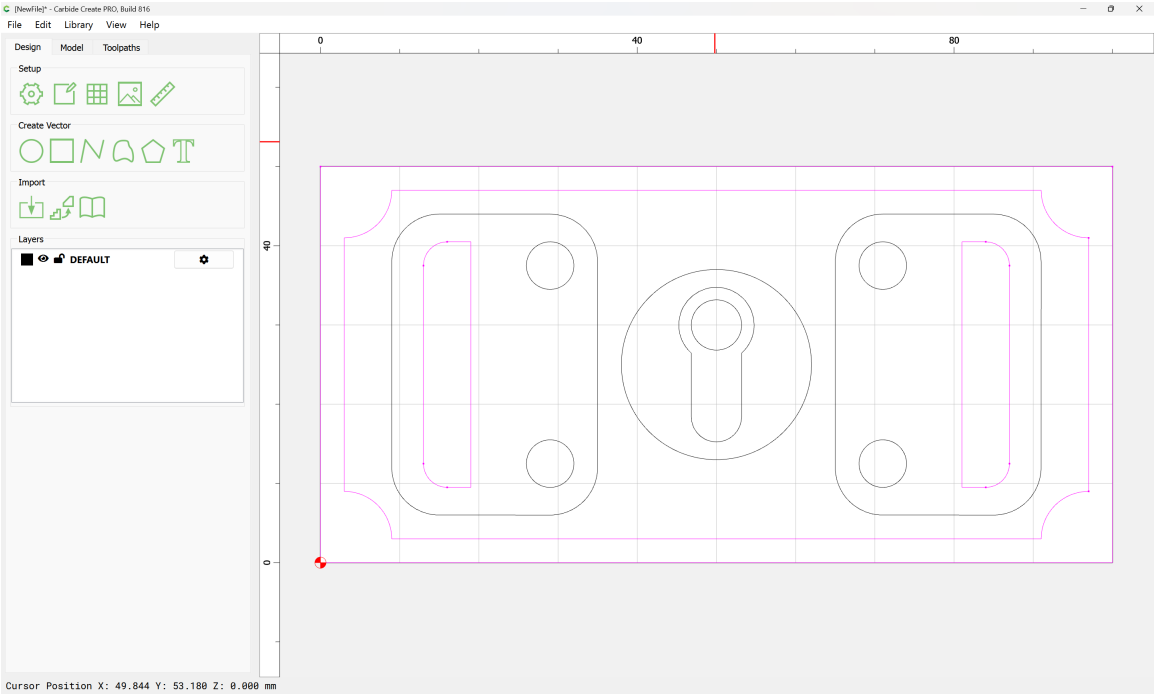
which creates a .dxf file which may be imported into any CAD program:



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 a simplified version of 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 gcpcutdxfp
96 gcpcutdxfp gcp.rapid(stockXwidth/2 + (radius * 2) + inset + radius, radius, 0,
    "laser")

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

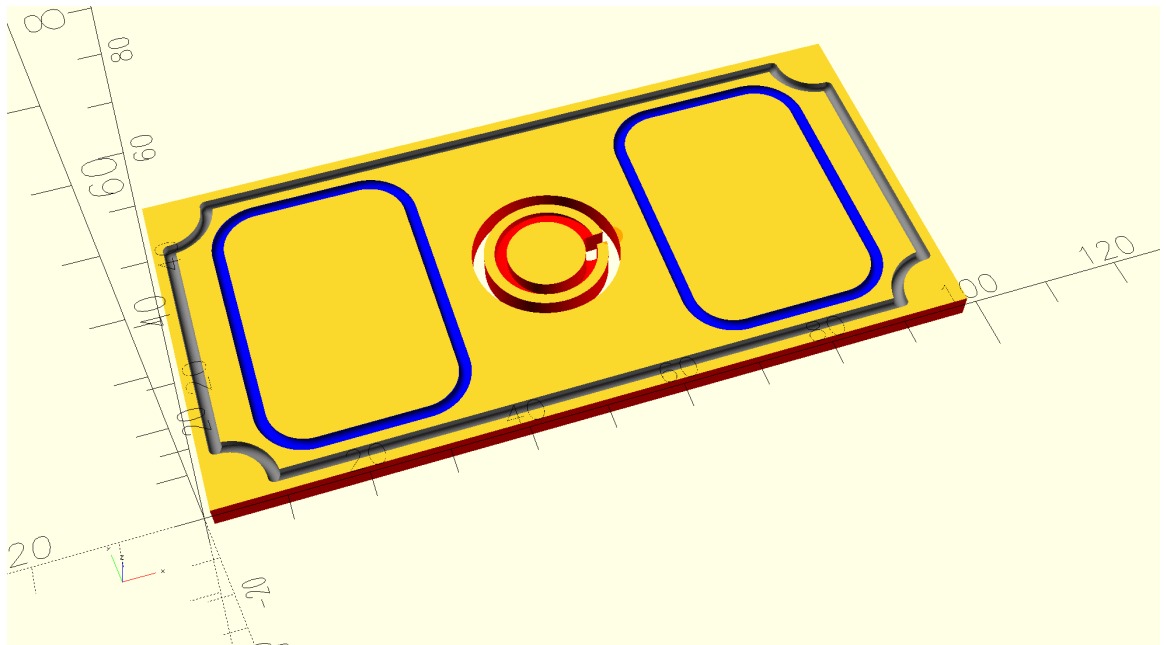
99 gcpcutdxfp
100 gcpcutdxfp gcp.setdxcolor("Red")
101 gcpcutdxfp
102 gcpcutdxfp gcp.rapid(stockXwidth/2 + radius, stockYheight/2, 0, "laser")
103 gcpcutdxfp
104 gcpcutdxfp gcp.toolchange(large_square_tool_num)
105 gcpcutdxfp
106 gcpcutdxfp gcp.cutcircleCC(stockXwidth/2, stockYheight/2, 0, -stockZthickness,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

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

109 gcpcutdxfp
110 gcpcutdxfp gcp.closedxf()
111 gcpcutdxfp
112 gcpcutdxfp gcp.stockandtoolpaths()

```

which creates the design:



and which allows an interactive usage in working up a design such as for lasercutting, and which incorporates an option to the `rapid(x,y,z)` command which simulates turning a laser off, repositioning, then powering up the laser after.

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
    /BOCPJPTMP
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)

```

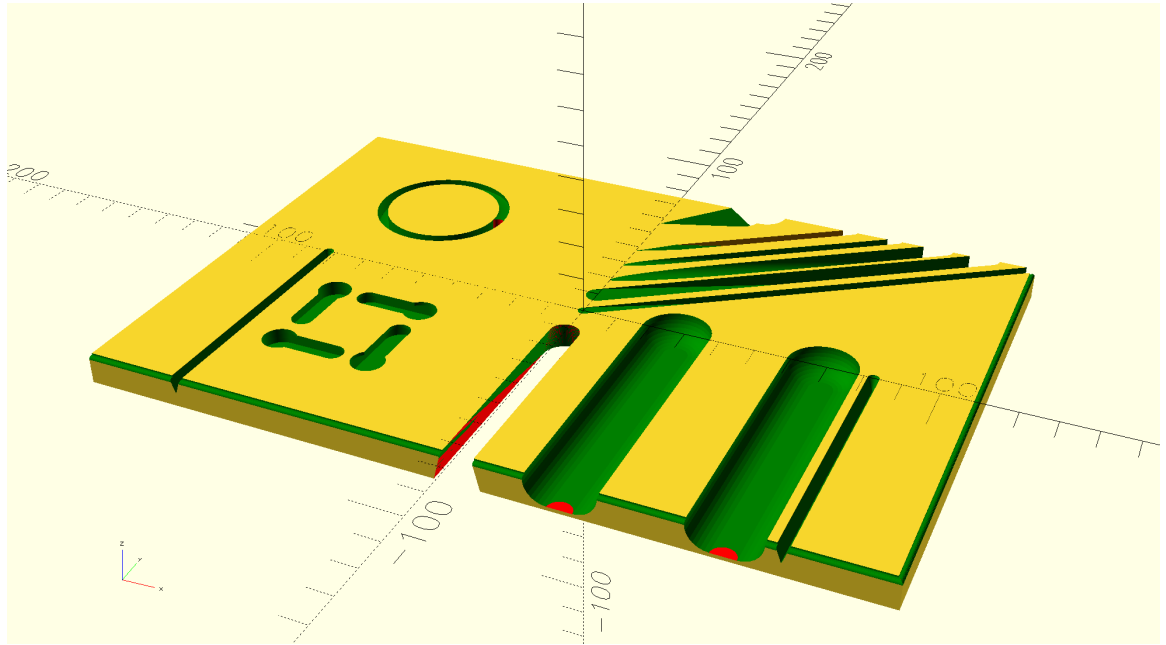


```

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/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 = 220;
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
175 gcptmpl rapidXY(-stockXwidth/4 + stockYheight/8, (stockYheight/4));
176 gcptmpl rapidZ(0);
177 gcptmpl
178 gcptmpl cutquarterCCNEdxf(xpos() - stockYheight/8, ypos() + stockYheight/8,
           -stockZthickness/4, stockYheight/8);
179 gcptmpl cutquarterCCNWdxf(xpos() - stockYheight/8, ypos() - stockYheight/8,
           -stockZthickness/2, stockYheight/8);
180 gcptmpl cutquarterCCSWdxf(xpos() + stockYheight/8, ypos() - stockYheight/8,
           -stockZthickness * 0.75, stockYheight/8);
181 gcptmpl //cutquarterCCSEdxf(xpos() + stockYheight/8, ypos() + stockYheight

```

```

        /8, -stockZthickness, stockYheight/8);
182 gcptmpl
183 gcptmpl movetosafeZ();
184 gcptmpl toolchange(201, 10000);
185 gcptmpl rapidXY(stockXwidth /2 -6.34, - stockYheight /2);
186 gcptmpl rapidZ(0);
187 gcptmpl //cutarcCW(180, 90, stockXwidth /2, -stockYheight/2, 6.34, -
        stockZthickness);
188 gcptmpl
189 gcptmpl movetosafeZ();
190 gcptmpl rapidXY(stockXwidth/2, -stockYheight/2);
191 gcptmpl rapidZ(0);
192 gcptmpl
193 gcptmpl //gcp.cutlinedxfgc(gcp.xpos(), gcp.ypos(), -stockZthickness);
194 gcptmpl
195 gcptmpl movetosafeZ();
196 gcptmpl toolchange(814, 10000);
197 gcptmpl rapidXY(0, -(stockYheight/2+12.7));
198 gcptmpl rapidZ(0);
199 gcptmpl
200 gcptmpl cutlinedxfgc(xpos(), ypos(), -stockZthickness);
201 gcptmpl cutlinedxfgc(xpos(), -12.7, -stockZthickness);
202 gcptmpl rapidXY(0, -(stockYheight/2+12.7));
203 gcptmpl
204 gcptmpl //rapidXY(stockXwidth/2-6.34, -stockYheight/2);
205 gcptmpl //rapidZ(0);
206 gcptmpl
207 gcptmpl //movetosafeZ();
208 gcptmpl //toolchange(374, 10000);
209 gcptmpl //rapidXY(-(stockXwidth/4 - stockXwidth /16), -(stockYheight/4 +
        stockYheight/16))
210 gcptmpl
211 gcptmpl //cutline(xpos(), ypos(), (stockZthickness/2) * -1);
212 gcptmpl //cutlinedxfgc(xpos() + stockYheight /9, ypos(), zpos());
213 gcptmpl //cutline(xpos() - stockYheight /9, ypos(), zpos());
214 gcptmpl //cutline(xpos(), ypos(), 0);
215 gcptmpl
216 gcptmpl movetosafeZ();
217 gcptmpl
218 gcptmpl toolchange(374, 10000);
219 gcptmpl rapidXY(-stockXwidth/4-stockXwidth/16, -(stockYheight/4+
        stockYheight/16))
220 gcptmpl //rapidXY(-(stockXwidth/4 - stockXwidth /16), -(stockYheight/4 +
        stockYheight/16))
221 gcptmpl rapidZ(0);
222 gcptmpl
223 gcptmpl cutline(xpos(), ypos(), (stockZthickness/2) * -1);
224 gcptmpl cutlinedxfgc(xpos() + stockYheight /9, ypos(), zpos());
225 gcptmpl cutline(xpos() - stockYheight /9, ypos(), zpos());
226 gcptmpl cutline(xpos(), ypos(), 0);
227 gcptmpl
228 gcptmpl rapidZ(retractheight);
229 gcptmpl rapidXY(-stockXwidth/4+stockXwidth/16, -(stockYheight/4+
        stockYheight/16));
230 gcptmpl rapidZ(0);
231 gcptmpl cutline(gcp.xpos(), gcp.ypos(), -stockZthickness/2);
232 gcptmpl cutlinedxfgc(gcp.xpos(), gcp.ypos()+stockYheight/9, gcp.zpos());
233 gcptmpl cutline(gcp.xpos(), gcp.ypos()-stockYheight/9, gcp.zpos());
234 gcptmpl cutline(gcp.xpos(), gcp.ypos(), 0);
235 gcptmpl
236 gcptmpl rapidZ(retractheight);
237 gcptmpl rapidXY(-stockXwidth/4+stockXwidth/16, -(stockYheight/4-
        stockYheight/8));
238 gcptmpl rapidZ(0);
239 gcptmpl cutline(gcp.xpos(), gcp.ypos(), -stockZthickness/2);
240 gcptmpl cutlinedxfgc(gcp.xpos()-stockYheight/9, gcp.ypos(), gcp.zpos());
241 gcptmpl cutline(gcp.xpos()+stockYheight/9, gcp.ypos(), gcp.zpos());
242 gcptmpl cutline(gcp.xpos(), gcp.ypos(), 0);
243 gcptmpl
244 gcptmpl rapidZ(retractheight);
245 gcptmpl rapidXY(-stockXwidth/4-stockXwidth/16, -(stockYheight/4-
        stockYheight/8));
246 gcptmpl rapidZ(0);
247 gcptmpl cutline(gcp.xpos(), gcp.ypos(), -stockZthickness/2);
248 gcptmpl cutlinedxfgc(gcp.xpos(), gcp.ypos()-stockYheight/9, gcp.zpos());
249 gcptmpl cutline(gcp.xpos(), gcp.ypos()+stockYheight/9, gcp.zpos());
250 gcptmpl cutline(gcp.xpos(), gcp.ypos(), 0);
251 gcptmpl

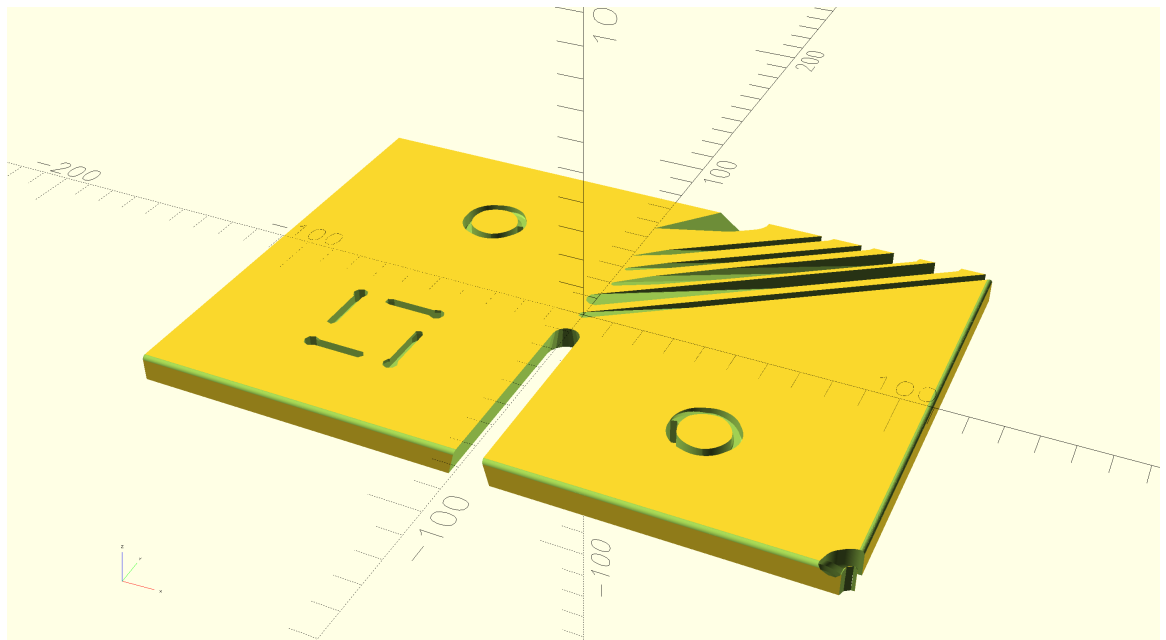
```

```

252 gcptmpl rapidZ(retractheight);
253 gcptmpl toolchange(45982, 10000);
254 gcptmpl rapidXY(stockXwidth/8, 0);
255 gcptmpl cutline(gcp.xpos(), gcp.ypos(), -(stockZthickness*7/8));
256 gcptmpl cutlinedxfgc(gcp.xpos(), -stockYheight/2, -(stockZthickness*7/8));
257 gcptmpl
258 gcptmpl rapidZ(retractheight);
259 gcptmpl toolchange(204, 10000);
260 gcptmpl rapidXY(stockXwidth*0.3125, 0);
261 gcptmpl cutline(gcp.xpos(), gcp.ypos(), -(stockZthickness*7/8));
262 gcptmpl cutlinedxfgc(gcp.xpos(), -stockYheight/2, -(stockZthickness*7/8));
263 gcptmpl
264 gcptmpl rapidZ(retractheight);
265 gcptmpl toolchange(502, 10000);
266 gcptmpl rapidXY(stockXwidth*0.375, 0);
267 gcptmpl cutline(gcp.xpos(), gcp.ypos(), -4.24);
268 gcptmpl cutlinedxfgc(gcp.xpos(), -stockYheight/2, -4.24);
269 gcptmpl
270 gcptmpl rapidZ(retractheight);
271 gcptmpl toolchange(13921, 10000);
272 gcptmpl rapidXY(-stockXwidth*0.375, 0);
273 gcptmpl cutline(gcp.xpos(), gcp.ypos(), -stockZthickness/2);
274 gcptmpl cutlinedxfgc(gcp.xpos(), -stockYheight/2, -stockZthickness/2);
275 gcptmpl
276 gcptmpl rapidZ(retractheight);
277 gcptmpl gcp.toolchange(56142, 10000);
278 gcptmpl gcp.rapidXY(-stockXwidth/2, -(stockYheight/2+0.508/2));
279 gcptmpl cutlineZgcfeed(-1.531, plunge);
280 gcptmpl //cutline(gcp.xpos(), gcp.ypos(), -1.531);
281 gcptmpl cutlinedxfgc(stockXwidth/2+0.508/2, -(stockYheight/2+0.508/2),
    -1.531);
282 gcptmpl
283 gcptmpl rapidZ(retractheight);
284 gcptmpl //#gcp.toolchange(56125, 10000)
285 gcptmpl cutlineZgcfeed(-1.531, plunge);
286 gcptmpl //toolpaths.append(gcp.cutline(gcp.xpos(), gcp.ypos(), -1.531))
287 gcptmpl cutlinedxfgc(stockXwidth/2+0.508/2, (stockYheight/2+0.508/2),
    -1.531);
288 gcptmpl
289 gcptmpl stockandtoolpaths();
290 gcptmpl //stockwotoolpaths();
291 gcptmpl //outputtoolpaths();
292 gcptmpl
293 gcptmpl //makecube(3, 2, 1);
294 gcptmpl
295 gcptmpl //instantiatecube();
296 gcptmpl
297 gcptmpl closegcodefile();
298 gcptmpl closedxffiles();
299 gcptmpl closedxffile();

```

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 4th additional axis may be worked up as a future option and supporting the work-around of two-sided (flip) machining by using an imported file as the Stock 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* (*gcpscad*) — which uses the Python file and which is included allowing it to access OpenSCAD variables for branching

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

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 Cutviewer

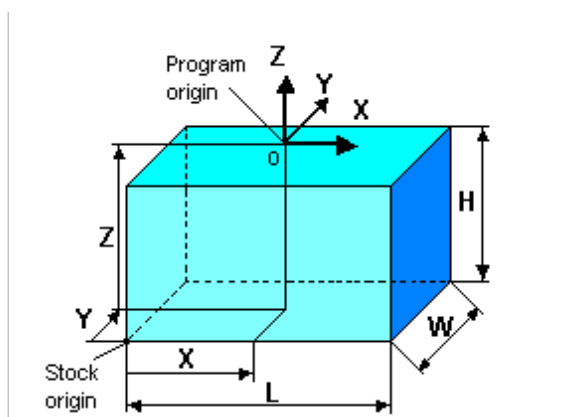
This problem space, showing the result of cutting stock using tooling in 3D has a number of tools addressing it, Camotics (formerly OpenSCAM) is an opensource option. Many tools simply create a wireframe preview such as <https://ncviewer.com/>. Cutviewer is a notable commercial program which has a unique approach centered on G-code where specially formatted comments fill in the dimensions needed for showing the 3D preview.

3.1.1 Stock size and placement

Setting the dimensions of the stock, and placing it in 3D space relative to the origin must be done very early in the G-code file.

The CutViewer comments are in the form:

(STOCK/BLOCK, Length, Width, Height, Origin X, Origin Y, Origin Z)

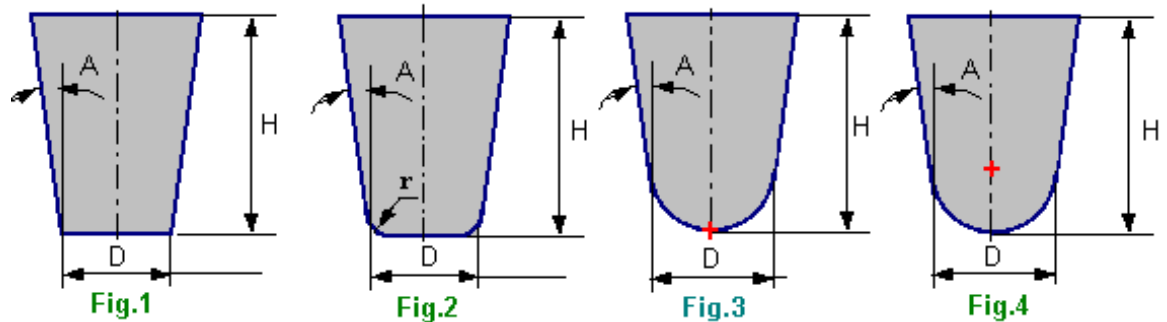


3.1.2 Tool Shapes

Cutviewer is unable to show tools which undercut, but other tool shapes are represented in a straight-forward and flexible fashion.

3.1.2.1 Tool/Mill (Square, radiused, ball-nose, and tapered-ball) The CutViewer values include:

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

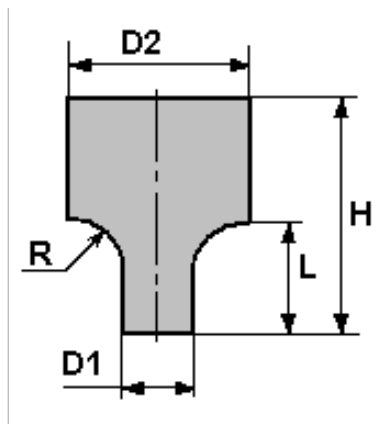


Note that it is possible to use these definitions for a wide variety of tooling, e.g., a Carbide 3D #301 V tool being represented as:

(TOOL/MILL, 0.10, 0.05, 6.35, 45.00)

3.1.2.2 Corner Rounding, (roundover) One notable tool option which cannot be supported using the Tool/Mill description is corner rounding/roundover tooling:

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



3.1.2.3 V shaped tooling (and variations) Cutviewer has multiple V shaped tooling definitions:

- ;TOOL/CHAMFER, Diameter, Point Angle, Height
- ;TOOL/CHAMFER, Diameter, Point Angle, Height, Chamfer Length (note that this is the definition of a flat-bottomed V tool)
- ;TOOL/DRILL, Diameter, Point Angle, Height
- ;TOOL/CDRILL, D1, A1, L, D2, A2, H

Since such tooling may be represented (albeit with a slight compromise which arguably is a nod to the real world) using the Tool/Mill definition from above, it seems unlikely that such tooling definitions will be supported.

3.2 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	dxf	gcode	cut	dxf	gcode
cut	cutline		cutlinegc	cutarc		cutarcgc
dxf	cutlinedxf	dxfline		cutarc dxf	dxfarc	
gcode	cutlinegc		linegc	cutarcgc		arcgc
	cutlinedxfgc			cutarc dxfgc		

Note that certain commands (dxflinegc, dxfarccgc, linegc, arccgc) 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 (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/

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

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

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

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

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

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

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

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

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

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

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

Lastly note that this style of programming was abandoned in favour of object-oriented dot notation for versions after v0.6 (see below) and that this technique was extended to class nested within another class.

3.2.1 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.

3.3 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


```

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

```

3.3.1 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:

```
108 gcpy      def xpos(self):
109 gcpy          return self.mpx
110 gcpy
111 gcpy      def ypos(self):
112 gcpy          return self.mpy
113 gcpy
114 gcpy      def zpos(self):
115 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 117 gcpy      def setxpos(self, newxpos):
118 gcpy          self.mpx = newxpos
119 gcpy
120 gcpy      def setypos(self, newypos):
121 gcpy          self.mpy = newypos
122 gcpy
123 gcpy      def setzpos(self, newzpos):
124 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.

3.3.2 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.

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.

toolmovement

rapids

toolpaths

3.3.2.1 setupstock 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 gcodepreview setupstock initializemachinestate() above), but the rewritten standalone version handles all necessary actions.

gcp.setupstock Since part of a class, it will be called as gcp.setupstock. It requires that the user set parameters for stock dimensions and so forth, and will create comments in the G-code (if generating that file is enabled) which incorporate the stock dimensions and its position relative to the zero as set relative to the stock.

```
126 gcpy      def setupstock(self, stockXwidth,
127 gcpy                      stockYheight,
128 gcpy                      stockZthickness,
129 gcpy                      zeroheight,
130 gcpy                      stockzero,
131 gcpy                      retractheight):
132 gcpy      """
133 gcpy      Set up blank/stock for material and position/zero.
134 gcpy
135 gcpy      Parameters
136 gcpy      -----
137 gcpy      stockXwidth : float
138 gcpy                      X extent/dimension
139 gcpy      stockYheight : float
140 gcpy                      Y extent/dimension
141 gcpy      stockZthickness : boolean
142 gcpy                      Z extent/dimension
143 gcpy      zeroheight : string
144 gcpy                      Top or Bottom, determines if Z extent will
                        be positive or negative
145 gcpy      stockzero : string
146 gcpy                      Lower-Left, Center-Left, Top-Left, Center,
                        determines XY position of stock
147 gcpy      retractheight : float
148 gcpy                      Distance which tool retracts above surface
                        of stock.
149 gcpy
150 gcpy      Returns
151 gcpy      -----
152 gcpy      none
153 gcpy      """
154 gcpy      self.stockXwidth = stockXwidth
155 gcpy      self.stockYheight = stockYheight
156 gcpy      self.stockZthickness = stockZthickness
157 gcpy      self.zeroheight = zeroheight
158 gcpy      self.stockzero = stockzero
159 gcpy      self.retractheight = retractheight
160 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.

```
162 gcpy      if self.zeroheight == "Top":
163 gcpy          if self.stockzero == "Lower-Left":
164 gcpy              self.stock = self.stock.translate([0, 0, -self.
                        stockZthickness])
165 gcpy          if self.generategcode == True:
166 gcpy              self.writegc("(stockMin:0.00mm,␣0.00mm,␣-", str
                        (self.stockZthickness), "mm)")
167 gcpy              self.writegc("(stockMax:", str(self.stockXwidth
                        ), "mm,␣", str(stockYheight), "mm,␣0.00mm)")
168 gcpy              self.writegc("(STOCK/BLOCK,␣", str(self.
                        stockXwidth), ",␣", str(self.stockYheight),
                        ",␣", str(self.stockZthickness), ",␣0.00,␣
                        0.00,␣", str(self.stockZthickness), ")")
169 gcpy          if self.stockzero == "Center-Left":
170 gcpy              self.stock = self.stock.translate([0, -stockYheight
                        / 2, -stockZthickness])
171 gcpy          if self.generategcode == True:
172 gcpy              self.writegc("(stockMin:0.00mm,␣-", str(self.
                        stockYheight/2), "mm,␣-", str(self.
                        stockZthickness), "mm)")
173 gcpy              self.writegc("(stockMax:", str(self.stockXwidth
                        ), "mm,␣", str(self.stockYheight/2), "mm,␣
                        0.00mm)")
174 gcpy              self.writegc("(STOCK/BLOCK,␣", str(self.
```

```

        stockXwidth), ",␣", str(self.stockYheight),
        ",␣", str(self.stockZthickness), ",␣0.00,␣",
        str(self.stockYheight/2), ",␣", str(self.
        stockZthickness), ")");
175 gcpy      if self.stockzero == "Top-Left":
176 gcpy      self.stock = self.stock.translate([0, -self.
        stockYheight, -self.stockZthickness])
177 gcpy      if self.generategcode == True:
178 gcpy      self.writegc("(stockMin:0.00mm,␣-", str(self.
        stockYheight), "mm,␣-", str(self.
        stockZthickness), "mm)")
179 gcpy      self.writegc("(stockMax:", str(self.stockXwidth
        ), "mm,␣0.00mm,␣0.00mm)")
180 gcpy      self.writegc("(STOCK/BLOCK,␣", str(self.
        stockXwidth), ",␣", str(self.stockYheight),
        ",␣", str(self.stockZthickness), ",␣0.00,␣",
        str(self.stockYheight), ",␣", str(self.
        stockZthickness), ")")
181 gcpy      if self.stockzero == "Center":
182 gcpy      self.stock = self.stock.translate([-self.
        stockXwidth / 2, -self.stockYheight / 2, -self.
        stockZthickness])
183 gcpy      if self.generategcode == True:
184 gcpy      self.writegc("(stockMin:␣-", str(self.
        stockXwidth/2), ",␣-", str(self.stockYheight
        /2), "mm,␣-", str(self.stockZthickness), "mm
        )")
185 gcpy      self.writegc("(stockMax:", str(self.stockXwidth
        /2), "mm,␣", str(self.stockYheight/2), "mm,␣
        0.00mm)")
186 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), ")")
187 gcpy      if self.zeroheight == "Bottom":
188 gcpy      if self.stockzero == "Lower-Left":
189 gcpy      self.stock = self.stock.translate([0, 0, 0])
190 gcpy      if self.generategcode == True:
191 gcpy      self.writegc("(stockMin:0.00mm,␣0.00mm,␣0.00mm
        )")
192 gcpy      self.writegc("(stockMax:", str(self.
        stockXwidth), "mm,␣", str(self.stockYheight
        ), "mm,␣", str(self.stockZthickness), "mm)"
        )
193 gcpy      self.writegc("(STOCK/BLOCK,␣", str(self.
        stockXwidth), ",␣", str(self.stockYheight),
        ",␣", str(self.stockZthickness), ",␣0.00,␣
        0.00,␣0.00)")
194 gcpy      if self.stockzero == "Center-Left":
195 gcpy      self.stock = self.stock.translate([0, -self.
        stockYheight / 2, 0])
196 gcpy      if self.generategcode == True:
197 gcpy      self.writegc("(stockMin:0.00mm,␣-", str(self.
        stockYheight/2), "mm,␣0.00mm)")
198 gcpy      self.writegc("(stockMax:", str(self.stockXwidth
        ), "mm,␣", str(self.stockYheight/2), "mm,␣-",
        , str(self.stockZthickness), "mm)")
199 gcpy      self.writegc("(STOCK/BLOCK,␣", str(self.
        stockXwidth), ",␣", str(self.stockYheight),
        ",␣", str(self.stockZthickness), ",␣0.00,␣",
        str(self.stockYheight/2), ",␣0.00mm)");
200 gcpy      if self.stockzero == "Top-Left":
201 gcpy      self.stock = self.stock.translate([0, -self.
        stockYheight, 0])
202 gcpy      if self.generategcode == True:
203 gcpy      self.writegc("(stockMin:0.00mm,␣-", str(self.
        stockYheight), "mm,␣0.00mm)")
204 gcpy      self.writegc("(stockMax:", str(self.stockXwidth
        ), "mm,␣0.00mm,␣", str(self.stockZthickness)
        , "mm)")
205 gcpy      self.writegc("(STOCK/BLOCK,␣", str(self.
        stockXwidth), ",␣", str(self.stockYheight),
        ",␣", str(self.stockZthickness), ",␣0.00,␣",
        str(self.stockYheight), ",␣0.00)")
206 gcpy      if self.stockzero == "Center":
207 gcpy      self.stock = self.stock.translate([-self.

```



```

stockXwidth / 2, -self.stockYheight / 2, 0])
208 gcpy      if self.generategcode == True:
209 gcpy          self.writegc("(stockMin:␣-", str(self.
stockXwidth/2), "␣-", str(self.stockYheight
/2), "mm,␣0.00mm)")
210 gcpy          self.writegc("(stockMax:", str(self.stockXwidth
/2), "mm,␣", str(self.stockYheight/2), "mm,␣
", str(self.stockZthickness), "mm)")
211 gcpy          self.writegc("(STOCK/BLOCK,␣", str(self.
stockXwidth), "␣", str(self.stockYheight),
"␣", str(self.stockZthickness), "␣", str(
self.stockXwidth/2), "␣", str(self.
stockYheight/2), "␣0.00)")
212 gcpy      if self.generategcode == True:
213 gcpy          self.writegc("G90");
214 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:

```

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

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

```

216 gcpy      def setupcuttingarea(self, sizeX, sizeY, sizeZ, extentleft,
extentfb, extentd):
217 gcpy #          self.initializemachinestate()
218 gcpy          c=cube([sizeX,sizeY,sizeZ])
219 gcpy          c = c.translate([extentleft,extentfb,extentd])
220 gcpy          self.stock = c
221 gcpy          self.toolpaths = []
222 gcpy          return c
```

3.3.3 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:

```

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

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

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

```

227 gcpy      def addtostock(self, stockXwidth, stockYheight, stockZthickness
,
228 gcpy          ,
229 gcpy          shiftX = 0,
230 gcpy          shiftY = 0,
231 gcpy          shiftZ = 0):
232 gcpy          addedpart = cube([stockXwidth, stockYheight,
stockZthickness])
233 gcpy          addedpart = addedpart.translate([shiftX, shiftY, shiftZ])
234 gcpy          self.stock = self.stock.union(addedpart)
```

the OpenSCAD module is a descriptor as expected:

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

3.4 Tools and Shapes 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 (rapids, toolpaths) 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 and the specifics of the current tool to 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).

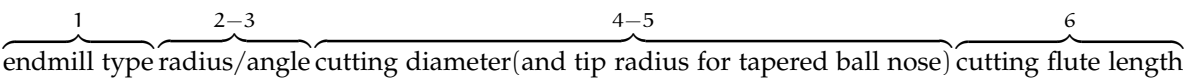
Note that it will also be necessary to write out a tool description compatible with the program CutViewer as a G-code comment so that it may be used as a 3D previewer for the G-code for tool changes in G-code. Several forms are available as described below.

3.4.1 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 - 1/32nd
 - 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.

```
235 gcpy      def currenttoolnumber(self):
236 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 3.4.1.1 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.

settoolparameters or when making a 3D file, settoolparameters and a second version which processes a toolchange
toolchange when presented with a tool number, toolchange (it may be that the latter will be set up to call the
former).

```
238 gcpy      def settoolparameters(self, tooltype, first, second, third,
239 gcpy      fourth, length = 0):
240 gcpy      diameter = first
241 gcpy      cornerradius = second
242 gcpy      height = third
243 gcpy      taperangle = fourth
244 gcpy      if cornerradius = 0:
245 gcpy      #M6T122 (TOOL/MILL,0.80, 0.00, 1.59, 0.00)
246 gcpy      #M6T112 (TOOL/MILL,1.59, 0.00, 6.35, 0.00)
247 gcpy      #M6T102 (TOOL/MILL,3.17, 0.00, 12.70, 0.00)
248 gcpy      #M6T201 (TOOL/MILL,6.35, 0.00, 19.05, 0.00)
249 gcpy      #M6T205 (TOOL/MILL,6.35, 0.00, 25.40, 0.00)
250 gcpy      #M6T251 (TOOL/MILL,6.35, 0.00, 19.05, 0.00)
251 gcpy      #M6T322 (TOOL/MILL,6.35, 0.00, 19.05, 0.00)
252 gcpy      #M6T324 (TOOL/MILL,6.35, 0.00, 22.22, 0.00)
253 gcpy      #M6T326 (TOOL/MILL,3.17, 0.00, 12.70, 0.00)
254 gcpy      #M6T602 (TOOL/MILL,25.40, 0.00, 9.91, 0.00)
255 gcpy      #M6T603 (TOOL/MILL,25.40, 0.00, 9.91, 0.00)
256 gcpy      #M6T274 (TOOL/MILL,3.17, 0.00, 12.70, 0.00)
```

```
257 gcpy #M6T278 (TOOL/MILL,6.35, 0.00, 19.05, 0.00)
258 gcpy #M6T282 (TOOL/MILL,2.00, 0.00, 6.35, 0.00)
259 gcpy         self.endmilltype = "square"
260 gcpy         self.diameter = diameter
261 gcpy         self.flute = height
262 gcpy         self.shaftdiameter = diameter
263 gcpy         self.shaftheight = height
264 gcpy         self.shaftlength = height
265 gcpy #
266 gcpy         elif cornerradius > 0 and taperangle = 0:
267 gcpy #M6T121 (TOOL/MILL,0.80, 0.40, 1.59, 0.00)
268 gcpy #M6T111 (TOOL/MILL,1.59, 0.79, 6.35, 0.00)
269 gcpy #M6T101 (TOOL/MILL,3.17, 1.59, 12.70, 0.00)
270 gcpy #M6T202 (TOOL/MILL,6.35, 3.17, 19.05, 0.00)
271 gcpy #M6T325 (TOOL/MILL,6.35, 3.17, 25.40, 0.00)
272 gcpy         self.endmilltype = "ball"
273 gcpy         self.diameter = diameter
274 gcpy         self.flute = height
275 gcpy         self.shaftdiameter = diameter
276 gcpy         self.shaftheight = height
277 gcpy         self.shaftlength = height
278 gcpy #
279 gcpy         elif taperangle > 0:
280 gcpy #M6T301 (TOOL/MILL,0.10, 0.05, 6.35, 45.00)
281 gcpy #M6T302 (TOOL/MILL,0.10, 0.05, 6.35, 30.00)
282 gcpy #M6T327 (TOOL/MILL,0.10, 0.05, 23.39, 30.00)
283 gcpy         self.endmilltype = "V"
284 gcpy         self.diameter = Tan(taperangle / 2) * height
285 gcpy         self.flute = height
286 gcpy         self.angle = taperangle
287 gcpy         self.shaftdiameter = Tan(taperangle / 2) * height
288 gcpy         self.shaftheight = height
289 gcpy         self.shaftlength = height
290 gcpy #
291 gcpy         elif tooltype = "chamfer":
292 gcpy         tipdiameter = first
293 gcpy         radius = second
294 gcpy         height = third
295 gcpy         taperangle = fourth
```

toolchange toolchange

```
245 gcpy      def toolchange(self, tool_number, speed = 10000):
246 gcpy          self.currenttoolnum = tool_number
247 gcpy
248 gcpy          if (self.generategcode == True):
249 gcpy              self.writegc("(Toolpath)")
250 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 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.

toolmovement

3.4.1.2 Square (including O-flute) The simplest sort of tool, they are defined as a cylinder.

```
252 gcpy      if (tool_number == 201): #201/251/322 (Amana 46202-K) ==
253 gcpy          100047
254 gcpy          self.writegc("(TOOL/MILL,▯6.35,▯0.00,▯0.00,▯0.00)")
255 gcpy          self.endmilltype = "square"
256 gcpy          self.diameter = 6.35
257 gcpy          self.flute = 19.05
258 gcpy          self.shaftdiameter = 6.35
259 gcpy          self.shaftheight = 19.05
260 gcpy          self.shaftlength = 20.0
261 gcpy          self.toolnumber = "100047"
262 gcpy      elif (tool_number == 102): #102/326 == 100036
263 gcpy          self.writegc("(TOOL/MILL,▯3.175,▯0.00,▯0.00,▯0.00)")
264 gcpy          self.endmilltype = "square"
265 gcpy          self.diameter = 3.175
266 gcpy          self.flute = 12.7
267 gcpy          self.shaftdiameter = 3.175
```

```
267 gcpy          self.shaftheight = 12.7
268 gcpy          self.shaftlength = 20.0
269 gcpy          self.toolnumber = 100036
270 gcpy          elif (tool_number == 112): #112 == 100024
271 gcpy            self.writegc("(T00L/MILL,␣1.5875,␣0.00,␣0.00,␣0.00)")
272 gcpy            self.endmilltype = "square"
273 gcpy            self.diameter = 1.5875
274 gcpy            self.flute = 6.35
275 gcpy            self.shaftdiameter = 3.175
276 gcpy            self.shaftheight = 6.35
277 gcpy            self.shaftlength = 12.0
278 gcpy            self.toolnumber = "100024"
279 gcpy          elif (tool_number == 122): #122 == 100012
280 gcpy            self.writegc("(T00L/MILL,␣0.79375,␣0.00,␣0.00,␣0.00)")
281 gcpy            self.endmilltype = "square"
282 gcpy            self.diameter = 0.79375
283 gcpy            self.flute = 1.5875
284 gcpy            self.shaftdiameter = 3.175
285 gcpy            self.shaftheight = 1.5875
286 gcpy            self.shaftlength = 12.0
287 gcpy            self.toolnumber = "100012"
288 gcpy          elif (tool_number == 324): #324 (Amana 46170-K) == 100048
289 gcpy            self.writegc("(T00L/MILL,␣6.35,␣0.00,␣0.00,␣0.00)")
290 gcpy            self.endmilltype = "square"
291 gcpy            self.diameter = 6.35
292 gcpy            self.flute = 22.225
293 gcpy            self.shaftdiameter = 6.35
294 gcpy            self.shaftheight = 22.225
295 gcpy            self.shaftlength = 20.0
296 gcpy            self.toolnumber = "100048"
297 gcpy          elif (tool_number == 205): #205 == 100048
298 gcpy            self.writegc("(T00L/MILL,␣6.35,␣0.00,␣0.00,␣0.00)")
299 gcpy            self.endmilltype = "square"
300 gcpy            self.diameter = 6.35
301 gcpy            self.flute = 25.4
302 gcpy            self.shaftdiameter = 6.35
303 gcpy            self.shaftheight = 25.4
304 gcpy            self.shaftlength = 20.0
305 gcpy            self.toolnumber = "100048"
306 gcpy          #
```

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

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

3.4.1.3 Ball-nose (including tapered-ball) The `elifs` continue with ball-nose and tapered-ball tooling which are defined as one would expect by spheres and cylinders. Note that the Cutviewer definition of a the measurement point of a tool being at the center is not yet set up — potentially it opens up greatly simplified toolpath calculations and may be implemented in a future version.

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

3.4.1.4 V Note that one V tool is described as an Engraver in Carbide Create. While CutViewer has specialty Tool/chamfer and Tool/drill parameters, it is possible to describe a V tool as a Tool/mill (using a very small tip radius).

```
381 gcpy      elif (tool_number == 301): #301 == 390074
382 gcpy      self.writegc("(T00L/MILL,␣0.10,␣0.05,␣6.35,␣45.00)")
383 gcpy      self.endmilltype = "V"
384 gcpy      self.diameter = 12.7
385 gcpy      self.flute = 6.35
386 gcpy      self.angle = 90
387 gcpy      self.shaftdiameter = 6.35
388 gcpy      self.shaftheight = 6.35
389 gcpy      self.shaftlength = 20.0
390 gcpy      self.toolnumber = "390074"
391 gcpy      elif (tool_number == 302): #302 == 360071
392 gcpy      self.writegc("(T00L/MILL,␣0.10,␣0.05,␣6.35,␣30.00)")
393 gcpy      self.endmilltype = "V"
394 gcpy      self.diameter = 12.7
395 gcpy      self.flute = 11.067
396 gcpy      self.angle = 60
397 gcpy      self.shaftdiameter = 6.35
398 gcpy      self.shaftheight = 11.067
399 gcpy      self.shaftlength = 20.0
400 gcpy      self.toolnumber = "360071"
401 gcpy      elif (tool_number == 390): #390 == 390032
402 gcpy      self.writegc("(T00L/MILL,␣0.03,␣0.00,␣1.5875,␣45.00)")
403 gcpy      self.endmilltype = "V"
404 gcpy      self.diameter = 3.175
```



```
405 gcpy          self.flute = 1.5875
406 gcpy          self.angle = 90
407 gcpy          self.shaftdiameter = 3.175
408 gcpy          self.shaftheight = 1.5875
409 gcpy          self.shaftlength = 20.0
410 gcpy          self.toolnumber = "390032"
411 gcpy          elif (tool_number == 327): #327 (Amana RC-1148) == 360098
412 gcpy          self.writegc("(T00L/MILL,␣0.03,␣0.00,␣13.4874,␣30.00)")
413 gcpy          self.endmilltype = "V"
414 gcpy          self.diameter = 25.4
415 gcpy          self.flute = 22.134
416 gcpy          self.angle = 60
417 gcpy          self.shaftdiameter = 6.35
418 gcpy          self.shaftheight = 22.134
419 gcpy          self.shaftlength = 20.0
420 gcpy          self.toolnumber = "360098"
421 gcpy          elif (tool_number == 323): #323 == 330041 30 degree V Amana
422 gcpy          , 45771-K
423 gcpy          self.writegc("(T00L/MILL,␣0.10,␣0.05,␣11.18,␣15.00)")
424 gcpy          self.endmilltype = "V"
425 gcpy          self.diameter = 6.35
426 gcpy          self.flute = 11.849
427 gcpy          self.angle = 30
428 gcpy          self.shaftdiameter = 6.35
429 gcpy          self.shaftheight = 11.849
430 gcpy          self.shaftlength = 20.0
431 gcpy          self.toolnumber = "330041"
431 gcpy #
```

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

```
432 gcpy          elif (tool_number == 374): #374 == 906043
433 gcpy          self.writegc("(T00L/MILL,␣9.53,␣0.00,␣3.17,␣0.00)")
434 gcpy          self.endmilltype = "keyhole"
435 gcpy          self.diameter = 9.525
436 gcpy          self.flute = 3.175
437 gcpy          self.radius = 6.35
438 gcpy          self.shaftdiameter = 6.35
439 gcpy          self.shaftheight = 3.175
440 gcpy          self.shaftlength = 20.0
441 gcpy          self.toolnumber = "906043"
442 gcpy          elif (tool_number == 375): #375 == 906053
443 gcpy          self.writegc("(T00L/MILL,␣9.53,␣0.00,␣3.17,␣0.00)")
444 gcpy          self.endmilltype = "keyhole"
445 gcpy          self.diameter = 9.525
446 gcpy          self.flute = 3.175
447 gcpy          self.radius = 8
448 gcpy          self.shaftdiameter = 6.35
449 gcpy          self.shaftheight = 3.175
450 gcpy          self.shaftlength = 20.0
451 gcpy          self.toolnumber = "906053"
452 gcpy          elif (tool_number == 376): #376 == 907040
453 gcpy          self.writegc("(T00L/MILL,␣12.7,␣0.00,␣4.77,␣0.00)")
454 gcpy          self.endmilltype = "keyhole"
455 gcpy          self.diameter = 12.7
456 gcpy          self.flute = 4.7625
457 gcpy          self.radius = 6.35
458 gcpy          self.shaftdiameter = 6.35
459 gcpy          self.shaftheight = 4.7625
460 gcpy          self.shaftlength = 20.0
461 gcpy          self.toolnumber = "907040"
462 gcpy          elif (tool_number == 378): #378 == 907050
463 gcpy          self.writegc("(T00L/MILL,␣12.7,␣0.00,␣4.77,␣0.00)")
464 gcpy          self.endmilltype = "keyhole"
465 gcpy          self.diameter = 12.7
466 gcpy          self.flute = 4.7625
467 gcpy          self.radius = 8
468 gcpy          self.shaftdiameter = 6.35
469 gcpy          self.shaftheight = 4.7625
470 gcpy          self.shaftlength = 20.0
471 gcpy          self.toolnumber = "907050"
472 gcpy #
```

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

```

473 gcpy          elif (tool_number == 45981): #45981 == 445981
474 gcpy #Amana Carbide Tipped Bowl & Tray 1/8 Radius x 1/2 Dia x 1/2 x 1/4
            Inch Shank
475 gcpy          self.writegc("(T00L/MILL,0.03,␣0.00,␣10.00,␣30.00)")
476 gcpy          self.writegc("(T00L/MILL,␣15.875,␣6.35,␣19.05,␣0.00)")
477 gcpy          self.endmilltype = "bowl"
478 gcpy          self.diameter = 12.7
479 gcpy          self.flute = 12.7
480 gcpy          self.radius = 3.175
481 gcpy          self.shaftdiameter = 6.35
482 gcpy          self.shaftheight = 12.7
483 gcpy          self.shaftlength = 20.0
484 gcpy          self.toolnumber = "445981"
485 gcpy          elif (tool_number == 45982):#0.507/2, 4.509
486 gcpy          self.writegc("(T00L/MILL,␣15.875,␣6.35,␣19.05,␣0.00)")
487 gcpy          self.endmilltype = "bowl"
488 gcpy          self.diameter = 19.05
489 gcpy          self.flute = 15.875
490 gcpy          self.radius = 6.35
491 gcpy          self.shaftdiameter = 6.35
492 gcpy          self.shaftheight = 15.875
493 gcpy          self.shaftlength = 20.0
494 gcpy          self.toolnumber = "445982"
495 gcpy #          elif (tool_number == 1370): #1370 == 401370
496 gcpy #Whiteside Bowl & Tray Bit 1/4"SH, 1/8"R, 7/16"CD (5/16" cutting
            flute length)
497 gcpy          self.writegc("(T00L/MILL,␣11.1125,␣8,␣3.175,␣0.00)")
498 gcpy          self.endmilltype = "bowl"
499 gcpy          self.diameter = 11.1125
500 gcpy          self.flute = 8
501 gcpy          self.radius = 3.175
502 gcpy          self.shaftdiameter = 6.35
503 gcpy          self.shaftheight = 8
504 gcpy          self.shaftlength = 20.0
505 gcpy          self.toolnumber = "401370"
506 gcpy #          elif (tool_number == 1372): #1372/45982 == 401372
507 gcpy #Whiteside Bowl & Tray Bit 1/4"SH, 1/4"R, 3/4"CD (5/8" cutting
            flute length)
508 gcpy #Amana Carbide Tipped Bowl & Tray 1/4 Radius x 3/4 Dia x 5/8 x 1/4
            Inch Shank
509 gcpy          self.writegc("(T00L/MILL,␣19.5,␣15.875,␣6.35,␣0.00)")
510 gcpy          self.endmilltype = "bowl"
511 gcpy          self.diameter = 19.5
512 gcpy          self.flute = 15.875
513 gcpy          self.radius = 6.35
514 gcpy          self.shaftdiameter = 6.35
515 gcpy          self.shaftheight = 15.875
516 gcpy          self.shaftlength = 20.0
517 gcpy          self.toolnumber = "401372"
518 gcpy #

```

3.4.1.7 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.

```

519 gcpy          elif (tool_number == 501): #501 == 530131
520 gcpy          self.writegc("(T00L/MILL,0.03,␣0.00,␣10.00,␣30.00)")
521 gcpy #          self.currenttoolshape = self.toolshapes("tapered ball
            ", 3.175, 5.561, 30, 0.254)
522 gcpy          self.endmilltype = "tapered␣ball"
523 gcpy          self.diameter = 3.175
524 gcpy          self.flute = 5.561
525 gcpy          self.angle = 30
526 gcpy          self.tip = 0.254
527 gcpy          self.shaftdiameter = 3.175
528 gcpy          self.shaftheight = 5.561
529 gcpy          self.shaftlength = 10.0
530 gcpy          self.toolnumber = "530131"
531 gcpy          elif (tool_number == 502): #502 == 540131
532 gcpy          self.writegc("(T00L/MILL,0.03,␣0.00,␣10.00,␣20.00)")
533 gcpy #          self.currenttoolshape = self.toolshapes("tapered ball
            ", 3.175, 4.117, 40, 0.254)
534 gcpy          self.endmilltype = "tapered␣ball"

```

```

535 gcpy                self.diameter = 3.175
536 gcpy                self.flute = 4.117
537 gcpy                self.angle = 40
538 gcpy                self.tip = 0.254
539 gcpy                self.shaftdiameter = 3.175
540 gcpy                self.shaftheight = 4.117
541 gcpy                self.shaftlength = 10.0
542 gcpy                self.toolnumber = "540131"
543 gcpy #                elif (tool_number == 204):#
544 gcpy #                    self.writegc("(")
545 gcpy #                    self.currenttoolshape = self.tapered_ball(1.5875,
6.35, 38.1, 3.6)
546 gcpy #                elif (tool_number == 304):#
547 gcpy #                    self.writegc("(")
548 gcpy #                    self.currenttoolshape = self.tapered_ball(3.175, 6.35,
38.1, 2.4)
549 gcpy #

```

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

```

550 gcpy                elif (tool_number == 56125):#0.508/2, 1.531 56125 == 603042
551 gcpy                self.writegc("(TOOL/CRMILL, 0.508, 6.35, 3.175, 7.9375,
3.175)")
552 gcpy                self.endmilltype = "roundover"
553 gcpy                self.tip = 0.508
554 gcpy                self.diameter = 6.35 - self.tip
555 gcpy                self.flute = 8 - self.tip
556 gcpy                self.radius = 3.175 - self.tip
557 gcpy                self.shaftdiameter = 6.35
558 gcpy                self.shaftheight = 8
559 gcpy                self.shaftlength = 10.0
560 gcpy                self.toolnumber = "603042"
561 gcpy                elif (tool_number == 56142):#0.508/2, 2.921 56142 == 602032
562 gcpy                self.writegc("(TOOL/CRMILL, 0.508, 3.571875, 1.5875,
5.55625, 1.5875)")
563 gcpy                self.endmilltype = "roundover"
564 gcpy                self.tip = 0.508
565 gcpy                self.diameter = 3.175 - self.tip
566 gcpy                self.flute = 4.7625 - self.tip
567 gcpy                self.radius = 1.5875 - self.tip
568 gcpy                self.shaftdiameter = 3.175
569 gcpy                self.shaftheight = 4.7625
570 gcpy                self.shaftlength = 10.0
571 gcpy                self.toolnumber = "602032"
572 gcpy #                elif (tool_number == 312):#1.524/2, 3.175
573 gcpy #                    self.writegc("(TOOL/CRMILL, Diameter1, Diameter2,
Radius, Height, Length)")
574 gcpy #                elif (tool_number == 1568):#0.507/2, 4.509 1568 == 603032
575 gcpy ##FIX                self.writegc("(TOOL/CRMILL, 0.17018, 9.525,
4.7625, 12.7, 4.7625)")
576 gcpy ##                self.currenttoolshape = self.toolshapes("roundover",
3.175, 6.35, 3.175, 0.396875)
577 gcpy #                self.endmilltype = "roundover"
578 gcpy #                self.diameter = 3.175
579 gcpy #                self.flute = 6.35
580 gcpy #                self.radius = 3.175
581 gcpy #                self.tip = 0.396875
582 gcpy #                self.toolnumber = "603032"
583 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
584 gcpy #                elif (tool_number == 1570):#0.507/2, 4.509 1570 == 600002
?!!
585 gcpy #                self.writegc("(TOOL/CRMILL, 0.17018, 9.525, 4.7625,
12.7, 4.7625)")
586 gcpy ##                self.currenttoolshape = self.toolshapes("roundover",
4.7625, 9.525, 4.7625, 0.396875)
587 gcpy #                self.endmilltype = "roundover"
588 gcpy #                self.diameter = 4.7625
589 gcpy #                self.flute = 9.525
590 gcpy #                self.radius = 4.7625
591 gcpy #                self.tip = 0.396875
592 gcpy #                self.toolnumber = "600002"
593 gcpy #                elif (tool_number == 1572): #1572 = 604042
594 gcpy ##FIX                self.writegc("(TOOL/CRMILL, 0.17018, 9.525,
4.7625, 12.7, 4.7625)")

```

```

595 gcpy ##          self.currenttoolshape = self.toolshapes("roundover",
6.35, 12.7, 6.35, 0.396875)
596 gcpy #          self.endmilltype = "roundover"
597 gcpy #          self.diameter = 6.35
598 gcpy #          self.flute = 12.7
599 gcpy #          self.radius = 6.35
600 gcpy #          self.tip = 0.396875
601 gcpy #          self.toolnumber = "604042"
602 gcpy #          elif (tool_number == 1574): #1574 == 600062
603 gcpy ##FIX          self.writegc("(TOOL/CRMILL, 0.17018, 9.525,
4.7625, 12.7, 4.7625)")
604 gcpy ##          self.currenttoolshape = self.toolshapes("roundover",
9.525, 19.5, 9.515, 0.396875)
605 gcpy #          self.endmilltype = "roundover"
606 gcpy #          self.diameter = 9.525
607 gcpy #          self.flute = 19.5
608 gcpy #          self.radius = 9.515
609 gcpy #          self.tip = 0.396875
610 gcpy #          self.toolnumber = "600062"
611 gcpy #

```

3.4.1.9 Dovetails Unfortunately, tools which support undercuts such as dovetails are not supported by many CAM tools including Carbide Create and 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).

```

612 gcpy          elif (tool_number == 814): #814 == 814071
613 gcpy #Item 18J1607, 1/2" 14ř Dovetail Bit, 8mm shank
614 gcpy          self.writegc("(TOOL/MILL, 12.7, 6.367, 12.7, 0.00)")
615 gcpy          #          dt_bottomdiameter, dt_topdiameter, dt_height, dt_angle
)
616 gcpy          #          https://www.leevalley.com/en-us/shop/tools/power-tool-
accessories/router-bits/30172-dovetail-bits?item=18J1607
617 gcpy #          self.currenttoolshape = self.toolshapes("dovetail",
12.7, 12.7, 14)
618 gcpy          self.endmilltype = "dovetail"
619 gcpy          self.diameter = 12.7
620 gcpy          self.flute = 12.7
621 gcpy          self.angle = 14
622 gcpy          self.toolnumber = "814071"
623 gcpy          elif (tool_number == 808079): #45828 == 808071
624 gcpy          self.writegc("(TOOL/MILL, 12.7, 6.816, 20.95, 0.00)")
625 gcpy          #          http://www.amanatool.com/45828-carbide-tipped-dovetail
-8-deg-x-1-2-dia-x-825-x-1-4-inch-shank.html
626 gcpy #          self.currenttoolshape = self.toolshapes("dovetail",
12.7, 20.955, 8)
627 gcpy          self.endmilltype = "dovetail"
628 gcpy          self.diameter = 12.7
629 gcpy          self.flute = 20.955
630 gcpy          self.angle = 8
631 gcpy          self.toolnumber = "808071"
632 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:

```
48 gcpscad module toolchange(tool_number, speed){
49 gcpscad     gcp.toolchange(tool_number, speed);
50 gcpscad }
```

For example:

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

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

3.4.1.10 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.

```
633 gcpy          self.writegc("M6T", str(tool_number))
634 gcpy          self.writegc("M03S", str(speed))
```

3.4.2 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
G0X0Y8
S1000
G1X100F800
S0
M5 S0
M3 S0
S0
G0X0Y4
S1000
G1X100F600
S0
M5 S0
M3 S0
S0
G0X0Y0
S1000
G1X100F400
S0
M5 S0
```

3.5 Shapes and tool movement

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

rapid... 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.

3.5.0.1 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.

3.5.1 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.

3.5.2 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.

```
636 gcpy      def setcolor(self,
637 gcpy                               cutcolor = "green",
638 gcpy                               rapidcolor = "orange",
639 gcpy                               shaftcolor = "red"):
640 gcpy      self.cutcolor = cutcolor
641 gcpy      self.rapidcolor = rapidcolor
642 gcpy      self.shaftcolor = shaftcolor
```

The possible colors are those of 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.

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

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)

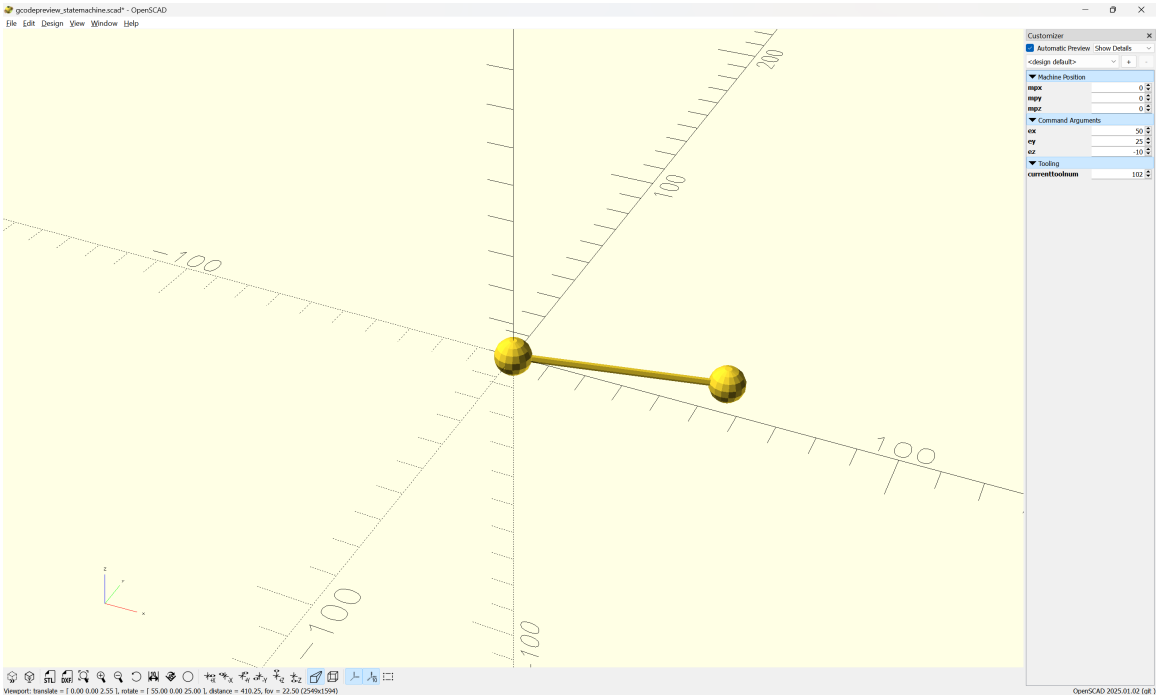
possible to color-code the shaft which may afford indication of instances of it rubbing against the stock.

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.

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

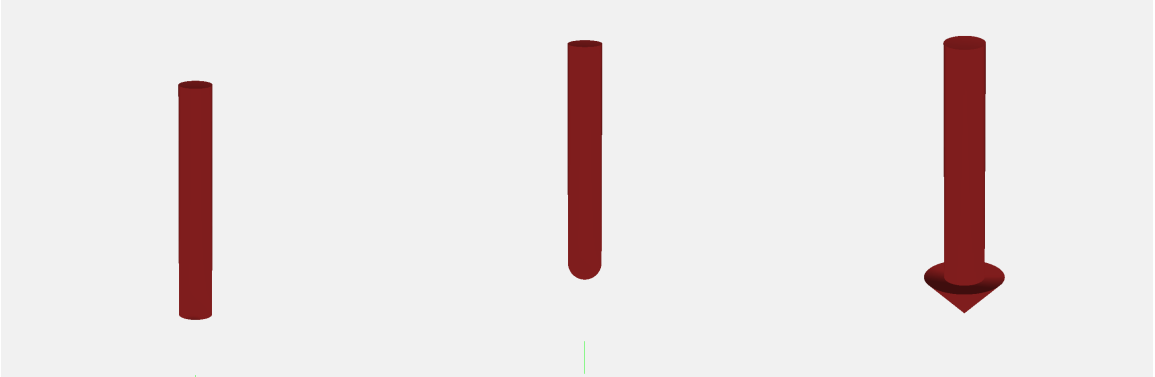
Diagramming this is quite straight-forward — there is simply a movement made from the current position to the end. If we start at the origin, `X0, Y0, Z0`, then it is simply a straight-line movement (rapid)/cut (possibly a partial cut in the instance of a keyhole or roundover tool), and no variables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



```
644 gcpy      def toolmovement(self, bx, by, bz, ex, ey, ez, step = 0):
645 gcpy          tslist = []
646 gcpy          if step > 0:
647 gcpy              steps = step
648 gcpy          else:
649 gcpy              steps = self.steps
```

3.5.2.2 Normal Tooling/toolshapes Most tooling has quite standard shapes and are defined by their profile as defined in the `toolmovement` command which simply defines/declares their shape and `hull()`s them together:



- Square (#201 and 102) — able to cut a flat bottom, perpendicular side and right angle, their simple and easily understood geometry makes them a standard choice
- Ballnose (#202 and 101) — rounded, they are the standard choice for concave and organic shapes
- V tooling (#301, 302 and 390) — pointed at the tip, they are available in a variety of angles and diameters and may be used for decorative V carving, or for chamfering or cutting specific angles

Note that the module for creating movement of the tool will need to handle all of the different tool shapes, generating a list of `hull()` commands which describe the 3D region which tool movement describes.

endmill square **3.5.2.3 Square (including O-flute)** The `endmill square` is a simple cylinder:

```
651 gcpy      if self.endmilltype == "square":
652 gcpy          ts = cylinder(r1=(self.diameter / 2), r2=(self.diameter
                    / 2), h=self.flute, center = False)
653 gcpy          tslist.append(hull(ts.translate([bx, by, bz]), ts.
                    translate([ex, ey, ez])))
654 gcpy          return tslist
655 gcpy
656 gcpy      if self.endmilltype == "O-flute":
657 gcpy          ts = cylinder(r1=(self.diameter / 2), r2=(self.diameter
                    / 2), h=self.flute, center = False)
658 gcpy          tslist.append(hull(ts.translate([bx, by, bz]), ts.
                    translate([ex, ey, ez])))
659 gcpy          return tslist
```

ballnose **3.5.2.4 Ball nose (including tapered ball nose)** The `ballnose` is modeled as a hemisphere joined with a cylinder:

```
661 gcpy      if self.endmilltype == "ball":
662 gcpy          b = sphere(r=(self.diameter / 2))
663 gcpy          s = cylinder(r1=(self.diameter / 2), r2=(self.diameter
                    / 2), h=self.flute, center=False)
664 gcpy          bs = union(b, s)
665 gcpy          bs = bs.translate([0, 0, (self.diameter / 2)])
666 gcpy          tslist.append(hull(bs.translate([bx, by, bz]), bs.
                    translate([ex, ey, ez])))
667 gcpy          return tslist
668 gcpy #
```

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

```
669 gcpy      if self.endmilltype == "bowl":
670 gcpy          inner = cylinder(r1 = self.diameter/2 - self.radius, r2
                    = self.diameter/2 - self.radius, h = self.flute)
671 gcpy          outer = cylinder(r1 = self.diameter/2, r2 = self.
                    diameter/2, h = self.flute - self.radius)
672 gcpy          outer = outer.translate([0,0, self.radius])
673 gcpy          slices = hull(outer, inner)
674 gcpy #      slices = cylinder(r1 = 0.0001, r2 = 0.0001, h = 0.0001, center
                    =False)
```

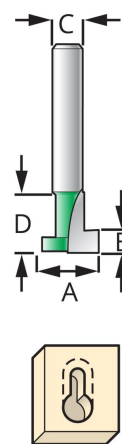


```
675 gcpy          for i in range(1, 90 - self.steps, self.steps):
676 gcpy              slice = cylinder(r1 = self.diameter / 2 - self.
                                radius + self.radius * Sin(i), r2 = self.
                                diameter / 2 - self.radius + self.radius * Sin(
                                i + self.steps), h = self.radius/90, center=False)
677 gcpy              slices = hull(slices, slice.translate([0, 0, self.
                                radius - self.radius * Cos(i+self.steps)]))
678 gcpy          tslist.append(hull(slices.translate([bx, by, bz]),
                                slices.translate([ex, ey, ez])))
679 gcpy          return tslist
680 gcpy #
```

endmill v **3.5.2.6 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 if using it, but fortunately, trigonometric commands have been added to OpenPythonSCAD (Sin, Cos, Tan, Atan)):

```
681 gcpy          if self.endmilltype == "V":
682 gcpy              v = cylinder(r1=0, r2=(self.diameter / 2), h=((self.
                                diameter / 2) / Tan((self.angle / 2))), center=False
                                )
683 gcpy #              s = cylinder(r1=(self.diameter / 2), r2=(self.
                                diameter / 2), h=self.flute, center=False)
684 gcpy #              sh = s.translate([0, 0, ((self.diameter / 2) / Tan
                                ((self.angle / 2)))]))
685 gcpy              tslist.append(hull(v.translate([bx, by, bz]), v.
                                translate([ex, ey, ez])))
686 gcpy          return tslist
```

3.5.2.7 Keyhole Keyhole toolpaths (see: subsection 3.8.0.2.3 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

```
688 gcpy          if self.endmilltype == "keyhole":
689 gcpy              kh = cylinder(r1=(self.diameter / 2), r2=(self.diameter
                                / 2), h=self.flute, center=False)
690 gcpy              sh = (cylinder(r1=(self.radius / 2), r2=(self.radius /
                                2), h=self.flute*2, center=False))
691 gcpy              tslist.append(hull(kh.translate([bx, by, bz]), kh.
                                translate([ex, ey, ez])))
692 gcpy              tslist.append(hull(sh.translate([bx, by, bz]), sh.
                                translate([ex, ey, ez])))
693 gcpy          return tslist
```

3.5.2.8 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.

```
695 gcpy          if self.endmilltype == "tapered_ball":
696 gcpy              b = sphere(r=(self.tip / 2))
697 gcpy              s = cylinder(r1=(self.tip / 2), r2=(self.diameter / 2),
                                h=self.flute, center=False)
698 gcpy              bshape = union(b, s)
```

```
699 gcpy          tslist.append(hull(bshape.translate([bx, by, bz]),
700 gcpy          bshape.translate([ex, ey, ez])))
              return tslist
```

dovetail 3.5.2.9 **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)

```
702 gcpy          if self.endmilltype == "dovetail":
703 gcpy          dt = cylinder(r1=(self.diameter / 2), r2=(self.diameter
              / 2) - self.flute * Tan(self.angle), h= self.flute,
              center=False)
704 gcpy          tslist.append(hull(dt.translate([bx, by, bz]), dt.
              translate([ex, ey, ez])))
705 gcpy          return tslist
706 gcpy          if self.endmilltype == "other":
707 gcpy          tslist = []
708 gcpy #          def dovetail(self, dt_bottomdiameter, dt_topdiameter,
              dt_height, dt_angle):
709 gcpy #          return cylinder(r1=(dt_bottomdiameter / 2), r2=(
              dt_topdiameter / 2), h= dt_height, center=False)
```

3.5.2.10 **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.

3.5.2.11 **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 3.5.2.10 — roundover tooling will need to generate a list of slices of the tool shape hulled together.

roundover

```
711 gcpy          if self.endmilltype == "roundover":
712 gcpy          shaft = cylinder(self.steps, self.tip/2, self.tip/2)
713 gcpy          toolpath = hull(shaft.translate([bx, by, bz]), shaft.
              translate([ex, ey, ez]))
714 gcpy          shaft = cylinder(self.flute, self.diameter/2 + self.tip
              /2, self.diameter/2 + self.tip/2)
715 gcpy          toolpath = toolpath.union(hull(shaft.translate([bx, by,
              bz + self.radius]), shaft.translate([ex, ey, ez +
              self.radius])))
716 gcpy          tslist = [toolpath]
717 gcpy          slice = cylinder(0.0001, 0.0001, 0.0001)
718 gcpy          slices = slice
719 gcpy          for i in range(1, 90 - self.steps, self.steps):
720 gcpy          dx = self.radius*Cos(i)
721 gcpy          dxx = self.radius*Cos(i + self.steps)
722 gcpy          dzz = self.radius*Sin(i)
723 gcpy          dz = self.radius*Sin(i + self.steps)
724 gcpy          dh = dz - dzz
725 gcpy          slice = cylinder(r1 = self.tip/2+self.radius-dx, r2
              = self.tip/2+self.radius-dxx, h = dh)
726 gcpy          slices = slices.union(hull(slice.translate([bx, by,
              bz+dz]), slice.translate([ex, ey, ez+dz])))
727 gcpy          tslist.append(slices)
728 gcpy          return tslist
729 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 3.5.2.12 **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:

```
730 gcpy      def shaftmovement(self, bx, by, bz, ex, ey, ez):
731 gcpy          tslist = []
732 gcpy          ts = cylinder(r1=(self.shaftdiameter / 2), r2=(self.
              shaftdiameter / 2), h=self.shaftlength, center = False)
733 gcpy          ts = ts.translate([0, 0, self.shaftheight])
734 gcpy          tslist.append(hull(ts.translate([bx, by, bz]), ts.translate
              ([ex, ey, ez])))
735 gcpy      return tslist
```

rapid 3.5.2.13 **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.

```
737 gcpy      def rapid(self, ex, ey, ez, laser = 0):
738 gcpy      #          print(self.rapidcolor)
739 gcpy      if laser == 0:
740 gcpy          tm = self.toolmovement(self.xpos(), self.ypos(), self.
              zpos(), ex, ey, ez)
741 gcpy          tm = color(tm, self.shaftcolor)
742 gcpy          ts = self.shaftmovement(self.xpos(), self.ypos(), self.
              zpos(), ex, ey, ez)
743 gcpy          ts = color(ts, self.rapidcolor)
744 gcpy          self.toolpaths.extend([tm, ts])
745 gcpy          self.setxpos(ex)
746 gcpy          self.setypos(ey)
747 gcpy          self.setzpos(ez)
748 gcpy
749 gcpy      def cutline(self, ex, ey, ez):
750 gcpy      #          print(self.cutcolor)
751 gcpy      #          print(ex, ey, ez)
752 gcpy          tm = self.toolmovement(self.xpos(), self.ypos(), self.zpos
              (), ex, ey, ez)
753 gcpy          tm = color(tm, self.cutcolor)
754 gcpy          ts = self.shaftmovement(self.xpos(), self.ypos(), self.zpos
              (), ex, ey, ez)
755 gcpy          ts = color(ts, self.rapidcolor)
756 gcpy          self.setxpos(ex)
757 gcpy          self.setypos(ey)
758 gcpy          self.setzpos(ez)
759 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:

```
761 gcpy      def movetosafeZ(self):
762 gcpy          rapid = self.rapid(self.xpos(), self.ypos(), self.
              retractheight)
763 gcpy      #          if self.generatepaths == True:
764 gcpy      #              rapid = self.rapid(self.xpos(), self.ypos(), self.
              retractheight)
765 gcpy      #              self.rapids = self.rapids.union(rapid)
766 gcpy      #          else:
767 gcpy      #              if (generategcode == true) {
768 gcpy      #                  //      writecomment("PREPOSITION FOR RAPID PLUNGE");Z25.650
769 gcpy      #                  //G1Z24.663F381.0, "F", str(plunge)
770 gcpy      #                  if self.generatepaths == False:
771 gcpy      #                      return rapid
772 gcpy      #                  else:
773 gcpy      #                      return cube([0.001, 0.001, 0.001])
774 gcpy          return rapid
775 gcpy
776 gcpy      def rapidXYZ(self, ex, ey, ez):
777 gcpy          rapid = self.rapid(ex, ey, ez)
778 gcpy      #          if self.generatepaths == False:
779 gcpy          return rapid
780 gcpy
781 gcpy      def rapidXY(self, ex, ey):
782 gcpy          rapid = self.rapid(ex, ey, self.zpos())
```

```

783 gcpy #         if self.generatepaths == True:
784 gcpy #             self.rapids = self.rapids.union(rapid)
785 gcpy #         else:
786 gcpy #             if self.generatepaths == False:
787 gcpy                 return rapid
788 gcpy
789 gcpy     def rapidXZ(self, ex, ez):
790 gcpy         rapid = self.rapid(ex, self.ypos(), ez)
791 gcpy #         if self.generatepaths == False:
792 gcpy             return rapid
793 gcpy
794 gcpy     def rapidYZ(self, ey, ez):
795 gcpy         rapid = self.rapid(self.xpos(), ey, ez)
796 gcpy #         if self.generatepaths == False:
797 gcpy             return rapid
798 gcpy
799 gcpy     def rapidX(self, ex):
800 gcpy         rapid = self.rapid(ex, self.ypos(), self.zpos())
801 gcpy #         if self.generatepaths == False:
802 gcpy             return rapid
803 gcpy
804 gcpy     def rapidY(self, ey):
805 gcpy         rapid = self.rapid(self.xpos(), ey, self.zpos())
806 gcpy #         if self.generatepaths == False:
807 gcpy             return rapid
808 gcpy
809 gcpy     def rapidZ(self, ez):
810 gcpy         rapid = [self.rapid(self.xpos(), self.ypos(), ez)]
811 gcpy #         if self.generatepaths == True:
812 gcpy #             self.rapids = self.rapids.union(rapid)
813 gcpy #         else:
814 gcpy #             if self.generatepaths == False:
815 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):

```

52 gcpscad module movetosafeZ(){
53 gcpscad     gcp.rapid(gcp.xpos(), gcp.ypos(), retractheight);
54 gcpscad }
55 gcpscad
56 gcpscad module rapid(ex, ey, ez) {
57 gcpscad     gcp.rapid(ex, ey, ez);
58 gcpscad }
59 gcpscad
60 gcpscad module rapidXY(ex, ey) {
61 gcpscad     gcp.rapid(ex, ey, gcp.zpos());
62 gcpscad }
63 gcpscad
64 gcpscad module rapidXZ(ex, ez) {
65 gcpscad     gcp.rapid(ex, gcp.zpos(), ez);
66 gcpscad }
67 gcpscad
68 gcpscad module rapidZ(ez) {
69 gcpscad     gcp.rapid(gcp.xpos(), gcp.ypos(), ez);
70 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 current position and the end position of the move. For `cutline`, this is a straight-forward connection of the current (beginning) and ending coordinates:

```

817 gcpy     def cutlinedxf(self, ex, ey, ez):
818 gcpy         self.dxfline(self.currenttoolnumber(), self.xpos(), self.
            ypos(), ex, ey)
819 gcpy         self.cutline(ex, ey, ez)
820 gcpy
821 gcpy     def cutlinedxfgc(self, ex, ey, ez):
822 gcpy         self.dxfline(self.currenttoolnumber(), self.xpos(), self.
            ypos(), ex, ey)
823 gcpy         self.writegc("G01_X", str(ex), "Y", str(ey), "Z", str(ez)
            )
824 gcpy         self.cutline(ex, ey, ez)
825 gcpy
826 gcpy     def cutvertexdxf(self, ex, ey, ez):
827 gcpy         self.addvertex(self.currenttoolnumber(), ex, ey)

```

```
828 gcpy          self.writegc("G01_X", str(ex), "Y", str(ey), "Z", str(ez)
829 gcpy          self.cutline(ex, ey, ez)
830 gcpy
831 gcpy          def cutlineXYZwithfeed(self, ex, ey, ez, feed):
832 gcpy              return self.cutline(ex, ey, ez)
833 gcpy
834 gcpy          def cutlineXYZ(self, ex, ey, ez):
835 gcpy              return self.cutline(ex, ey, ez)
836 gcpy
837 gcpy          def cutlineXYwithfeed(self, ex, ey, feed):
838 gcpy              return self.cutline(ex, ey, self.zpos())
839 gcpy
840 gcpy          def cutlineXY(self, ex, ey):
841 gcpy              return self.cutline(ex, ey, self.zpos())
842 gcpy
843 gcpy          def cutlineXZwithfeed(self, ex, ez, feed):
844 gcpy              return self.cutline(ex, self.ypos(), ez)
845 gcpy
846 gcpy          def cutlineXZ(self, ex, ez):
847 gcpy              return self.cutline(ex, self.ypos(), ez)
848 gcpy
849 gcpy          def cutlineXwithfeed(self, ex, feed):
850 gcpy              return self.cutline(ex, self.ypos(), self.zpos())
851 gcpy
852 gcpy          def cutlineX(self, ex):
853 gcpy              return self.cutline(ex, self.ypos(), self.zpos())
854 gcpy
855 gcpy          def cutlineYZ(self, ey, ez):
856 gcpy              return self.cutline(self.xpos(), ey, ez)
857 gcpy
858 gcpy          def cutlineYwithfeed(self, ey, feed):
859 gcpy              return self.cutline(self.xpos(), ey, self.zpos())
860 gcpy
861 gcpy          def cutlineY(self, ey):
862 gcpy              return self.cutline(self.xpos(), ey, self.zpos())
863 gcpy
864 gcpy          def cutlineZgcfeed(self, ez, feed):
865 gcpy              self.writegc("G01_Z", str(ez), "F", str(feed))
866 gcpy              return self.cutline(self.xpos(), self.ypos(), ez)
867 gcpy
868 gcpy          def cutlineZwithfeed(self, ez, feed):
869 gcpy              return self.cutline(self.xpos(), self.ypos(), ez)
870 gcpy
871 gcpy          def cutlineZ(self, ez):
872 gcpy              return self.cutline(self.xpos(), self.ypos(), ez)
```

The matching OpenSCAD command is a descriptor:

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

3.5.2.14 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
- cutcircleCCdxf
- cutcircleCWdxf

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

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

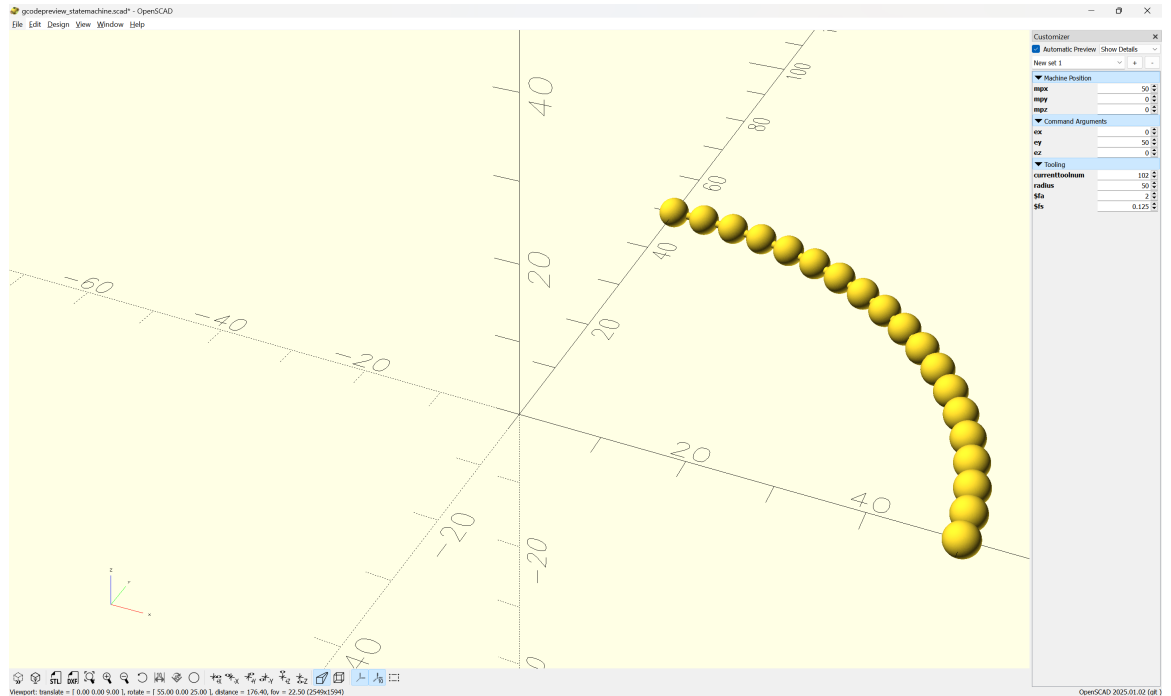
Cutting the quadrant arcs greatly simplifies the calculation and interface for the modules. A full set of 8 will be necessary, then circles will have a pair of modules (one for each cut direction) made for them.

Parameters which will need to be passed in are:

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

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

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



The code for which makes this obvious:

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

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

/* [Tooling] */
currenttoolnum = 102;

machine_extents();

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

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

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

```

}
}

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

```

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

```

874 gcpy      def cutarcCC(self, barc, earc, xcenter, ycenter, radius,
tpzreldim, stepsizearc=1):
875 gcpy      tpzinc = tpzreldim / (earc - barc)
876 gcpy      i = barc
877 gcpy      while i < earc:
878 gcpy          self.cutline(xcenter + radius * Cos(math.radians(i)),
ycenter + radius * Sin(math.radians(i)), self.zpos()
+tpzinc)
879 gcpy          i += stepsizearc
880 gcpy      self.setxpos(xcenter + radius * Cos(math.radians(earc)))
881 gcpy      self.setypos(ycenter + radius * Sin(math.radians(earc)))
882 gcpy
883 gcpy      def cutarcCW(self, barc, earc, xcenter, ycenter, radius,
tpzreldim, stepsizearc=1):
884 gcpy #          print(str(self.zpos()))
885 gcpy #          print(str(ez))
886 gcpy #          print(str(barc - earc))
887 gcpy #          tpzinc = ez - self.zpos() / (barc - earc)
888 gcpy #          print(str(tzinc))
889 gcpy #          global toolpath
890 gcpy #          print("Entering n toolpath")
891 gcpy      tpzinc = tpzreldim / (barc - earc)
892 gcpy #          cts = self.currenttoolshape
893 gcpy #          toolpath = cts
894 gcpy #          toolpath = toolpath.translate([self.xpos(), self.ypos(),
self.zpos()])
895 gcpy #          toolpath = []
896 gcpy      i = barc
897 gcpy      while i > earc:
898 gcpy          self.cutline(xcenter + radius * Cos(math.radians(i)),
ycenter + radius * Sin(math.radians(i)), self.zpos()
+tpzinc)
899 gcpy #          self.setxpos(xcenter + radius * Cos(math.radians(i)))
900 gcpy #          self.setypos(ycenter + radius * Sin(math.radians(i)))
901 gcpy #          print(str(self.xpos()), str(self.ypos()), str(self.zpos
()))
902 gcpy #          self.setzpos(self.zpos()+tpzinc)
903 gcpy      i += abs(stepsizearc) * -1
904 gcpy #          self.dxfarc(self.currenttoolnumber(), xcenter, ycenter,
radius, barc, earc)
905 gcpy #          if self.generatepaths == True:
906 gcpy #              print("Unioning n toolpath")
907 gcpy #              self.toolpaths = self.toolpaths.union(toolpath)
908 gcpy #          else:
909 gcpy      self.setxpos(xcenter + radius * Cos(math.radians(earc)))
910 gcpy      self.setypos(ycenter + radius * Sin(math.radians(earc)))
911 gcpy #          self.toolpaths.extend(toolpath)
912 gcpy #          if self.generatepaths == False:
913 gcpy #              return toolpath
914 gcpy #          else:
915 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:

```

917 gcpy      def cutarcCWdxf(self, barc, earc, xcenter, ycenter, radius,
tpzreldim, stepsizearc=1):
918 gcpy      self.cutarcCW(barc, earc, xcenter, ycenter, radius,
tpzreldim, stepsizearc=1)
919 gcpy      self.dxfarc(self.currenttoolnumber(), xcenter, ycenter,
radius, earc, barc)
920 gcpy #          if self.generatepaths == False:

```



```

921 gcpy #          return toolpath
922 gcpy #          else:
923 gcpy #              return cube([0.01, 0.01, 0.01])
924 gcpy
925 gcpy      def cutarcCCdxf(self, barc, earc, xcenter, ycenter, radius,
tpzreldim, stepsizearc=1):
926 gcpy          self.cutarcCC(barc, earc, xcenter, ycenter, radius,
tpzreldim, stepsizearc=1)
927 gcpy          self.dxfarc(self.currenttoolnumber(), xcenter, ycenter,
radius, barc, earc)

```

Matching OpenSCAD modules are easily made:

```

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

```

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

```

929 gcpy      def cutquarterCCNE(self, ex, ey, ez, radius):
930 gcpy          if self.zpos() == ez:
931 gcpy              tpzinc = 0
932 gcpy          else:
933 gcpy              tpzinc = (ez - self.zpos()) / 90
934 gcpy #          print("tpzinc ", tpzinc)
935 gcpy          i = 1
936 gcpy          while i < 91:
937 gcpy              self.cutline(ex + radius * Cos(i), ey - radius + radius
* Sin(i), self.zpos()+tpzinc)
938 gcpy              i += 1
939 gcpy
940 gcpy      def cutquarterCCNW(self, ex, ey, ez, radius):
941 gcpy          if self.zpos() == ez:
942 gcpy              tpzinc = 0
943 gcpy          else:
944 gcpy              tpzinc = (ez - self.zpos()) / 90
945 gcpy #          tpzinc = (self.zpos() + ez) / 90
946 gcpy          print("tpzinc_", tpzinc)
947 gcpy          i = 91
948 gcpy          while i < 181:
949 gcpy              self.cutline(ex + radius + radius * Cos(i), ey + radius
* Sin(i), self.zpos()+tpzinc)
950 gcpy              i += 1
951 gcpy
952 gcpy      def cutquarterCCSW(self, ex, ey, ez, radius):
953 gcpy          if self.zpos() == ez:
954 gcpy              tpzinc = 0
955 gcpy          else:
956 gcpy              tpzinc = (ez - self.zpos()) / 90
957 gcpy #          tpzinc = (self.zpos() + ez) / 90
958 gcpy          print("tpzinc_", tpzinc)
959 gcpy          i = 181
960 gcpy          while i < 271:
961 gcpy              self.cutline(ex + radius * Cos(i), ey + radius + radius
* Sin(i), self.zpos()+tpzinc)
962 gcpy              i += 1
963 gcpy
964 gcpy      def cutquarterCCSE(self, ex, ey, ez, radius):
965 gcpy          if self.zpos() == ez:
966 gcpy              tpzinc = 0
967 gcpy          else:
968 gcpy              tpzinc = (ez - self.zpos()) / 90
969 gcpy #          tpzinc = (self.zpos() + ez) / 90
970 gcpy #          print("tpzinc ", tpzinc)
971 gcpy          i = 271
972 gcpy          while i < 361:
973 gcpy              self.cutline(ex - radius + radius * Cos(i), ey + radius
* Sin(i), self.zpos()+tpzinc)
974 gcpy              i += 1
975 gcpy
976 gcpy      def cutquarterCCNEdxf(self, ex, ey, ez, radius):
977 gcpy          self.cutquarterCCNE(ex, ey, ez, radius)

```

```
978 gcpy          self.dxfarc(self.currenttoolnumber(), ex, ey - radius,
                        radius, 0, 90)
979 gcpy
980 gcpy          def cutquarterCCNWdxf(self, ex, ey, ez, radius):
981 gcpy              self.cutquarterCCNW(ex, ey, ez, radius)
982 gcpy              self.dxfarc(self.currenttoolnumber(), ex + radius, ey,
                        radius, 90, 180)
983 gcpy
984 gcpy          def cutquarterCCSWdxf(self, ex, ey, ez, radius):
985 gcpy              self.cutquarterCCSW(ex, ey, ez, radius)
986 gcpy              self.dxfarc(self.currenttoolnumber(), ex, ey + radius,
                        radius, 180, 270)
987 gcpy
988 gcpy          def cutquarterCCSEdxf(self, ex, ey, ez, radius):
989 gcpy              self.cutquarterCCSE(ex, ey, ez, radius)
990 gcpy              self.dxfarc(self.currenttoolnumber(), ex - radius, ey,
                        radius, 270, 360)
```

```
92 gcpscad module cutquarterCCNE(ex, ey, ez, radius){
93 gcpscad     gcp.cutquarterCCNE(ex, ey, ez, radius);
94 gcpscad }
95 gcpscad
96 gcpscad module cutquarterCCNW(ex, ey, ez, radius){
97 gcpscad     gcp.cutquarterCCNW(ex, ey, ez, radius);
98 gcpscad }
99 gcpscad
100 gcpscad module cutquarterCCSW(ex, ey, ez, radius){
101 gcpscad     gcp.cutquarterCCSW(ex, ey, ez, radius);
102 gcpscad }
103 gcpscad
104 gcpscad module cutquarterCCSE(self, ex, ey, ez, radius){
105 gcpscad     gcp.cutquarterCCSE(ex, ey, ez, radius);
106 gcpscad }
107 gcpscad
108 gcpscad module cutquarterCCNEdxf(ex, ey, ez, radius){
109 gcpscad     gcp.cutquarterCCNEdxf(ex, ey, ez, radius);
110 gcpscad }
111 gcpscad
112 gcpscad module cutquarterCCNWdxf(ex, ey, ez, radius){
113 gcpscad     gcp.cutquarterCCNWdxf(ex, ey, ez, radius);
114 gcpscad }
115 gcpscad
116 gcpscad module cutquarterCCSWdxf(ex, ey, ez, radius){
117 gcpscad     gcp.cutquarterCCSWdxf(ex, ey, ez, radius);
118 gcpscad }
119 gcpscad
120 gcpscad module cutquarterCCSEdxf(self, ex, ey, ez, radius){
121 gcpscad     gcp.cutquarterCCSEdxf(ex, ey, ez, radius);
122 gcpscad }
```

3.5.3 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 are 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:

```
124 gcpscad function tool_diameter(td_tool, td_depth) = otool_diameter(td_tool,
                        td_depth);
```

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

```
992 gcpy          def tool_diameter(self, ptd_tool, ptd_depth):
993 gcpy # Square 122, 112, 102, 201
994 gcpy          if ptd_tool == 122:
995 gcpy              return 0.79375
996 gcpy          if ptd_tool == 112:
997 gcpy              return 1.5875
998 gcpy          if ptd_tool == 102:
```

```

999 gcpy                return 3.175
1000 gcpy               if ptd_tool == 201:
1001 gcpy                return 6.35
1002 gcpy # Ball 121, 111, 101, 202
1003 gcpy               if ptd_tool == 122:
1004 gcpy                 if ptd_depth > 0.396875:
1005 gcpy                   return 0.79375
1006 gcpy                 else:
1007 gcpy                   return ptd_tool
1008 gcpy               if ptd_tool == 112:
1009 gcpy                 if ptd_depth > 0.79375:
1010 gcpy                   return 1.5875
1011 gcpy                 else:
1012 gcpy                   return ptd_tool
1013 gcpy               if ptd_tool == 101:
1014 gcpy                 if ptd_depth > 1.5875:
1015 gcpy                   return 3.175
1016 gcpy                 else:
1017 gcpy                   return ptd_tool
1018 gcpy               if ptd_tool == 202:
1019 gcpy                 if ptd_depth > 3.175:
1020 gcpy                   return 6.35
1021 gcpy                 else:
1022 gcpy                   return ptd_tool
1023 gcpy # V 301, 302, 390
1024 gcpy               if ptd_tool == 301:
1025 gcpy                 return ptd_tool
1026 gcpy               if ptd_tool == 302:
1027 gcpy                 return ptd_tool
1028 gcpy               if ptd_tool == 390:
1029 gcpy                 return ptd_tool
1030 gcpy # Keyhole
1031 gcpy               if ptd_tool == 374:
1032 gcpy                 if ptd_depth < 3.175:
1033 gcpy                   return 9.525
1034 gcpy                 else:
1035 gcpy                   return 6.35
1036 gcpy               if ptd_tool == 375:
1037 gcpy                 if ptd_depth < 3.175:
1038 gcpy                   return 9.525
1039 gcpy                 else:
1040 gcpy                   return 8
1041 gcpy               if ptd_tool == 376:
1042 gcpy                 if ptd_depth < 4.7625:
1043 gcpy                   return 12.7
1044 gcpy                 else:
1045 gcpy                   return 6.35
1046 gcpy               if ptd_tool == 378:
1047 gcpy                 if ptd_depth < 4.7625:
1048 gcpy                   return 12.7
1049 gcpy                 else:
1050 gcpy                   return 8
1051 gcpy # Dovetail
1052 gcpy               if ptd_tool == 814:
1053 gcpy                 if ptd_depth > 12.7:
1054 gcpy                   return 6.35
1055 gcpy                 else:
1056 gcpy                   return ptd_tool
1057 gcpy               if ptd_tool == 808079:
1058 gcpy                 if ptd_depth > 20.95:
1059 gcpy                   return 6.816
1060 gcpy                 else:
1061 gcpy                   return ptd_tool
1062 gcpy # Bowl Bit
1063 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
1064 gcpy               if ptd_tool == 45982:
1065 gcpy                 if ptd_depth > 6.35:
1066 gcpy                   return 15.875
1067 gcpy                 else:
1068 gcpy                   return ptd_tool
1069 gcpy # Tapered Ball Nose
1070 gcpy               if ptd_tool == 204:
1071 gcpy                 if ptd_depth > 6.35:
1072 gcpy                   return ptd_tool
1073 gcpy               if ptd_tool == 304:
1074 gcpy                 if ptd_depth > 6.35:
1075 gcpy                   return ptd_tool

```

```
1076 gcpy                else:
1077 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:

```
1079 gcpy    def tool_radius(self, ptd_tool, ptd_depth):
1080 gcpy        tr = self.tool_diameter(ptd_tool, ptd_depth)/2
1081 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.)

3.5.4 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

3.6 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:

```
1083 gcpy    def stockandtoolpaths(self, option = "stockandtoolpaths"):
1084 gcpy        if option == "stock":
1085 gcpy            show(self.stock)
1086 gcpy        elif option == "toolpaths":
1087 gcpy            show(self.toolpaths)
1088 gcpy        elif option == "rapids":
1089 gcpy            show(self.rapids)
1090 gcpy        else:
1091 gcpy            part = self.stock.difference(self.rapids)
1092 gcpy            part = self.stock.difference(self.toolpaths)
1093 gcpy            show(part)
```

Note that because of the differences in behaviour between OpenPythonSCAD (the show() command results in an explicit display of the requested element) and OpenSCAD (there is an implicit mechanism where the 3D element whihc is returned is displayed), the most expedient mechanism is to have an explicit Python command which returns the 3D model:

```
1095 gcpy    def returnstockandtoolpaths(self):
1096 gcpy        part = self.stock.difference(self.toolpaths)
1097 gcpy        return part
```

and then make use of that specific command for OpenSCAD:

```
126 gcpscad module stockandtoolpaths(){
127 gcpscad     gcp.returnstockandtoolpaths();
128 gcpscad }
```

forgoing the options of showing toolpaths and/or rapids separately.

3.7 Output files

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

3.7.1 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.

```
1099 gcpy      def writegc(self, *arguments):
1100 gcpy          if self.generategcode == True:
1101 gcpy              line_to_write = ""
1102 gcpy              for element in arguments:
1103 gcpy                  line_to_write += element
1104 gcpy              self.gc.write(line_to_write)
1105 gcpy              self.gc.write("\n")
1106 gcpy
1107 gcpy      def writedxf(self, toolnumber, *arguments):
1108 gcpy          # global dxfclosed
1109 gcpy          line_to_write = ""
1110 gcpy          for element in arguments:
1111 gcpy              line_to_write += element
1112 gcpy          if self.generatedxif == True:
1113 gcpy              if self.dxfclosed == False:
1114 gcpy                  self.dxf.write(line_to_write)
1115 gcpy                  self.dxf.write("\n")
1116 gcpy          if self.generatedxifs == True:
1117 gcpy              self.writedxifs(toolnumber, line_to_write)
1118 gcpy
1119 gcpy      def writedxifs(self, toolnumber, line_to_write):
1120 gcpy          # print("Processing writing toolnumber", toolnumber)
1121 gcpy          # line_to_write = ""
1122 gcpy          # for element in arguments:
1123 gcpy          #     line_to_write += element
1124 gcpy          if (toolnumber == 0):
1125 gcpy              return
1126 gcpy          elif self.generatedxifs == True:
1127 gcpy              if (self.large_square_tool_num == toolnumber):
1128 gcpy                  self.dxfllsq.write(line_to_write)
1129 gcpy                  self.dxfllsq.write("\n")
1130 gcpy              if (self.small_square_tool_num == toolnumber):
1131 gcpy                  self.dxfllsq.write(line_to_write)
1132 gcpy                  self.dxfllsq.write("\n")
1133 gcpy              if (self.large_ball_tool_num == toolnumber):
1134 gcpy                  self.dxfllbl.write(line_to_write)
1135 gcpy                  self.dxfllbl.write("\n")
1136 gcpy              if (self.small_ball_tool_num == toolnumber):
1137 gcpy                  self.dxfllbl.write(line_to_write)
1138 gcpy                  self.dxfllbl.write("\n")
1139 gcpy              if (self.large_V_tool_num == toolnumber):
1140 gcpy                  self.dxfllV.write(line_to_write)
1141 gcpy                  self.dxfllV.write("\n")
1142 gcpy              if (self.small_V_tool_num == toolnumber):
1143 gcpy                  self.dxfllV.write(line_to_write)
1144 gcpy                  self.dxfllV.write("\n")
1145 gcpy              if (self.DT_tool_num == toolnumber):
1146 gcpy                  self.dxfDT.write(line_to_write)
1147 gcpy                  self.dxfDT.write("\n")
1148 gcpy              if (self.KH_tool_num == toolnumber):
1149 gcpy                  self.dxfKH.write(line_to_write)
1150 gcpy                  self.dxfKH.write("\n")
1151 gcpy              if (self.Roundover_tool_num == toolnumber):
1152 gcpy                  self.dxfRt.write(line_to_write)
1153 gcpy                  self.dxfRt.write("\n")
1154 gcpy              if (self.MISC_tool_num == toolnumber):
1155 gcpy                  self.dxfMt.write(line_to_write)
1156 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:

```
130 gpcscad module opendxfile(basefilename){
131 gpcscad     gcp.opendxfile(basefilename);
132 gpcscad }
133 gpcscad
134 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) {
135 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)
    ;
136 gpcscad }
```

opengcodefile With matching OpenSCAD commands: opengcodefile for OpenSCAD:

```
138 gpcscad module opengcodefile(basefilename, currenttoolnum, toolradius,
    plunge, feed, speed) {
139 gpcscad     gcp.opengcodefile(basefilename, currenttoolnum, toolradius,
    plunge, feed, speed);
140 gpcscad }
```

and Python:

```
1158 gcpy      def opengcodefile(self, basefilename = "export",
1159 gcpy          currenttoolnum = 102,
1160 gcpy          toolradius = 3.175,
1161 gcpy          plunge = 400,
1162 gcpy          feed = 1600,
1163 gcpy          speed = 10000
1164 gcpy          ):
1165 gcpy          self.basefilename = basefilename
1166 gcpy          self.currenttoolnum = currenttoolnum
1167 gcpy          self.toolradius = toolradius
1168 gcpy          self.plunge = plunge
1169 gcpy          self.feed = feed
1170 gcpy          self.speed = speed
1171 gcpy          if self.generategcode == True:
1172 gcpy              self.gcodefilename = basefilename + ".nc"
1173 gcpy              self.gc = open(self.gcodefilename, "w")
1174 gcpy              self.writegc("(Design␣File:␣" + self.basefilename + ")")
1175 gcpy
1176 gcpy      def opendxfile(self, basefilename = "export"):
1177 gcpy          self.basefilename = basefilename
1178 gcpy          # global generatedxfs
1179 gcpy          # global dxfclosed
1180 gcpy          self.dxfclosed = False
1181 gcpy          self.dxfcolor = "Black"
1182 gcpy          if self.generatedxfs == True:
1183 gcpy              self.generatedxfs = False
1184 gcpy              self.dxffilename = basefilename + ".dxf"
1185 gcpy              self.dxf = open(self.dxffilename, "w")
1186 gcpy              self.dxfpreamble(-1)
1187 gcpy
1188 gcpy      def opendxfiles(self, basefilename = "export",
1189 gcpy          large_square_tool_num = 0,
1190 gcpy          small_square_tool_num = 0,
1191 gcpy          large_ball_tool_num = 0,
1192 gcpy          small_ball_tool_num = 0,
1193 gcpy          large_V_tool_num = 0,
1194 gcpy          small_V_tool_num = 0,
1195 gcpy          DT_tool_num = 0,
1196 gcpy          KH_tool_num = 0,
1197 gcpy          Roundover_tool_num = 0,
1198 gcpy          MISC_tool_num = 0):
1199 gcpy          # global generatedxfs
1200 gcpy          self.basefilename = basefilename
1201 gcpy          self.generatedxfs = True
1202 gcpy          self.large_square_tool_num = large_square_tool_num
1203 gcpy          self.small_square_tool_num = small_square_tool_num
1204 gcpy          self.large_ball_tool_num = large_ball_tool_num
1205 gcpy          self.small_ball_tool_num = small_ball_tool_num
1206 gcpy          self.large_V_tool_num = large_V_tool_num
```

```
1207 gcpy self.small_V_tool_num = small_V_tool_num
1208 gcpy self.DT_tool_num = DT_tool_num
1209 gcpy self.KH_tool_num = KH_tool_num
1210 gcpy self.Roundover_tool_num = Roundover_tool_num
1211 gcpy self.MISC_tool_num = MISC_tool_num
1212 gcpy if self.generatedxf == True:
1213 gcpy     if (large_square_tool_num > 0):
1214 gcpy         self.dxfllsqfilename = basefilename + str(
                large_square_tool_num) + ".dxf"
1215 gcpy #         print("Opening ", str(self.dxfllsqfilename))
1216 gcpy         self.dxfllsq = open(self.dxfllsqfilename, "w")
1217 gcpy     if (small_square_tool_num > 0):
1218 gcpy #         print("Opening small square")
1219 gcpy         self.dxfllsqfilename = basefilename + str(
                small_square_tool_num) + ".dxf"
1220 gcpy         self.dxfllsq = open(self.dxfllsqfilename, "w")
1221 gcpy     if (large_ball_tool_num > 0):
1222 gcpy #         print("Opening large ball")
1223 gcpy         self.dxfllbfilename = basefilename + str(
                large_ball_tool_num) + ".dxf"
1224 gcpy         self.dxfllb = open(self.dxfllbfilename, "w")
1225 gcpy     if (small_ball_tool_num > 0):
1226 gcpy #         print("Opening small ball")
1227 gcpy         self.dxfllbfilename = basefilename + str(
                small_ball_tool_num) + ".dxf"
1228 gcpy         self.dxfllb = open(self.dxfllbfilename, "w")
1229 gcpy     if (large_V_tool_num > 0):
1230 gcpy #         print("Opening large V")
1231 gcpy         self.dxfllvfilename = basefilename + str(
                large_V_tool_num) + ".dxf"
1232 gcpy         self.dxfllv = open(self.dxfllvfilename, "w")
1233 gcpy     if (small_V_tool_num > 0):
1234 gcpy #         print("Opening small V")
1235 gcpy         self.dxfllvfilename = basefilename + str(
                small_V_tool_num) + ".dxf"
1236 gcpy         self.dxfllv = open(self.dxfllvfilename, "w")
1237 gcpy     if (DT_tool_num > 0):
1238 gcpy #         print("Opening DT")
1239 gcpy         self.dxfllDTfilename = basefilename + str(DT_tool_num
                ) + ".dxf"
1240 gcpy         self.dxfllDT = open(self.dxfllDTfilename, "w")
1241 gcpy     if (KH_tool_num > 0):
1242 gcpy #         print("Opening KH")
1243 gcpy         self.dxfllKHfilename = basefilename + str(KH_tool_num
                ) + ".dxf"
1244 gcpy         self.dxfllKH = open(self.dxfllKHfilename, "w")
1245 gcpy     if (Roundover_tool_num > 0):
1246 gcpy #         print("Opening Rt")
1247 gcpy         self.dxfllRtfilename = basefilename + str(
                Roundover_tool_num) + ".dxf"
1248 gcpy         self.dxfllRt = open(self.dxfllRtfilename, "w")
1249 gcpy     if (MISC_tool_num > 0):
1250 gcpy #         print("Opening Mt")
1251 gcpy         self.dxfllMtfilename = basefilename + str(
                MISC_tool_num) + ".dxf"
1252 gcpy         self.dxfllMt = open(self.dxfllMtfilename, "w")
```

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

```
1253 gcpy     if (large_square_tool_num > 0):
1254 gcpy         self.dxfpreamble(large_square_tool_num)
1255 gcpy     if (small_square_tool_num > 0):
1256 gcpy         self.dxfpreamble(small_square_tool_num)
1257 gcpy     if (large_ball_tool_num > 0):
1258 gcpy         self.dxfpreamble(large_ball_tool_num)
1259 gcpy     if (small_ball_tool_num > 0):
1260 gcpy         self.dxfpreamble(small_ball_tool_num)
1261 gcpy     if (large_V_tool_num > 0):
1262 gcpy         self.dxfpreamble(large_V_tool_num)
1263 gcpy     if (small_V_tool_num > 0):
1264 gcpy         self.dxfpreamble(small_V_tool_num)
1265 gcpy     if (DT_tool_num > 0):
1266 gcpy         self.dxfpreamble(DT_tool_num)
1267 gcpy     if (KH_tool_num > 0):
1268 gcpy         self.dxfpreamble(KH_tool_num)
1269 gcpy     if (Roundover_tool_num > 0):
1270 gcpy         self.dxfpreamble(Roundover_tool_num)
```

```
1271 gcpy          if (MISC_tool_num > 0):
1272 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,

3.7.2 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
```

3.7.2.1 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

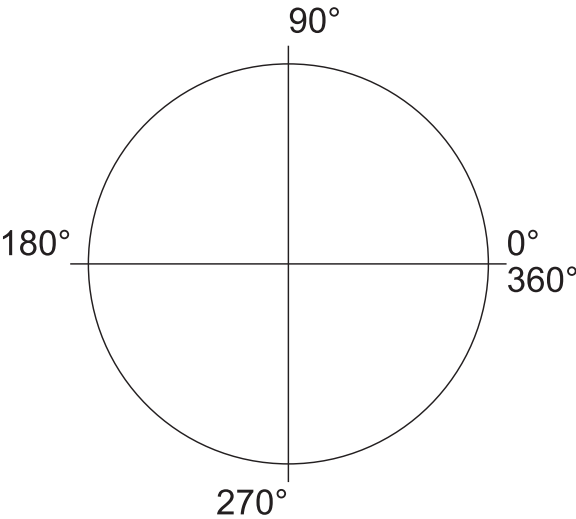
dxfpreamble This module requires that the tool number be passed in, and after writing out dxfpreamble, that value will be used to write out to the appropriate file with a series of if statements.

```
1274 gcpy      def dxfpreamble(self, tn):
1275 gcpy #          self.writedxf(tn, str(tn))
1276 gcpy          self.writedxf(tn, "0")
1277 gcpy          self.writedxf(tn, "SECTION")
1278 gcpy          self.writedxf(tn, "2")
1279 gcpy          self.writedxf(tn, "ENTITIES")
```

3.7.2.1.1 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:

```
1281 gcpy      def setdxfcOLOR(self, color):
1282 gcpy          self.dxfcolor = color
1283 gcpy          self.cutcolor = color
1284 gcpy
1285 gcpy      def writedxfcOLOR(self, tn):
1286 gcpy          self.writedxf(tn, "8")
1287 gcpy          if (self.dxfcolor == "Black"):
1288 gcpy              self.writedxf(tn, "Layer_Black")
1289 gcpy          if (self.dxfcolor == "Red"):
1290 gcpy              self.writedxf(tn, "Layer_Red")
1291 gcpy          if (self.dxfcolor == "Yellow"):
1292 gcpy              self.writedxf(tn, "Layer_Yellow")
1293 gcpy          if (self.dxfcolor == "Green"):
1294 gcpy              self.writedxf(tn, "Layer_Green")
1295 gcpy          if (self.dxfcolor == "Cyan"):
1296 gcpy              self.writedxf(tn, "Layer_Cyan")
1297 gcpy          if (self.dxfcolor == "Blue"):
1298 gcpy              self.writedxf(tn, "Layer_Blue")
1299 gcpy          if (self.dxfcolor == "Magenta"):
1300 gcpy              self.writedxf(tn, "Layer_Magenta")
1301 gcpy          if (self.dxfcolor == "White"):
1302 gcpy              self.writedxf(tn, "Layer_White")
1303 gcpy          if (self.dxfcolor == "Dark_Gray"):
1304 gcpy              self.writedxf(tn, "Layer_Dark_Gray")
1305 gcpy          if (self.dxfcolor == "Light_Gray"):
1306 gcpy              self.writedxf(tn, "Layer_Light_Gray")
1307 gcpy
1308 gcpy          self.writedxf(tn, "62")
1309 gcpy          if (self.dxfcolor == "Black"):
1310 gcpy              self.writedxf(tn, "0")
1311 gcpy          if (self.dxfcolor == "Red"):
1312 gcpy              self.writedxf(tn, "1")
1313 gcpy          if (self.dxfcolor == "Yellow"):
1314 gcpy              self.writedxf(tn, "2")
1315 gcpy          if (self.dxfcolor == "Green"):
1316 gcpy              self.writedxf(tn, "3")
1317 gcpy          if (self.dxfcolor == "Cyan"):
1318 gcpy              self.writedxf(tn, "4")
1319 gcpy          if (self.dxfcolor == "Blue"):
1320 gcpy              self.writedxf(tn, "5")
1321 gcpy          if (self.dxfcolor == "Magenta"):
1322 gcpy              self.writedxf(tn, "6")
1323 gcpy          if (self.dxfcolor == "White"):
1324 gcpy              self.writedxf(tn, "7")
1325 gcpy          if (self.dxfcolor == "Dark_Gray"):
1326 gcpy              self.writedxf(tn, "8")
1327 gcpy          if (self.dxfcolor == "Light_Gray"):
1328 gcpy              self.writedxf(tn, "9")
1329 gcpy
1330 gcpy
1331 gcpy
1332 gcpy
1333 gcpy #
1334 gcpy      self.writedxfcOLOR(tn)
1335 gcpy #
1336 gcpy      self.writedxf(tn, "10")
```

```
142 gcpSCAD module setdxfcOLOR(color){
143 gcpSCAD     gcp.setdxfcOLOR(color);
144 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.

```
1330 gcpy      def dxfline(self, tn, xbegin, ybegin, xend, yend):
1331 gcpy          self.writedxf(tn, "0")
1332 gcpy          self.writedxf(tn, "LINE")
1333 gcpy #
1334 gcpy          self.writedxfcOLOR(tn)
1335 gcpy #
1336 gcpy          self.writedxf(tn, "10")
```

```
1337 gcpy          self.writedxf(tn, str(xbegin))
1338 gcpy          self.writedxf(tn, "20")
1339 gcpy          self.writedxf(tn, str(ybegin))
1340 gcpy          self.writedxf(tn, "30")
1341 gcpy          self.writedxf(tn, "0.0")
1342 gcpy          self.writedxf(tn, "11")
1343 gcpy          self.writedxf(tn, str(xend))
1344 gcpy          self.writedxf(tn, "21")
1345 gcpy          self.writedxf(tn, str(yend))
1346 gcpy          self.writedxf(tn, "31")
1347 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:

```
1349 gcpy          def beginpolyline(self, tn):#, xbegin, ybegin
1350 gcpy          self.writedxf(tn, "0")
1351 gcpy          self.writedxf(tn, "POLYLINE")
1352 gcpy          self.writedxf(tn, "8")
1353 gcpy          self.writedxf(tn, "default")
1354 gcpy          self.writedxf(tn, "66")
1355 gcpy          self.writedxf(tn, "1")
1356 gcpy #
1357 gcpy          self.writedxfcolor(tn)
1358 gcpy #
1359 gcpy #          self.writedxf(tn, "10")
1360 gcpy #          self.writedxf(tn, str(xbegin))
1361 gcpy #          self.writedxf(tn, "20")
1362 gcpy #          self.writedxf(tn, str(ybegin))
1363 gcpy #          self.writedxf(tn, "30")
1364 gcpy #          self.writedxf(tn, "0.0")
1365 gcpy          self.writedxf(tn, "70")
1366 gcpy          self.writedxf(tn, "0")
```

then add as many vertexes as are wanted:

```
1368 gcpy          def addvertex(self, tn, xend, yend):
1369 gcpy          self.writedxf(tn, "0")
1370 gcpy          self.writedxf(tn, "VERTEX")
1371 gcpy          self.writedxf(tn, "8")
1372 gcpy          self.writedxf(tn, "default")
1373 gcpy          self.writedxf(tn, "70")
1374 gcpy          self.writedxf(tn, "32")
1375 gcpy          self.writedxf(tn, "10")
1376 gcpy          self.writedxf(tn, str(xend))
1377 gcpy          self.writedxf(tn, "20")
1378 gcpy          self.writedxf(tn, str(yend))
1379 gcpy          self.writedxf(tn, "30")
1380 gcpy          self.writedxf(tn, "0.0")
```

then end the sequence:

```
1382 gcpy          def closepolyline(self, tn):
1383 gcpy          self.writedxf(tn, "0")
1384 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).

```
1386 gcpy          def dxfarc(self, tn, xcenter, ycenter, radius, anglebegin,
                             endangle):
1387 gcpy          if (self.generatedxf == True):
1388 gcpy              self.writedxf(tn, "0")
1389 gcpy              self.writedxf(tn, "ARC")
1390 gcpy #
1391 gcpy          self.writedxfcolor(tn)
1392 gcpy #
1393 gcpy          self.writedxf(tn, "10")
1394 gcpy          self.writedxf(tn, str(xcenter))
1395 gcpy          self.writedxf(tn, "20")
1396 gcpy          self.writedxf(tn, str(ycenter))
1397 gcpy          self.writedxf(tn, "40")
1398 gcpy          self.writedxf(tn, str(radius))
```

```

1399 gcpy          self.writedxf(tn, "50")
1400 gcpy          self.writedxf(tn, str(anglebegin))
1401 gcpy          self.writedxf(tn, "51")
1402 gcpy          self.writedxf(tn, str(endangle))
1403 gcpy
1404 gcpy          def gcodearc(self, tn, xcenter, ycenter, radius, anglebegin,
endangle):
1405 gcpy              if (self.generategcode == True):
1406 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.

```

1408 gcpy          def cutarcNECCdxf(self, ex, ey, ez, xcenter, ycenter, radius):
1409 gcpy          #          global toolpath
1410 gcpy          #          toolpath = self.currenttool()
1411 gcpy          #          toolpath = toolpath.translate([self.xpos(), self.ypos(),
self.zpos()])
1412 gcpy          self.dxfarc(self.currenttoolnumber(), xcenter, ycenter,
radius, 0, 90)
1413 gcpy          if (self.zpos == ez):
1414 gcpy              self.settzpos(0)
1415 gcpy          else:
1416 gcpy              self.settzpos((self.zpos()-ez)/90)
1417 gcpy          #          self.setxpos(ex)
1418 gcpy          #          self.setypos(ey)
1419 gcpy          #          self.setzpos(ez)
1420 gcpy          #          if self.generatepaths == True:
1421 gcpy          #              print("Unioning cutarcNECCdxf toolpath")
1422 gcpy          self.arcloop(1, 90, xcenter, ycenter, radius)
1423 gcpy          #          self.toolpaths = self.toolpaths.union(toolpath)
1424 gcpy          #          else:
1425 gcpy          #              toolpath = self.arcloop(1, 90, xcenter, ycenter,
radius)
1426 gcpy          #              print("Returning cutarcNECCdxf toolpath")
1427 gcpy          return toolpath
1428 gcpy
1429 gcpy          def cutarcNWCCdxf(self, ex, ey, ez, xcenter, ycenter, radius):
1430 gcpy          #          global toolpath
1431 gcpy          #          toolpath = self.currenttool()
1432 gcpy          #          toolpath = toolpath.translate([self.xpos(), self.ypos(),
self.zpos()])
1433 gcpy          self.dxfarc(self.currenttoolnumber(), xcenter, ycenter,
radius, 90, 180)
1434 gcpy          if (self.zpos == ez):
1435 gcpy              self.settzpos(0)
1436 gcpy          else:
1437 gcpy              self.settzpos((self.zpos()-ez)/90)
1438 gcpy          #          self.setxpos(ex)
1439 gcpy          #          self.setypos(ey)
1440 gcpy          #          self.setzpos(ez)
1441 gcpy          #          if self.generatepaths == True:
1442 gcpy          #              self.arcloop(91, 180, xcenter, ycenter, radius)
1443 gcpy          #              self.toolpaths = self.toolpaths.union(toolpath)
1444 gcpy          #          else:
1445 gcpy          toolpath = self.arcloop(91, 180, xcenter, ycenter, radius)
1446 gcpy          return toolpath
1447 gcpy
1448 gcpy          def cutarcSWCCdxf(self, ex, ey, ez, xcenter, ycenter, radius):
1449 gcpy          #          global toolpath
1450 gcpy          #          toolpath = self.currenttool()
1451 gcpy          #          toolpath = toolpath.translate([self.xpos(), self.ypos(),
self.zpos()])
1452 gcpy          self.dxfarc(self.currenttoolnumber(), xcenter, ycenter,
radius, 180, 270)
1453 gcpy          if (self.zpos == ez):
1454 gcpy              self.settzpos(0)
1455 gcpy          else:
1456 gcpy              self.settzpos((self.zpos()-ez)/90)
1457 gcpy          #          self.setxpos(ex)
1458 gcpy          #          self.setypos(ey)
1459 gcpy          #          self.setzpos(ez)
1460 gcpy          #          if self.generatepaths == True:
1461 gcpy          #              self.arcloop(181, 270, xcenter, ycenter, radius)
1462 gcpy          #              self.toolpaths = self.toolpaths.union(toolpath)
1463 gcpy          #          else:
1464 gcpy          toolpath = self.arcloop(181, 270, xcenter, ycenter,
radius)

```

```

1465 gcpy                return toolpath
1466 gcpy
1467 gcpy    def cutarcSECCdxf(self, ex, ey, ez, xcenter, ycenter, radius):
1468 gcpy #        global toolpath
1469 gcpy #        toolpath = self.currenttool()
1470 gcpy #        toolpath = toolpath.translate([self.xpos(), self.ypos(),
self.zpos()])
1471 gcpy        self.dxfarc(self.currenttoolnumber(), xcenter, ycenter,
radius, 270, 360)
1472 gcpy        if (self.zpos == ez):
1473 gcpy            self.settzpos(0)
1474 gcpy        else:
1475 gcpy            self.settzpos((self.zpos()-ez)/90)
1476 gcpy #        self.setxpos(ex)
1477 gcpy #        self.setypos(ey)
1478 gcpy #        self.setzpos(ez)
1479 gcpy        if self.generatepaths == True:
1480 gcpy            self.arcloop(271, 360, xcenter, ycenter, radius)
1481 gcpy #            self.toolpaths = self.toolpaths.union(toolpath)
1482 gcpy        else:
1483 gcpy            toolpath = self.arcloop(271, 360, xcenter, ycenter,
radius)
1484 gcpy            return toolpath
1485 gcpy
1486 gcpy    def cutarcNECWdxf(self, ex, ey, ez, xcenter, ycenter, radius):
1487 gcpy #        global toolpath
1488 gcpy #        toolpath = self.currenttool()
1489 gcpy #        toolpath = toolpath.translate([self.xpos(), self.ypos(),
self.zpos()])
1490 gcpy        self.dxfarc(self.currenttoolnumber(), xcenter, ycenter,
radius, 0, 90)
1491 gcpy        if (self.zpos == ez):
1492 gcpy            self.settzpos(0)
1493 gcpy        else:
1494 gcpy            self.settzpos((self.zpos()-ez)/90)
1495 gcpy #        self.setxpos(ex)
1496 gcpy #        self.setypos(ey)
1497 gcpy #        self.setzpos(ez)
1498 gcpy        if self.generatepaths == True:
1499 gcpy            self.narcloop(89, 0, xcenter, ycenter, radius)
1500 gcpy #            self.toolpaths = self.toolpaths.union(toolpath)
1501 gcpy        else:
1502 gcpy            toolpath = self.narcloop(89, 0, xcenter, ycenter,
radius)
1503 gcpy            return toolpath
1504 gcpy
1505 gcpy    def cutarcSECWdxf(self, ex, ey, ez, xcenter, ycenter, radius):
1506 gcpy #        global toolpath
1507 gcpy #        toolpath = self.currenttool()
1508 gcpy #        toolpath = toolpath.translate([self.xpos(), self.ypos(),
self.zpos()])
1509 gcpy        self.dxfarc(self.currenttoolnumber(), xcenter, ycenter,
radius, 270, 360)
1510 gcpy        if (self.zpos == ez):
1511 gcpy            self.settzpos(0)
1512 gcpy        else:
1513 gcpy            self.settzpos((self.zpos()-ez)/90)
1514 gcpy #        self.setxpos(ex)
1515 gcpy #        self.setypos(ey)
1516 gcpy #        self.setzpos(ez)
1517 gcpy        if self.generatepaths == True:
1518 gcpy            self.narcloop(359, 270, xcenter, ycenter, radius)
1519 gcpy #            self.toolpaths = self.toolpaths.union(toolpath)
1520 gcpy        else:
1521 gcpy            toolpath = self.narcloop(359, 270, xcenter, ycenter,
radius)
1522 gcpy            return toolpath
1523 gcpy
1524 gcpy    def cutarcSWCWdxf(self, ex, ey, ez, xcenter, ycenter, radius):
1525 gcpy #        global toolpath
1526 gcpy #        toolpath = self.currenttool()
1527 gcpy #        toolpath = toolpath.translate([self.xpos(), self.ypos(),
self.zpos()])
1528 gcpy        self.dxfarc(self.currenttoolnumber(), xcenter, ycenter,
radius, 180, 270)
1529 gcpy        if (self.zpos == ez):
1530 gcpy            self.settzpos(0)
1531 gcpy        else:

```

```
1532 gcpy          self.settzpos((self.zpos()-ez)/90)
1533 gcpy #        self.setxpos(ex)
1534 gcpy #        self.setypos(ey)
1535 gcpy #        self.setzpos(ez)
1536 gcpy          if self.generatepaths == True:
1537 gcpy              self.narcloop(269, 180, xcenter, ycenter, radius)
1538 gcpy #          self.toolpaths = self.toolpaths.union(toolpath)
1539 gcpy          else:
1540 gcpy              toolpath = self.narcloop(269, 180, xcenter, ycenter,
1541                                     radius)
1542 gcpy              return toolpath
1543 gcpy          def cutarcNWCWdxf(self, ex, ey, ez, xcenter, ycenter, radius):
1544 gcpy #              global toolpath
1545 gcpy #              toolpath = self.currentttool()
1546 gcpy #              toolpath = toolpath.translate([self.xpos(), self.ypos(),
self.zpos()])
1547 gcpy              self.dxfarc(self.currenttoolnumber(), xcenter, ycenter,
1548                                     radius, 90, 180)
1549 gcpy              if (self.zpos == ez):
1550 gcpy                  self.settzpos(0)
1551 gcpy              else:
1552 gcpy                  self.settzpos((self.zpos()-ez)/90)
1553 gcpy #              self.setxpos(ex)
1554 gcpy #              self.setypos(ey)
1555 gcpy #              self.setzpos(ez)
1556 gcpy              if self.generatepaths == True:
1557 gcpy                  self.narcloop(179, 90, xcenter, ycenter, radius)
1558 gcpy                  self.toolpaths = self.toolpaths.union(toolpath)
1559 gcpy              else:
1560 gcpy                  toolpath = self.narcloop(179, 90, xcenter, ycenter,
1561                                     radius)
1562 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

```
1562 gcpy          def arcCCgc(self, ex, ey, ez, xcenter, ycenter, radius):
1563 gcpy              self.writegc("G03_X", str(ex), "Y", str(ey), "Z", str(ez)
, "R", str(radius))
1564 gcpy
1565 gcpy          def arcCWgc(self, ex, ey, ez, xcenter, ycenter, radius):
1566 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:

```
1568 gcpy          def cutarcNECCdxfgc(self, ex, ey, ez, xcenter, ycenter, radius)
:
1569 gcpy              self.arcCCgc(ex, ey, ez, xcenter, ycenter, radius)
1570 gcpy              if self.generatepaths == True:
1571 gcpy                  self.cutarcNECCdxf(ex, ey, ez, xcenter, ycenter, radius
)
1572 gcpy              else:
1573 gcpy                  return self.cutarcNECCdxf(ex, ey, ez, xcenter, ycenter,
radius)
1574 gcpy
1575 gcpy          def cutarcNWCCdxfgc(self, ex, ey, ez, xcenter, ycenter, radius)
:
1576 gcpy              self.arcCCgc(ex, ey, ez, xcenter, ycenter, radius)
1577 gcpy              if self.generatepaths == False:
1578 gcpy                  return self.cutarcNWCCdxf(ex, ey, ez, xcenter, ycenter,
radius)
1579 gcpy
1580 gcpy          def cutarcSWCCdxfgc(self, ex, ey, ez, xcenter, ycenter, radius)
:
1581 gcpy              self.arcCCgc(ex, ey, ez, xcenter, ycenter, radius)
1582 gcpy              if self.generatepaths == False:
1583 gcpy                  return self.cutarcSWCCdxf(ex, ey, ez, xcenter, ycenter,
radius)
1584 gcpy
1585 gcpy          def cutarcSECCdxfgc(self, ex, ey, ez, xcenter, ycenter, radius)
:
```

```

1586 gcpy      self.arcCCgc(ex, ey, ez, xcenter, ycenter, radius)
1587 gcpy      if self.generatepaths == False:
1588 gcpy          return self.cutarcSECCdxfc(ex, ey, ez, xcenter, ycenter,
            radius)

1589 gcpy
1590 gcpy      def cutarcNECWdxfgc(self, ex, ey, ez, xcenter, ycenter, radius)
            :
1591 gcpy          self.arcCWgc(ex, ey, ez, xcenter, ycenter, radius)
1592 gcpy          if self.generatepaths == False:
1593 gcpy              return self.cutarcNECWdxfc(ex, ey, ez, xcenter, ycenter,
            radius)

1594 gcpy
1595 gcpy      def cutarcSECWdxfgc(self, ex, ey, ez, xcenter, ycenter, radius)
            :
1596 gcpy          self.arcCWgc(ex, ey, ez, xcenter, ycenter, radius)
1597 gcpy          if self.generatepaths == False:
1598 gcpy              return self.cutarcSECWdxfc(ex, ey, ez, xcenter, ycenter,
            radius)

1599 gcpy
1600 gcpy      def cutarcSWCWdxfgc(self, ex, ey, ez, xcenter, ycenter, radius)
            :
1601 gcpy          self.arcCWgc(ex, ey, ez, xcenter, ycenter, radius)
1602 gcpy          if self.generatepaths == False:
1603 gcpy              return self.cutarcSWCWdxfc(ex, ey, ez, xcenter, ycenter,
            radius)

1604 gcpy
1605 gcpy      def cutarcNWCWdxfgc(self, ex, ey, ez, xcenter, ycenter, radius)
            :
1606 gcpy          self.arcCWgc(ex, ey, ez, xcenter, ycenter, radius)
1607 gcpy          if self.generatepaths == False:
1608 gcpy              return self.cutarcNWCWdxfc(ex, ey, ez, xcenter, ycenter,
            radius)

1609 gcpy

```

```

146 gcpscad module cutarcNECCdxfgc(ex, ey, ez, xcenter, ycenter, radius){
147 gcpscad     gcp.cutarcNECCdxfgc(ex, ey, ez, xcenter, ycenter, radius);
148 gcpscad }
149 gcpscad
150 gcpscad module cutarcNWCCdxfgc(ex, ey, ez, xcenter, ycenter, radius){
151 gcpscad     gcp.cutarcNWCCdxfgc(ex, ey, ez, xcenter, ycenter, radius);
152 gcpscad }
153 gcpscad
154 gcpscad module cutarcSWCCdxfgc(ex, ey, ez, xcenter, ycenter, radius){
155 gcpscad     gcp.cutarcSWCCdxfgc(ex, ey, ez, xcenter, ycenter, radius);
156 gcpscad }
157 gcpscad
158 gcpscad module cutarcSECCdxfgc(ex, ey, ez, xcenter, ycenter, radius){
159 gcpscad     gcp.cutarcSECCdxfgc(ex, ey, ez, xcenter, ycenter, radius);
160 gcpscad }

```

3.7.3 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)

3.7.3.1 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 `closedxfile` additional information, depending on the file format. Note that these commands will need to be within the `gcodepreview` class.

```
1610 gcpy      def dxfpostamble(self, tn):
1611 gcpy #          self.writedxf(tn, str(tn))
1612 gcpy          self.writedxf(tn, "0")
1613 gcpy          self.writedxf(tn, "ENDSEC")
1614 gcpy          self.writedxf(tn, "0")
1615 gcpy          self.writedxf(tn, "EOF")

1617 gcpy      def gcodepostamble(self):
1618 gcpy          self.writegc("Z12.700")
1619 gcpy          self.writegc("M05")
1620 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.

```
1622 gcpy      def closegcodefile(self):
1623 gcpy          if self.generategcode == True:
1624 gcpy              self.gcodepostamble()
1625 gcpy              self.gc.close()

1627 gcpy      def closedxfile(self):
1628 gcpy          if self.generatedxfile == True:
1629 gcpy #              global dxfclose
1630 gcpy              self.dxfpostamble(-1)
1631 gcpy #              self.dxfclosed = True
1632 gcpy              self.dxf.close()

1634 gcpy      def closedxfiles(self):
1635 gcpy          if self.generatedxfiles == True:
1636 gcpy              if (self.large_square_tool_num > 0):
1637 gcpy                  self.dxfpostamble(self.large_square_tool_num)
1638 gcpy              if (self.small_square_tool_num > 0):
1639 gcpy                  self.dxfpostamble(self.small_square_tool_num)
1640 gcpy              if (self.large_ball_tool_num > 0):
1641 gcpy                  self.dxfpostamble(self.large_ball_tool_num)
1642 gcpy              if (self.small_ball_tool_num > 0):
1643 gcpy                  self.dxfpostamble(self.small_ball_tool_num)
1644 gcpy              if (self.large_V_tool_num > 0):
1645 gcpy                  self.dxfpostamble(self.large_V_tool_num)
1646 gcpy              if (self.small_V_tool_num > 0):
1647 gcpy                  self.dxfpostamble(self.small_V_tool_num)
1648 gcpy              if (self.DT_tool_num > 0):
1649 gcpy                  self.dxfpostamble(self.DT_tool_num)
1650 gcpy              if (self.KH_tool_num > 0):
1651 gcpy                  self.dxfpostamble(self.KH_tool_num)
1652 gcpy              if (self.Roundover_tool_num > 0):
1653 gcpy                  self.dxfpostamble(self.Roundover_tool_num)
1654 gcpy              if (self.MISC_tool_num > 0):
1655 gcpy                  self.dxfpostamble(self.MISC_tool_num)
1656 gcpy
1657 gcpy              if (self.large_square_tool_num > 0):
1658 gcpy                  self.dxfclose()
1659 gcpy              if (self.small_square_tool_num > 0):
1660 gcpy                  self.dxfclose()
1661 gcpy              if (self.large_ball_tool_num > 0):
1662 gcpy                  self.dxfclose()
1663 gcpy              if (self.small_ball_tool_num > 0):
1664 gcpy                  self.dxfclose()
1665 gcpy              if (self.large_V_tool_num > 0):
1666 gcpy                  self.dxfclose()
1667 gcpy              if (self.small_V_tool_num > 0):
1668 gcpy                  self.dxfclose()
1669 gcpy              if (self.DT_tool_num > 0):
1670 gcpy                  self.dxfclose()
1671 gcpy              if (self.KH_tool_num > 0):
1672 gcpy                  self.dxfclose()
1673 gcpy              if (self.Roundover_tool_num > 0):
1674 gcpy                  self.dxfclose()
1675 gcpy              if (self.MISC_tool_num > 0):
1676 gcpy                  self.dxfclose()
```

`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.

```

162 gpcscad module closegcodefile(){
163 gpcscad     gcp.closegcodefile();
164 gpcscad }
165 gpcscad
166 gpcscad module closedxfiles(){
167 gpcscad     gcp.closedxfiles();
168 gpcscad }
169 gpcscad
170 gpcscad module closedxfile(){
171 gpcscad     gcp.closedxfile();
172 gpcscad }

```

3.8 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 3.4.1.1) 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.

3.8.0.1 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.

3.8.0.2 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 sector¹
- two convex, one concave arc
- 4
 - rectangle (including square) — `dxfrectangle`, `dxfrectangleround`
 - parallelogram
 - rhombus
 - trapezoid/trapezium
 - kite
 - ring/annulus segment (straight line, concave arc, straight line, convex arc)
 - astroid (four concave arcs)
 - salinon (four semicircles)
 - three straight lines and one concave arc

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

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

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

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

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

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

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?

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

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

```
1678 gcpy      def dxfcircle(self, tool_num, xcenter, ycenter, radius):
1679 gcpy          self.dxfarc(tool_num, xcenter, ycenter, radius, 0, 90)
1680 gcpy          self.dxfarc(tool_num, xcenter, ycenter, radius, 90, 180)
1681 gcpy          self.dxfarc(tool_num, xcenter, ycenter, radius, 180, 270)
1682 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.

```
1684 gcpy      def cutcircleCC(self, xcenter, ycenter, bz, ez, radius):
1685 gcpy          self.setzpos(bz)
1686 gcpy          self.cutquarterCCNE(xcenter, ycenter + radius, self.zpos()
                                     + ez/4, radius)
1687 gcpy          self.cutquarterCCNW(xcenter - radius, ycenter, self.zpos()
                                     + ez/4, radius)
1688 gcpy          self.cutquarterCCSW(xcenter, ycenter - radius, self.zpos()
                                     + ez/4, radius)
1689 gcpy          self.cutquarterCCSE(xcenter + radius, ycenter, self.zpos()
                                     + ez/4, radius)
1690 gcpy
1691 gcpy      def cutcircleCCdxf(self, xcenter, ycenter, bz, ez, radius):
1692 gcpy          self.cutcircleCC(self, xcenter, ycenter, bz, ez, radius)
1693 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.

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

```
1695 gcpy      def dxfrectangle(self, tool_num, xorigin, yorigin, xwidth,
                               yheight, corners = "Square", radius = 6):
1696 gcpy          if corners == "Square":
1697 gcpy              self.dxfline(tool_num, xorigin, yorigin, xorigin +
                                   xwidth, yorigin)
1698 gcpy              self.dxfline(tool_num, xorigin + xwidth, yorigin,
                                   xorigin + xwidth, yorigin + yheight)
1699 gcpy              self.dxfline(tool_num, xorigin + xwidth, yorigin +
                                   yheight, xorigin, yorigin + yheight)
1700 gcpy              self.dxfline(tool_num, xorigin, yorigin + yheight,
                                   xorigin, yorigin)
1701 gcpy          elif corners == "Fillet":
1702 gcpy              self.dxfrectangleround(tool_num, xorigin, yorigin,
                                             xwidth, yheight, radius)
1703 gcpy          elif corners == "Chamfer":
1704 gcpy              self.dxfrectanglechamfer(tool_num, xorigin, yorigin,
                                             xwidth, yheight, radius)
1705 gcpy          elif corners == "Flipped_Fillet":
1706 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.

```
1708 gcpy      def dxfrectangleround(self, tool_num, xorigin, yorigin, xwidth,
                                     yheight, radius):
1709 gcpy      # begin section
1710 gcpy          self.writedxf(tool_num, "0")
1711 gcpy          self.writedxf(tool_num, "SECTION")
1712 gcpy          self.writedxf(tool_num, "2")
1713 gcpy          self.writedxf(tool_num, "ENTITIES")
1714 gcpy          self.writedxf(tool_num, "0")
1715 gcpy          self.writedxf(tool_num, "LWPOLYLINE")
1716 gcpy          self.writedxf(tool_num, "5")
1717 gcpy          self.writedxf(tool_num, "4E")
1718 gcpy          self.writedxf(tool_num, "100")
1719 gcpy          self.writedxf(tool_num, "AcDbEntity")
1720 gcpy          self.writedxf(tool_num, "8")
1721 gcpy          self.writedxf(tool_num, "0")
1722 gcpy          self.writedxf(tool_num, "6")
1723 gcpy          self.writedxf(tool_num, "ByLayer")
1724 gcpy      #
```

```

1725 gcpy          self.writedxfcolor(tool_num)
1726 gcpy #
1727 gcpy          self.writedxf(tool_num, "370")
1728 gcpy          self.writedxf(tool_num, "-1")
1729 gcpy          self.writedxf(tool_num, "100")
1730 gcpy          self.writedxf(tool_num, "AcDbPolyline")
1731 gcpy          self.writedxf(tool_num, "90")
1732 gcpy          self.writedxf(tool_num, "8")
1733 gcpy          self.writedxf(tool_num, "70")
1734 gcpy          self.writedxf(tool_num, "1")
1735 gcpy          self.writedxf(tool_num, "43")
1736 gcpy          self.writedxf(tool_num, "0")
1737 gcpy #1 upper right corner before arc (counter-clockwise)
1738 gcpy          self.writedxf(tool_num, "10")
1739 gcpy          self.writedxf(tool_num, str(xorigin + xwidth))
1740 gcpy          self.writedxf(tool_num, "20")
1741 gcpy          self.writedxf(tool_num, str(yorigin + yheight - radius))
1742 gcpy          self.writedxf(tool_num, "42")
1743 gcpy          self.writedxf(tool_num, "0.414213562373095")
1744 gcpy #2 upper right corner after arc
1745 gcpy          self.writedxf(tool_num, "10")
1746 gcpy          self.writedxf(tool_num, str(xorigin + xwidth - radius))
1747 gcpy          self.writedxf(tool_num, "20")
1748 gcpy          self.writedxf(tool_num, str(yorigin + yheight))
1749 gcpy #3 upper left corner before arc (counter-clockwise)
1750 gcpy          self.writedxf(tool_num, "10")
1751 gcpy          self.writedxf(tool_num, str(xorigin + radius))
1752 gcpy          self.writedxf(tool_num, "20")
1753 gcpy          self.writedxf(tool_num, str(yorigin + yheight))
1754 gcpy          self.writedxf(tool_num, "42")
1755 gcpy          self.writedxf(tool_num, "0.414213562373095")
1756 gcpy #4 upper left corner after arc
1757 gcpy          self.writedxf(tool_num, "10")
1758 gcpy          self.writedxf(tool_num, str(xorigin))
1759 gcpy          self.writedxf(tool_num, "20")
1760 gcpy          self.writedxf(tool_num, str(yorigin + yheight - radius))
1761 gcpy #5 lower left corner before arc (counter-clockwise)
1762 gcpy          self.writedxf(tool_num, "10")
1763 gcpy          self.writedxf(tool_num, str(xorigin))
1764 gcpy          self.writedxf(tool_num, "20")
1765 gcpy          self.writedxf(tool_num, str(yorigin + radius))
1766 gcpy          self.writedxf(tool_num, "42")
1767 gcpy          self.writedxf(tool_num, "0.414213562373095")
1768 gcpy #6 lower left corner after arc
1769 gcpy          self.writedxf(tool_num, "10")
1770 gcpy          self.writedxf(tool_num, str(xorigin + radius))
1771 gcpy          self.writedxf(tool_num, "20")
1772 gcpy          self.writedxf(tool_num, str(yorigin))
1773 gcpy #7 lower right corner before arc (counter-clockwise)
1774 gcpy          self.writedxf(tool_num, "10")
1775 gcpy          self.writedxf(tool_num, str(xorigin + xwidth - radius))
1776 gcpy          self.writedxf(tool_num, "20")
1777 gcpy          self.writedxf(tool_num, str(yorigin))
1778 gcpy          self.writedxf(tool_num, "42")
1779 gcpy          self.writedxf(tool_num, "0.414213562373095")
1780 gcpy #8 lower right corner after arc
1781 gcpy          self.writedxf(tool_num, "10")
1782 gcpy          self.writedxf(tool_num, str(xorigin + xwidth))
1783 gcpy          self.writedxf(tool_num, "20")
1784 gcpy          self.writedxf(tool_num, str(yorigin + radius))
1785 gcpy # end current section
1786 gcpy          self.writedxf(tool_num, "0")
1787 gcpy          self.writedxf(tool_num, "SEQEND")

```

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

```

1789 gcpy      def dxfrectanglechamfer(self, tool_num, xorigin, yorigin,
1790 gcpy          xwidth, yheight, radius):
1791 gcpy          self.dxflines(tool_num, xorigin + radius, yorigin, xorigin,
1792 gcpy          yorigin + radius)
1793 gcpy          self.dxflines(tool_num, xorigin, yorigin + yheight - radius,
1794 gcpy          xorigin + radius, yorigin + yheight)
1795 gcpy          self.dxflines(tool_num, xorigin + xwidth - radius, yorigin +
1796 gcpy          yheight, xorigin + xwidth, yorigin + yheight - radius)
1797 gcpy          self.dxflines(tool_num, xorigin + xwidth - radius, yorigin,
1798 gcpy          xorigin + xwidth, yorigin + radius)

```

```
1795 gcpy          self.dxfline(tool_num, xorigin + radius, yorigin, xorigin +
                    xwidth - radius, yorigin)
1796 gcpy          self.dxfline(tool_num, xorigin + xwidth, yorigin + radius,
                    xorigin + xwidth, yorigin + yheight - radius)
1797 gcpy          self.dxfline(tool_num, xorigin + xwidth - radius, yorigin +
                    yheight, xorigin + radius, yorigin + yheight)
1798 gcpy          self.dxfline(tool_num, xorigin, yorigin + yheight - radius,
                    xorigin, yorigin + radius)
```

Flipped Fillet:

```
1800 gcpy      def dxfrectangleflippedfillet(self, tool_num, xorigin, yorigin,
                    xwidth, yheight, radius):
1801 gcpy          self.dxfarc(tool_num, xorigin, yorigin, radius, 0, 90)
1802 gcpy          self.dxfarc(tool_num, xorigin + xwidth, yorigin, radius,
                    90, 180)
1803 gcpy          self.dxfarc(tool_num, xorigin + xwidth, yorigin + yheight,
                    radius, 180, 270)
1804 gcpy          self.dxfarc(tool_num, xorigin, yorigin + yheight, radius,
                    270, 360)
1805 gcpy
1806 gcpy          self.dxfline(tool_num, xorigin + radius, yorigin, xorigin +
                    xwidth - radius, yorigin)
1807 gcpy          self.dxfline(tool_num, xorigin + xwidth, yorigin + radius,
                    xorigin + xwidth, yorigin + yheight - radius)
1808 gcpy          self.dxfline(tool_num, xorigin + xwidth - radius, yorigin +
                    yheight, xorigin + radius, yorigin + yheight)
1809 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.

```
1811 gcpy      def cutrectangle(self, tool_num, bx, by, bz, xwidth, yheight,
                    zdepth):
1812 gcpy          self.cutline(bx, by, bz)
1813 gcpy          self.cutline(bx, by, bz - zdepth)
1814 gcpy          self.cutline(bx + xwidth, by, bz - zdepth)
1815 gcpy          self.cutline(bx + xwidth, by + yheight, bz - zdepth)
1816 gcpy          self.cutline(bx, by + yheight, bz - zdepth)
1817 gcpy          self.cutline(bx, by, bz - zdepth)
1818 gcpy
1819 gcpy      def cutrectangledxf(self, tool_num, bx, by, bz, xwidth, yheight
                    , zdepth):
1820 gcpy          self.cutrectangle(tool_num, bx, by, bz, xwidth, yheight,
                    zdepth)
1821 gcpy          self.dxfrectangle(tool_num, bx, by, xwidth, yheight, "
                    Square")
```

The rounded forms instantiate a radius:

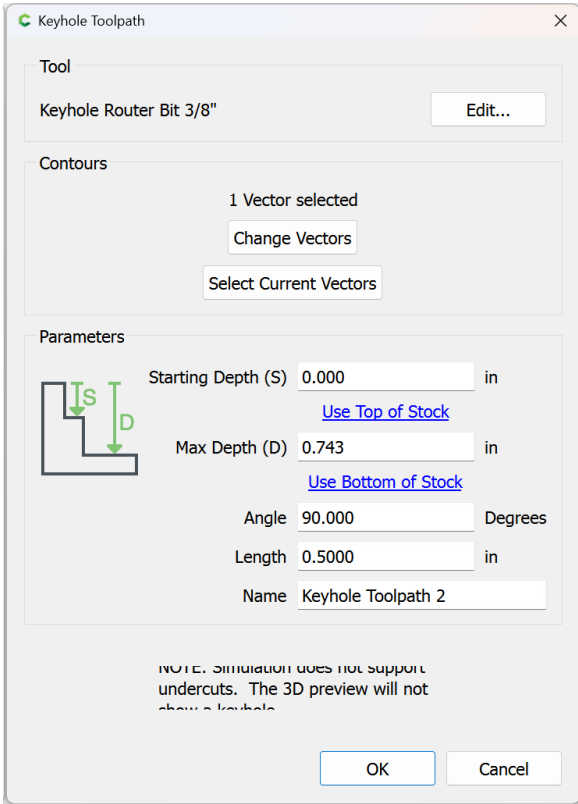
```
1823 gcpy      def cutrectangleround(self, tool_num, bx, by, bz, xwidth,
                    yheight, zdepth, radius):
1824 gcpy #          self.rapid(bx + radius, by, bz)
1825 gcpy          self.cutline(bx + radius, by, bz + zdepth)
1826 gcpy          self.cutline(bx + xwidth - radius, by, bz + zdepth)
1827 gcpy          self.cutquarterCCSE(bx + xwidth, by + radius, bz + zdepth,
                    radius)
1828 gcpy          self.cutline(bx + xwidth, by + yheight - radius, bz +
                    zdepth)
1829 gcpy          self.cutquarterCCNE(bx + xwidth - radius, by + yheight, bz
                    + zdepth, radius)
1830 gcpy          self.cutline(bx + radius, by + yheight, bz + zdepth)
1831 gcpy          self.cutquarterCCNW(bx, by + yheight - radius, bz + zdepth,
                    radius)
1832 gcpy          self.cutline(bx, by + radius, bz + zdepth)
1833 gcpy          self.cutquarterCCSW(bx + radius, by, bz + zdepth, radius)
```

```
1834 gcpy
1835 gcpy      def cutrectanglerounddx(self, tool_num, bx, by, bz, xwidth,
1836 gcpy          yheight, zdepth, radius):
1837 gcpy          self.cutrectangleround(tool_num, bx, by, bz, xwidth,
          yheight, zdepth, radius)
          self.dxfrectangleround(tool_num, bx, by, xwidth, yheight,
          radius)
```

3.8.0.2.3 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 3.5.0.1

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:

```
1839 gcpy      def cutkeyholegcdxf(self, kh_tool_num, kh_start_depth,
1840 gcpy          kh_max_depth, kht_direction, kh_distance):
1841 gcpy          toolpath = self.cutKHgcdxf(kh_tool_num, kh_start_depth,
1842 gcpy              kh_max_depth, 90, kh_distance)
1843 gcpy          elif (kht_direction == "S"):
1844 gcpy              toolpath = self.cutKHgcdxf(kh_tool_num, kh_start_depth,
1845 gcpy                  kh_max_depth, 270, kh_distance)
1846 gcpy          elif (kht_direction == "E"):
1847 gcpy              toolpath = self.cutKHgcdxf(kh_tool_num, kh_start_depth,
1848 gcpy                  kh_max_depth, 0, kh_distance)
1849 gcpy          elif (kht_direction == "W"):
1850 gcpy              toolpath = self.cutKHgcdxf(kh_tool_num, kh_start_depth,
1851 gcpy                  kh_max_depth, 180, kh_distance)
1852 gcpy          if self.generatepaths == True:
1853 gcpy              self.toolpaths = union([self.toolpaths, toolpath])
1854 gcpy          return toolpath
1855 gcpy          else:
1856 gcpy              return cube([0.01, 0.01, 0.01])

174 gcpscad module cutkeyholegcdxf(kh_tool_num, kh_start_depth, kh_max_depth,
175 gcpscad     kht_direction, kh_distance){
176 gcpscad     gcp.cutkeyholegcdxf(kh_tool_num, kh_start_depth, kh_max_depth,
177 gcpscad         kht_direction, kh_distance);
178 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:

```
1854 gcpy      def cutKHgcdxf(self, kh_tool_num, kh_start_depth, kh_max_depth,
1855 gcpy          kh_angle, kh_distance):
1856 gcpy          oXpos = self.xpos()
1857 gcpy          oYpos = self.ypos()
1858 gcpy          self.dxfKH(kh_tool_num, self.xpos(), self.ypos(),
1859 gcpy              kh_start_depth, kh_max_depth, kh_angle, kh_distance)
1860 gcpy          toolpath = self.cutline(self.xpos(), self.ypos(), -
1861 gcpy              kh_max_depth)
1862 gcpy          self.setxpos(oXpos)
1863 gcpy          self.setypos(oYpos)
1864 gcpy          if self.generatepaths == False:
1865 gcpy              return toolpath
1866 gcpy          else:
1867 gcpy              return cube([0.001, 0.001, 0.001])

1866 gcpy      def dxfKH(self, kh_tool_num, oXpos, oYpos, kh_start_depth,
1867 gcpy          kh_max_depth, kh_angle, kh_distance):
1868 gcpy          oXpos = self.xpos()
1869 gcpy          oYpos = self.ypos()
1870 gcpy          #Circle at entry hole
1871 gcpy          self.dxfarc(kh_tool_num, oXpos, oYpos, self.tool_radius(
1872 gcpy              kh_tool_num, 7), 0, 90)
1873 gcpy          self.dxfarc(kh_tool_num, oXpos, oYpos, self.tool_radius(
1874 gcpy              kh_tool_num, 7), 90, 180)
1875 gcpy          self.dxfarc(kh_tool_num, oXpos, oYpos, self.tool_radius(
1876 gcpy              kh_tool_num, 7), 180, 270)
1877 gcpy          self.dxfarc(kh_tool_num, oXpos, oYpos, self.tool_radius(
1878 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:

```
1874 gcpy #pre-calculate needed values
1875 gcpy     r = self.tool_radius(kh_tool_num, 7)
1876 gcpy     print(r)
1877 gcpy     rt = self.tool_radius(kh_tool_num, 1)
1878 gcpy     print(rt)
1879 gcpy     ro = math.sqrt((self.tool_radius(kh_tool_num, 1))**2-(self.
1880 gcpy         tool_radius(kh_tool_num, 7))**2)
1881 gcpy     print(ro)
1882 gcpy     angle = math.degrees(math.acos(ro/rt))
1883 gcpy #Outlines of entry hole and slot
```



```

1883 gcpy          if (kh_angle == 0):
1884 gcpy #Lower left of entry hole
1885 gcpy          self.dxfarc(kh_tool_num, self.xpos(), self.ypos(), self
                        .tool_radius(kh_tool_num, 1), 180, 270)
1886 gcpy #Upper left of entry hole
1887 gcpy          self.dxfarc(kh_tool_num, self.xpos(), self.ypos(), self
                        .tool_radius(kh_tool_num, 1), 90, 180)
1888 gcpy #Upper right of entry hole
1889 gcpy #          self.dxfarc(kh_tool_num, self.xpos(), self.ypos(), rt,
                        41.810, 90)
1890 gcpy          self.dxfarc(kh_tool_num, self.xpos(), self.ypos(), rt,
                        angle, 90)
1891 gcpy #Lower right of entry hole
1892 gcpy          self.dxfarc(kh_tool_num, self.xpos(), self.ypos(), rt,
                        270, 360-angle)
1893 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))))
1894 gcpy #Actual line of cut
1895 gcpy #          self.dxfline(kh_tool_num, self.xpos(), self.ypos(),
                        self.xpos()+kh_distance, self.ypos())
1896 gcpy #upper right of end of slot (kh_max_depth+4.36))/2
1897 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)
1898 gcpy #lower right of end of slot
1899 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)
1900 gcpy #upper right slot
1901 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))
1902 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)
1903 gcpy #end position at top of slot
1904 gcpy #lower right slot
1905 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))
1906 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)
1907 gcpy #end position at top of slot
1908 gcpy #          hull(){
1909 gcpy #              translate([xpos(), ypos(), zpos()]){
1910 gcpy #                  keyhole_shaft(6.35, 9.525);
1911 gcpy #              }
1912 gcpy #              translate([xpos(), ypos(), zpos()-kh_max_depth]){
1913 gcpy #                  keyhole_shaft(6.35, 9.525);
1914 gcpy #              }
1915 gcpy #          }
1916 gcpy #          hull(){
1917 gcpy #              translate([xpos(), ypos(), zpos()-kh_max_depth]){
1918 gcpy #                  keyhole_shaft(6.35, 9.525);
1919 gcpy #              }
1920 gcpy #              translate([xpos()+kh_distance, ypos(), zpos()-kh_max_depth])
                        {
1921 gcpy #                  keyhole_shaft(6.35, 9.525);
1922 gcpy #              }
1923 gcpy #          }
1924 gcpy #          cutwithfeed(getxpos(), getypos(), -kh_max_depth, feed);
1925 gcpy #          cutwithfeed(getxpos()+kh_distance, getypos(), -kh_max_depth,
                        feed);
1926 gcpy #          setxpos(getxpos()-kh_distance);
1927 gcpy #      } else if (kh_angle > 0 && kh_angle < 90) {

```

```

1928 gcpy #//echo(kh_angle);
1929 gcpy #   dxfarc(getxpos(), getypos(), tool_diameter(KH_tool_num, (
        kh_max_depth))/2, 90+kh_angle, 180+kh_angle, KH_tool_num);
1930 gcpy #   dxfarc(getxpos(), getypos(), tool_diameter(KH_tool_num, (
        kh_max_depth))/2, 180+kh_angle, 270+kh_angle, KH_tool_num);
1931 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);
1932 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);
1933 gcpy #dxfarc(getxpos()+(kh_distance*cos(kh_angle)),
1934 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);
1935 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);
1936 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))),
1937 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))),
1938 gcpy #   getxpos()+(kh_distance*cos(kh_angle))-((tool_diameter(KH_tool_num
        , (kh_max_depth+4.36))/2)*sin(kh_angle)),
1939 gcpy #   getypos()+(kh_distance*sin(kh_angle))+((tool_diameter(KH_tool_num
        , (kh_max_depth+4.36))/2)*cos(kh_angle)), KH_tool_num);
1940 gcpy #//echo("a", tool_diameter(KH_tool_num, (kh_max_depth+4.36))/2);
1941 gcpy #//echo("c", tool_diameter(KH_tool_num, (kh_max_depth))/2);
1942 gcpy #echo("Aangle", asin((tool_diameter(KH_tool_num, (kh_max_depth
        +4.36))/2)/(tool_diameter(KH_tool_num, (kh_max_depth))/2)));
1943 gcpy #//echo(kh_angle);
1944 gcpy #   cutwithfeed(getxpos()+(kh_distance*cos(kh_angle)), getypos()+(
        kh_distance*sin(kh_angle)), -kh_max_depth, feed);
1945 gcpy #           toolpath = toolpath.union(self.cutline(self.xpos()+
        kh_distance, self.ypos(), -kh_max_depth))
1946 gcpy         elif (kh_angle == 90):
1947 gcpy #Lower left of entry hole
1948 gcpy         self.dxfarc(kh_tool_num, oXpos, oYpos, self.tool_radius
        (kh_tool_num, 1), 180, 270)
1949 gcpy #Lower right of entry hole
1950 gcpy         self.dxfarc(kh_tool_num, oXpos, oYpos, self.tool_radius
        (kh_tool_num, 1), 270, 360)
1951 gcpy #left slot
1952 gcpy         self.dxfline(kh_tool_num, oXpos-r, oYpos+ro, oXpos-r,
        oYpos+kh_distance)
1953 gcpy #right slot
1954 gcpy         self.dxfline(kh_tool_num, oXpos+r, oYpos+ro, oXpos+r,
        oYpos+kh_distance)
1955 gcpy #upper left of end of slot
1956 gcpy         self.dxfarc(kh_tool_num, oXpos, oYpos+kh_distance, r,
        90, 180)
1957 gcpy #upper right of end of slot
1958 gcpy         self.dxfarc(kh_tool_num, oXpos, oYpos+kh_distance, r,
        0, 90)
1959 gcpy #Upper right of entry hole
1960 gcpy         self.dxfarc(kh_tool_num, oXpos, oYpos, rt, 0, 90-angle)
1961 gcpy #Upper left of entry hole
1962 gcpy         self.dxfarc(kh_tool_num, oXpos, oYpos, rt, 90+angle,
        180)
1963 gcpy #           toolpath = toolpath.union(self.cutline(oXpos, oYpos+
        kh_distance, -kh_max_depth))
1964 gcpy         elif (kh_angle == 180):
1965 gcpy #Lower right of entry hole
1966 gcpy         self.dxfarc(kh_tool_num, oXpos, oYpos, self.tool_radius
        (kh_tool_num, 1), 270, 360)
1967 gcpy #Upper right of entry hole
1968 gcpy         self.dxfarc(kh_tool_num, oXpos, oYpos, self.tool_radius
        (kh_tool_num, 1), 0, 90)
1969 gcpy #Upper left of entry hole
1970 gcpy         self.dxfarc(kh_tool_num, oXpos, oYpos, rt, 90, 180-
        angle)
1971 gcpy #Lower left of entry hole
1972 gcpy         self.dxfarc(kh_tool_num, oXpos, oYpos, rt, 180+angle,
        270)
1973 gcpy #upper slot

```

```

1974 gcpy                self.dxfline(kh_tool_num, oXpos-ro, oYpos-r, oXpos-
                           kh_distance, oYpos-r)
1975 gcpy #lower slot
1976 gcpy                self.dxfline(kh_tool_num, oXpos-ro, oYpos+r, oXpos-
                           kh_distance, oYpos+r)
1977 gcpy #upper left of end of slot
1978 gcpy                self.dxfarc(kh_tool_num, oXpos-kh_distance, oYpos, r,
                                   90, 180)
1979 gcpy #lower left of end of slot
1980 gcpy                self.dxfarc(kh_tool_num, oXpos-kh_distance, oYpos, r,
                                   180, 270)
1981 gcpy #                toolpath = toolpath.union(self.cutline(oXpos-
                           kh_distance, oYpos, -kh_max_depth))
1982 gcpy                elif (kh_angle == 270):
1983 gcpy #Upper left of entry hole
1984 gcpy                self.dxfarc(kh_tool_num, oXpos, oYpos, self.tool_radius
                                   (kh_tool_num, 1), 90, 180)
1985 gcpy #Upper right of entry hole
1986 gcpy                self.dxfarc(kh_tool_num, oXpos, oYpos, self.tool_radius
                                   (kh_tool_num, 1), 0, 90)
1987 gcpy #left slot
1988 gcpy                self.dxfline(kh_tool_num, oXpos-r, oYpos-ro, oXpos-r,
                                   oYpos-kh_distance)
1989 gcpy #right slot
1990 gcpy                self.dxfline(kh_tool_num, oXpos+r, oYpos-ro, oXpos+r,
                                   oYpos-kh_distance)
1991 gcpy #lower left of end of slot
1992 gcpy                self.dxfarc(kh_tool_num, oXpos, oYpos-kh_distance, r,
                                   180, 270)
1993 gcpy #lower right of end of slot
1994 gcpy                self.dxfarc(kh_tool_num, oXpos, oYpos-kh_distance, r,
                                   270, 360)
1995 gcpy #lower right of entry hole
1996 gcpy                self.dxfarc(kh_tool_num, oXpos, oYpos, rt, 180, 270-
                                   angle)
1997 gcpy #lower left of entry hole
1998 gcpy                self.dxfarc(kh_tool_num, oXpos, oYpos, rt, 270+angle,
                                   360)
1999 gcpy #                toolpath = toolpath.union(self.cutline(oXpos, oYpos-
                           kh_distance, -kh_max_depth))
2000 gcpy #                print(self.zpos())
2001 gcpy #                self.setxpos(oXpos)
2002 gcpy #                self.setypos(oYpos)
2003 gcpy #                if self.generatepaths == False:
2004 gcpy #                    return toolpath
2005 gcpy
2006 gcpy # } else if (kh_angle == 90) {
2007 gcpy # //Lower left of entry hole
2008 gcpy # dxfarc(getxpos(), getypos(), 9.525/2, 180, 270, KH_tool_num);
2009 gcpy # //Lower right of entry hole
2010 gcpy # dxfarc(getxpos(), getypos(), 9.525/2, 270, 360, KH_tool_num);
2011 gcpy # //Upper right of entry hole
2012 gcpy # dxfarc(getxpos(), getypos(), 9.525/2, 0, acos(tool_diameter(
KH_tool_num, 5)/tool_diameter(KH_tool_num, 1)), KH_tool_num);
2013 gcpy # //Upper left of entry hole
2014 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
);
2015 gcpy # //Actual line of cut
2016 gcpy # dxfline(getxpos(), getypos(), getxpos(), getypos()+kh_distance
);
2017 gcpy # //upper right of slot
2018 gcpy # dxfarc(getxpos(), getypos()+kh_distance, tool_diameter(
KH_tool_num, (kh_max_depth+4.36))/2, 0, 90, KH_tool_num);
2019 gcpy # //upper left of slot
2020 gcpy # dxfarc(getxpos(), getypos()+kh_distance, tool_diameter(
KH_tool_num, (kh_max_depth+6.35))/2, 90, 180, KH_tool_num);
2021 gcpy # //right of slot
2022 gcpy # dxfline(
2023 gcpy #     getxpos()+tool_diameter(KH_tool_num, (kh_max_depth))/2,
2024 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,
2025 gcpy #     getxpos()+tool_diameter(KH_tool_num, (kh_max_depth))/2,
2026 gcpy # //end position at top of slot
2027 gcpy #     getypos()+kh_distance,
2028 gcpy #     KH_tool_num);
2029 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);
2030 gcpy #   hull(){
2031 gcpy #       translate([xpos(), ypos(), zpos()]){
2032 gcpy #           keyhole_shaft(6.35, 9.525);
2033 gcpy #       }
2034 gcpy #       translate([xpos(), ypos(), zpos()-kh_max_depth]){
2035 gcpy #           keyhole_shaft(6.35, 9.525);
2036 gcpy #       }
2037 gcpy #   }
2038 gcpy #   hull(){
2039 gcpy #       translate([xpos(), ypos(), zpos()-kh_max_depth]){
2040 gcpy #           keyhole_shaft(6.35, 9.525);
2041 gcpy #       }
2042 gcpy #       translate([xpos(), ypos()+kh_distance, zpos()-kh_max_depth])
{
2043 gcpy #           keyhole_shaft(6.35, 9.525);
2044 gcpy #       }
2045 gcpy #   }
2046 gcpy #   cutwithfeed(getxpos(), getypos(), -kh_max_depth, feed);
2047 gcpy #   cutwithfeed(getxpos(), getypos()+kh_distance, -kh_max_depth,
feed);
2048 gcpy #   setypos(getypos()-kh_distance);
2049 gcpy # } else if (kh_angle == 180) {
2050 gcpy #     //Lower right of entry hole
2051 gcpy #     dxfarc(getxpos(), getypos(), 9.525/2, 270, 360, KH_tool_num);
2052 gcpy #     //Upper right of entry hole
2053 gcpy #     dxfarc(getxpos(), getypos(), 9.525/2, 0, 90, KH_tool_num);
2054 gcpy #     //Upper left of entry hole
2055 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);
2056 gcpy #     //Lower left of entry hole
2057 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
);
2058 gcpy #     //upper left of slot
2059 gcpy #     dxfarc(getxpos()-kh_distance, getypos(), tool_diameter(
KH_tool_num, (kh_max_depth+6.35))/2, 90, 180, KH_tool_num);
2060 gcpy #     //lower left of slot
2061 gcpy #     dxfarc(getxpos()-kh_distance, getypos(), tool_diameter(
KH_tool_num, (kh_max_depth+6.35))/2, 180, 270, KH_tool_num);
2062 gcpy #     //Actual line of cut
2063 gcpy #     dxfline(getxpos(), getypos(), getxpos()-kh_distance, getypos()
);
2064 gcpy #     //upper left slot
2065 gcpy #     dxfline(
2066 gcpy #         getxpos()-(math.sqrt((tool_diameter(KH_tool_num, 1)^2)-(
tool_diameter(KH_tool_num, 5)^2))/2),
2067 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,
2068 gcpy #         getxpos()-kh_distance,
2069 gcpy #         //end position at top of slot
2070 gcpy #         getypos()+tool_diameter(KH_tool_num, (kh_max_depth))/2,
KH_tool_num);
2071 gcpy #     //lower right slot
2072 gcpy #     dxfline(
2073 gcpy #         getxpos()-(math.sqrt((tool_diameter(KH_tool_num, 1)^2)-(
tool_diameter(KH_tool_num, 5)^2))/2),
2074 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,
2075 gcpy #         getxpos()-kh_distance,
2076 gcpy #         //end position at top of slot
2077 gcpy #         getypos()-tool_diameter(KH_tool_num, (kh_max_depth))/2,
KH_tool_num);
2078 gcpy #     hull(){
2079 gcpy #         translate([xpos(), ypos(), zpos()]){
2080 gcpy #             keyhole_shaft(6.35, 9.525);
2081 gcpy #         }
2082 gcpy #         translate([xpos(), ypos(), zpos()-kh_max_depth]){
2083 gcpy #             keyhole_shaft(6.35, 9.525);
2084 gcpy #         }
2085 gcpy #     }
2086 gcpy # }
2087 gcpy # hull(){
2088 gcpy #

```

```

2089 gcpy #         translate([xpos(), ypos(), zpos()-kh_max_depth]){
2090 gcpy #             keyhole_shaft(6.35, 9.525);
2091 gcpy #         }
2092 gcpy #         translate([xpos()-kh_distance, ypos(), zpos()-kh_max_depth])
2093 gcpy #     {
2094 gcpy #         keyhole_shaft(6.35, 9.525);
2095 gcpy #     }
2096 gcpy #     cutwithfeed(getxpos(), getypos(), -kh_max_depth, feed);
2097 gcpy #     cutwithfeed(getxpos()-kh_distance, getypos(), -kh_max_depth,
2098 gcpy #         feed);
2099 gcpy #     setxpos(getxpos()+kh_distance);
2100 gcpy # } else if (kh_angle == 270) {
2101 gcpy #     //Upper right of entry hole
2102 gcpy #     dxfarc(getxpos(), getypos(), 9.525/2, 0, 90, KH_tool_num);
2103 gcpy #     //Upper left of entry hole
2104 gcpy #     dxfarc(getxpos(), getypos(), 9.525/2, 90, 180, KH_tool_num);
2105 gcpy #     //lower right of slot
2106 gcpy #     dxfarc(getxpos(), getypos()-kh_distance, tool_diameter(
2107 gcpy #         KH_tool_num, (kh_max_depth+4.36))/2, 270, 360, KH_tool_num);
2108 gcpy #     //lower left of slot
2109 gcpy #     dxfarc(getxpos(), getypos()-kh_distance, tool_diameter(
2110 gcpy #         KH_tool_num, (kh_max_depth+4.36))/2, 180, 270, KH_tool_num);
2111 gcpy #     //Actual line of cut
2112 gcpy #     dxfline(getxpos(), getypos(), getxpos(), getypos()-kh_distance
2113 gcpy #     );
2114 gcpy #     //right of slot
2115 gcpy #     dxfline(
2116 gcpy #         getxpos()+tool_diameter(KH_tool_num, (kh_max_depth))/2,
2117 gcpy #         getypos()-(math.sqrt((tool_diameter(KH_tool_num, 1)^2)-(
2118 gcpy #         tool_diameter(KH_tool_num, 5)^2))/2), //( (kh_max_depth-6.34))
2119 gcpy #         /2)^2-(tool_diameter(KH_tool_num, (kh_max_depth-6.34))/2)^2,
2120 gcpy #         getxpos()+tool_diameter(KH_tool_num, (kh_max_depth))/2,
2121 gcpy #         //end position at top of slot
2122 gcpy #         getypos()-kh_distance,
2123 gcpy #         KH_tool_num);
2124 gcpy #     //left of slot
2125 gcpy #     dxfline(
2126 gcpy #         getxpos()-tool_diameter(KH_tool_num, (kh_max_depth))/2,
2127 gcpy #         getypos()-(math.sqrt((tool_diameter(KH_tool_num, 1)^2)-(
2128 gcpy #         tool_diameter(KH_tool_num, 5)^2))/2), //( (kh_max_depth-6.34))
2129 gcpy #         /2)^2-(tool_diameter(KH_tool_num, (kh_max_depth-6.34))/2)^2,
2130 gcpy #         getxpos()-tool_diameter(KH_tool_num, (kh_max_depth))/2,
2131 gcpy #         //end position at top of slot
2132 gcpy #         getypos()-kh_distance,
2133 gcpy #         KH_tool_num);
2134 gcpy #     //Lower right of entry hole
2135 gcpy #     dxfarc(getxpos(), getypos(), 9.525/2, 360-acos(tool_diameter(
2136 gcpy #         KH_tool_num, 5)/tool_diameter(KH_tool_num, 1)), 360, KH_tool_num
2137 gcpy #     );
2138 gcpy #     //Lower left of entry hole
2139 gcpy #     dxfarc(getxpos(), getypos(), 9.525/2, 180, 180+acos(
2140 gcpy #         tool_diameter(KH_tool_num, 5)/tool_diameter(KH_tool_num, 1)),
2141 gcpy #         KH_tool_num);
2142 gcpy #     hull(){
2143 gcpy #         translate([xpos(), ypos(), zpos()]){
2144 gcpy #             keyhole_shaft(6.35, 9.525);
2145 gcpy #         }
2146 gcpy #         translate([xpos(), ypos(), zpos()-kh_max_depth]){
2147 gcpy #             keyhole_shaft(6.35, 9.525);
2148 gcpy #         }
2149 gcpy #     }
2150 gcpy #     hull(){
2151 gcpy #         translate([xpos(), ypos(), zpos()-kh_max_depth]){
2152 gcpy #             keyhole_shaft(6.35, 9.525);
2153 gcpy #         }
2154 gcpy #         cutwithfeed(getxpos(), getypos(), -kh_max_depth, feed);
2155 gcpy #         cutwithfeed(getxpos(), getypos()-kh_distance, -kh_max_depth,
2156 gcpy #             feed);
2157 gcpy #         setypos(getypos()+kh_distance);
2158 gcpy #     }
2159 gcpy # }

```

3.8.0.2.4 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:

```
2152 gcpy      def cut_pins(self, Joint_Width, stockZthickness,
2153 gcpy          Number_of_Dovetails, Spacing, Proportion, DTT_diameter,
                DTT_angle):
2154 gcpy          DT0 = Tan(math.radians(DTT_angle)) * (stockZthickness *
2155 gcpy              Proportion)
2156 gcpy          DTR = DTT_diameter/2 - DT0
2157 gcpy          cpr = self.rapidXY(0, stockZthickness + Spacing/2)
2158 gcpy          ctp = self.cutlinedxfgc(self.xpos(), self.ypos(), -
                stockZthickness * Proportion)
2159 gcpy          #      ctp = ctp.union(self.cutlinedxfgc(Joint_Width / (
2160 gcpy              Number_of_Dovetails * 2), self.ypos(), -stockZthickness *
                Proportion))
2161 gcpy          i = 1
2162 gcpy          while i < Number_of_Dovetails * 2:
2163 gcpy              print(i)
2164 gcpy              ctp = ctp.union(self.cutlinedxfgc(i * (Joint_Width / (
2165 gcpy                  Number_of_Dovetails * 2)), self.ypos(), -
2166 gcpy                      stockZthickness * Proportion))
2167 gcpy              ctp = ctp.union(self.cutlinedxfgc(i * (Joint_Width / (
2168 gcpy                  Number_of_Dovetails * 2)), (stockZthickness +
2169 gcpy                      Spacing) + (stockZthickness * Proportion) - (
2170 gcpy                          DTT_diameter/2), -(stockZthickness * Proportion)))
2171 gcpy              ctp = ctp.union(self.cutlinedxfgc(i * (Joint_Width / (
2172 gcpy                  Number_of_Dovetails * 2)), stockZthickness + Spacing
2173 gcpy                      /2, -(stockZthickness * Proportion)))
2174 gcpy              ctp = ctp.union(self.cutlinedxfgc((i + 1) * (
2175 gcpy                  Joint_Width / (Number_of_Dovetails * 2)),
2176 gcpy                      stockZthickness + Spacing/2, -(stockZthickness *
2177 gcpy                          Proportion)))
2178 gcpy              self.dxfrectangleround(self.currenttoolnumber(),
2179 gcpy                  i * (Joint_Width / (Number_of_Dovetails * 2))-DTR,
2180 gcpy                      stockZthickness + (Spacing/2) - DTR,
2181 gcpy                      DTR * 2,
2182 gcpy                      (stockZthickness * Proportion) + Spacing/2 + DTR *
2183 gcpy                          2 - (DTT_diameter/2),
2184 gcpy                      DTR)
2185 gcpy              i += 2
2186 gcpy              self.rapidZ(0)
2187 gcpy              return ctp
```

and

```
2175 gcpy      def cut_tails(self, Joint_Width, stockZthickness,
2176 gcpy          Number_of_Dovetails, Spacing, Proportion, DTT_diameter,
                DTT_angle):
2177 gcpy          DT0 = Tan(math.radians(DTT_angle)) * (stockZthickness *
2178 gcpy              Proportion)
2179 gcpy          DTR = DTT_diameter/2 - DT0
2180 gcpy          cpr = self.rapidXY(0, 0)
2181 gcpy          ctp = self.cutlinedxfgc(self.xpos(), self.ypos(), -
                stockZthickness * Proportion)
2182 gcpy          ctp = ctp.union(self.cutlinedxfgc(
2183 gcpy              Joint_Width / (Number_of_Dovetails * 2) - (DTT_diameter
2184 gcpy                  - DT0),
2185 gcpy                  self.ypos(),
2186 gcpy                  -stockZthickness * Proportion))
2187 gcpy          i = 1
2188 gcpy          while i < Number_of_Dovetails * 2:
2189 gcpy              ctp = ctp.union(self.cutlinedxfgc(
2190 gcpy                  i * (Joint_Width / (Number_of_Dovetails * 2)) - (
2191 gcpy                      DTT_diameter - DT0),
2192 gcpy                      stockZthickness * Proportion - DTT_diameter / 2,
```

```

2189 gcpy          -(stockZthickness * Proportion)))
2190 gcpy          ctp = ctp.union(self.cutarcCWdxf(180, 90,
2191 gcpy              i * (Joint_Width / (Number_of_Dovetails * 2)),
2192 gcpy              stockZthickness * Proportion - DTT_diameter / 2,
2193 gcpy #          self.ypos(),
2194 gcpy              DTT_diameter - DT0, 0, 1))
2195 gcpy          ctp = ctp.union(self.cutarcCWdxf(90, 0,
2196 gcpy              i * (Joint_Width / (Number_of_Dovetails * 2)),
2197 gcpy              stockZthickness * Proportion - DTT_diameter / 2,
2198 gcpy              DTT_diameter - DT0, 0, 1))
2199 gcpy          ctp = ctp.union(self.cutlinedxfgc(
2200 gcpy              i * (Joint_Width / (Number_of_Dovetails * 2)) + (
2201 gcpy                  DTT_diameter - DT0),
2202 gcpy              0,
2203 gcpy              -(stockZthickness * Proportion)))
2204 gcpy          ctp = ctp.union(self.cutlinedxfgc(
2205 gcpy              (i + 2) * (Joint_Width / (Number_of_Dovetails * 2))
2206 gcpy              - (DTT_diameter - DT0),
2207 gcpy              0,
2208 gcpy              -(stockZthickness * Proportion)))
2209 gcpy          i += 2
2210 gcpy          self.rapidZ(0)
2211 gcpy          self.rapidXY(0, 0)
2212 gcpy          ctp = ctp.union(self.cutlinedxfgc(self.xpos(), self.ypos(),
2213 gcpy              -stockZthickness * Proportion))
2214 gcpy          self.dxfarc(self.currenttoolnumber(), 0, 0, DTR, 180, 270)
2215 gcpy          self.dxfline(self.currenttoolnumber(), -DTR, 0, -DTR,
2216 gcpy              stockZthickness + DTR)
2217 gcpy          self.dxfarc(self.currenttoolnumber(), 0, stockZthickness +
2218 gcpy              DTR, DTR, 90, 180)
2219 gcpy          self.dxfline(self.currenttoolnumber(), 0, stockZthickness +
2220 gcpy              DTR * 2, Joint_Width, stockZthickness + DTR * 2)
2221 gcpy          i = 0
2222 gcpy          while i < Number_of_Dovetails * 2:
2223 gcpy              ctp = ctp.union(self.cutline(i * (Joint_Width / (
2224 gcpy                  Number_of_Dovetails * 2)), stockZthickness + DT0, -(
2225 gcpy                      stockZthickness * Proportion)))
2226 gcpy              ctp = ctp.union(self.cutline((i+2) * (Joint_Width / (
2227 gcpy                  Number_of_Dovetails * 2)), stockZthickness + DT0, -(
2228 gcpy                      stockZthickness * Proportion)))
2229 gcpy              ctp = ctp.union(self.cutline((i+2) * (Joint_Width / (
2230 gcpy                  Number_of_Dovetails * 2)), 0, -(stockZthickness *
2231 gcpy                      Proportion)))
2232 gcpy              self.dxfarc(self.currenttoolnumber(), i * (Joint_Width
2233 gcpy                  / (Number_of_Dovetails * 2)), 0, DTR, 270, 360)
2234 gcpy              self.dxfline(self.currenttoolnumber(),
2235 gcpy                  i * (Joint_Width / (Number_of_Dovetails * 2)) + DTR
2236 gcpy                  ,
2237 gcpy                  0,
2238 gcpy                  i * (Joint_Width / (Number_of_Dovetails * 2)) + DTR
2239 gcpy                  , stockZthickness * Proportion - DTT_diameter /
2240 gcpy                  2)
2241 gcpy              self.dxfarc(self.currenttoolnumber(), (i + 1) * (
2242 gcpy                  Joint_Width / (Number_of_Dovetails * 2)),
2243 gcpy                  stockZthickness * Proportion - DTT_diameter / 2, (
2244 gcpy                      Joint_Width / (Number_of_Dovetails * 2)) - DTR, 90,
2245 gcpy                      180)
2246 gcpy              self.dxfarc(self.currenttoolnumber(), (i + 1) * (
2247 gcpy                  Joint_Width / (Number_of_Dovetails * 2)),
2248 gcpy                  stockZthickness * Proportion - DTT_diameter / 2, (
2249 gcpy                      Joint_Width / (Number_of_Dovetails * 2)) - DTR, 0,
2250 gcpy                      90)
2251 gcpy              self.dxfline(self.currenttoolnumber(),
2252 gcpy                  (i + 2) * (Joint_Width / (Number_of_Dovetails * 2))
2253 gcpy                  - DTR,
2254 gcpy                  0,
2255 gcpy                  (i + 2) * (Joint_Width / (Number_of_Dovetails * 2))
2256 gcpy                  - DTR, stockZthickness * Proportion -
2257 gcpy                      DTT_diameter / 2)
2258 gcpy              self.dxfarc(self.currenttoolnumber(), (i + 2) * (
2259 gcpy                  Joint_Width / (Number_of_Dovetails * 2)), 0, DTR,
2260 gcpy                  180, 270)
2261 gcpy              i += 2
2262 gcpy              self.dxfarc(self.currenttoolnumber(), Joint_Width,
2263 gcpy                  stockZthickness + DTR, DTR, 0, 90)
2264 gcpy              self.dxfline(self.currenttoolnumber(), Joint_Width + DTR,
2265 gcpy                  stockZthickness + DTR, Joint_Width + DTR, 0)
2266 gcpy              self.dxfarc(self.currenttoolnumber(), Joint_Width, 0, DTR,

```

```
270, 360)
2236 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.

3.8.0.2.5 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:

```
2238 gcpy         def Full_Blind_Finger_Joint_square(self, bx, by, orientation,
                side, width, thickness, Number_of_Pins, largeVdiameter,
                smallDiameter, normalormirror = "Default"):
2239 gcpy         # Joint_Orientation = "Horizontal" "Even" == "Lower", "Odd" ==
                "Upper"
2240 gcpy         # Joint_Orientation = "Vertical" "Even" == "Left", "Odd" == "
                Right"
2241 gcpy         if (orientation == "Vertical"):
2242 gcpy             if (normalormirror == "Default" and side != "Both"):
2243 gcpy                 if (side == "Left"):
2244 gcpy                     normalormirror = "Even"
2245 gcpy                 if (side == "Right"):
2246 gcpy                     normalormirror = "Odd"
2247 gcpy         if (orientation == "Horizontal"):
2248 gcpy             if (normalormirror == "Default" and side != "Both"):
2249 gcpy                 if (side == "Lower"):
2250 gcpy                     normalormirror = "Even"
2251 gcpy                 if (side == "Upper"):
2252 gcpy                     normalormirror = "Odd"
2253 gcpy         Finger_Width = ((Number_of_Pins * 2) - 1) * smallDiameter *
                1.1
2254 gcpy         Finger_Origin = width/2 - Finger_Width/2
2255 gcpy         rapid = self.rapidZ(0)
2256 gcpy         self.setdxfcolor("Cyan")
2257 gcpy         rapid = rapid.union(self.rapidXY(bx, by))
2258 gcpy         toolpath = (self.Finger_Joint_square(bx, by, orientation,
                side, width, thickness, Number_of_Pins, Finger_Origin,
                smallDiameter))
2259 gcpy         if (orientation == "Vertical"):
2260 gcpy             if (side == "Both"):
2261 gcpy                 toolpath = self.cutrectanglerounddx(self.
                    currenttoolnum, bx - (thickness - smallDiameter
                    /2), by-smallDiameter/2, 0, (thickness * 2) -
                    smallDiameter, width+smallDiameter, (
                    smallDiameter / 2) / Tan(math.radians(45)),
                    smallDiameter/2)
2262 gcpy             if (side == "Left"):
2263 gcpy                 toolpath = self.cutrectanglerounddx(self.
                    currenttoolnum, bx - (smallDiameter/2), by-
                    smallDiameter/2, 0, thickness, width+
                    smallDiameter, ((smallDiameter / 2) / Tan(math.
                    radians(45))), smallDiameter/2)
2264 gcpy             if (side == "Right"):
2265 gcpy                 toolpath = self.cutrectanglerounddx(self.
                    currenttoolnum, bx - (thickness - smallDiameter
                    /2), by-smallDiameter/2, 0, thickness, width+
                    smallDiameter, ((smallDiameter / 2) / Tan(math.
                    radians(45))), smallDiameter/2)
```



```

2266 gcpy        toolpath = toolpath.union(self.Finger_Joint_square(bx, by,
2267 gcpy        orientation, side, width, thickness, Number_of_Pins,
2268 gcpy        Finger-Origin, smallDiameter))
2267 gcpy        if (orientation == "Horizontal"):
2268 gcpy            if (side == "Both"):
2269 gcpy                toolpath = self.cutrectanglerounddxf(
2270 gcpy                    self.currenttoolnum,
2271 gcpy                    bx-smallDiameter/2,
2272 gcpy                    by - (thickness - smallDiameter/2),
2273 gcpy                    0,
2274 gcpy                    width+smallDiameter,
2275 gcpy                    (thickness * 2) - smallDiameter,
2276 gcpy                    (smallDiameter / 2) / Tan(math.radians(45)),
2277 gcpy                    smallDiameter/2)
2278 gcpy            if (side == "Lower"):
2279 gcpy                toolpath = self.cutrectanglerounddxf(
2280 gcpy                    self.currenttoolnum,
2281 gcpy                    bx - (smallDiameter/2),
2282 gcpy                    by - smallDiameter/2,
2283 gcpy                    0,
2284 gcpy                    width+smallDiameter,
2285 gcpy                    thickness,
2286 gcpy                    ((smallDiameter / 2) / Tan(math.radians(45))),
2287 gcpy                    smallDiameter/2)
2288 gcpy            if (side == "Upper"):
2289 gcpy                toolpath = self.cutrectanglerounddxf(
2290 gcpy                    self.currenttoolnum,
2291 gcpy                    bx - smallDiameter/2,
2292 gcpy                    by - (thickness - smallDiameter/2),
2293 gcpy                    0,
2294 gcpy                    width+smallDiameter,
2295 gcpy                    thickness,
2296 gcpy                    ((smallDiameter / 2) / Tan(math.radians(45))),
2297 gcpy                    smallDiameter/2)
2298 gcpy        toolpath = toolpath.union(self.Finger_Joint_square(bx, by,
2299 gcpy        orientation, side, width, thickness, Number_of_Pins,
2300 gcpy        Finger-Origin, smallDiameter))
2301 gcpy        return toolpath
2301 gcpy
2301 gcpy    def Finger_Joint_square(self, bx, by, orientation, side, width,
2302 gcpy        thickness, Number_of_Pins, Finger-Origin, smallDiameter,
2303 gcpy        normalormirror = "Default"):
2304 gcpy        jointdepth = -(thickness - (smallDiameter / 2) / Tan(math.
2305 gcpy            radians(45)))
2306 gcpy        # Joint_Orientation = "Horizontal" "Even" == "Lower", "Odd" ==
2307 gcpy            "Upper"
2308 gcpy        # Joint_Orientation = "Vertical" "Even" == "Left", "Odd" == "
2309 gcpy            Right"
2310 gcpy        if (orientation == "Vertical"):
2311 gcpy            if (normalormirror == "Default" and side != "Both"):
2312 gcpy                if (side == "Left"):
2313 gcpy                    normalormirror = "Even"
2314 gcpy                if (side == "Right"):
2315 gcpy                    normalormirror = "Odd"
2316 gcpy        if (orientation == "Horizontal"):
2317 gcpy            if (normalormirror == "Default" and side != "Both"):
2318 gcpy                if (side == "Lower"):
2319 gcpy                    normalormirror = "Even"
2320 gcpy                if (side == "Upper"):
2321 gcpy                    normalormirror = "Odd"
2322 gcpy        radius = smallDiameter/2
2323 gcpy        jointwidth = thickness - smallDiameter
2324 gcpy        toolpath = self.currenttool()
2325 gcpy        rapid = self.rapidZ(0)
2326 gcpy        self.setdxfcolor("Blue")
2327 gcpy        toolpath = toolpath.union(self.cutlineZgcfeed(jointdepth
2328 gcpy            ,1000))
2329 gcpy        self.beginpolyline(self.currenttool())
2330 gcpy        if (orientation == "Vertical"):
2331 gcpy            rapid = rapid.union(self.rapidXY(bx, by + Finger-Origin
2332 gcpy            ))
2333 gcpy        self.addvertex(self.currenttoolnumber(), self.xpos(),
2334 gcpy            self.ypos())
2335 gcpy        toolpath = toolpath.union(self.cutlineZgcfeed(
2336 gcpy            jointdepth,1000))
2337 gcpy        i = 0
2338 gcpy        while i <= Number_of_Pins - 1:
2339 gcpy            if (side == "Right"):
2340 gcpy

```

```

2331 gcpy                toolpath = toolpath.union(self.cutvertexdxf(
                        self.xpos(), self.ypos() + smallDiameter +
                        radius/5, jointdepth))
2332 gcpy                if (side == "Left" or side == "Both"):
2333 gcpy                toolpath = toolpath.union(self.cutvertexdxf(
                        self.xpos(), self.ypos() + radius,
                        jointdepth))
2334 gcpy                toolpath = toolpath.union(self.cutvertexdxf(
                        self.xpos() + jointwidth, self.ypos(),
                        jointdepth))
2335 gcpy                toolpath = toolpath.union(self.cutvertexdxf(
                        self.xpos(), self.ypos() + radius/5,
                        jointdepth))
2336 gcpy                toolpath = toolpath.union(self.cutvertexdxf(
                        self.xpos() - jointwidth, self.ypos(),
                        jointdepth))
2337 gcpy                toolpath = toolpath.union(self.cutvertexdxf(
                        self.xpos(), self.ypos() + radius,
                        jointdepth))
2338 gcpy                if (side == "Left"):
2339 gcpy                toolpath = toolpath.union(self.cutvertexdxf(
                        self.xpos(), self.ypos() + smallDiameter +
                        radius/5, jointdepth))
2340 gcpy                if (side == "Right" or side == "Both"):
2341 gcpy                    if (i < (Number_of_Pins - 1)):
2342 gcpy                        # print(i)
2343 gcpy                        toolpath = toolpath.union(self.cutvertexdxf(
                                self.xpos(), self.ypos() + radius,
                                jointdepth))
2344 gcpy                        toolpath = toolpath.union(self.cutvertexdxf(
                                self.xpos() - jointwidth, self.ypos(),
                                jointdepth))
2345 gcpy                        toolpath = toolpath.union(self.cutvertexdxf(
                                self.xpos(), self.ypos() + radius/5,
                                jointdepth))
2346 gcpy                        toolpath = toolpath.union(self.cutvertexdxf(
                                self.xpos() + jointwidth, self.ypos(),
                                jointdepth))
2347 gcpy                        toolpath = toolpath.union(self.cutvertexdxf(
                                self.xpos(), self.ypos() + radius,
                                jointdepth))
2348 gcpy                    i += 1
2349 gcpy                # Joint_Orientation = "Horizontal" "Even" == "Lower", "Odd" ==
                        "Upper"
2350 gcpy                if (orientation == "Horizontal"):
2351 gcpy                    rapid = rapid.union(self.rapidXY(bx + Finger_Origin, by
                        ))
2352 gcpy                    self.addvertex(self.currenttoolnumber(), self.xpos(),
                        self.ypos())
2353 gcpy                    toolpath = toolpath.union(self.cutlineZgcfeed(
                        jointdepth,1000))
2354 gcpy                    i = 0
2355 gcpy                    while i <= Number_of_Pins - 1:
2356 gcpy                        if (side == "Upper"):
2357 gcpy                            toolpath = toolpath.union(self.cutvertexdxf(
                                    self.xpos() + smallDiameter + radius/5, self
                                    .ypos(), jointdepth))
2358 gcpy                        if (side == "Lower" or side == "Both"):
2359 gcpy                            toolpath = toolpath.union(self.cutvertexdxf(
                                    self.xpos() + radius, self.ypos(),
                                    jointdepth))
2360 gcpy                            toolpath = toolpath.union(self.cutvertexdxf(
                                    self.xpos(), self.ypos() + jointwidth,
                                    jointdepth))
2361 gcpy                            toolpath = toolpath.union(self.cutvertexdxf(
                                    self.xpos() + radius/5, self.ypos(),
                                    jointdepth))
2362 gcpy                            toolpath = toolpath.union(self.cutvertexdxf(
                                    self.xpos(), self.ypos() - jointwidth,
                                    jointdepth))
2363 gcpy                            toolpath = toolpath.union(self.cutvertexdxf(
                                    self.xpos() + radius, self.ypos(),
                                    jointdepth))
2364 gcpy                        if (side == "Lower"):
2365 gcpy                            toolpath = toolpath.union(self.cutvertexdxf(
                                    self.xpos() + smallDiameter + radius/5, self
                                    .ypos(), jointdepth))
2366 gcpy                        if (side == "Upper" or side == "Both"):

```

```

2367 gcpy                                if (i < (Number_of_Pins - 1)):
2368 gcpy                                #         print(i)
2369 gcpy                                toolpath = toolpath.union(self.cutvertexdxf
                                         (self.xpos() + radius, self.ypos(),
                                         jointdepth))
2370 gcpy                                toolpath = toolpath.union(self.cutvertexdxf
                                         (self.xpos(), self.ypos() - jointwidth,
                                         jointdepth))
2371 gcpy                                toolpath = toolpath.union(self.cutvertexdxf
                                         (self.xpos() + radius/5, self.ypos(),
                                         jointdepth))
2372 gcpy                                toolpath = toolpath.union(self.cutvertexdxf
                                         (self.xpos(), self.ypos() + jointwidth,
                                         jointdepth))
2373 gcpy                                toolpath = toolpath.union(self.cutvertexdxf
                                         (self.xpos() + radius, self.ypos(),
                                         jointdepth))

2374 gcpy                                i += 1
2375 gcpy                                self.closepolyline(self.currenttoolnumber())
2376 gcpy                                return toolpath
2377 gcpy
2378 gcpy                                def Full_Blind_Finger_Joint_smallV(self, bx, by, orientation,
                                         side, width, thickness, Number_of_Pins, largeVdiameter,
                                         smallDiameter):
2379 gcpy                                    rapid = self.rapidZ(0)
2380 gcpy                                #         rapid = rapid.union(self.rapidXY(bx, by))
2381 gcpy                                    self.setdxfcolor("Red")
2382 gcpy                                    if (orientation == "Vertical"):
2383 gcpy                                        rapid = rapid.union(self.rapidXY(bx, by - smallDiameter
                                         /6))
2384 gcpy                                        toolpath = self.cutlineZgcfeed(-thickness,1000)
2385 gcpy                                        toolpath = self.cutlinedxfgc(bx, by + width +
                                         smallDiameter/6, - thickness)
2386 gcpy                                    if (orientation == "Horizontal"):
2387 gcpy                                        rapid = rapid.union(self.rapidXY(bx - smallDiameter/6,
                                         by))
2388 gcpy                                        toolpath = self.cutlineZgcfeed(-thickness,1000)
2389 gcpy                                        toolpath = self.cutlinedxfgc(bx + width + smallDiameter
                                         /6, by, -thickness)
2390 gcpy                                #         rapid = self.rapidZ(0)
2391 gcpy
2392 gcpy                                    return toolpath
2393 gcpy
2394 gcpy                                def Full_Blind_Finger_Joint_largeV(self, bx, by, orientation,
                                         side, width, thickness, Number_of_Pins, largeVdiameter,
                                         smallDiameter):
2395 gcpy                                    radius = smallDiameter/2
2396 gcpy                                    rapid = self.rapidZ(0)
2397 gcpy                                    Finger_Width = ((Number_of_Pins * 2) - 1) * smallDiameter *
                                         1.1
2398 gcpy                                    Finger-Origin = width/2 - Finger_Width/2
2399 gcpy                                #         rapid = rapid.union(self.rapidXY(bx, by))
2400 gcpy                                # Joint_Orientation = "Horizontal" "Even" == "Lower", "Odd" ==
                                         "Upper"
2401 gcpy                                # Joint_Orientation = "Vertical" "Even" == "Left", "Odd" == "
                                         Right"
2402 gcpy                                    if (orientation == "Vertical"):
2403 gcpy                                        rapid = rapid.union(self.rapidXY(bx, by))
2404 gcpy                                        toolpath = self.cutlineZgcfeed(-thickness,1000)
2405 gcpy                                        toolpath = toolpath.union(self.cutlinedxfgc(bx, by +
                                         Finger-Origin, -thickness))
2406 gcpy                                    rapid = self.rapidZ(0)
2407 gcpy                                    rapid = rapid.union(self.rapidXY(bx, by + width -
                                         Finger-Origin))
2408 gcpy                                    self.setdxfcolor("Blue")
2409 gcpy                                    toolpath = toolpath.union(self.cutlineZgcfeed(-
                                         thickness,1000))
2410 gcpy                                    toolpath = toolpath.union(self.cutlinedxfgc(bx, by +
                                         width, -thickness))
2411 gcpy                                    if (side == "Left" or side == "Both"):
2412 gcpy                                        rapid = self.rapidZ(0)
2413 gcpy                                        self.setdxfcolor("Dark_Gray")
2414 gcpy                                        rapid = rapid.union(self.rapidXY(bx+thickness-(
                                         smallDiameter / 2) / Tan(math.radians(45)), by -
                                         radius/2))
2415 gcpy                                    toolpath = toolpath.union(self.cutlineZgcfeed(-(
                                         smallDiameter / 2) / Tan(math.radians(45))
                                         ,10000))

```

```

2416 gcpy          toolpath = toolpath.union(self.cutlinedxfgc(bx+
                    thickness-(smallDiameter / 2) / Tan(math.radians
                    (45))), by + width + radius/2, -(smallDiameter /
                    2) / Tan(math.radians(45))))
2417 gcpy          rapid = self.rapidZ(0)
2418 gcpy          self.setdxfc("Green")
2419 gcpy          rapid = rapid.union(self.rapidXY(bx+thickness/2, by
                    +width))
2420 gcpy          toolpath = toolpath.union(self.cutlineZgcfeed(-
                    thickness/2,1000))
2421 gcpy          toolpath = toolpath.union(self.cutlinedxfgc(bx+
                    thickness/2, by + width -thickness, -thickness
                    /2))
2422 gcpy          rapid = self.rapidZ(0)
2423 gcpy          rapid = rapid.union(self.rapidXY(bx+thickness/2, by
                    ))
2424 gcpy          toolpath = toolpath.union(self.cutlineZgcfeed(-
                    thickness/2,1000))
2425 gcpy          toolpath = toolpath.union(self.cutlinedxfgc(bx+
                    thickness/2, by +thickness, -thickness/2))
2426 gcpy          if (side == "Right" or side == "Both"):
2427 gcpy              rapid = self.rapidZ(0)
2428 gcpy              self.setdxfc("Dark_Gray")
2429 gcpy              rapid = rapid.union(self.rapidXY(bx-(thickness-(
                    smallDiameter / 2) / Tan(math.radians(45))), by
                    - radius/2))
2430 gcpy          toolpath = toolpath.union(self.cutlineZgcfeed(-(
                    smallDiameter / 2) / Tan(math.radians(45))
                    ,10000))
2431 gcpy          toolpath = toolpath.union(self.cutlinedxfgc(bx-(
                    thickness-(smallDiameter / 2) / Tan(math.radians
                    (45))), by + width + radius/2, -(smallDiameter /
                    2) / Tan(math.radians(45))))
2432 gcpy          rapid = self.rapidZ(0)
2433 gcpy          self.setdxfc("Green")
2434 gcpy          rapid = rapid.union(self.rapidXY(bx-thickness/2, by
                    +width))
2435 gcpy          toolpath = toolpath.union(self.cutlineZgcfeed(-
                    thickness/2,1000))
2436 gcpy          toolpath = toolpath.union(self.cutlinedxfgc(bx-
                    thickness/2, by + width -thickness, -thickness
                    /2))
2437 gcpy          rapid = self.rapidZ(0)
2438 gcpy          rapid = rapid.union(self.rapidXY(bx-thickness/2, by
                    ))
2439 gcpy          toolpath = toolpath.union(self.cutlineZgcfeed(-
                    thickness/2,1000))
2440 gcpy          toolpath = toolpath.union(self.cutlinedxfgc(bx-
                    thickness/2, by +thickness, -thickness/2))
2441 gcpy          # Joint_Orientation = "Horizontal" "Even" == "Lower", "Odd" ==
                    "Upper"
2442 gcpy          if (orientation == "Horizontal"):
2443 gcpy              rapid = rapid.union(self.rapidXY(bx, by))
2444 gcpy              self.setdxfc("Blue")
2445 gcpy              toolpath = self.cutlineZgcfeed(-thickness,1000)
2446 gcpy              toolpath = toolpath.union(self.cutlinedxfgc(bx +
                    Finger_Origin, by, -thickness))
2447 gcpy              rapid = rapid.union(self.rapidZ(0))
2448 gcpy              rapid = rapid.union(self.rapidXY(bx + width -
                    Finger_Origin, by))
2449 gcpy              toolpath = toolpath.union(self.cutlineZgcfeed(-
                    thickness,1000))
2450 gcpy              toolpath = toolpath.union(self.cutlinedxfgc(bx + width,
                    by, -thickness))
2451 gcpy          if (side == "Lower" or side == "Both"):
2452 gcpy              rapid = self.rapidZ(0)
2453 gcpy              self.setdxfc("Dark_Gray")
2454 gcpy              rapid = rapid.union(self.rapidXY(bx - radius, by+
                    thickness-(smallDiameter / 2) / Tan(math.radians
                    (45))))
2455 gcpy              toolpath = toolpath.union(self.cutlineZgcfeed(-(
                    smallDiameter / 2) / Tan(math.radians(45))
                    ,10000))
2456 gcpy              toolpath = toolpath.union(self.cutlinedxfgc(bx +
                    width + radius, by+thickness-(smallDiameter / 2)
                    / Tan(math.radians(45)), -(smallDiameter / 2) /
                    Tan(math.radians(45))))
2457 gcpy              rapid = self.rapidZ(0)

```

```
2458 gcpy                self.setdxfcolor("Green")
2459 gcpy                rapid = rapid.union(self.rapidXY(bx+width, by+
                        thickness/2))
2460 gcpy                toolpath = toolpath.union(self.cutlineZgcfeed(-
                        thickness/2,1000))
2461 gcpy                toolpath = toolpath.union(self.cutlinedxfgc(bx +
                        width -thickness, by+thickness/2, -thickness/2))
2462 gcpy                rapid = self.rapidZ(0)
2463 gcpy                rapid = rapid.union(self.rapidXY(bx, by+thickness
                        /2))
2464 gcpy                toolpath = toolpath.union(self.cutlineZgcfeed(-
                        thickness/2,1000))
2465 gcpy                toolpath = toolpath.union(self.cutlinedxfgc(bx +
                        thickness, by+thickness/2, -thickness/2))
2466 gcpy                if (side == "Upper" or side == "Both"):
2467 gcpy                    rapid = self.rapidZ(0)
2468 gcpy                    self.setdxfcolor("DarkGray")
2469 gcpy                    rapid = rapid.union(self.rapidXY(bx - radius, by-(
                        thickness-(smallDiameter / 2) / Tan(math.radians
                        (45)))))
2470 gcpy                toolpath = toolpath.union(self.cutlineZgcfeed(-(
                        smallDiameter / 2) / Tan(math.radians(45))
                        ,10000))
2471 gcpy                toolpath = toolpath.union(self.cutlinedxfgc(bx +
                        width + radius, by-(thickness-(smallDiameter /
                        2) / Tan(math.radians(45))), -(smallDiameter /
                        2) / Tan(math.radians(45))))
2472 gcpy                rapid = self.rapidZ(0)
2473 gcpy                self.setdxfcolor("Green")
2474 gcpy                rapid = rapid.union(self.rapidXY(bx+width, by-
                        thickness/2))
2475 gcpy                toolpath = toolpath.union(self.cutlineZgcfeed(-
                        thickness/2,1000))
2476 gcpy                toolpath = toolpath.union(self.cutlinedxfgc(bx +
                        width -thickness, by-thickness/2, -thickness/2))
2477 gcpy                rapid = self.rapidZ(0)
2478 gcpy                rapid = rapid.union(self.rapidXY(bx, by-thickness
                        /2))
2479 gcpy                toolpath = toolpath.union(self.cutlineZgcfeed(-
                        thickness/2,1000))
2480 gcpy                toolpath = toolpath.union(self.cutlinedxfgc(bx +
                        thickness, by-thickness/2, -thickness/2))
2481 gcpy                rapid = self.rapidZ(0)
2482 gcpy                return toolpath
2483 gcpy
2484 gcpy                def Full_Blind_Finger_Joint(self, bx, by, orientation, side,
                        width, thickness, largeVdiameter, smallDiameter,
                        normalormirror = "Default", squaretool = 102, smallV = 390,
                        largeV = 301):
2485 gcpy                    Number_of_Pins = int(((width - thickness * 2) / (
                        smallDiameter * 2.2) / 2) + 0.0) * 2 + 1
2486 gcpy #                    print("Number of Pins: ",Number_of_Pins)
2487 gcpy                    self.movetosafeZ()
2488 gcpy                    self.toolchange(squaretool, 17000)
2489 gcpy                    toolpath = self.Full_Blind_Finger_Joint_square(bx, by,
                        orientation, side, width, thickness, Number_of_Pins,
                        largeVdiameter, smallDiameter)
2490 gcpy                    self.movetosafeZ()
2491 gcpy                    self.toolchange(smallV, 17000)
2492 gcpy                    toolpath = toolpath.union(self.
                        Full_Blind_Finger_Joint_smallV(bx, by, orientation, side
                        , width, thickness, Number_of_Pins, largeVdiameter,
                        smallDiameter))
2493 gcpy                    self.toolchange(largeV, 17000)
2494 gcpy                    toolpath = toolpath.union(self.
                        Full_Blind_Finger_Joint_largeV(bx, by, orientation, side
                        , width, thickness, Number_of_Pins, largeVdiameter,
                        smallDiameter))
2495 gcpy                return toolpath
```

3.9 (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 #Requires OpenPythonSCAD, so load support for 3D modeling in that
    tool:
2 gcpncpy from openscad import *
3 gcpncpy
4 gcpncpy #The gcodepreview library must be loaded, either from github (first
    line below) or from a local library (second line below),
    uncomment one and comment out the other, depending on where one
    wishes to load from
5 gcpncpy #nimport("https://raw.githubusercontent.com/WillAdams/gcodepreview/
    refs/heads/main/gcodepreview.py")
6 gcpncpy from gcodepreview import *
7 gcpncpy
8 gcpncpy #The file to be loaded must be specified:
9 gcpncpy #gc_file = "filename_of_G-code_file_to_process.nc"
10 gcpncpy #
11 gcpncpy #if using windows the full filepath should be provided with
    backslashes replaced with double slashes and wrapped in quotes
    since it is provided as a string:
12 gcpncpy gc_file = "C:\\Users\\willla\\OneDrive\\Desktop\\19mm_1_32_depth.nc"
13 gcpncpy
14 gcpncpy #Create the gcodepreview object:
15 gcpncpy gcp = gcodepreview(False, False)
16 gcpncpy
17 gcpncpy #Process the file (this could be combined with the variable
    definition above, directly inputting the string)
18 gcpncpy gcp.previewgcodefile(gc_file)
```

previewgcodefile Which simply needs to call the previewgcodefile command:

```
2497 gcpy      def previewgcodefile(self, gc_file):
2498 gcpy          gc_file = open(gc_file, 'r')
2499 gcpy          gcfilecontents = []
2500 gcpy          with gc_file as file:
2501 gcpy              for line in file:
2502 gcpy                  command = line
2503 gcpy                  gcfilecontents.append(line)
2504 gcpy
2505 gcpy          numlinesfound = 0
2506 gcpy          for line in gcfilecontents:
2507 gcpy              # print(line)
2508 gcpy              if line[:10] == "(stockMin:":
2509 gcpy                  subdivisions = line.split()
2510 gcpy                  extentleft = float(subdivisions[0][10:-3])
2511 gcpy                  extentfb = float(subdivisions[1][:-3])
2512 gcpy                  extentd = float(subdivisions[2][:-3])
2513 gcpy                  numlinesfound = numlinesfound + 1
2514 gcpy              if line[:13] == "(STOCK/BLOCK,":
2515 gcpy                  subdivisions = line.split()
2516 gcpy                  sizeX = float(subdivisions[0][13:-1])
2517 gcpy                  sizeY = float(subdivisions[1][:-1])
2518 gcpy                  sizeZ = float(subdivisions[4][:-1])
2519 gcpy                  numlinesfound = numlinesfound + 1
2520 gcpy              if line[:3] == "G21":
2521 gcpy                  units = "mm"
2522 gcpy                  numlinesfound = numlinesfound + 1
2523 gcpy              if numlinesfound >=3:
2524 gcpy                  break
2525 gcpy              # print(numlinesfound)
```

Once the initial parameters are parsed, the stock may be set up:

```
2526 gcpy          self.setupcuttingarea(sizeX, sizeY, sizeZ, extentleft,
    extentfb, extentd)
2527 gcpy
2528 gcpy          commands = []
2529 gcpy          for line in gcfilecontents:
2530 gcpy              Xc = 0
2531 gcpy              Yc = 0
2532 gcpy              Zc = 0
2533 gcpy              Fc = 0
2534 gcpy              Xp = 0.0
2535 gcpy              Yp = 0.0
2536 gcpy              Zp = 0.0
2537 gcpy              if line == "G53G0Z-5.000\n":
2538 gcpy                  self.movetosafeZ()
2539 gcpy              if line[:3] == "M6T":
2540 gcpy                  tool = int(line[3:])
```

```
2541 gcpy self.toolchange(tool)
```

Processing tool changes will require examining lines such as:

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

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

;TOOL/CHAMFER, Diameter, Point Angle, Height
```

which once parsed will be passed to a command which uses them to set the variables necessary to effect the toolchange:

```
if line[:11] == "(TOOL/MILL,"
    subdivisions = line.split()
    diameter = float(subdivisions[1][:3])
    cornerradius = float(subdivisions[2][:3])
    height = float(subdivisions[3][:3])
    taperangle = float(subdivisions[4][:3])
    self.settoolparameters("mill", diameter, cornerradius, height, taperangle)

if line[:14] == "(TOOL/CHAMFER,"
    subdivisions = line.split()
    tipdiameter = float(subdivisions[1][:3])
    diameter = float(subdivisions[2][:3])
    radius = float(subdivisions[3][:3])
    height = float(subdivisions[4][:3])
    length = float(subdivisions[4][:3])
    self.settoolparameters("chamfer", tipdiameter, diameter, radius, height, length)
```

```
2544 gcpy if line[:2] == "G0":
2545 gcpy     machinestate = "rapid"
2546 gcpy if line[:2] == "G1":
2547 gcpy     machinestate = "cutline"
2548 gcpy if line[:2] == "G0" or line[:2] == "G1" or line[:1] ==
        "X" or line[:1] == "Y" or line[:1] == "Z":
2549 gcpy     if "F" in line:
2550 gcpy         Fplus = line.split("F")
2551 gcpy         Fc = 1
2552 gcpy         fr = float(Fplus[1])
2553 gcpy         line = Fplus[0]
2554 gcpy     if "Z" in line:
2555 gcpy         Zplus = line.split("Z")
2556 gcpy         Zc = 1
2557 gcpy         Zp = float(Zplus[1])
2558 gcpy         line = Zplus[0]
2559 gcpy     if "Y" in line:
2560 gcpy         Yplus = line.split("Y")
2561 gcpy         Yc = 1
2562 gcpy         Yp = float(Yplus[1])
2563 gcpy         line = Yplus[0]
2564 gcpy     if "X" in line:
2565 gcpy         Xplus = line.split("X")
2566 gcpy         Xc = 1
2567 gcpy         Xp = float(Xplus[1])
2568 gcpy     if Zc == 1:
2569 gcpy         if Yc == 1:
2570 gcpy             if Xc == 1:
2571 gcpy                 if machinestate == "rapid":
2572 gcpy                     command = "rapidXYZ(" + str(Xp) + "
                        ,\u" + str(Yp) + ",\u" + str(Zp) +
                        ")"
2573 gcpy                     self.rapidXYZ(Xp, Yp, Zp)
2574 gcpy                 else:
2575 gcpy                     command = "cutlineXYZ(" + str(Xp) +
                        ",\u" + str(Yp) + ",\u" + str(Zp) +
                        + ")"
2576 gcpy                     self.cutlineXYZ(Xp, Yp, Zp)
2577 gcpy             else:
2578 gcpy                 if machinestate == "rapid":
2579 gcpy                     command = "rapidYZ(" + str(Yp) + ",
                        \u" + str(Zp) + ")"
2580 gcpy                     self.rapidYZ(Yp, Zp)
2581 gcpy                 else:
2582 gcpy                     command = "cutlineYZ(" + str(Yp) +
                        ",\u" + str(Zp) + ")"
2583 gcpy                     self.cutlineYZ(Yp, Zp)
```

```
2584 gcpy                                     else:
2585 gcpy                                     if Xc == 1:
2586 gcpy                                     if machinestate == "rapid":
2587 gcpy                                     command = "rapidXZ(" + str(Xp) + ",
                                                "\u" + str(Zp) + ")"
2588 gcpy                                     self.rapidXZ(Xp, Zp)
2589 gcpy                                     else:
2590 gcpy                                     command = "cutlineXZ(" + str(Xp) +
                                                ",\u" + str(Zp) + ")"
2591 gcpy                                     self.cutlineXZ(Xp, Zp)
2592 gcpy                                     else:
2593 gcpy                                     if machinestate == "rapid":
2594 gcpy                                     command = "rapidZ(" + str(Zp) + ")"
2595 gcpy                                     self.rapidZ(Zp)
2596 gcpy                                     else:
2597 gcpy                                     command = "cutlineZ(" + str(Zp) + "
                                                )"
2598 gcpy                                     self.cutlineZ(Zp)
2599 gcpy
2600 gcpy                                     else:
2601 gcpy                                     if Yc == 1:
2602 gcpy                                     if Xc == 1:
2603 gcpy                                     if machinestate == "rapid":
2604 gcpy                                     command = "rapidXY(" + str(Xp) + ",
                                                "\u" + str(Yp) + ")"
2605 gcpy                                     self.rapidXY(Xp, Yp)
2606 gcpy                                     else:
2607 gcpy                                     command = "cutlineXY(" + str(Xp) +
                                                ",\u" + str(Yp) + ")"
2608 gcpy                                     self.cutlineXY(Xp, Yp)
2609 gcpy                                     else:
2610 gcpy                                     if machinestate == "rapid":
2611 gcpy                                     command = "rapidY(" + str(Yp) + ")"
2612 gcpy                                     self.rapidY(Yp)
2613 gcpy                                     else:
2614 gcpy                                     command = "cutlineY(" + str(Yp) + "
                                                )"
2615 gcpy                                     self.cutlineY(Yp)
2616 gcpy                                     else:
2617 gcpy                                     if Xc == 1:
2618 gcpy                                     if machinestate == "rapid":
2619 gcpy                                     command = "rapidX(" + str(Xp) + ")"
2620 gcpy                                     self.rapidX(Xp)
2621 gcpy                                     else:
2622 gcpy                                     command = "cutlineX(" + str(Xp) + "
                                                )"
2623 gcpy                                     self.cutlineX(Xp)
2624 gcpy #                                     commands.append(command)
2625 gcpy #                                     print(line)
2626 gcpy #                                     print(command)
2627 gcpy #                                     print(machinestate, Xc, Yc, Zc)
2628 gcpy #                                     print(Xp, Yp, Zp)
2629 gcpy #                                     print("/n")
2630 gcpy #
2631 gcpy #                                     for command in commands:
2632 gcpy #                                     print(command)
2633 gcpy #
2634 gcpy #                                     show(self.stockandtoolpaths())
2635 gcpy #                                     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

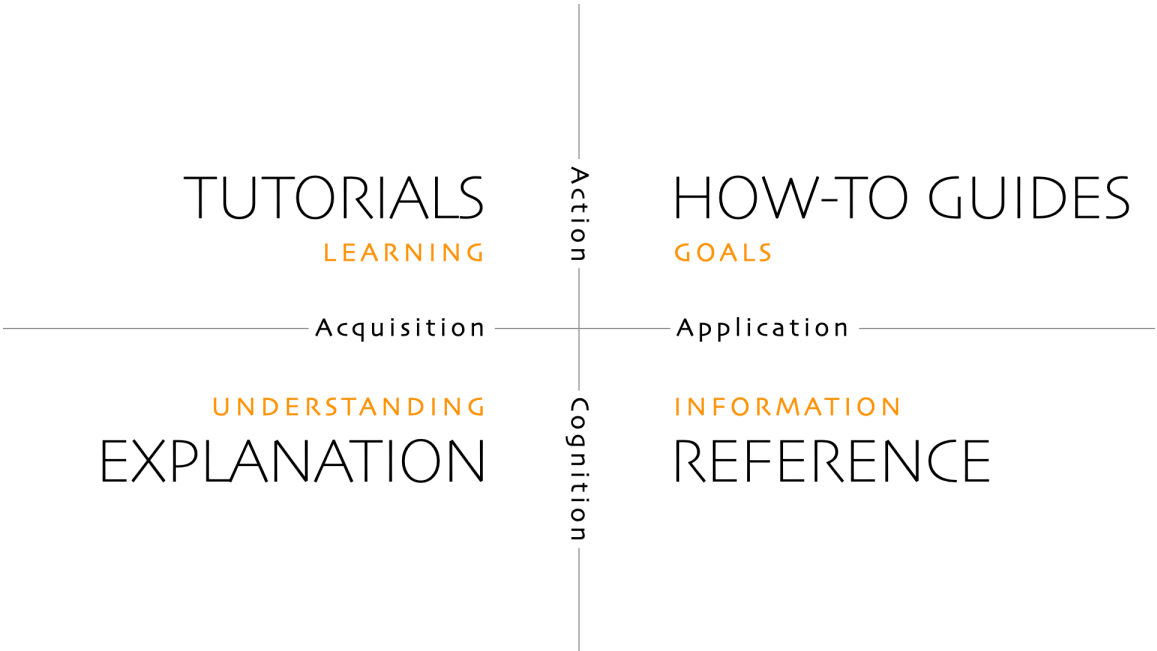
<https://thewhitetulip.gitbook.io/py/06-file-handling>

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:

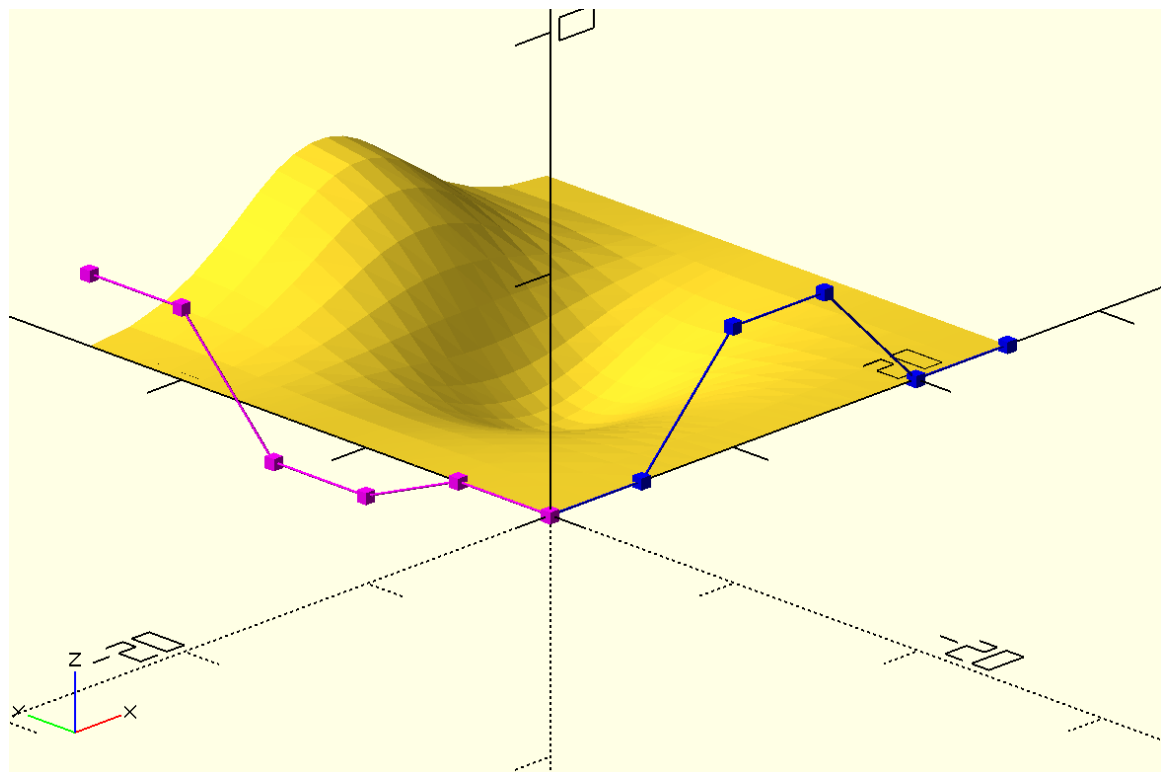
3 planes * 3 Béziers * (2 on-curve + 2 off-curve points) == 36 coordinate pairs

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/

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://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, 65
- ballnose, 48
- beginpolyline, 65
- closedxfile, 73, 74
- closegcodefile, 73, 74
- closepolyline, 65
- currenttoolnum, 30
- cut..., 45, 52
- cutarcCC, 54
- cutarcCW, 54
- cutkeyhole toolpath, 79
- cutKHgcdxf, 80
- cutline, 52
- cutrectangle, 78
- diameter, 30
- dovetail, 50
- dxfarc, 65
- dxfcircle, 65
- dxfline, 65
- dxfpreamble, 73, 74
- dxfpreamble, 65
- dxfwrite, 64
- endmill square, 48
- endmill v, 49
- endmilltype, 30
- feed, 60
- flute, 30
- gcodepreview, 28
 - writeln, 61
- gcp.setupstock, 31
- init, 28
- mpx, 30
- mpy, 30
- mpz, 30
- opendxfile, 61
- opengcodefile, 61, 62
- plunge, 60
- previewgcodefile, 94
- ra, 30
- rapid, 51
- rapid..., 45
- rapids, 30, 34
- roundover, 50
- settoolparameters, 37
- setupstock, 31
 - gcodepreview, 31
- setxpos, 30
- setypos, 30
- setzpos, 30
- shaftmovement, 46, 50
- speed, 60
- stockzero, 31
- subroutine
 - gcodepreview, 31
 - writeln, 61
- tip, 30
- tool diameter, 58
- tool number, 37
- tool radius, 60
- toolchange, 37, 38
- toolmovement, 30, 34, 38, 46
- toolpaths, 30, 34
- tpzinc, 30
- writedxfileDT, 65
- writedxfileKH, 65
- writedxfilegbl, 64
- writedxfilegsq, 64
- writedxfileV, 64
- writedxfilesmbl, 64
- writedxfilesmsq, 64
- writedxfilesmV, 65
- xpos, 30
- ypos, 30
- zeroheight, 31
- zpos, 30

Routines

addvertex, 65	previewgcodefile, 94
ballnose, 48	rapid, 51
beginpolyline, 65	rapid..., 45
	roundover, 50
closedxfile, 73, 74	setupstock, 31
closegcodefile, 73, 74	setxpos, 30
closepolyline, 65	setypos, 30
cut..., 45, 52	setzpos, 30
cutarcCC, 54	shaftmovement, 46, 50
cutarcCW, 54	
cutkeyhole toolpath, 79	tool diameter, 58
cutKHgcdxf, 80	tool radius, 60
cutline, 52	toolchange, 37, 38
cutrectangle, 78	toolmovement, 30, 34, 38, 46
dovetail, 50	writedxDT, 65
dxarc, 65	writedxKH, 65
dxcircle, 65	writedxflgbl, 64
dxline, 65	writedxflgsq, 64
dxpostamble, 73, 74	writedxflgV, 64
dxfpreamble, 65	writedxfsmb, 64
dxfwrite, 64	writedxfsmsq, 64
	writedxsmV, 65
endmill square, 48	writeln, 61
endmill v, 49	
	xpos, 30
gcodepreview, 28, 31	
gcp.setupstock, 31	ypos, 30
init, 28	zpos, 30
opendxfile, 61	
opengcodefile, 61, 62	

Variables

currenttoolnum, 30	ra, 30
diameter, 30	rapids, 30, 34
endmilltype, 30	settoolparameters, 37
feed, 60	speed, 60
flute, 30	stockzero, 31
mpx, 30	tip, 30
mpy, 30	tool number, 37
mpz, 30	toolchange, 37, 38
	toolpaths, 30, 34
	tpzinc, 30
plunge, 60	zeroheight, 31