

The gcodepreview PythonSCAD library*

Author: William F. Adams
willadams at aol dot com

2026/02/25

Abstract

The gcodepreview library allows using PythonSCAD (Python in OpenSCAD) 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

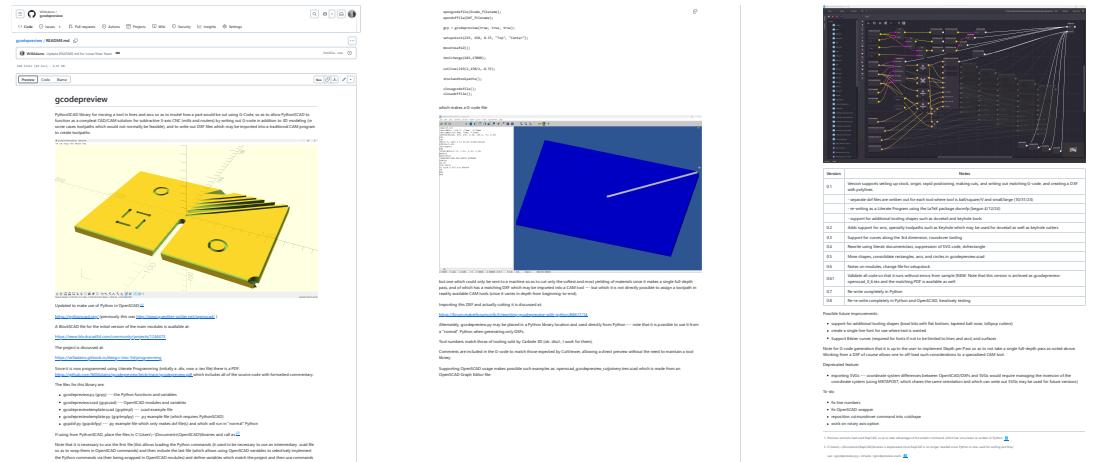
1	readme.md	4
2	Usage and Templates	8
2.1	gcpdxf.py	9
2.2	gcpcutdxf.py	12
2.3	gcodepreviewtemplate.py	14
2.4	gcodepreviewtemplate.scad	19
2.5	gpcthreedp.py	23
2.6	gcodepreviewtemplate.txt	24
3	gcodepreview	25
3.1	Tools for 3D Previewing G-code	25
3.1.1	Cutviewer	25
3.2	Module Naming Convention	27
3.2.1	Parameters and Default Values	29
3.3	Implementation files and gcodepreview class	29
3.3.1	init	31
3.3.2	Position and Variables	33
3.3.3	Initial Modules	33
3.3.4	Adjustments and Additions	37
3.4	Tools and Shapes and Changes	37
3.4.1	Numbering for Tools	38
3.4.2	Laser support	52
3.5	Shapes and tool movement	52
3.5.1	Tooling for Undercutting Toolpaths	53
3.5.2	Generalized commands and cuts	53
3.5.3	Movement and color	53
3.5.4	tooldiameter	70
3.5.5	Feeds and Speeds	71
3.5.6	3D Printing	71
3.6	Difference of Stock, Rapids, and Toolpaths	84
3.7	Output files	84
3.7.1	Python and OpenSCAD File Handling	84
3.7.2	DXF Overview	86
3.7.3	G-code Overview	94
3.8	Cutting shapes and expansion	95
3.8.1	Building blocks	95
3.9	(Reading) G-code Files	115
4	Notes	118
4.1	Other Resources	118
4.1.1	Coding Style	118
4.1.2	Coding References	118
4.1.3	Documentation Style	118
4.2	Future	119
4.2.1	Images	119
4.2.2	Bézier curves in 2 dimensions	119
4.2.3	Bézier curves in 3 dimensions	119
4.2.4	Mathematics	120
Index		123
Routines	124
Variables	125

Contents

1	readme.md	4
2	Usage and Templates	8
2.1	gcpdxf.py	9
2.2	gcpcutdxf.py	12
2.3	gcodepreviewtemplate.py	14
2.4	gcodepreviewtemplate.scad	19
2.5	gpcthreedp.py	23
2.6	gcodepreviewtemplate.txt	24
3	gcodepreview	25
3.1	Tools for 3D Previewing G-code	25
3.1.1	Cutviewer	25
3.1.1.1	Stock size and placement	25
3.1.1.2	Tool Shapes	26
3.1.1.2.1	Tool/Mill (Square, radiused, ball-nose, and tapered-ball)	26
3.1.1.2.2	Corner Rounding, (roundover)	26
3.1.1.2.3	V shaped tooling (and variations)	27
3.2	Module Naming Convention	27
3.2.1	Parameters and Default Values	29
3.3	Implementation files and gcodepreview class	29
3.3.1	init	31
3.3.2	Position and Variables	33
3.3.3	Initial Modules	33
3.3.3.1	setupstock	34
3.3.3.2	setupcuttingarea	36
3.3.3.3	debug	36
3.3.4	Adjustments and Additions	37
3.4	Tools and Shapes and Changes	37
3.4.1	Numbering for Tools	38
3.4.1.1	toolchange	41
3.4.1.1.1	settoolparameters	42
3.4.1.1.2	toolchange	42
3.4.1.1.3	Square (including O-flute)	43
3.4.1.1.4	Ball-nose	45
3.4.1.1.5	V	46
3.4.1.1.6	Keyhole	47
3.4.1.1.7	Bowl	48
3.4.1.1.8	Tapered ball nose	49
3.4.1.1.9	Roundover (cove tooling)	49
3.4.1.1.10	Dovetails	51
3.4.1.1.11	closing G-code	52
3.4.2	Laser support	52
3.5	Shapes and tool movement	52
3.5.1	Tooling for Undercutting Toolpaths	53
3.5.2	Generalized commands and cuts	53
3.5.3	Movement and color	53
3.5.3.1	toolmovement	55
3.5.3.1.1	Square (including O-flute)	55
3.5.3.1.2	Ball nose	56
3.5.3.1.3	bowl	56
3.5.3.1.4	V	56
3.5.3.1.5	Keyhole	56
3.5.3.1.6	Tapered ball nose	57
3.5.3.1.7	Dovetails	57
3.5.3.2	Concave toolshapes	57
3.5.3.2.1	Roundover tooling	58
3.5.3.3	shaftmovement	58
3.5.3.4	tool outlines	58
3.5.3.4.1	defineshaft	59
3.5.3.4.2	Square (including O-flute)	60
3.5.3.4.3	Ball-nose	60
3.5.3.4.4	V tool outline	60
3.5.3.4.5	Keyhole outline	61
3.5.3.4.6	Bowl outline	61
3.5.3.4.7	Tapered ball nose	61
3.5.3.4.8	Roundover (cove tooling)	61
3.5.3.5	rapid and cut (lines)	62
3.5.3.6	Arcs	64
3.5.4	tooldiameter	70

3.5.5	Feeds and Speeds	71
3.5.6	3D Printing	71
3.5.6.1	fullcontrolgcode	72
3.5.6.2	Previewing/verifying G-code for 3D printers	76
3.5.6.3	Time and Firmware for 3D printers	76
3.5.6.4	Sample 3D printing file	76
3.5.6.5	Initialize	80
3.5.6.6	extrude	82
3.5.6.7	Shutdown	83
3.5.6.8	fullcontrolgcode commands	83
3.6	Difference of Stock, Rapids, and Toolpaths	84
3.7	Output files	84
3.7.1	Python and OpenSCAD File Handling	84
3.7.2	DXF Overview	86
3.7.2.1	Writing a preamble to DXF files	87
3.7.2.1.1	DXF Lines and Arcs	88
3.7.3	G-code Overview	94
3.7.3.1	Closings	95
3.8	Cutting shapes and expansion	95
3.8.1	Building blocks	95
3.8.1.1	List of shapes	96
3.8.1.1.1	circles	97
3.8.1.1.2	rectangles	98
3.8.1.1.3	Keyhole toolpath and undercut tooling	99
3.8.1.1.4	Dovetail joinery and tooling	107
3.8.1.1.5	Full-blind box joints	109
3.8.1.2	Fonts	114
3.9	(Reading) G-code Files	115
4	Notes	118
4.1	Other Resources	118
4.1.1	Coding Style	118
4.1.2	Coding References	118
4.1.3	Documentation Style	118
4.2	Future	119
4.2.1	Images	119
4.2.2	Bézier curves in 2 dimensions	119
4.2.3	Bézier curves in 3 dimensions	119
4.2.4	Mathematics	120
	Index	123
	Routines	124
	Variables	125

1 readme.md



```

1 rdme # gcodepreview
2 rdme
3 rdme OpenPythonSCAD library for moving a tool in lines and arcs so as to
      model how a part would be cut or extruded using G-Code, so as
      to allow use as a compleat CAD/CAM solution for subtractive or
      additive 3-axis CNC (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 in
      typical tools), 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 Uses Python in OpenSCAD: https://pythonscad.org/[^pythonscad]
8 rdme
9 rdme [^pythonscad]: Previously this was http://www.guenther-sohler.net/
      openscad/
10 rdme
11 rdme A BlockSCAD file for the initial version of the
12 rdme main modules is available at:
13 rdme
14 rdme https://www.blockscad3d.com/community/projects/1244473
15 rdme
16 rdme The project is discussed at:
17 rdme
18 rdme https://willadams.gitbook.io/design-into-3d/programming
19 rdme
20 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 commentary.
21 rdme
22 rdme The files for this library are:
23 rdme
24 rdme - gcodepreview.py (gcpy) --- the Python class/functions and
      variables
25 rdme - gcodepreview.scad (gcpscad) --- OpenSCAD modules and parameters
26 rdme
27 rdme And there several sample/template files which may be used as the
      starting point for a given project:
28 rdme
29 rdme - gcodepreviewtemplate.txt (gcptmpl) --- .txt file collecting
      various commands with brief comments which may be used as a
      quick reference or copy-pasting from
30 rdme - gcodepreviewtemplate.py (gcptmplpy) --- .py example file
31 rdme - gcodepreviewtemplate.scad (gcptmplscad) --- .scad example file
32 rdme - gcpdxf.py (gcpdxfpy) --- .py example file which only makes dxf
      file(s) and which will run in "normal" Python in addition to
      PythonSCAD
33 rdme - gcpgc.py (gcpgc) --- .py example which loads a G-code file and
      generates a 3D preview showing how the G-code will cut
34 rdme - gcptreepd.py --- Template for 3D printing using Full Control G-
      code https://fullcontrolgcode.com/
35 rdme
36 rdme Note that additional templates are in: https://github.com/WillAdams/
      /gcodepreview/tree/main/templates

```

```

37 rdme
38 rdme If using from PythonSCAD, place the files in C:\Users\\~\Documents
         \OpenSCAD\libraries or, load them from Github using the command:
39 rdme
40 rdme     nimport("https://raw.githubusercontent.com/WillAdams/
                  gcodepreview/refs/heads/main/gcodepreview.py")
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     opendxf(file(DXF_filename));
51 rdme
52 rdme     gcp = gcodepreview("cut", 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     closedxf(file());
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 to generate DXFs as shown
         in gcpdxf.py (generating a 3D preview requires OpenPythonSCAD
         and generating G-code without a preview is not supported).
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 worked up and used
         as desired.
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 Written as a [Literate Program](http://literateprogramming.com/) in
     [lualatex](https://www.luatex.org/) which is a version of
     Donald E. Knuth's [TeX typesetting program](https://tug.org/
     whatis.html) extended by the [Lua programming language](https://
     www.lua.org/) using a custom package, [literati.sty](https://
     github.com/WillAdams/gcodepreview/blob/main/literati.sty)
     developed with a bit of help on [tex.stackexchange.com](https://
     tex.stackexchange.com/questions/722886/how-to-write-out-multiple-
     -text-files-from-multiple-instances-of-latex-environment) rather
     than using a more typical IDE because of the need for keeping
     multiple files in sync and so as to have a single point of
     control (the .tex source file used to generate the .pdf and all
     other files for this project).

88 rdme
89 rdme | Version      | Notes          |
90 rdme | ----- | ----- |
91 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.
92 rdme |           | - separate dxf files are written out for each
     tool where tool is ball/square/V and small/large (10/31/23)
     |
93 rdme |           | - re-writing as a Literate Program using the
     LaTeX package docmfp (begun 4/12/24)
     |
94 rdme |           | - support for additional tooling shapes such as
     dovetail and keyhole tools
     |
95 rdme | 0.2           | Adds support for arcs, specialty toolpaths such
     as Keyhole which may be used for dovetail as well as keyhole
     cutters
     |
96 rdme | 0.3           | Support for curves along the 3rd dimension,
     roundover tooling
     |
97 rdme | 0.4           | Rewrite using literati documentclass, suppression
     of SVG code, dxfractangle
     |
98 rdme | 0.5           | More shapes, consolidate rectangles, arcs, and
     circles in gcodepreview.scad
     |
99 rdme | 0.6           | Notes on modules, change file for setupstock
     |
100 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)
101 rdme | 0.7           | Re-write completely in Python
     |
102 rdme | 0.8           | Re-re-write completely in Python and OpenSCAD,
     iteratively testing
     |
103 rdme | 0.801          | Add support for bowl bits with flat bottom
     |
104 rdme | 0.802          | Add support for tapered ball-nose and V tools
     with flat bottom
     |
105 rdme | 0.803          | Implement initial color support and joinery
     modules (dovetail and full blind box joint modules)
     |
106 rdme | 0.9           | Re-write to use Python lists for 3D shapes for
     toolpaths and rapids.
     |
107 rdme | 0.91          | Finish converting to native OpenPythonSCAD

```

```

trigonometric functions.

|
108 rdme | 0.92           | Remove multiple DXFs and unimplemented features,
           add hooks for 3D printing.

|
109 rdme | 0.93           | Initial support for 3D printing.

|
110 rdme | 0.931          | Update support for OpenSCAD modules.

|
111 rdme | 0.932          | Update DXF file-handling for Carbide Create 839.

|
112 rdme
113 rdme To do:
114 rdme
115 rdme - implement OpenSCAD commands for 3D printing
116 rdme - implement 3D printing commands beyond straight-line extrude
117 rdme - add toolpath for cutting countersinks using ball-nose tool from
           inside working out
118 rdme - create additional template and sample files
119 rdme - fully implement/verify describing/saving/loading tools using
           CutViewer comments
120 rdme - support for additional tooling shapes (lollipop cutters)
121 rdme - threadmilling
122 rdme - create font using lines and arcs with parameters for overshoot
           and width/spacing
123 rdme
124 rdme Possible future improvements:
125 rdme
126 rdme - implement skin()
127 rdme - support for 4th-axis
128 rdme - support for post-processors
129 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)
130 rdme - create a single line font for use where text is wanted
131 rdme - Support for METAPOST and Bézier curves (latter required for
           fonts if not to be limited to lines and arcs) and surfaces
132 rdme
133 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.
134 rdme
135 rdme Issues/Research:
136 rdme
137 rdme - determine why one quadrant of arc command doesn't work in
           OpenSCAD
138 rdme - clock-wise arcs
139 rdme - verify OpenSCAD wrapper and add any missing commands for Python
140 rdme - verify support for shaft on tooling
141 rdme
142 rdme Deprecated features:
143 rdme
144 rdme - polylines
145 rdme - exporting SVGs --- coordinate system differences between
           OpenSCAD/DXF 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)
146 rdme - using linear/rotate_extrude --- 2D geometry is rotated to match
           the arc of the movement, which is appropriate to a 5-axis
           machine, but not workable for a 3-axis. Adding an option to
           support the use of such commands for horizontal movement is
           within the realm of possibility.
147 rdme - multiple DXF files
148 rdme - RapCAD support

```

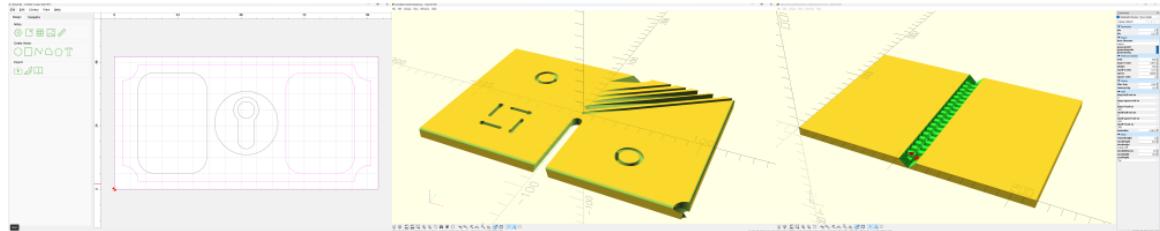
2 Usage and Templates

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).

Written as a Literate Program¹ in lualatex² which is a version of Donald E. Knuth’s TeX typesetting program³ extended by the Lua programming language⁴ using a custom package, literati.sty⁵ developed with a bit of help on tex.stackexchange.com⁶ rather than using a more typical IDE because of the need for keeping multiple files in synch and so as to have a single point of control (the .tex source file used to generate the .pdf and all other files for this project).

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 supported 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 which will be collected in a plain text template file which is concatenated to provide a usable example of each command with (brief) commentary (potentially serving as a how-to guide as well as documenting the code in a minimalistic fashion) 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 efficiency 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.

¹<http://literateprogramming.com/>

²<https://www.luatex.org/>

³<https://tug.org/whatis.html>

⁴<https://www.lua.org/>

⁵<https://github.com/WillAdams/gcodepreview/blob/main/literati.sty>

⁶<https://tex.stackexchange.com/questions/722886/how-to-write-out-multiple-text-files-from-multiple-instances-of-tex>

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

```

```

        inset + radius), stockYheight - inset)
63 gcpdxfpy
64 gcpdxfpy gcp.setdxfcolor("Blue")
65 gcpdxfpy gcp.setdxflayer("Blue")
66 gcpdxfpy
67 gcpdxfpy gcp.dxfrectangle(radius + inset, radius, stockXwidth/2 - (radius *
    4), stockYheight - (radius * 2), cornerstyle, radius)
68 gcpdxfpy gcp.dxrectangle(stockXwidth/2 + (radius * 2) + inset, radius,
    stockXwidth/2 - (radius * 4), stockYheight - (radius * 2),
    cornerstyle, radius)
69 gcpdxfpy
70 gcpdxfpy gcp.setdxfcolor("Black")
71 gcpdxfpy gcp.setdxflayer("DEFAULT")
72 gcpdxfpy
73 gcpdxfpy gcp.beginpolyline(stockXwidth*0.75+radius*1.5, stockYheight/4-radius
    /2)
74 gcpdxfpy gcp.addvertex(stockXwidth*0.75+radius, stockYheight/4-radius/2)
75 gcpdxfpy gcp.addvertex(stockXwidth*0.75+radius, stockYheight*0.75+radius/2)
76 gcpdxfpy gcp.addvertex(stockXwidth*0.75+radius*1.5, stockYheight*0.75+radius
    /2)
77 gcpdxfpy gcp.closepolyline()
78 gcpdxfpy
79 gcpdxfpy gcp.dxfarc(stockXwidth*0.75+radius*1.5, stockYheight*0.75, radius
    /2, 0, 90)
80 gcpdxfpy
81 gcpdxfpy gcp.beginpolyline(stockXwidth*0.75+radius*2, stockYheight*0.75)
82 gcpdxfpy gcp.addvertex(stockXwidth*0.75+radius*2, stockYheight/4)
83 gcpdxfpy gcp.closepolyline()
84 gcpdxfpy
85 gcpdxfpy gcp.dxfarc(stockXwidth*0.75+radius*1.5, stockYheight/4, radius/2,
    270, 360)
86 gcpdxfpy
87 gcpdxfpy gcp.setdxfcolor("LightGray")
88 gcpdxfpy gcp.setdxflayer("LightGray")
89 gcpdxfpy
90 gcpdxfpy gcp.beginpolyline(stockXwidth*0.25-radius*1.5, stockYheight/4-radius
    /2)
91 gcpdxfpy gcp.addvertex(stockXwidth*0.25-radius, stockYheight/4-radius/2)
92 gcpdxfpy gcp.addvertex(stockXwidth*0.25-radius, stockYheight*0.75+radius/2)
93 gcpdxfpy gcp.addvertex(stockXwidth*0.25-radius*1.5, stockYheight*0.75+radius
    /2)
94 gcpdxfpy gcp.closepolyline()
95 gcpdxfpy
96 gcpdxfpy gcp.dxfarc(stockXwidth*0.25-radius*1.5, stockYheight*0.75, radius
    /2, 90, 180)
97 gcpdxfpy
98 gcpdxfpy gcp.beginpolyline(stockXwidth*0.25-radius*2, stockYheight*0.75)
99 gcpdxfpy gcp.addvertex(stockXwidth*0.25-radius*2, stockYheight/4)
100 gcpdxfpy gcp.closepolyline()
101 gcpdxfpy
102 gcpdxfpy gcp.dxfarc(stockXwidth*0.25-radius*1.5, stockYheight/4, radius/2,
    180, 270)
103 gcpdxfpy
104 gcpdxfpy gcp.setdxfcolor("Yellow")
105 gcpdxfpy gcp.setdxflayer("Yellow")
106 gcpdxfpy gcp.dxfcircle(stockXwidth/4+1+radius/2, stockYheight/4, radius/2)
107 gcpdxfpy
108 gcpdxfpy gcp.setdxfcolor("Green")
109 gcpdxfpy gcp.setdxflayer("Green")
110 gcpdxfpy gcp.dxfcircle(stockXwidth*0.75-(1+radius/2), stockYheight*0.75,
    radius/2)
111 gcpdxfpy
112 gcpdxfpy gcp.setdxfcolor("Cyan")
113 gcpdxfpy gcp.setdxflayer("Cyan")
114 gcpdxfpy gcp.dxfcircle(stockXwidth/4+1+radius/2, stockYheight*0.75, radius
    /2)
115 gcpdxfpy
116 gcpdxfpy gcp.setdxfcolor("Magenta")
117 gcpdxfpy gcp.setdxflayer("Magenta")
118 gcpdxfpy gcp.dxfcircle(stockXwidth*0.75-(1+radius/2), stockYheight/4, radius
    /2)
119 gcpdxfpy
120 gcpdxfpy gcp.setdxfcolor("DarkGray")
121 gcpdxfpy gcp.setdxflayer("DarkGray")
122 gcpdxfpy gcp.dxfcircle(stockXwidth/2, stockYheight/2, radius * 2)
123 gcpdxfpy
124 gcpdxfpy gcp.setdxfcolor("LightGray")
125 gcpdxfpy gcp.setdxflayer("LightGray")

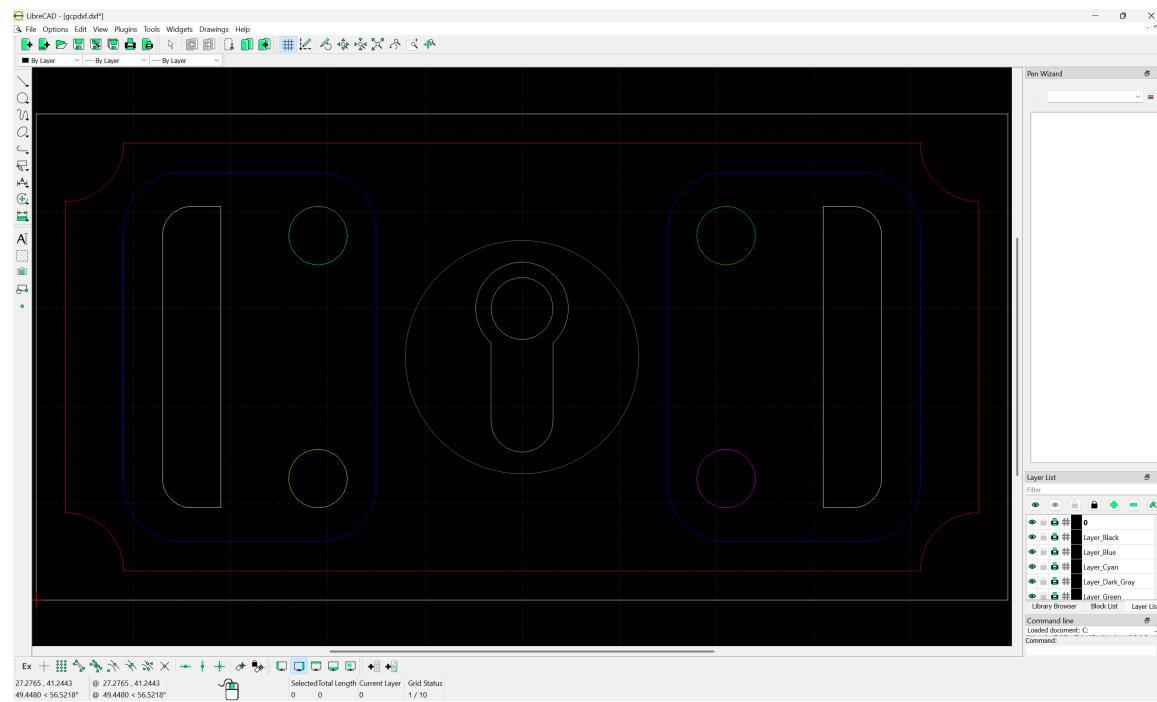
```

```

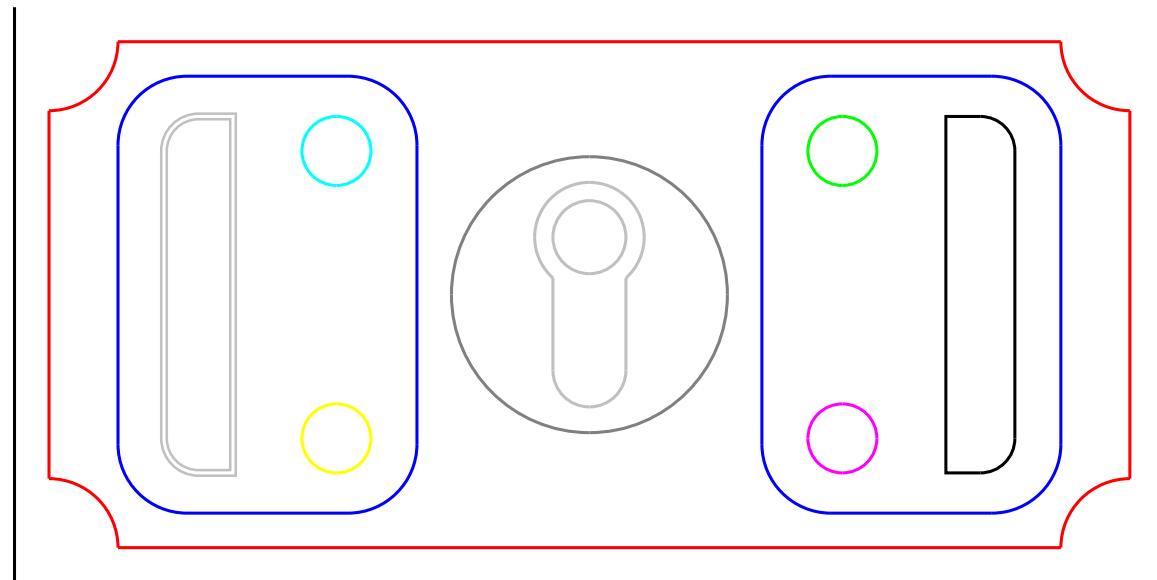
126 gcpdxfpy
127 gcpdxfpy gcp.toolchange(374)
128 gcpdxfpy
129 gcpdxfpy gcp.dxfKH(stockXwidth/2, stockYheight/5*3, 0, -7, 270, 11.5875)
130 gcpdxfpy
131 gcpdxfpy gcp.closedxfffile()

```

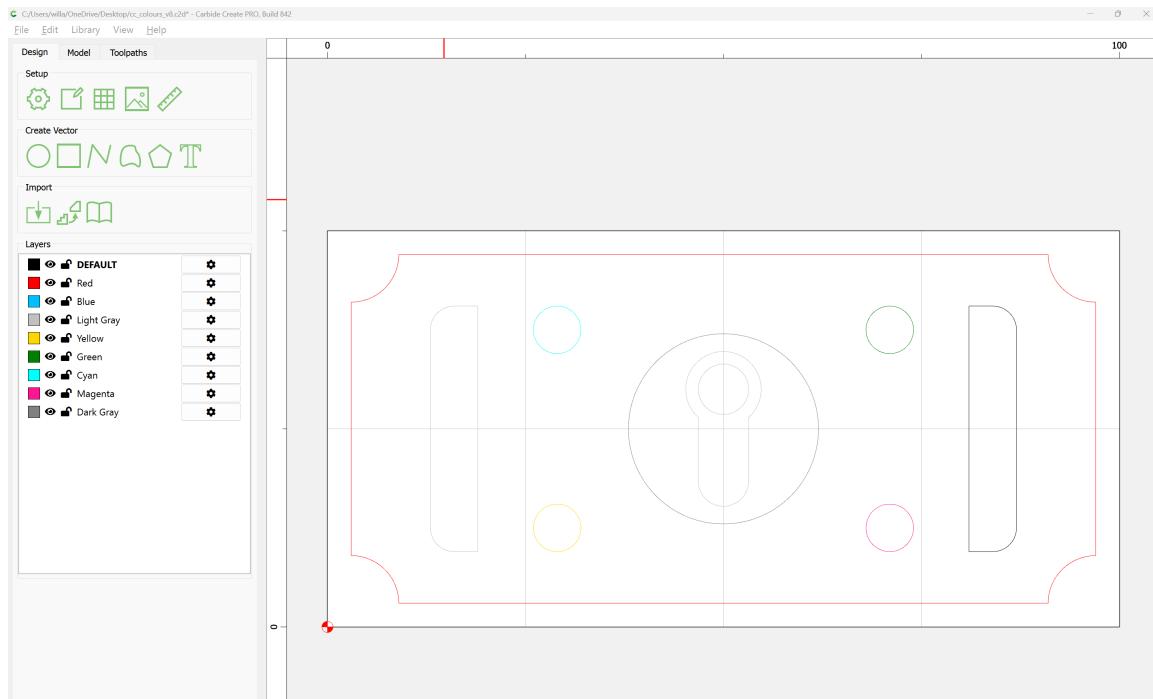
which creates a .dxf file which may be imported into any CAD program:



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 corner location and width (X)/height(Y))
 - `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 gpcutdxf.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 in OpenPythonSCAD is a straight-forward task:

```

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

```

```

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

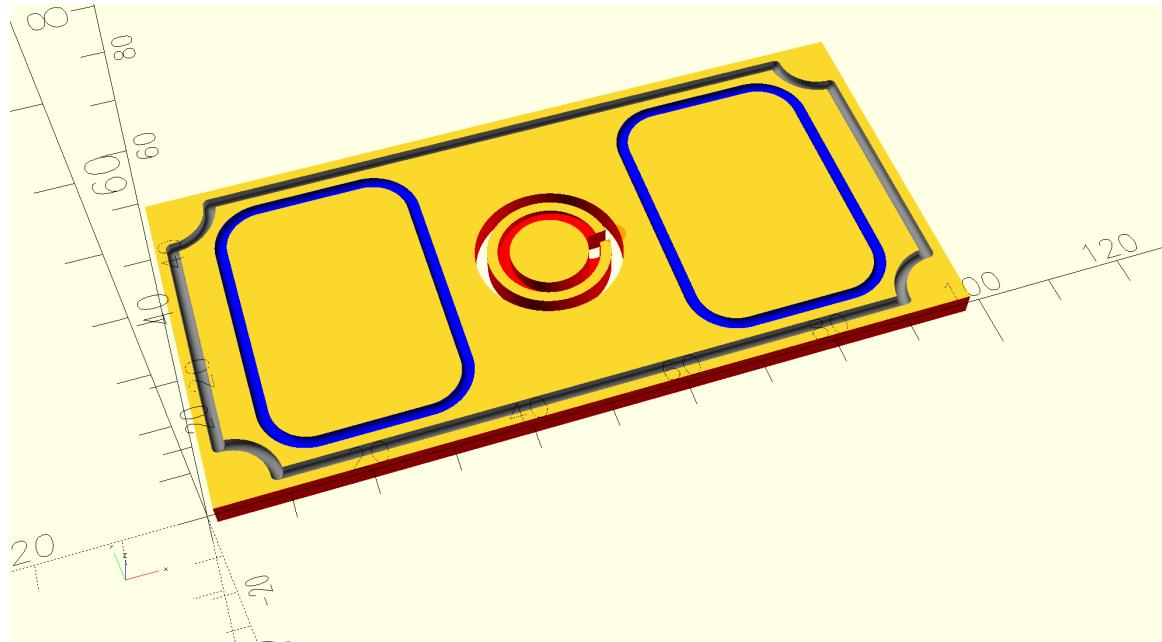
```

```

96 gcpcutdxfpy gcp.setdxflayer("Blue")
97 gcpcutdxfpy
98 gcpcutdxfpy gcp.rapidZ(retractheight)
99 gcpcutdxfpy gcp.rapid(radius + inset + radius, radius, 0)
100 gcpcutdxfpy
101 gcpcutdxfpy gcp.cutrectanglerounddxf(radius + inset, radius, 0, stockXwidth/2 -
    (radius * 4), stockYheight - (radius * 2), -stockZthickness/4,
    radius)
102 gcpcutdxfpy
103 gcpcutdxfpy gcp.rapidZ(retractheight)
104 gcpcutdxfpy gcp.rapid(stockXwidth/2 + (radius * 2) + inset + radius, radius, 0)
105 gcpcutdxfpy
106 gcpcutdxfpy gcp.cutrectanglerounddxf(stockXwidth/2 + (radius * 2) + inset,
    radius, 0, stockXwidth/2 - (radius * 4), stockYheight - (radius
    * 2), -stockZthickness/4, radius)
107 gcpcutdxfpy
108 gcpcutdxfpy gcp.setdxfcolor("Green")
109 gcpcutdxfpy gcp.setdxflayer("Green")
110 gcpcutdxfpy
111 gcpcutdxfpy gcp.rapidZ(retractheight)
112 gcpcutdxfpy gcp.rapid(stockXwidth/2, stockYheight/2 - radius, 0)
113 gcpcutdxfpy
114 gcpcutdxfpy gcp.toolchange(large_square_tool_num)
115 gcpcutdxfpy
116 gcpcutdxfpy gcp.cutquarterCCSEdxf(stockXwidth/2 + radius, stockYheight/2, -
    stockZthickness/4, radius)
117 gcpcutdxfpy gcp.cutquarterCCNEdxf(stockXwidth/2, stockYheight/2 + radius, -
    stockZthickness/2, radius)
118 gcpcutdxfpy gcp.cutquarterCCNWdxr(stockXwidth/2 - radius, stockYheight/2, -
    stockZthickness*0.75, radius)
119 gcpcutdxfpy gcp.cutquarterCCSWdxr(stockXwidth/2, stockYheight/2 - radius, -
    stockZthickness, radius)
120 gcpcutdxfpy
121 gcpcutdxfpy gcp.closeddfffile()
122 gcpcutdxfpy
123 gcpcutdxfpy 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 to resume cutting at the new position.

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", 3.6"
61 gcptmplpy #304 tapered ball nose 0.1250", 0.2500", 1.50", 2.4"
62 gcptmplpy #648 threadmill_shaft(2.4, 0.75, 18)
63 gcptmplpy #45982 Carbide Tipped Bowl & Tray 1/4 Radius x 3/4 Dia x 5/8 x 1/4
      Inch Shank
64 gcptmplpy #13921 https://www.amazon.com/Yonico-Groove-Bottom-Router-Degree/dp
      /B0CPJPTMPP
65 gcptmplpy
66 gcptmplpy # [Feeds and Speeds] */
67 gcptmplpy plunge = 100
68 gcptmplpy # [Feeds and Speeds] */
69 gcptmplpy feed = 400
70 gcptmplpy # [Feeds and Speeds] */
71 gcptmplpy speed = 16000
72 gcptmplpy # [Feeds and Speeds] */
73 gcptmplpy small_square_ratio = 0.75 # [0.25:2]
74 gcptmplpy # [Feeds and Speeds] */
75 gcptmplpy large_ball_ratio = 1.0 # [0.25:2]
76 gcptmplpy # [Feeds and Speeds] */
77 gcptmplpy small_ball_ratio = 0.75 # [0.25:2]
78 gcptmplpy # [Feeds and Speeds] */
79 gcptmplpy large_V_ratio = 0.875 # [0.25:2]
80 gcptmplpy # [Feeds and Speeds] */

```



```

151 gcptmplpy #gcp.cutarcCC(90, 180, gcp.xpos(), gcp.ypos()-stockYheight/16,
                      stockYheight/16, -stockZthickness/4)
152 gcptmplpy #gcp.cutarcCC(180, 270, gcp.xpos()+stockYheight/16, gcp.ypos(),
                      stockYheight/16, -stockZthickness/4)
153 gcptmplpy #gcp.cutarcCC(270, 360, gcp.xpos(), gcp.ypos()+stockYheight/16,
                      stockYheight/16, -stockZthickness/4)
154 gcptmplpy gcp.cutquarterCCNEdxf(gcp.xpos() - stockYheight/8, gcp.ypos() +
                      stockYheight/8, -stockZthickness/4, stockYheight/8)
155 gcptmplpy gcp.cutquarterCCNWdxf(gcp.xpos() - stockYheight/8, gcp.ypos() -
                      stockYheight/8, -stockZthickness/2, stockYheight/8)
156 gcptmplpy gcp.cutquarterCCSWdxf(gcp.xpos() + stockYheight/8, gcp.ypos() -
                      stockYheight/8, -stockZthickness * 0.75, stockYheight/8)
157 gcptmplpy gcp.cutquarterCCSEdxf(gcp.xpos() + stockYheight/8, gcp.ypos() +
                      stockYheight/8, -stockZthickness, stockYheight/8)
158 gcptmplpy
159 gcptmplpy gcp.movetosafeZ()
160 gcptmplpy gcp.rapidXY(stockXwidth/4-stockYheight/16, -stockYheight/4)
161 gcptmplpy gcp.rapidZ(0)
162 gcptmplpy
163 gcptmplpy
164 gcptmplpy #gcp.cutarcCW(180, 90, gcp.xpos()+stockYheight/16, gcp.ypos(),
                      stockYheight/16, -stockZthickness/4)
165 gcptmplpy #gcp.cutarcCW(90, 0, gcp.xpos(), gcp.ypos()-stockYheight/16,
                      stockYheight/16, -stockZthickness/4)
166 gcptmplpy #gcp.cutarcCW(360, 270, gcp.xpos()-stockYheight/16, gcp.ypos(),
                      stockYheight/16, -stockZthickness/4)
167 gcptmplpy #gcp.cutarcCW(270, 180, gcp.xpos(), gcp.ypos()+stockYheight/16,
                      stockYheight/16, -stockZthickness/4)
168 gcptmplpy
169 gcptmplpy #gcp.movetosafeZ()
170 gcptmplpy #gcp.toolchange(201, 10000)
171 gcptmplpy #gcp.rapidXY(stockXwidth/2, -stockYheight/2)
172 gcptmplpy #gcp.rapidZ(0)
173 gcptmplpy
174 gcptmplpy #gcp.cutlinedxfgc(gcp.xpos(), gcp.ypos(), -stockZthickness)
175 gcptmplpy #test = gcp.cutlinedxfgc(gcp.xpos(), gcp.ypos(), -stockZthickness)
176 gcptmplpy
177 gcptmplpy #gcp.movetosafeZ()
178 gcptmplpy #gcp.rapidXY(stockXwidth/2-6.34, -stockYheight/2)
179 gcptmplpy #gcp.rapidZ(0)
180 gcptmplpy
181 gcptmplpy #gcp.cutarcCW(180, 90, stockXwidth/2, -stockYheight/2, 6.34, -
                      stockZthickness)
182 gcptmplpy
183 gcptmplpy
184 gcptmplpy gcp.movetosafeZ()
185 gcptmplpy gcp.toolchange(814, 10000)
186 gcptmplpy gcp.rapidXY(0, -(stockYheight/2+12.7))
187 gcptmplpy gcp.rapidZ(0)
188 gcptmplpy
189 gcptmplpy gcp.cutlinedxfgc(gcp.xpos(), gcp.ypos(), -stockZthickness)
190 gcptmplpy gcp.cutlinedxfgc(gcp.xpos(), -12.7, -stockZthickness)
191 gcptmplpy
192 gcptmplpy gcp.rapidXY(0, -(stockYheight/2+12.7))
193 gcptmplpy gcp.movetosafeZ()
194 gcptmplpy gcp.toolchange(374, 10000)
195 gcptmplpy gcp.rapidXY(stockXwidth/4-stockXwidth/16, -(stockYheight/4+
                      stockYheight/16))
196 gcptmplpy gcp.rapidZ(0)
197 gcptmplpy
198 gcptmplpy gcp.rapidZ(retractheight)
199 gcptmplpy gcp.toolchange(374, 10000)
200 gcptmplpy gcp.rapidXY(-stockXwidth/4-stockXwidth/16, -(stockYheight/4+
                      stockYheight/16))
201 gcptmplpy gcp.rapidZ(0)
202 gcptmplpy
203 gcptmplpy gcp.cutline(gcp.xpos(), gcp.ypos(), -stockZthickness/2)
204 gcptmplpy gcp.cutlinedxfgc(gcp.xpos()+stockYheight/9, gcp.ypos(), gcp.zpos())
205 gcptmplpy
206 gcptmplpy gcp.cutline(gcp.xpos()-stockYheight/9, gcp.ypos(), gcp.zpos())
207 gcptmplpy gcp.cutline(gcp.xpos(), gcp.ypos(), 0)
208 gcptmplpy
209 gcptmplpy #key = gcp.cutkeyholegcdxf(KH_tool_num, 0, stockZthickness*0.75, "E
                      ", stockYheight/9)
210 gcptmplpy #key = gcp.cutKHgcdxf(374, 0, stockZthickness*0.75, 90,
                      stockYheight/9)
211 gcptmplpy #toolpaths = toolpaths.union(key)
212 gcptmplpy

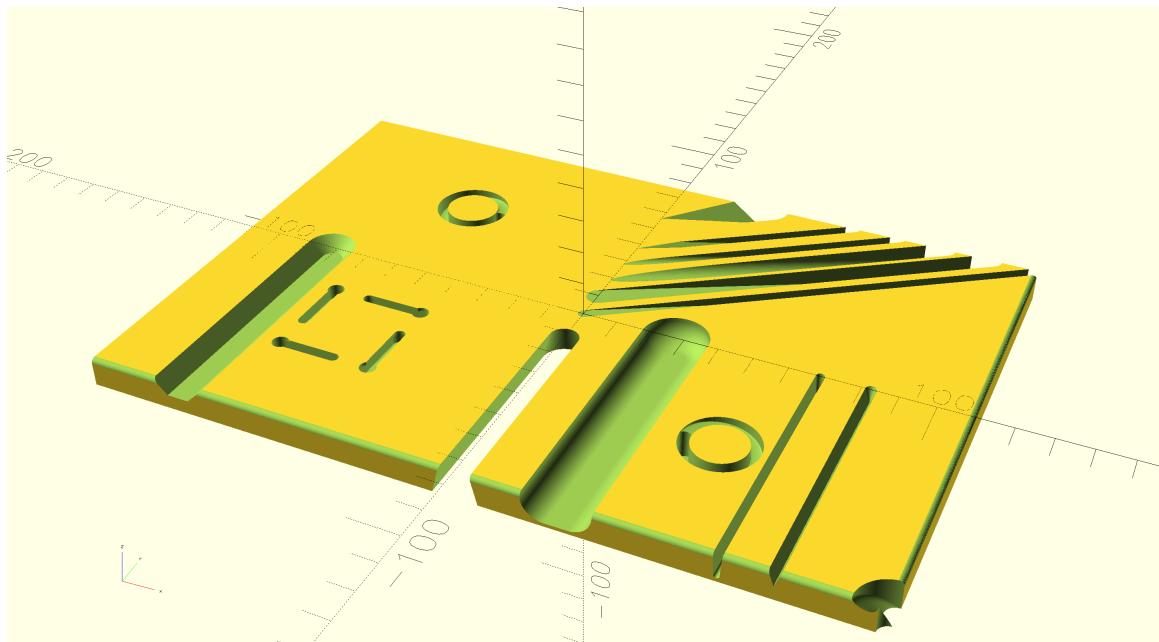
```

```

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

```

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 gcptmplscad //!OpenSCAD
2 gcptmplscad
3 gcptmplscad use <gcodepreview.py>
4 gcptmplscad include <gcodepreview.scad>
5 gcptmplscad
6 gcptmplscad $fn = $preview ? 32 : 256;
7 gcptmplscad fn = $preview ? 32 : 256;
8 gcptmplscad
9 gcptmplscad /* [Stock] */
10 gcptmplscad stockXwidth = 220;
11 gcptmplscad /* [Stock] */
12 gcptmplscad stockYheight = 150;
13 gcptmplscad /* [Stock] */
14 gcptmplscad stockZthickness = 8.35;
15 gcptmplscad /* [Stock] */
16 gcptmplscad zeroheight = "Top"; // [Top, Bottom]
17 gcptmplscad /* [Stock] */
18 gcptmplscad stockzero = "Center"; // [Lower-Left, Center-Left, Top-Left, Center]
19 gcptmplscad /* [Stock] */
20 gcptmplscad retractheight = 9;
21 gcptmplscad
22 gcptmplscad /* [Export] */
23 gcptmplscad Base_filename = "export";
24 gcptmplscad /* [Export] */
25 gcptmplscad generatedxf = true;
26 gcptmplscad /* [Export] */
27 gcptmplscad generategcode = true;
28 gcptmplscad
29 gcptmplscad /* [CAM] */
30 gcptmplscad toolradius = 1.5875;
31 gcptmplscad /* [CAM] */
32 gcptmplscad large_square_tool_num = 0; // [0:0, 112:112, 102:102, 201:201]
33 gcptmplscad /* [CAM] */
34 gcptmplscad small_square_tool_num = 102; // [0:0, 122:122, 112:112, 102:102]
35 gcptmplscad /* [CAM] */
36 gcptmplscad large_ball_tool_num = 0; // [0:0, 111:111, 101:101, 202:202]
37 gcptmplscad /* [CAM] */
38 gcptmplscad small_ball_tool_num = 0; // [0:0, 121:121, 111:111, 101:101]
39 gcptmplscad /* [CAM] */
40 gcptmplscad large_V_tool_num = 0; // [0:0, 301:301, 690:690]
41 gcptmplscad /* [CAM] */
42 gcptmplscad small_V_tool_num = 0; // [0:0, 390:390, 301:301]
43 gcptmplscad /* [CAM] */

```

```

44 gcptmplscad DT_tool_num = 0; // [0:0, 814:814, 808079:808079]
45 gcptmplscad /* [CAM] */
46 gcptmplscad KH_tool_num = 0; // [0:0, 374:374, 375:375, 376:376, 378:378]
47 gcptmplscad /* [CAM] */
48 gcptmplscad Roundover_tool_num = 0; // [56142:56142, 56125:56125, 1570:1570]
49 gcptmplscad /* [CAM] */
50 gcptmplscad MISC_tool_num = 0; // [648:648, 45982:45982]
51 gcptmplscad //648 threadmill_shaft(2.4, 0.75, 18)
52 gcptmplscad //45982 Carbide Tipped Bowl & Tray 1/4 Radius x 3/4 Dia x 5/8 x 1/4
   Inch Shank
53 gcptmplscad
54 gcptmplscad /* [Feeds and Speeds] */
55 gcptmplscad plunge = 100;
56 gcptmplscad /* [Feeds and Speeds] */
57 gcptmplscad feed = 400;
58 gcptmplscad /* [Feeds and Speeds] */
59 gcptmplscad speed = 16000;
60 gcptmplscad /* [Feeds and Speeds] */
61 gcptmplscad small_square_ratio = 0.75; // [0.25:2]
62 gcptmplscad /* [Feeds and Speeds] */
63 gcptmplscad large_ball_ratio = 1.0; // [0.25:2]
64 gcptmplscad /* [Feeds and Speeds] */
65 gcptmplscad small_ball_ratio = 0.75; // [0.25:2]
66 gcptmplscad /* [Feeds and Speeds] */
67 gcptmplscad large_V_ratio = 0.875; // [0.25:2]
68 gcptmplscad /* [Feeds and Speeds] */
69 gcptmplscad small_V_ratio = 0.625; // [0.25:2]
70 gcptmplscad /* [Feeds and Speeds] */
71 gcptmplscad DT_ratio = 0.75; // [0.25:2]
72 gcptmplscad /* [Feeds and Speeds] */
73 gcptmplscad KH_ratio = 0.75; // [0.25:2]
74 gcptmplscad /* [Feeds and Speeds] */
75 gcptmplscad R0_ratio = 0.5; // [0.25:2]
76 gcptmplscad /* [Feeds and Speeds] */
77 gcptmplscad MISC_ratio = 0.5; // [0.25:2]
78 gcptmplscad
79 gcptmplscad thegeneratedxf = generatedxf == true ? 1 : 0;
80 gcptmplscad thegenerategcode = generategcode == true ? 1 : 0;
81 gcptmplscad
82 gcptmplscad gcp = gcodepreview("cut", // or "print" (no preview not suited to
   OpenSCAD)
83 gcptmplscad           thegenerategcode,
84 gcptmplscad           thegeneratedxf,
85 gcptmplscad           );
86 gcptmplscad
87 gcptmplscad opengcodefile(Base_filename);
88 gcptmplscad opendxf(file(Base_filename));
89 gcptmplscad
90 gcptmplscad setupstock(stockXwidth, stockYheight, stockZthickness, zeroheight,
   stockzero);
91 gcptmplscad
92 gcptmplscad //echo(gcp);
93 gcptmplscad //gcpversion();
94 gcptmplscad
95 gcptmplscad //c = myfunc(4);
96 gcptmplscad //echo(c);
97 gcptmplscad
98 gcptmplscad //echo(getvv());
99 gcptmplscad
100 gcptmplscad cutline(stockXwidth/2, stockYheight/2, -stockZthickness);
101 gcptmplscad
102 gcptmplscad rapidZ(retractheight);
103 gcptmplscad toolchange(201, 10000);
104 gcptmplscad rapidXY(0, stockYheight/16);
105 gcptmplscad rapidZ(0);
106 gcptmplscad cutlinedxfgc(stockXwidth/16*7, stockYheight/2, -stockZthickness);
107 gcptmplscad
108 gcptmplscad
109 gcptmplscad rapidZ(retractheight);
110 gcptmplscad toolchange(202, 10000);
111 gcptmplscad rapidXY(0, stockYheight/8);
112 gcptmplscad rapidZ(0);
113 gcptmplscad cutlinedxfgc(stockXwidth/16*6, stockYheight/2, -stockZthickness);
114 gcptmplscad
115 gcptmplscad rapidZ(retractheight);
116 gcptmplscad toolchange(101, 10000);
117 gcptmplscad rapidXY(0, stockYheight/16*3);
118 gcptmplscad rapidZ(0);

```

```

119 gcptmplscad cutlinedxfgc(stockXwidth/16*5, stockYheight/2, -stockZthickness);
120 gcptmplscad
121 gcptmplscad rapidZ(retractheight);
122 gcptmplscad toolchange(390, 10000);
123 gcptmplscad rapidXY(0, stockYheight/16*4);
124 gcptmplscad rapidZ(0);
125 gcptmplscad
126 gcptmplscad cutlinedxfgc(stockXwidth/16*4, stockYheight/2, -stockZthickness);
127 gcptmplscad rapidZ(retractheight);
128 gcptmplscad
129 gcptmplscad toolchange(301, 10000);
130 gcptmplscad rapidXY(0, stockYheight/16*6);
131 gcptmplscad rapidZ(0);
132 gcptmplscad
133 gcptmplscad cutlinedxfgc(stockXwidth/16*2, stockYheight/2, -stockZthickness);
134 gcptmplscad
135 gcptmplscad
136 gcptmplscad movetosafeZ();
137 gcptmplscad rapid(gcp.xpos(), gcp.ypos(), retractheight);
138 gcptmplscad toolchange(102, 10000);
139 gcptmplscad
140 gcptmplscad //rapidXY(stockXwidth/4+stockYheight/8+stockYheight/16, +
               stockYheight/8);
141 gcptmplscad rapidXY(-stockXwidth/4+stockXwidth/16, (stockYheight/4));//+
               stockYheight/16
142 gcptmplscad rapidZ(0);
143 gcptmplscad
144 gcptmplscad //cutarcCW(360, 270, gcp.xpos()-stockYheight/16, gcp.ypos(),
               stockYheight/16, -stockZthickness);
145 gcptmplscad //gcp.cutarcCW(270, 180, gcp.xpos(), gcp.ypos()+stockYheight/16,
               stockYheight/16)
146 gcptmplscad //cutarcCC(0, 90, gcp.xpos()-stockYheight/16, gcp.ypos(),
               stockYheight/16, -stockZthickness/4);
147 gcptmplscad //cutarcCC(90, 180, gcp.xpos(), gcp.ypos()-stockYheight/16,
               stockYheight/16, -stockZthickness/4);
148 gcptmplscad //cutarcCC(180, 270, gcp.xpos()+stockYheight/16, gcp.ypos(),
               stockYheight/16, -stockZthickness/4);
149 gcptmplscad //cutarcCC(270, 360, gcp.xpos(), gcp.ypos()+stockYheight/16,
               stockYheight/16, -stockZthickness/4);
150 gcptmplscad
151 gcptmplscad movetosafeZ();
152 gcptmplscad //rapidXY(stockXwidth/4+stockYheight/8-stockYheight/16, -
               stockYheight/8);
153 gcptmplscad rapidXY(stockXwidth/4-stockYheight/16, -(stockYheight/4));
154 gcptmplscad rapidZ(0);
155 gcptmplscad
156 gcptmplscad //cutarcCW(180, 90, gcp.xpos()+stockYheight/16, gcp.ypos(),
               stockYheight/16, -stockZthickness/4);
157 gcptmplscad //cutarcCW(90, 0, gcp.xpos(), gcp.ypos()-stockYheight/16,
               stockYheight/16, -stockZthickness/4);
158 gcptmplscad //cutarcCW(360, 270, gcp.xpos()-stockYheight/16, gcp.ypos(),
               stockYheight/16, -stockZthickness/4);
159 gcptmplscad //cutarcCW(270, 180, gcp.xpos(), gcp.ypos()+stockYheight/16,
               stockYheight/16, -stockZthickness/4);
160 gcptmplscad
161 gcptmplscad movetosafeZ();
162 gcptmplscad
163 gcptmplscad rapidXY(-stockXwidth/4 + stockYheight/8, (stockYheight/4));
164 gcptmplscad rapidZ(0);
165 gcptmplscad
166 gcptmplscad cutquarterCCNEdxf(xpos() - stockYheight/8, ypos() + stockYheight/8,
               -stockZthickness/4, stockYheight/8);
167 gcptmplscad cutquarterCCNWdxf(xpos() - stockYheight/8, ypos() - stockYheight/8,
               -stockZthickness/2, stockYheight/8);
168 gcptmplscad cutquarterCCSWdxf(xpos() + stockYheight/8, ypos() - stockYheight/8,
               -stockZthickness * 0.75, stockYheight/8);
169 gcptmplscad //cutquarterCCSEdxf(xpos() + stockYheight/8, ypos() + stockYheight
               /8, -stockZthickness, stockYheight/8);
170 gcptmplscad
171 gcptmplscad movetosafeZ();
172 gcptmplscad toolchange(201, 10000);
173 gcptmplscad rapidXY(stockXwidth /2 -6.34, - stockYheight /2);
174 gcptmplscad rapidZ(0);
175 gcptmplscad //cutarcCW(180, 90, stockXwidth /2, -stockYheight/2, 6.34, -
               stockZthickness);
176 gcptmplscad
177 gcptmplscad movetosafeZ();
178 gcptmplscad rapidXY(stockXwidth/2, -stockYheight/2);

```

```

179 gcptmplscad rapidZ(0);
180 gcptmplscad
181 gcptmplscad //gcp.cutlinedxfgc(gcp.xpos(), gcp.ypos(), -stockZthickness);
182 gcptmplscad
183 gcptmplscad movetosafeZ();
184 gcptmplscad toolchange(814, 10000);
185 gcptmplscad rapidXY(0, -(stockYheight/2+12.7));
186 gcptmplscad rapidZ(0);
187 gcptmplscad
188 gcptmplscad cutlinedxfgc(xpos(), ypos(), -stockZthickness);
189 gcptmplscad cutlinedxfgc(xpos(), -12.7, -stockZthickness);
190 gcptmplscad rapidXY(0, -(stockYheight/2+12.7));
191 gcptmplscad
192 gcptmplscad //rapidXY(stockXwidth/2-6.34, -stockYheight/2);
193 gcptmplscad //rapidZ(0);
194 gcptmplscad
195 gcptmplscad //movetosafeZ();
196 gcptmplscad //toolchange(374, 10000);
197 gcptmplscad //rapidXY(-(stockXwidth/4 - stockXwidth /16), -(stockYheight/4 +
    stockYheight/16))
198 gcptmplscad
199 gcptmplscad //cutline(xpos(), ypos(), (stockZthickness/2) * -1);
200 gcptmplscad //cutlinedxfgc(xpos() + stockYheight /9, ypos(), zpos());
201 gcptmplscad //cutline(xpos() - stockYheight /9, ypos(), zpos());
202 gcptmplscad //cutline(xpos(), ypos(), 0);
203 gcptmplscad
204 gcptmplscad movetosafeZ();
205 gcptmplscad
206 gcptmplscad toolchange(374, 10000);
207 gcptmplscad rapidXY(-stockXwidth/4-stockXwidth/16, -(stockYheight/4+
    stockYheight/16))
208 gcptmplscad //rapidXY(-(stockXwidth/4 - stockXwidth /16), -(stockYheight/4 +
    stockYheight/16))
209 gcptmplscad rapidZ(0);
210 gcptmplscad
211 gcptmplscad cutline(xpos(), ypos(), (stockZthickness/2) * -1);
212 gcptmplscad cutlinedxfgc(xpos() + stockYheight /9, ypos(), zpos());
213 gcptmplscad cutline(xpos() - stockYheight /9, ypos(), zpos());
214 gcptmplscad cutline(xpos(), ypos(), 0);
215 gcptmplscad
216 gcptmplscad rapidZ(retractheight);
217 gcptmplscad rapidXY(-stockXwidth/4+stockXwidth/16, -(stockYheight/4+
    stockYheight/16));
218 gcptmplscad rapidZ(0);
219 gcptmplscad cutline(gcp.xpos(), gcp.ypos(), -stockZthickness/2);
220 gcptmplscad cutlinedxfgc(gcp.xpos(), gcp.ypos()+stockYheight/9, gcp.zpos());
221 gcptmplscad cutline(gcp.xpos(), gcp.ypos()-stockYheight/9, gcp.zpos());
222 gcptmplscad cutline(gcp.xpos(), gcp.ypos(), 0);
223 gcptmplscad
224 gcptmplscad rapidZ(retractheight);
225 gcptmplscad rapidXY(-stockXwidth/4+stockXwidth/16, -(stockYheight/4-
    stockYheight/8));
226 gcptmplscad rapidZ(0);
227 gcptmplscad cutline(gcp.xpos(), gcp.ypos(), -stockZthickness/2);
228 gcptmplscad cutlinedxfgc(gcp.xpos()-stockYheight/9, gcp.ypos(), gcp.zpos());
229 gcptmplscad cutline(gcp.xpos()+stockYheight/9, gcp.ypos(), gcp.zpos());
230 gcptmplscad cutline(gcp.xpos(), gcp.ypos(), 0);
231 gcptmplscad
232 gcptmplscad rapidZ(retractheight);
233 gcptmplscad rapidXY(-stockXwidth/4-stockXwidth/16, -(stockYheight/4-
    stockYheight/8));
234 gcptmplscad rapidZ(0);
235 gcptmplscad cutline(gcp.xpos(), gcp.ypos(), -stockZthickness/2);
236 gcptmplscad cutlinedxfgc(gcp.xpos(), gcp.ypos()-stockYheight/9, gcp.zpos());
237 gcptmplscad cutline(gcp.xpos(), gcp.ypos()+stockYheight/9, gcp.zpos());
238 gcptmplscad cutline(gcp.xpos(), gcp.ypos(), 0);
239 gcptmplscad
240 gcptmplscad rapidZ(retractheight);
241 gcptmplscad toolchange(45982, 10000);
242 gcptmplscad rapidXY(stockXwidth/8, 0);
243 gcptmplscad cutline(gcp.xpos(), gcp.ypos(), -(stockZthickness*7/8));
244 gcptmplscad cutlinedxfgc(gcp.xpos(), -stockYheight/2, -(stockZthickness*7/8));
245 gcptmplscad
246 gcptmplscad rapidZ(retractheight);
247 gcptmplscad toolchange(204, 10000);
248 gcptmplscad rapidXY(stockXwidth*0.3125, 0);
249 gcptmplscad cutline(gcp.xpos(), gcp.ypos(), -(stockZthickness*7/8));
250 gcptmplscad cutlinedxfgc(gcp.xpos(), -stockYheight/2, -(stockZthickness*7/8));

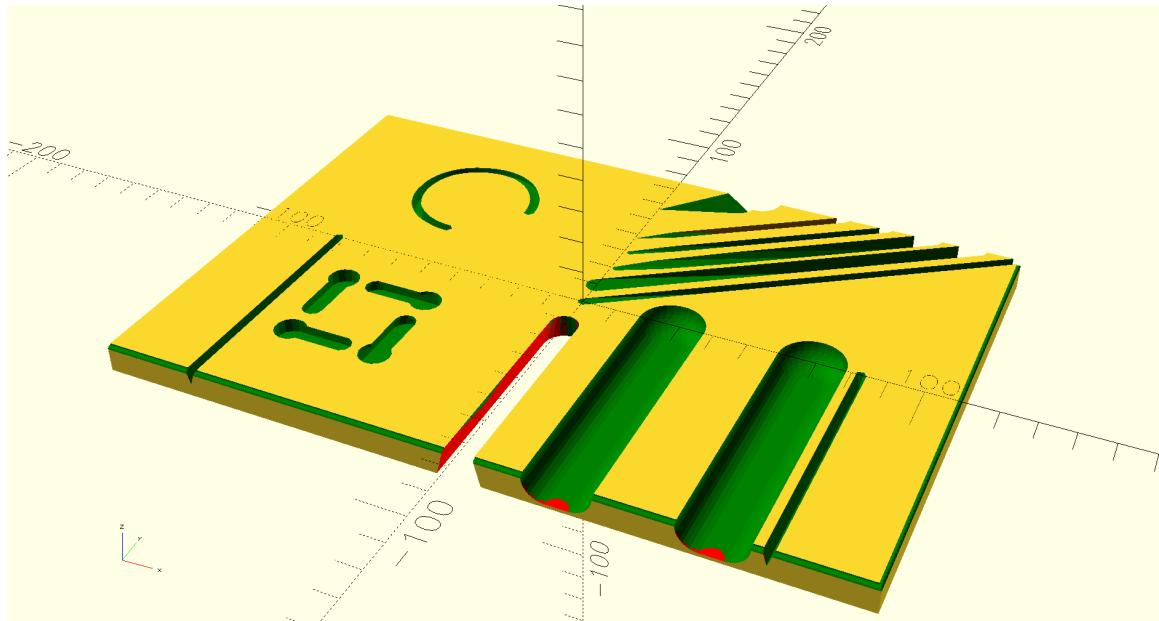
```

```

251 gcptmplscad
252 gcptmplscad rapidZ(retractheight);
253 gcptmplscad toolchange(502, 10000);
254 gcptmplscad rapidXY(stockXwidth*0.375, 0);
255 gcptmplscad cutline(gcp.xpos(), gcp.ypos(), -4.24);
256 gcptmplscad cutlinedxfgc(gcp.xpos(), -stockYheight/2, -4.24);
257 gcptmplscad
258 gcptmplscad rapidZ(retractheight);
259 gcptmplscad toolchange(13921, 10000);
260 gcptmplscad rapidXY(-stockXwidth*0.375, 0);
261 gcptmplscad cutline(gcp.xpos(), gcp.ypos(), -stockZthickness/2);
262 gcptmplscad cutlinedxfgc(gcp.xpos(), -stockYheight/2, -stockZthickness/2);
263 gcptmplscad
264 gcptmplscad rapidZ(retractheight);
265 gcptmplscad gcp.toolchange(56142, 10000);
266 gcptmplscad gcp.rapidXY(-stockXwidth/2, -(stockYheight/2+0.508/2));
267 gcptmplscad cutlineZgcfeed(-1.531, plunge);
268 gcptmplscad //cutline(gcp.xpos(), gcp.ypos(), -1.531);
269 gcptmplscad cutlinedxfgc(stockXwidth/2+0.508/2, -(stockYheight/2+0.508/2),
-1.531);
270 gcptmplscad
271 gcptmplscad rapidZ(retractheight);
272 gcptmplscad //#gcp.toolchange(56125, 10000)
273 gcptmplscad cutlineZgcfeed(-1.531, plunge);
274 gcptmplscad //toolpaths.append(gcp.cutline(gcp.xpos(), gcp.ypos(), -1.531))
275 gcptmplscad cutlinedxfgc(stockXwidth/2+0.508/2, (stockYheight/2+0.508/2),
-1.531);
276 gcptmplscad
277 gcptmplscad stockandtoolpaths();
278 gcptmplscad //stockwotoolpaths();
279 gcptmplscad //outputtoolpaths();
280 gcptmplscad
281 gcptmplscad //makecube(3, 2, 1);
282 gcptmplscad
283 gcptmplscad //instantiatecube();
284 gcptmplscad
285 gcptmplscad closegcodefile();
286 gcptmplscad closedxffile();

```

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



2.5 gpcthreedp.py

Setting up 3D printing will require accommodating the requirements of both the printer *and* filament being used. The most straight-forward and expedient way to arrive at this is to leverage a traditional 3D printer slicer which has settings appropriate to the machine and filament being used which are tuned to the sort of part being made/printing being done, export the G-code, and use that as a template for setting up 3D printing.

Towards that end, a G-code file for a very basic 3D printer was output for printing PLA from an Orbot Quantum

```

1 gcpthreedp #gcpthreedp.py --- Template for 3D printing
2 gcpthreedp #                                     Initial version.
3 gcpthreedp #!/usr/bin/env python
4 gcpthreedp
5 gcpthreedp import sys
6 gcpthreedp
7 gcpthreedp try:
8 gcpthreedp     if 'gcodepreview' in sys.modules:
9 gcpthreedp         del sys.modules['gcodepreview']
10 gcpthreedp except AttributeError:
11 gcpthreedp     pass
12 gcpthreedp
13 gcpthreedp from gcodepreview import *
14 gcpthreedp
15 gcpthreedp fa = 2
16 gcpthreedp fs = 0.125
17 gcpthreedp
18 gcpthreedp # [Export] */
19 gcpthreedp Base_filename = "aexport"
20 gcpthreedp # [Export] */
21 gcpthreedp generatedxf = False
22 gcpthreedp # [Export] */
23 gcpthreedp generategcode = True
24 gcpthreedp # [3D Printing] */
25 gcpthreedp printer_name = 'prusa_i3' # generic / ultimaker2plus / prusa_i3 /
                                         ender_3 / cr_10 / bambulab_x1 / toolchanger_T0
26 gcpthreedp # [3D Printing] */
27 gcpthreedp nozzlediameter = 0.4
28 gcpthreedp # [3D Printing] */
29 gcpthreedp filamentdiameter = 1.75
30 gcpthreedp # [3D Printing] */
31 gcpthreedp extrusionwidth = 0.6
32 gcpthreedp # [3D Printing] */
33 gcpthreedp layerheight = 0.2
34 gcpthreedp # [3D Printing] */
35 gcpthreedp extruder_temperature = 200
36 gcpthreedp # [3D Printing] */
37 gcpthreedp bed_temperature = 60
38 gcpthreedp
39 gcpthreedp gcp = gcodepreview("print", # "cut" or "no_preview"
40 gcpthreedp                               generategcode,
41 gcpthreedp                               generatedxf,
42 gcpthreedp                               )
43 gcpthreedp
44 gcpthreedp gcp.initializeforprinting(nozzlediameter,
45 gcpthreedp                               filamentdiameter,
46 gcpthreedp                               extrusionwidth,
47 gcpthreedp                               layerheight,
48 gcpthreedp                               "absolute",
49 gcpthreedp                               extruder_temperature,
50 gcpthreedp                               bed_temperature,
51 gcpthreedp                               printer_name,
52 gcpthreedp                               Base_filename)
53 gcpthreedp
54 gcpthreedp gcp.extrude(9, 18, layerheight)
55 gcpthreedp
56 gcpthreedp gcp.rapid(125, 125, layerheight)
57 gcpthreedp gcp.extrude(150, 125, layerheight)
58 gcpthreedp gcp.extrude(150, 150, layerheight)
59 gcpthreedp gcp.extrude(125, 150, layerheight)
60 gcpthreedp gcp.extrude(125, 125, layerheight)
61 gcpthreedp
62 gcpthreedp gcp.stockandtoolpaths("toolpaths")
63 gcpthreedp
64 gcpthreedp gcp.shutdownafterprinting()

```

2.6 gcodepreviewtemplate.txt

Throughout this document, examples of commands will be shown and then collected in gcodepreviewtemplate.txt for easy copy-pasting (insert old computer joke about how many original Cobol programs have been written).

```

1 gcptmpl #gcptemplate.txt --- this file will collect example usages of each
2 gcptmpl #                                     command with a brief commentary.

```

3 gcodereview

This library for OpenPythonSCAD works by using Python code to persistently store and access variables which denote the machine position and describe the characteristics of tools, 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 loading up to two files:

- A Python file: gcodereview.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 gcodereview_standalone as gcp
or used in an OpenSCAD file:
```

```
use <gcodereview.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: gcodereview.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 is expedient, or even 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 stored 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 are added where appropriate to help define the functionality of each defined module in Python. <https://numpydoc.readthedocs.io/en/latest/>.

3.1 Tools for 3D Previewing G-code

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/>.

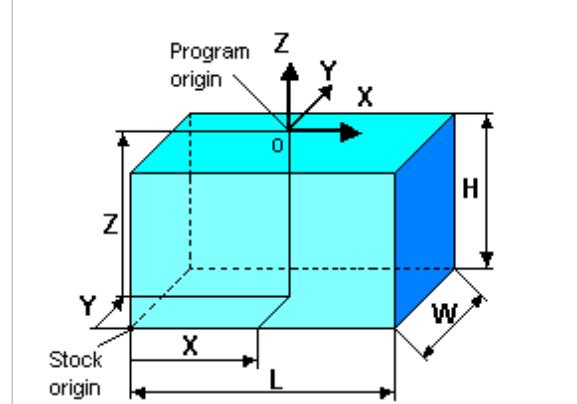
3.1.1 Cutviewer

Cutviewer is a notable commercial program which has a unique approach centered on G-code where specially formatted comments fill in the dimensions and other parameters needed for showing the 3D preview.

3.1.1.1 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.

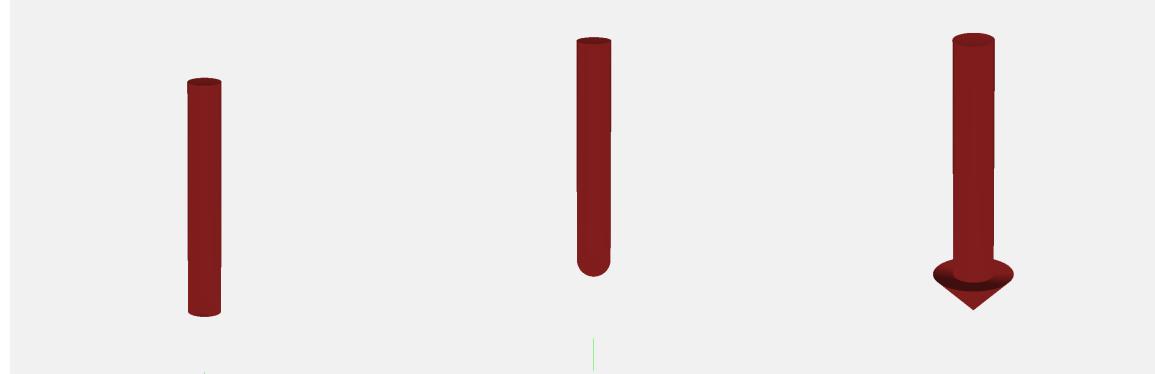
The CutViewer comments are in the form:

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



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

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

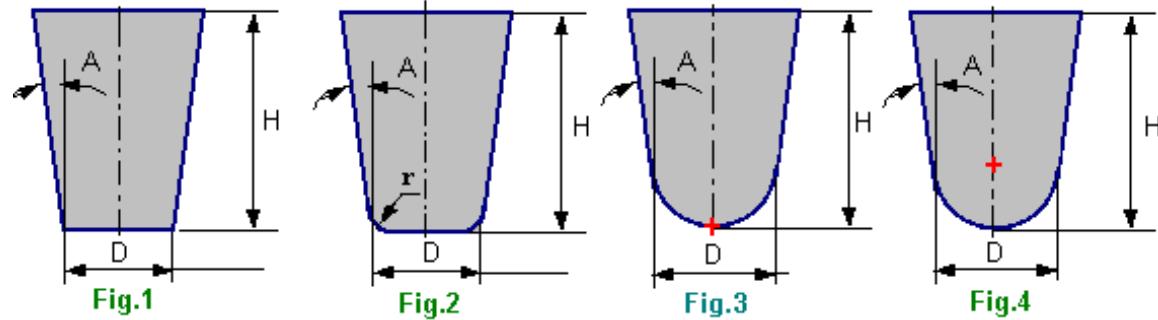


- Square (#201 and 102) — able to cut a flat bottom, perpendicular side and right angle, their simple and easily understood geometry makes them a standard choice
- Ballnose (#202 and 101) — rounded, they are the standard choice for concave and organic shapes
- V tooling (#301, 302, 311 and 312) — pointed 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() or rotate_extrude commands which describe the 3D region which tool movement describes.

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

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

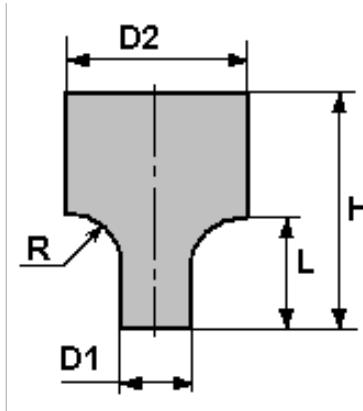


Note that it is possible to use these definitions for a wide variety of tooling, e.g., a Carbide 3D #301 V tool being represented as:

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

3.1.1.2.2 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



3.1.1.2.3 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 beginning of wisdom is to call things by their right names.

— CONFUCIUS

One of the hard things in computer science, naming modules (and certain variables) requires that the conventions of G-code, the various file types which are written to, and the actions which the system takes are all taken into due consideration so as to arrive at a consistent scheme.

Number will be abbreviated as num rather than no, and the short form will be used internally for variable names, while the compleat 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.

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
 - extrude (action) — 3D printing equivalent to cut
 - 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 (geometry/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 theory, 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		cutarcdxf	dxfarc	
gcode	cutlinegc		linegc	cutarcgc		arcgc
			cutlinedxfgc			cutarcdxfgc

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

The gcode and dxf columns and the matter of having specific commands which encompass generategcode those file types is tied up in having the internal variables generategcode, generatedxf and im-generatedxf plementations, and a strong argument could be made that this should simply be handled by generatecut if...then structures using those variables. The addition of a generatecut variable adds the necessary symmetry. Note that an early option to output a separate file for each tool used has since been deprecated and removed. The need for it was addressed by instead using colour-coded layers. A future update may add support for grouping elements.

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/.gcode 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)`, `dxf`, `gcode`, `tool`, etc.) note the `gcodepreview` class which will normally be imported as `gcp` so that module `<foo>` will be called as `gcp.<foo>` from Python and by the same `<foo>` in OpenSCAD

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 `dxf` files and using mutable variables (this is done in current versions of this project)
- OpenSCAD — see: <https://openscad.org/documentation.html>
- Programming in Python, calling Python from OpenSCAD using dispatchers/descriptors (this is done in current versions of this project)

- 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 up through 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=\thegpcscad}
\begin{writecode}{a}{gcodepreview.scad}{scad}
module FOOBAR(...) {
    oFOOBAR(...);
}

\end{writecode}
\addtocounter{gpcscad}{4}

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

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

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

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

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

...

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

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

Lastly note that this style of programming was abandoned in favour of object-oriented dot notation 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 (gcpversion) 0.93, 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\u2022module\u2022not\u2022loaded.")
18 gcpy
19 gcpy def pygcpversion():
20 gcpy     thegcpversion = 0.931
21 gcpy     return thegcpversion

```

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

```

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

```

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

- Mandatory
 - gcodepreview (init)
 - * generatecut, generatedxf, generategcode
 - stocksetup:
 - * stockXwidth, stockYheight, stockZthickness, zeroheight, stockzero, retractheight
 - gcpfiles:
 - * basefilename
 - largesquaretool:
 - * large_square_tool_num, toolradius, plunge, feed, speed
 - currenttoolnum
 - * endmilltype
 - * diameter
 - * flute
 - * shaftdiameter
 - * shaftheight
 - * shaftlength
 - * toolnumber
 - * cutcolor

- * rapidcolor
- * shaftcolor
- Optional
 - smallsquaretool:
 - * small_square_tool_num, small_square_ratio
 - largeballtool:
 - * large_ball_tool_num, large_ball_ratio
 - largeVtool:
 - * large_V_tool_num, large_V_ratio
 - smallballtool:
 - * small_ball_tool_num, small_ball_ratio
 - smallVtool:
 - * small_V_tool_num, small_V_ratio
 - DTtool:
 - * DT_tool_num, DT_ratio
 - KHtool:
 - * KH_tool_num, KH_ratio
 - Roundovertool:
 - * Roundover_tool_num, RO_ratio
 - misctool:
 - * MISC_tool_num, MISC_ratio

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

3.3.1 init

Initialization of the `gcodepreview` object requires handling a number of different cases, two of which are exclusive to each other. It must also take into account the possibility of being called from OpenSCAD

```

23 gcpy class gcodepreview:
24 gcpy
25 gcpy     def __init__(self,
26 gcpy         cutterprint = "cut", # "cut", "print", "no_preview"
27 gcpy         generategcode = False,
28 gcpy         generatedxf = False,
29 gcpy         gcpfa = 2,
30 gcpy         gcpfs = 0.125,
31 gcpy         steps = 10
32 gcpy         ):
33 gcpy         """
34 gcpy             Initialize gcodepreview object.
35 gcpy
36 gcpy             Parameters
37 gcpy             -----
38 gcpy             cutterprint : string
39 gcpy                 Enables creation of 3D model for cutting or
40 gcpy                 printing.
41 gcpy             generategcode : boolean
42 gcpy                 Enables writing out G-code.
43 gcpy             generatedxf : boolean
44 gcpy                 Enables writing out a DXF file.
45 gcpy
46 gcpy             Returns
47 gcpy             -----
48 gcpy             object
49 gcpy                 The initialized gcodepreview object.
50 gcpy             """
51 gcpy             if cutterprint == "print":
52 gcpy                 self.generatecut = False
53 gcpy                 self.generateprint = True
54 gcpy                 self.gcodefileext = ".gcode"
55 gcpy             elif cutterprint == "cut":
56 gcpy                 self.generatecut = True
57 gcpy                 self.generateprint = False
58 gcpy                 self.gcodefileext = ".nc"
59 gcpy             else: # no_preview

```

```

59 gcpy           self.generatecut = False
60 gcpy           self.generateprint = False
61 gcpy           if generategcode == True:
62 gcpy             self.generategcode = True
63 gcpy           elif generategcode == 1:
64 gcpy             self.generategcode = True
65 gcpy           elif generategcode == 0:
66 gcpy             self.generategcode = False
67 gcpy           else:
68 gcpy             self.generategcode = generategcode
69 gcpy           if generatedxf == True:
70 gcpy             self.generatedxf = True
71 gcpy           elif generatedxf == 1:
72 gcpy             self.generatedxf = True
73 gcpy           elif generatedxf == 0:
74 gcpy             self.generatedxf = False
75 gcpy           else:
76 gcpy             self.generatedxf = generatedxf
77 gcpy # set up 3D previewing parameters
78 gcpy           fa = gcpfa
79 gcpy           fs = gcpfs
80 gcpy           self.steps = steps
81 gcpy # initialize the machine state
82 gcpy           self.mc = "Initialized"
83 gcpy           self.mpx = float(0)
84 gcpy           self.mpy = float(0)
85 gcpy           self.mpz = float(0)
86 gcpy           self.tpz = float(0)
87 gcpy # initialize the toolpath state
88 gcpy           self.retractheight = 5
89 gcpy # initialize the DEFAULT tool
90 gcpy           self.currenttoolnum = 102
91 gcpy           self.endmilltype = "square"
92 gcpy           self.diameter = 3.175
93 gcpy           self.flute = 12.7
94 gcpy           self.shaftdiameter = 3.175
95 gcpy           self.shaftheight = 12.7
96 gcpy           self.shaftlength = 19.5
97 gcpy           self.toolnumber = "100036"
98 gcpy           self.cutcolor = "green"
99 gcpy           self.rapidcolor = "orange"
100 gcpy          self.shaftcolor = "red"
101 gcpy # the command definesquaretool(3.175, 12.7, 20) is used in the
        toolchange command
102 gcpy          self.tooloutline = polygon( points
        = [[0,0],[3.175,0],[3.175,12.7],[0,12.7]] )
103 gcpy          self.toolprofile = polygon( points
        = [[0,0],[1.5875,0],[1.5875,12.7],[0,12.7]] )
104 gcpy          self.shaftoutline = polygon( points
        = [[0,12.7],[3.175,12.7],[3.175,25.4],[0,25.4]] )
105 gcpy          self.shaftprofile = polygon( points
        = [[0,12.7],[1.5875,12.7],[1.5875,25.4],[0,25.4]] )
106 gcpy          self.currenttoolshape = cylinder(h = self.flute, r = self.
        shaftdiameter/2)
107 gcpy          sh = cylinder(h = self.flute, r = self.shaftdiameter/2)
108 gcpy          self.currenttoolshaft = sh.translate([0,0,self.flute])
109 gcpy # debug mode requires a variable to track if it is on or off
110 gcpy          self.debugenable = False
111 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
112 gcpy          self.rapids = []
113 gcpy          self.toolpaths = []
114 gcpy          print("gcodepreview\u00d7class\u00d7initialized")
115 gcpy
116 gcpy #     def myfunc(self, var):
117 gcpy #         self.vv = var * var
118 gcpy #         return self.vv
119 gcpy #
120 gcpy #     def getvv(self):
121 gcpy #         return self.vv
122 gcpy #
123 gcpy #     def checkint(self):
124 gcpy #         return self.mc
125 gcpy #
126 gcpy #     def makecube(self, xdim, ydim, zdim):
127 gcpy #         self.c=cube([xdim, ydim, zdim])
128 gcpy #

```

```

129 gcpy #     def placecube(self):
130 gcpy #         show(self.c)
131 gcpy #
132 gcpy #     def instantiatecube(self):
133 gcpy #         return self.c

```

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

`currenttool` The 3D model of the tool is stored in `currenttool`.

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

`zpos`

```

135 gcpy     def xpos(self):
136 gcpy         return self.mpx
137 gcpy
138 gcpy     def ypos(self):
139 gcpy         return self.mpy
140 gcpy
141 gcpy     def zpos(self):
142 gcpy         return self.mpz

```

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

```

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

```

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

`setypos`

```

144 gcpy     def setxpos(self, newxpos):
145 gcpy         self.mpx = newxpos
146 gcpy
147 gcpy     def setypos(self, newypos):
148 gcpy         self.mpy = newypos
149 gcpy
150 gcpy     def setzpos(self, newzpos):
151 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.

3.3.3 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.

`toolmovement` The `toolmovement` class requires that the tool be defined in terms of `endmilltype`, `diameter`, `endmilltype` `flute` (`length`), `ra` (radius or angle depending on context), and `tip`, and there is a mechanism `diameter` which defines an internal tool number as described below. Currently though, the interface calls `flute` the `toolchange` routine passing in a manufacturer tool number as an expedient/default/initial `ra` option.

`tip` There are two variables to record `toolmovement`, `rapids` and `toolpaths`. Initialized as empty `toolmovement` lists, `toolmovements` will be extended to the lists, then for output, the lists will be expanded and `rapids` subtracted from the stock separately so that `rapids` are colour-coded so that if there is an interaction with the stock at rapid speed it will be obvious. A similar method should be implemented for the shafts of tooling.

`gcodepreview 3.3.3.1 setupstock` The first such setup subroutine is `gcodepreview setupstock` which is `setupstock` appropriately enough, to set up the stock, and perform other initializations — initially, the only thing done in Python was to set the value of the persistent (Python) variables (see `initializemachinestate()` above), but the rewritten standalone version handles all necessary actions.

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

```

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

```

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

```

189 gcpy      if self.zeroheight == "Top":
190 gcpy          if self.stockzero == "Lower-Left":
191 gcpy              self.stock = self.stock.translate([0, 0, -self.
192 gcpy                      stockZthickness])
193 gcpy          if self.generategcode == True:

```

```

193 gcpy
    self.writegc("(stockMin:0.00mm,0.00mm,-", str
                  (self.stockZthickness), "mm"))
194 gcpy
    self.writegc("(stockMax:", str(self.stockXwidth
                  ), "mm,", str(stockYheight), "mm,0.00mm)")
195 gcpy
    self.writegc("(STOCK/BLOCK,", str(self.
                  stockXwidth), ",,", str(self.stockYheight),
                  ",,", str(self.stockZthickness), ",0.00,0.
                  00,", str(self.stockZthickness), ")")
196 gcpy
    if self.stockzero == "Center-Left":
        self.stock = self.stock.translate([0, -stockYheight
                  / 2, -stockZthickness])
197 gcpy
    if self.generategcode == True:
        self.writegc("(stockMin:0.00mm,-", str(self.
                  stockYheight/2), "mm,-", str(self.
                  stockZthickness), "mm)")
198 gcpy
199 gcpy
    self.writegc("(stockMax:", str(self.stockXwidth
                  ), "mm,", str(self.stockYheight/2), "mm,
                  0.00mm)")
200 gcpy
    self.writegc("(STOCK/BLOCK,", str(self.
                  stockXwidth), ",,", str(self.stockYheight),
                  ",,", str(self.stockZthickness), ",0.00,,",
                  str(self.stockYheight/2), ",,", str(self.
                  stockZthickness), ")");
201 gcpy
    if self.stockzero == "Top-Left":
        self.stock = self.stock.translate([0, -self.
                  stockYheight, -self.stockZthickness])
202 gcpy
    if self.generategcode == True:
        self.writegc("(stockMin:0.00mm,-", str(self.
                  stockYheight), "mm,-", str(self.
                  stockZthickness), "mm)")
203 gcpy
    self.writegc("(stockMax:", str(self.stockXwidth
                  ), "mm,0.00mm,0.00mm)")
204 gcpy
205 gcpy
    self.writegc("(STOCK/BLOCK,", str(self.
                  stockXwidth), ",,", str(self.stockYheight),
                  ",,", str(self.stockZthickness), ",0.00,,",
                  str(self.stockYheight), ",,", str(self.
                  stockZthickness), ")");
206 gcpy
    if self.stockzero == "Center":
        self.stock = self.stock.translate([-self.
                  stockXwidth / 2, -self.stockYheight / 2, -self.
                  stockZthickness])
207 gcpy
    if self.generategcode == True:
        self.writegc("(stockMin:-", str(self.
                  stockXwidth/2), ",-", str(self.stockYheight
                  /2), "mm,-", str(self.stockZthickness), "mm
                  )")
208 gcpy
    self.writegc("(stockMax:", str(self.stockXwidth
                  /2), "mm,", str(self.stockYheight/2), "mm,
                  0.00mm)")
209 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), ")");
210 gcpy
211 gcpy
    if self.generategcode == True:
        self.writegc("(stockMin:-", str(self.
                  stockXwidth/2), ",-", str(self.stockYheight
                  /2), "mm,-", str(self.stockZthickness), "mm
                  )")
212 gcpy
    self.writegc("(stockMax:", str(self.stockXwidth
                  /2), "mm,", str(self.stockYheight/2), "mm,
                  0.00mm)")
213 gcpy
    self.writegc("(STOCK/BLOCK,", str(self.
                  stockXwidth), ",,", str(self.stockYheight),
                  ",,", str(self.stockZthickness), ",",
                  str(self.stockXwidth/2), ",,", str(self.
                  stockYheight/2), ",,", str(self.
                  stockZthickness), ")");
214 gcpy
    if self.zeroheight == "Bottom":
        if self.stockzero == "Lower-Left":
            self.stock = self.stock.translate([0, 0, 0])
215 gcpy
        if self.generategcode == True:
            self.writegc("(stockMin:0.00mm,0.00mm,0.00mm
                          )")
216 gcpy
        self.writegc("(stockMax:", str(self.
                      stockXwidth), "mm,", str(self.stockYheight
                      ), "mm,", str(self.stockZthickness), "mm
                      )")
217 gcpy
218 gcpy
        self.writegc("(STOCK/BLOCK,", str(self.
                      stockXwidth), ",,", str(self.stockYheight),
                      ",,", str(self.stockZthickness), ",0.00,0.
                      00,0.00)")
219 gcpy
    if self.stockzero == "Center-Left":
        self.stock = self.stock.translate([0, -self.
                  stockYheight / 2, 0])
220 gcpy
        if self.generategcode == True:
            self.writegc("(stockMin:0.00mm,-", str(self.
                  stockYheight/2), "mm,0.00mm)")
221 gcpy
        self.writegc("(stockMax:", str(self.stockXwidth
                      ), "mm,", str(self.stockYheight/2), "mm,
                      -,", str(self.stockZthickness), "mm")
222 gcpy
    if self.generategcode == True:
        self.writegc("(stockMin:0.00mm,-", str(self.
                  stockYheight/2), "mm,0.00mm)")
223 gcpy
224 gcpy
        self.writegc("(stockMax:", str(self.stockXwidth
                      ), "mm,", str(self.stockYheight/2), "mm,-",
                      ",", str(self.stockZthickness), "mm)")
225 gcpy

```

```

226 gcpy                         self.writegc("(STOCK/BLOCK, " , str(self.
227 gcpy                           stockXwidth) , " , " , str(self.stockYheight) ,
228 gcpy                           " , " , str(self.stockZthickness) , " , "0.00 , " ,
229 gcpy                           str(self.stockYheight / 2) , " , "0.00mm)");
230 gcpy                           if self.stockzero == "Top-Left":
231 gcpy                             self.stock = self.stock.translate([0, -self.
232 gcpy                               stockYheight, 0])
233 gcpy                           if self.generategcode == True:
234 gcpy                             self.writegc("(stockMin:0.00mm," , str(self.
235 gcpy                               stockYheight) , "mm, "0.00mm)")
236 gcpy                           self.writegc("(stockMax:", str(self.stockXwidth
237 gcpy                               ) , "mm, "0.00mm , " , str(self.stockZthickness)
238 gcpy                               , "mm)" )
239 gcpy                           self.writegc("(STOCK/BLOCK," , str(self.
240 gcpy                             stockXwidth) , " , " , str(self.stockYheight),
241 gcpy                             " , " , str(self.stockZthickness) , " , "0.00 , " ,
242 gcpy                           str(self.stockYheight) , " , "0.00)" )
243 gcpy                           if self.generategcode == True:
244 gcpy                             self.writegc("G90");
245 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 gpcscad module setupstock(stockXwidth, stockYheight, stockZthickness,
37 gpcscad   zeroheight, stockzero, retractheight) {
38 gpcscad     gcp.setupstock(stockXwidth, stockYheight, stockZthickness,
39 gpcscad       zeroheight, stockzero, retractheight);
40 gpcscad }

```

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

```

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

```

(Note that since this command will only be done in the process of inputting a G-code file, there is no need for a matching OpenSCAD module.)

3.3.3.3 debug Rather than endlessly add and then comment out `print()` commands, it is easier to have a variable for this, and a command which wraps the command which checks for that:

```

251 gcpy     def debug(self, *args: any, sep: str = " " , end: str = "\n", **
252 gcpy       print_kwargs) -> None:
253 gcpy       """
254 gcpy         Print debug output if enabled.

```

```

255 gcpy      Accepts the same arguments as built-in print (except file
               is supported via print_kwargs).
256 gcpy      """
257 gcpy      if not self.debugenable:
258 gcpy          return
259 gcpy      # Build the message and print under a lock to avoid
               interleaving in multithreaded apps
260 gcpy      self.prefix = "DEBUG: "
261 gcpy      msg = self.prefix + sep.join(map(str, args))
262 gcpy      with self._lock:
263 gcpy          print(msg, end=end, **print_kwargs)

```

Note that it will be necessary to manually use commands such as:

```

3 gcptmpl self.debugenable = True
4 gcptmpl
5 gcptmpl testvariable = 1
6 gcptmpl
7 gcptmpl self.outputdebugnote("Current value of testvariable is: ",
                               testvariable)

```

3.3.4 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:

```

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

```

```

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

```

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

```

268 gcpy      def addtostock(self, stockXwidth, stockYheight, stockZthickness
                  ,
                  shiftX = 0,
                  shiftY = 0,
                  shiftZ = 0):
271 gcpy          addedpart = cube([stockXwidth, stockYheight,
                  stockZthickness])
272 gcpy          addedpart = addedpart.translate([shiftX, shiftY, shiftZ])
273 gcpy          self.stock = self.stock.union(addedpart)
274 gcpy

```

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 rapids variables (rapids, toolpaths) as lists processed/created using a command toolmovement which toolpaths will subsume all tool related functionality. As other routines need access to information about the toolmovement current tool, appropriate routines will allow its variables and the specifics of the current tool to be queried.

It will be necessary to describe the tool in four different fashions:

- variables — a full set of variables is required to allow defining a shape and to determine the appropriate fashion in which to treat each tool at need

```

        tooltype = "mill"
        diameter = first

```

```

cornerradius = second
height = third
taperangle
length

```

- profile — the profile is a definition of the tool from the centerline to the outer edge which is used when necessary to `rotate_extrude()` the design
- outline — the outline is the entire definition of the tool shape which is used when `rotate_extrude`ing an arc (which will also require a 3D version of the rotated tool profile at each end)
- shape — originally the program used the tool shape and `hull()`ed it from beginning to end of a movement — having the shape pre-made allows it to be `union()`ed at need.

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).

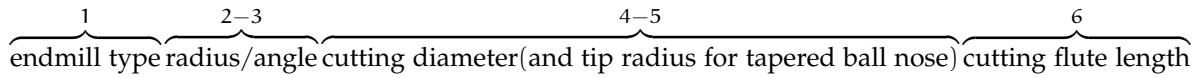
Note that it will also be necessary to write out a tool description compatible with the program CutViewer as a G-code comment so that it may be used as a 3D previewer for the G-code for tool changes in G-code. Several forms are available as described below.

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/completeness against simplicity and usability. The software application Carbide Create (as released by an employer of the main author) has a limit of six digits, which seems a reasonable length from a complexity/simplicity standpoint, but also potentially reasonably expressive.

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



- 1st digit: endmill type:
 - 0 - manufacturer number
 - 1 - square (incl. "O"-flute)
 - 2 - ball
 - 3 - V
 - 4 - bowl
 - 5 - tapered ball
 - 6 - roundover
 - 7 - thread-cutting
 - 8 - dovetail
 - 9 - other (e.g., user-defined, or unsupported tools, keyhole, lollipop, &c.)
- 2nd and 3rd digits shape radius (ball/roundover) or angle (V), 2nd and 3rd digit together 10–99 indicate measurement in tenth of a millimeter. 2nd digit:
 - 0 - Imperial (00 indicates n/a or square)
 - any other value for both the 2nd and 3rd digits together indicate a metric measurement or an angle in degrees
- 3rd digit (if 2nd is 0 indicating Imperial)
 - 1 - $1/32^{\text{nd}}$
 - 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 #274 and #326 (Amana 46200-K))
#201 == 100047 (also #251 and #322 (Amana 46202-K))
#205 == 100048 (also #324)
#211 == 100058 (also #213 and #214)
#213 == 100058 (Rougher, also #213 and #214)
#214 == 100058 (Compression, also #213 and #214)
#251 == 100047 (also #201, #278, and #322 (Amana 46202-K))
#274 == 100036 (also #102 and #326 (Amana 46200-K))
#278 == 100047
#282 == 100204
#322 == 100047 (also #201 and #251)
#324 == 100048 (also #205 (Amana 46170-K))
#326 == 100036 (also #102 and #274)
```

- Ball

```
#121 == 201012
#111 == 202024
#101 == 203036
#202 == 204047
#212 == 205058
#325 == 204048 (Amana 46376-K)
```

- V

```
#301 == 390074
#302 == 360071
#311 == 390121
#312 == 360121
#327 == 360098 (Amana RC-1148)
```

- 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:

- V

```
#390 == 390032
```

- 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
```

- Bowl bit

```
45981
45982
1370
1372
```

Notable limitations:

- No way to indicate number of flutes or flute geometry (an initial version used a first digit of 0 for O-flute, but they are now indicated by 1 (square))
- 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 typically 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 0... or 9#####)
- coatings are not considered, geometry/size only

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.*, given a hypothetical command/instruction:

```
self.currenttoolshape = self.toolshapes("square", 6.35, 19.05)
```

it could be viewed as representing any of three different tools (Carbide 3D #201 (upcut), #251 (downcut), and #322 (Amana 46202-K downcut)), it is worth noting that #205E is differentiated due to its longer flute length as-is #324 (Amana 46170-K compression), though the fact of its compression cutting geometry is not recorded. 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 continue to be used to identify tooling — it may be that in the future the generated number will be used internally, but the saved manufacturer number will be exported to the G-code file unless something changes.

276 gcopy	def currenttoolnumber(self):
277 gcopy	 return(self.currenttoolnum)

toolchange The toolchange command will need to set several variables.

Mandatory variables include:

- endmilltype
 - O-flute (this may be deprecated and removed)
 - square
 - ball
 - V
 - keyhole
 - dovetail
 - roundover
 - tapered ball
- diameter
- flute length

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.

Setting the shaft dimensions would be helpful, and might be used to facilitate keyhole cutters and similar tooling which calculates undercuts.

tool number **3.4.1.1 toolchange** This command accepts a tool number and assigns its characteristics as parameters. 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 previewing 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 the variable `misc` (as opposed to the list of explicitly supported variables/shapes/modules).

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, collecting all such in a list which is then parsed when the 3D model is generated.

There are two separate commands for handling a tool being changed, the first sets the parameters which describe the tool and may be used to effect the change of a tool either in a G-code file `settoolparameters` or when making a 3D file, `settoolparameters` (not currently used as noted below) and a second `toolchange` version which processes a toolchange when presented with a tool number, `toolchange` (it may be that the latter will be set up to call the former).

3.4.1.1.1 settoolparameters Not currently used, this command is intended for a future state/need where tools are defined in a vendor-neutral fashion.

```

279 gcpy      def settoolparameters(self, tooltype, first, second, third,
280 gcpy          fourth, length = 0):
281 gcpy          if tooltype == "mill":
282 gcpy              diameter = first
283 gcpy              cornerradius = second
284 gcpy              height = third
285 gcpy              taperangle = fourth
286 gcpy      #M6T122 (TOOL/MILL,0.80, 0.00, 1.59, 0.00)
287 gcpy      #M6T112 (TOOL/MILL,1.59, 0.00, 6.35, 0.00)
288 gcpy      #M6T102 (TOOL/MILL,3.17, 0.00, 12.70, 0.00)
289 gcpy      #M6T201 (TOOL/MILL,6.35, 0.00, 19.05, 0.00)
290 gcpy      #M6T205 (TOOL/MILL,6.35, 0.00, 25.40, 0.00)
291 gcpy      #M6T251 (TOOL/MILL,6.35, 0.00, 19.05, 0.00)
292 gcpy      #M6T322 (TOOL/MILL,6.35, 0.00, 19.05, 0.00)
293 gcpy      #M6T324 (TOOL/MILL,6.35, 0.00, 22.22, 0.00)
294 gcpy      #M6T326 (TOOL/MILL,3.17, 0.00, 12.70, 0.00)
295 gcpy      #M6T602 (TOOL/MILL,25.40, 0.00, 9.91, 0.00)
296 gcpy      #M6T603 (TOOL/MILL,25.40, 0.00, 9.91, 0.00)
297 gcpy      #M6T274 (TOOL/MILL,3.17, 0.00, 12.70, 0.00)
298 gcpy      #M6T278 (TOOL/MILL,6.35, 0.00, 19.05, 0.00)
299 gcpy      #M6T282 (TOOL/MILL,2.00, 0.00, 6.35, 0.00)
300 gcpy          self.endmilltype = "square"
301 gcpy          self.diameter = diameter
302 gcpy          self.flute = height
303 gcpy          self.shaftdiameter = diameter
304 gcpy          self.shaftheight = height
305 gcpy          self.shaftlength = height
306 gcpy #
307 gcpy          elif cornerradius > 0 and taperangle == 0:
308 gcpy      #M6T121 (TOOL/MILL,0.80, 0.40, 1.59, 0.00)
309 gcpy      #M6T111 (TOOL/MILL,1.59, 0.79, 6.35, 0.00)
310 gcpy      #M6T101 (TOOL/MILL,3.17, 1.59, 12.70, 0.00)
311 gcpy      #M6T202 (TOOL/MILL,6.35, 3.17, 19.05, 0.00)
312 gcpy      #M6T325 (TOOL/MILL,6.35, 3.17, 25.40, 0.00)
313 gcpy          self.endmilltype = "ball"
314 gcpy          self.diameter = diameter
315 gcpy          self.flute = height
316 gcpy          self.shaftdiameter = diameter
317 gcpy          self.shaftheight = height
318 gcpy          self.shaftlength = height
319 gcpy #
320 gcpy          elif taperangle > 0:
321 gcpy      #M6T301 (TOOL/MILL,0.10, 0.05, 6.35, 45.00)
322 gcpy      #M6T302 (TOOL/MILL,0.10, 0.05, 6.35, 30.00)
323 gcpy      #M6T327 (TOOL/MILL,0.10, 0.05, 23.39, 30.00)
324 gcpy          self.endmilltype = "V"
325 gcpy          self.diameter = Tan(taperangle / 2) * height
326 gcpy          self.flute = height
327 gcpy          self.angle = taperangle
328 gcpy          self.shaftdiameter = Tan(taperangle / 2) * height
329 gcpy          self.shaftheight = height
330 gcpy          self.shaftlength = height
331 gcpy #
332 gcpy          elif tooltype == "chamfer":
333 gcpy              tipdiameter = first
334 gcpy              radius = second
335 gcpy              height = third
336 gcpy              taperangle = fourth

```

toolchange 3.4.1.1.2 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 toolmovement for actually modeling the shapes and the path and the resultant material removal.

```

338 gcpy      def toolchange(self, tool_number, speed = 10000):
339 gcpy          self.currenttoolnum = tool_number
340 gcpy
341 gcpy          if (self.generategcode == True):
342 gcpy              self.writegc("(Toolpath)")
343 gcpy              self.writegc("M05")

```

3.4.1.1.3 Square (including O-flute)

The simplest sort of tool, they are defined as a cylinder.

```

345 gcpy           if (tool_number == 102) or (tool_number == 100036): #
346 gcpy             102/326 == 100036
347 gcpy             self.writegc("(TOOL/MILL, 3.175, 0.00, 0.00, 0.00)")
348 gcpy             self.endmilltype = "square"
349 gcpy             self.diameter = 3.175
350 gcpy             self.flute = 12.7
351 gcpy             self.shaftdiameter = 3.175
352 gcpy             self.shaftheight = 12.7
353 gcpy             self.shaftlength = 19.5

```

The outline definitions for linear/rotate extrude are the same for this tool as in the default tool definition in `__init__`, but the commands `definesquaretool` and `defineshaft` are used:

```

353 gcpy           self.definesquaretool(self.diameter, self.shaftheight,
354 gcpy             self.shaftlength)
355 gcpy             self.defineshaft(self.diameter, self.shaftdiameter,
356 gcpy               self.flute, 0, self.shaftlength)
357 gcpy             self.toolnumber = 10003
358 gcpy             elif (tool_number == 201) or (tool_number == 100047): #
359 gcpy               201/251/322 (Amana 46202-K) == 100047
360 gcpy               self.writegc("(TOOL/MILL, 6.35, 0.00, 0.00, 0.00)")
361 gcpy               self.endmilltype = "square"
362 gcpy               self.diameter = 6.35
363 gcpy               self.flute = 19.05
364 gcpy               self.shaftdiameter = 6.35
365 gcpy               self.shaftheight = 19.05
366 gcpy               self.shaftlength = 20.0
367 gcpy               self.definesquaretool(self.diameter, self.shaftheight,
368 gcpy                 self.shaftlength)
369 gcpy               self.defineshaft(self.diameter, self.shaftdiameter,
370 gcpy                 self.flute, 0, self.shaftlength)
371 gcpy               self.toolnumber = "100047"
372 gcpy               elif (tool_number == 112) or (tool_number == 100024): #112
373 gcpy                 == 100024
374 gcpy                 self.writegc("(TOOL/MILL, 1.5875, 0.00, 0.00, 0.00)")
375 gcpy                 self.endmilltype = "square"
376 gcpy                 self.diameter = 1.5875
377 gcpy                 self.flute = 6.35
378 gcpy                 self.shaftdiameter = 3.175
379 gcpy                 self.shaftheight = 6.35
380 gcpy                 self.shaftlength = 12.0
381 gcpy                 self.definesquaretool(self.diameter, self.shaftheight,
382 gcpy                   self.shaftlength, (self.shaftdiameter - self.
383 gcpy                     diameter)/2)
384 gcpy                 self.defineshaft(self.diameter, self.shaftdiameter,
385 gcpy                   self.flute, 0, self.shaftlength)
386 gcpy                   self.toolnumber = "100024"
387 gcpy                   elif (tool_number == 122) or (tool_number == 100012): #122
388 gcpy                     == 100012
389 gcpy                     self.writegc("(TOOL/MILL, 0.79375, 0.00, 0.00, 0.00)")
390 gcpy                     self.endmilltype = "square"
391 gcpy                     self.diameter = 0.79375
392 gcpy                     self.flute = 1.5875
393 gcpy                     self.shaftdiameter = 3.175
394 gcpy                     self.shaftheight = 1.5875
395 gcpy                     self.shaftlength = 12.0
396 gcpy                     self.definesquaretool(self.diameter, self.shaftheight,
397 gcpy                       self.shaftlength, (self.shaftdiameter - self.
398 gcpy                         diameter)/2)
399 gcpy                     self.defineshaft(self.diameter, self.shaftdiameter,
400 gcpy                       self.flute, 0, self.shaftlength)
401 gcpy                       self.toolnumber = "100012"
402 gcpy                       elif (tool_number == 324): #324 (Amana 46170-K) == 100048
403 gcpy                         self.writegc("(TOOL/MILL, 6.35, 0.00, 0.00, 0.00)")
404 gcpy                         self.endmilltype = "square"
405 gcpy                         self.diameter = 6.35
406 gcpy                         self.flute = 22.225
407 gcpy                         self.shaftdiameter = 6.35
408 gcpy                         self.shaftheight = 22.225
409 gcpy                         self.shaftlength = 20.0
410 gcpy                         self.definesquaretool(self.diameter, self.shaftheight,
411 gcpy                           self.shaftlength)

```

```

398 gcpy           self.defineshaft(self.diameter, self.shaftdiameter,
399 gcpy             self.flute, 0, self.shaftlength)
400 gcpy             self.toolnumber = "100048"
elif (tool_number == 205) or (tool_number == 100048): #205
    == 100048
401 gcpy             self.writegc("(TOOL/MILL, 6.35, 0.00, 0.00, 0.00)")
402 gcpy             self.endmilltype = "square"
403 gcpy             self.diameter = 6.35
404 gcpy             self.flute = 25.4
405 gcpy             self.shaftdiameter = 6.35
406 gcpy             self.shaftheight = 25.4
407 gcpy             self.shaftlength = 20.0
408 gcpy             self.definesquaretool(self.diameter, self.shaftheight,
409 gcpy               self.shaftlength)
410 gcpy             self.defineshaft(self.diameter, self.shaftdiameter,
411 gcpy               self.flute, 0, self.shaftlength)
412 gcpy             self.toolnumber = "100048"
elif (tool_number == 211) or (tool_number == 213) or
    (tool_number == 214): #211/213/214 == 100058
413 gcpy             self.writegc("(TOOL/MILL, 8.00, 0.00, 0.00, 0.00)")
414 gcpy             self.endmilltype = "square"
415 gcpy             self.diameter = 8.00
416 gcpy             self.flute = 26.0
417 gcpy             self.shaftdiameter = 8.00
418 gcpy             self.shaftheight = 26.0
419 gcpy             self.shaftlength = 49.0
420 gcpy             self.definesquaretool(self.diameter, self.shaftheight,
421 gcpy               self.shaftlength)
422 gcpy             self.defineshaft(self.diameter, self.shaftdiameter,
423 gcpy               self.flute, 0, self.shaftlength)
424 gcpy             self.toolnumber = "100058"
425 gcpy #

```

The former distinction betwixt Square and O-flute tooling has been removed from the current version.

```

423 gcpy           elif (tool_number == 282) or (tool_number == 100204): #282
    == 000204
424 gcpy             self.writegc("(TOOL/MILL, 2.0, 0.00, 0.00, 0.00)")
425 gcpy             self.endmilltype = "O-flute"
426 gcpy             self.diameter = 2.0
427 gcpy             self.flute = 6.35
428 gcpy             self.shaftdiameter = 6.35
429 gcpy             self.shaftheight = 6.35
430 gcpy             self.shaftlength = 12.0
431 gcpy             self.definesquaretool(self.diameter, self.shaftheight,
432 gcpy               self.shaftlength, (self.shaftdiameter - self.
433 gcpy                 diameter)/2)
434 gcpy             self.defineshaft(self.diameter, self.shaftdiameter,
435 gcpy               self.flute, 0, self.shaftlength)
436 gcpy             self.toolnumber = "100204"
elif (tool_number == 274) or (tool_number == 100036): #274
    == 000036
437 gcpy             self.writegc("(TOOL/MILL, 3.175, 0.00, 0.00, 0.00)")
438 gcpy             self.endmilltype = "O-flute"
439 gcpy             self.diameter = 3.175
440 gcpy             self.flute = 12.7
441 gcpy             self.shaftdiameter = 3.175
442 gcpy             self.shaftheight = 12.7
443 gcpy             self.shaftlength = 20.0
444 gcpy             self.definesquaretool(self.diameter, self.shaftheight,
445 gcpy               self.shaftlength)
446 gcpy             self.defineshaft(self.diameter, self.shaftdiameter,
447 gcpy               self.flute, 0, self.shaftlength)
448 gcpy             self.toolnumber = "100036"
elif (tool_number == 278) or (tool_number == 100047): #278
    == 000047
449 gcpy             self.writegc("(TOOL/MILL, 6.35, 0.00, 0.00, 0.00)")
450 gcpy             self.endmilltype = "O-flute"
451 gcpy             self.diameter = 6.35
452 gcpy             self.flute = 19.05
453 gcpy             self.shaftdiameter = 3.175
454 gcpy             self.shaftheight = 19.05
455 gcpy             self.shaftlength = 20.0
456 gcpy             self.definesquaretool(self.diameter, self.shaftheight,
457 gcpy               self.shaftlength)
458 gcpy             self.defineshaft(self.diameter, self.shaftdiameter,
459 gcpy               self.flute, 0, self.shaftlength)

```

```
455 gcpy           self.toolnumber = "100047"
456 gcpy #
```

3.4.1.1.4 Ball-nose The elifs continue with ball-nose (note that tapered-ball tooling is covered separately below) 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.

```
457 gcpy           elif (tool_number == 202) or (tool_number == 204047): #202
458 gcpy           == 204047
459 gcpy           self.writegc("(TOOL/MILL, 6.35, 3.175, 0.00, 0.00)")
460 gcpy           self.endmilltype = "ball"
461 gcpy           self.diameter = 6.35
462 gcpy           self.flute = 19.05
463 gcpy           self.shaftdiameter = 6.35
464 gcpy           self.shaftheight = 19.05
465 gcpy           self.shaftlength = 20.0
466 gcpy           self.defineballnose(tool(self.diameter, self.flute, self
467 gcpy           .shaftlength)
468 gcpy           self.defineshaft(self.diameter, self.shaftdiameter,
469 gcpy           self.flute, 0, self.shaftlength)
470 gcpy           self.toolnumber = "204047"
471 gcpy           elif (tool_number == 101) or (tool_number == 203036): #101
472 gcpy           == 203036
473 gcpy           self.writegc("(TOOL/MILL, 3.175, 1.5875, 0.00, 0.00)")
474 gcpy           self.endmilltype = "ball"
475 gcpy           self.diameter = 3.175
476 gcpy           self.flute = 12.7
477 gcpy           self.shaftdiameter = 3.175
478 gcpy           self.shaftheight = 12.7
479 gcpy           self.shaftlength = 20.0
480 gcpy           self.defineballnose(tool(self.diameter, self.flute, self
481 gcpy           .shaftlength)
482 gcpy           self.defineshaft(self.diameter, self.shaftdiameter,
483 gcpy           self.flute, 0, self.shaftlength)
484 gcpy           self.toolnumber = "203036"
485 gcpy           elif (tool_number == 111) or (tool_number == 202024): #111
486 gcpy           == 202024
487 gcpy           self.writegc("(TOOL/MILL, 1.5875, 0.79375, 0.00, 0.00)"
488 gcpy           )
489 gcpy           self.endmilltype = "ball"
490 gcpy           self.diameter = 1.5875
491 gcpy           self.flute = 6.35
492 gcpy           self.shaftdiameter = 3.175
493 gcpy           self.shaftheight = 6.35
494 gcpy           self.shaftlength = 20.0
495 gcpy           self.defineballnose(tool(self.diameter, self.flute, self
496 gcpy           .shaftlength, (self.shaftdiameter - self.diameter)
497 gcpy           /2)
498 gcpy           self.defineshaft(self.diameter, self.shaftdiameter,
499 gcpy           self.flute, 0, self.shaftlength)
500 gcpy           self.toolnumber = "202024"
501 gcpy           elif (tool_number == 121) or (tool_number == 201012): #121
502 gcpy           == 201012
503 gcpy           self.writegc("(TOOL/MILL, 3.175, 0.79375, 0.00, 0.00)")
504 gcpy           self.endmilltype = "ball"
505 gcpy           self.diameter = 0.79375
506 gcpy           self.flute = 1.5875
507 gcpy           self.shaftdiameter = 3.175
508 gcpy           self.shaftheight = 1.5875
509 gcpy           self.shaftlength = 20.0
510 gcpy           self.defineballnose(tool(self.diameter, self.flute, self
511 gcpy           .shaftlength, (self.shaftdiameter - self.diameter)
512 gcpy           /2)
513 gcpy           self.defineshaft(self.diameter, self.shaftdiameter,
514 gcpy           self.flute, 0, self.shaftlength)
515 gcpy           self.toolnumber = "201012"
516 gcpy           elif (tool_number == 325) or (tool_number == 204048): #325
517 gcpy           (Amana 46376-K) == 204048
518 gcpy           self.writegc("(TOOL/MILL, 6.35, 3.175, 0.00, 0.00)")
519 gcpy           self.endmilltype = "ball"
520 gcpy           self.diameter = 6.35
521 gcpy           self.flute = 25.4
522 gcpy           self.shaftdiameter = 6.35
523 gcpy           self.shaftheight = 25.4
```

```

508 gcpy           self.shaftlength = 20.0
509 gcpy           self.defineballnosetool(self.diameter, self.flute, self.
510 gcpy             .shaftlength, (self.shaftdiameter - self.diameter)
511 gcpy               /2)
512 gcpy             self.defineshaft(self.diameter, self.shaftdiameter,
513 gcpy               self.flute, 0, self.shaftlength)
514 gcpy             self.toolnumber = "204048"
515 gcpy           elif (tool_number == 212) or (tool_number == 205058): #212
516 gcpy             == 205058
517 gcpy             self.writegc("(TOOL/MILL, 8.00, 4.00, 0.00, 0.00)")
518 gcpy             self.endmilltype = "ball"
519 gcpy             self.diameter = 8.00
520 gcpy             self.flute = 26.0
521 gcpy             self.shaftdiameter = 8.00
522 gcpy             self.shaftheight = 26.0
523 gcpy             self.shaftlength = 49.0
524 gcpy             self.defineballnosetool(self.diameter, self.flute, self.
525 gcpy               .shaftlength, (self.shaftdiameter - self.diameter)
526 gcpy               /2)
527 gcpy             self.defineshaft(self.diameter, self.shaftdiameter,
528 gcpy               self.flute, 0, self.shaftlength)
529 gcpy             self.toolnumber = "204048"
530 gcpy           #

```

3.4.1.1.5 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).

```

513 gcpy           elif (tool_number == 301) or (tool_number == 390074): #301
514 gcpy             == 390074
515 gcpy             self.writegc("(TOOL/MILL, 0.10, 0.05, 6.35, 45.00)")
516 gcpy             self.endmilltype = "V"
517 gcpy             self.diameter = 12.7
518 gcpy             self.flute = 6.35
519 gcpy             self.angle = 90
520 gcpy             self.shaftdiameter = 6.35
521 gcpy             self.shaftheight = 6.35
522 gcpy             self.shaftlength = 20.0
523 gcpy             self.defineVtool(self.diameter, self.flute, self.
524 gcpy               shaftlength, self.shaftdiameter)
525 gcpy             self.toolnumber = "390074"
526 gcpy           elif (tool_number == 302) or (tool_number == 360071): #302
527 gcpy             == 360071
528 gcpy             self.writegc("(TOOL/MILL, 0.10, 0.05, 6.35, 30.00)")
529 gcpy             self.endmilltype = "V"
530 gcpy             self.diameter = 12.7
531 gcpy             self.flute = 11.067
532 gcpy             self.angle = 60
533 gcpy             self.shaftdiameter = 6.35
534 gcpy             self.shaftheight = 11.067
535 gcpy             self.shaftlength = 20.0
536 gcpy             self.defineVtool(self.diameter, self.flute, self.
537 gcpy               shaftlength, self.shaftdiameter)
538 gcpy             self.toolnumber = "360071"
539 gcpy           elif (tool_number == 311) or (tool_number == 390121): #311
540 gcpy             == 390121
541 gcpy             self.writegc("(TOOL/MILL, 0.10, 0.05, 6.00, 45.00)")
542 gcpy             self.endmilltype = "V"
543 gcpy             self.diameter = 12.0
544 gcpy             self.flute = 6.00
545 gcpy             self.angle = 90
546 gcpy             self.shaftdiameter = 8.00
547 gcpy             self.shaftheight = 8.00
548 gcpy             self.shaftlength = 20.0
549 gcpy             self.defineVtool(self.diameter, self.flute, self.
550 gcpy               shaftlength, self.shaftdiameter)
551 gcpy             self.toolnumber = "390121"
552 gcpy           elif (tool_number == 312) or (tool_number == 360121): #312
553 gcpy             == 360121
554 gcpy             self.writegc("(TOOL/MILL, 0.10, 0.05, 6.00, 30.00)")
555 gcpy             self.endmilltype = "V"
556 gcpy             self.diameter = 12.0
557 gcpy             self.flute = 10.39
558 gcpy             self.angle = 60
559 gcpy             self.shaftdiameter = 8.00
560 gcpy             self.shaftheight = 10.39

```

```

554 gcpy           self.shaftlength = 20.0
555 gcpy           self.defineVtool(self.diameter, self.flute, self.
556 gcpy             shaftlength, self.shaftdiameter)
557 gcpy           self.toolnumber = "360121"
558 gcpy   elif (tool_number == 327) or (tool_number == 360098): #327
559 gcpy     (Amana RC-1148) == 360098
560 gcpy     self.writegc("(TOOL/MILL, 0.03, 0.00, 13.4874, 30.00)")
561 gcpy     self.endmilltype = "V"
562 gcpy     self.diameter = 25.4
563 gcpy     self.flute = 22.134
564 gcpy     self.angle = 60
565 gcpy     self.shaftdiameter = 6.35
566 gcpy     self.shaftheight = 22.134
567 gcpy     self.shaftlength = 20.0
568 gcpy     self.defineVtool(self.diameter, self.flute, self.
569 gcpy       shaftlength, self.shaftdiameter)
570 gcpy     self.toolnumber = "360098"
571 gcpy   elif (tool_number == 323) or (tool_number == 330041): #323
572 gcpy     == 330041 30 degree V Amana, 45771-K
573 gcpy     self.writegc("(TOOL/MILL, 0.10, 0.05, 11.18, 15.00)")
574 gcpy     self.endmilltype = "V"
575 gcpy     self.diameter = 6.35
576 gcpy     self.flute = 11.849
577 gcpy     self.angle = 30
578 gcpy     self.shaftdiameter = 6.35
579 gcpy     self.shaftheight = 11.849
580 gcpy     self.shaftlength = 20.0
581 gcpy     self.defineVtool(self.diameter, self.flute, self.
582 gcpy       shaftlength, self.shaftdiameter)
583 gcpy     self.toolnumber = "330041"
584 gcpy   elif (tool_number == 390) or (tool_number == 390032): #390
585 gcpy     == 390032
586 gcpy     self.writegc("(TOOL/MILL, 0.03, 0.00, 1.5875, 45.00)")
587 gcpy     self.endmilltype = "V"
588 gcpy     self.diameter = 3.175
589 gcpy     self.flute = 1.5875
590 gcpy     self.angle = 90
591 gcpy     self.shaftdiameter = 3.175
592 gcpy     self.shaftheight = 1.5875
593 gcpy     self.shaftlength = 20.0
594 gcpy     self.defineVtool(self.diameter, self.flute, self.
595 gcpy       shaftlength, self.shaftdiameter)
596 gcpy     self.toolnumber = "390032"
597 gcpy #

```

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

```

591 gcpy   elif (tool_number == 374) or (tool_number == 906043): #374
592 gcpy     == 906043
593 gcpy     self.writegc("(TOOL/MILL, 9.53, 0.00, 3.17, 0.00)")
594 gcpy     self.endmilltype = "keyhole"
595 gcpy     self.diameter = 9.525
596 gcpy     self.flute = 3.175
597 gcpy     self.radius = 6.35
598 gcpy     self.shaftdiameter = 6.35
599 gcpy     self.shaftheight = 3.175
600 gcpy     self.shaftlength = 20.0
601 gcpy     self.defineKeyholetool(self.diameter, self.flute, self.
602 gcpy       shaftdiameter, self.shaftheight, self.shaftdiameter,
603 gcpy         self.shaftlength)
604 gcpy     self.toolnumber = "906043"
605 gcpy   elif (tool_number == 375) or (tool_number == 906053): #375
606 gcpy     == 906053
607 gcpy     self.writegc("(TOOL/MILL, 9.53, 0.00, 3.17, 0.00)")
608 gcpy     self.endmilltype = "keyhole"
609 gcpy     self.diameter = 9.525
610 gcpy     self.flute = 3.175
611 gcpy     self.radius = 8
612 gcpy     self.shaftdiameter = 6.35
613 gcpy     self.shaftheight = 3.175
614 gcpy     self.shaftlength = 20.0
615 gcpy     self.defineKeyholetool(self.diameter, self.flute, self.
616 gcpy       shaftdiameter, self.shaftheight, self.shaftdiameter,
617 gcpy         self.shaftlength)
618 gcpy     self.toolnumber = "906053"
619 gcpy   elif (tool_number == 376) or (tool_number == 907040): #376
620 gcpy     == 907040

```

```

614 gcpy           self.writegc("(TOOL/MILL,12.7,0.00,4.77,0.00)")
615 gcpy           self.endmilltype = "keyhole"
616 gcpy           self.diameter = 12.7
617 gcpy           self.flute = 4.7625
618 gcpy           self.radius = 6.35
619 gcpy           self.shaftdiameter = 6.35
620 gcpy           self.shaftheight = 4.7625
621 gcpy           self.shaftlength = 20.0
622 gcpy           self.defineKeyholetool(self.diameter, self.flute, self.
623 gcpy               shaftdiameter, self.shaftheight, self.shaftdiameter,
624 gcpy                   self.shaftlength)
625 gcpy           self.toolnumber = "907040"
626 gcpy           elif (tool_number == 378) or (tool_number == 907050): #378
627 gcpy               == 907050
628 gcpy           self.writegc("(TOOL/MILL,12.7,0.00,4.77,0.00)")
629 gcpy           self.endmilltype = "keyhole"
630 gcpy           self.diameter = 12.7
631 gcpy           self.flute = 4.7625
632 gcpy           self.radius = 8
633 gcpy           self.shaftdiameter = 6.35
634 gcpy           self.shaftheight = 4.7625
635 gcpy           self.shaftlength = 20.0
636 gcpy           self.defineKeyholetool(self.diameter, self.flute, self.
637 gcpy               shaftdiameter, self.shaftheight, self.shaftdiameter,
638 gcpy                   self.shaftlength)
639 gcpy           self.toolnumber = "907050"
640 gcpy           #

```

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

```

636 gcpy           elif (tool_number == 45981): #45981 == 445981
637 gcpy #Amana Carbide Tipped Bowl & Tray 1/8 Radius x 1/2 Dia x 1/2 x 1/4
638 gcpy           Inch Shank
639 gcpy           self.writegc("(TOOL/MILL,0.03,0.00,10.00,30.00)")
640 gcpy           self.writegc("(TOOL/MILL,15.875,6.35,19.05,0.00)")
641 gcpy           self.endmilltype = "bowl"
642 gcpy           self.diameter = 12.7
643 gcpy           self.flute = 12.7
644 gcpy           self.radius = 3.175
645 gcpy           self.shaftdiameter = 6.35
646 gcpy           self.shaftheight = 12.7
647 gcpy           self.shaftlength = 20.0
648 gcpy           self.definebowltool(self.diameter, self.flute, self.
649 gcpy               radius, self.shaftdiameter, self.shaftlength)
650 gcpy           self.toolnumber = "445981"
651 gcpy           elif (tool_number == 45982):#0.507/2, 4.509
652 gcpy           self.writegc("(TOOL/MILL,15.875,6.35,19.05,0.00)")
653 gcpy           self.endmilltype = "bowl"
654 gcpy           self.diameter = 19.05
655 gcpy           self.flute = 15.875
656 gcpy           self.radius = 6.35
657 gcpy           self.shaftdiameter = 6.35
658 gcpy           self.shaftheight = 15.875
659 gcpy           self.shaftlength = 20.0
660 gcpy           self.definebowltool(self.diameter, self.flute, self.
661 gcpy               radius, self.shaftdiameter, self.shaftlength)
662 gcpy           self.toolnumber = "445982"
663 gcpy           elif (tool_number == 1370): #1370 == 401370
664 gcpy #Whiteside Bowl & Tray Bit 1/4"SH, 1/8"R, 7/16"CD (5/16" cutting
665 gcpy           flute length)
666 gcpy           self.writegc("(TOOL/MILL,11.1125,8,3.175,0.00)")
667 gcpy           self.endmilltype = "bowl"
668 gcpy           self.diameter = 11.1125
669 gcpy           self.flute = 8
670 gcpy           self.radius = 3.175
671 gcpy           self.shaftdiameter = 6.35
672 gcpy           self.shaftheight = 8
673 gcpy           self.shaftlength = 20.0
674 gcpy           self.definebowltool(self.diameter, self.flute, self.
675 gcpy               radius, self.shaftdiameter, self.shaftlength)
676 gcpy           self.toolnumber = "401370"
677 gcpy           elif (tool_number == 1372): #1372/45982 == 401372
678 gcpy #Whiteside Bowl & Tray Bit 1/4"SH, 1/4"R, 3/4"CD (5/8" cutting
679 gcpy           flute length)
680 gcpy #Amana Carbide Tipped Bowl & Tray 1/4 Radius x 3/4 Dia x 5/8 x 1/4
681 gcpy           Inch Shank
682 gcpy           self.writegc("(TOOL/MILL,19.5,15.875,6.35,0.00)")

```

```

676 gcpy           self.endmilltype = "bowl"
677 gcpy           self.diameter = 19.5
678 gcpy           self.flute = 15.875
679 gcpy           self.radius = 6.35
680 gcpy           self.shaftdiameter = 6.35
681 gcpy           self.shaftheight = 15.875
682 gcpy           self.shaftlength = 20.0
683 gcpy           self.definebowltool(self.diameter, self.flute, self.
684 gcpy           radius, self.shaftdiameter, self.shaftlength)
685 gcpy #          self.toolnumber = "401372"

```

3.4.1.1.8 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 could be used. Similarly, the #501 and #502 PCB engravers from Carbide 3D are supported.

Outlines and profiles for these tools are stored in svg files:

```

501_outline.svg
501_profile.svg
501_shaft_outline.svg
501_shaft_profile.svg
502_outline.svg
502_profile.svg
502_shaft_outline.svg
502_shaft_profile.svg

```

which are then imported into the appropriate variables when a tool is loaded.

```

686 gcpy      elif (tool_number == 501) or (tool_number == 530131): #501
687 gcpy          == 530131
688 gcpy #         self.writegc("(TOOL/MILL,0.03,0.00,10.00,30.00)")
689 gcpy          self.currenttoolshape = self.toolshapes("tapered ball
690 gcpy          ", 3.175, 5.561, 30, 0.254)
691 gcpy          self.tooloutline = osimport("501_outline.svg")
692 gcpy          self.toolprofile = osimport("501_profile.svg")
693 gcpy          self.endmilltype = "tapered_ball"
694 gcpy          self.diameter = 3.175
695 gcpy          self.flute = 5.561
696 gcpy          self.angle = 30
697 gcpy          self.tip = 0.254
698 gcpy          self.shaftdiameter = 3.175
699 gcpy          self.shaftheight = 5.561
700 gcpy          self.shaftlength = 10.0
701 gcpy          self.toolnumber = "530131"
702 gcpy #        elif (tool_number == 502) or (tool_number == 540131): #502
703 gcpy          == 540131
704 gcpy          self.writegc("(TOOL/MILL,0.03,0.00,10.00,20.00)")
705 gcpy #        self.currenttoolshape = self.toolshapes("tapered ball
706 gcpy          ", 3.175, 4.117, 40, 0.254)
707 gcpy          self.endmilltype = "tapered_ball"
708 gcpy          self.diameter = 3.175
709 gcpy          self.flute = 4.117
710 gcpy          self.angle = 40
711 gcpy          self.tip = 0.254
712 gcpy #        self.shaftdiameter = 3.175
713 gcpy #        self.shaftheight = 4.117
714 gcpy #        self.shaftlength = 10.0
715 gcpy #        self.toolnumber = "540131"
716 gcpy #        elif (tool_number == 204):#
717 gcpy #          self.writegc("(")
718 gcpy #          self.currenttoolshape = self.tapered_ball(1.5875,
719 gcpy           6.35, 38.1, 3.6)
720 gcpy #        elif (tool_number == 304):#
721 gcpy #          self.writegc("(")
722 gcpy #          self.currenttoolshape = self.tapered_ball(3.175, 6.35,
723 gcpy           38.1, 2.4)
724 gcpy #

```

3.4.1.1.9 Roundover (cove tooling) Note that the parameters will need to incorporate the tip diameter into the overall diameter.

```

719 gcpy      elif (tool_number == 56125) or (tool_number == 603042):#
720 gcpy          0.508/2, 1.531 56125 == 603042

```

```

720 gcpy           self.writegc("(TOOL/CRMILL, 0.508, 6.35, 3.175, 7.9375,
721 gcpy           3.175)")
722 gcpy           self.endmilltype = "roundover"
723 gcpy           self.tipdiameter = 0.508
724 gcpy           self.diameter = 6.35 - self.tipdiameter
725 gcpy           self.flute = 8 - self.tipdiameter
726 gcpy           self.radius = 3.175 - self.tipdiameter/2
727 gcpy           self.shaftdiameter = 6.35
728 gcpy           self.shaftheight = 8
729 gcpy           self.shaftlength = 10.0
730 gcpy           self.defineRoundovertool(self.diameter, self.
731 gcpy           tipdiameter, self.flute, self.radius, self.
732 gcpy           shaftdiameter, self.shaftlength)
733 gcpy           self.toolnumber = "603042"
734 gcpy           elif (tool_number == 56142) or (tool_number == 602032):#
735 gcpy           0.508/2, 2.921 56142 == 602032
736 gcpy           self.writegc("(TOOL/CRMILL, 0.508, 3.571875, 1.5875, 5.55625, 1.5875)")
737 gcpy           self.endmilltype = "roundover"
738 gcpy           self.tip = 0.508
739 gcpy           self.diameter = 3.175 - self.tip
740 gcpy           self.flute = 4.7625 - self.tip
741 gcpy           self.radius = 1.5875 - self.tip/2
742 gcpy           self.shaftdiameter = 3.175
743 gcpy           self.shaftheight = 4.7625
744 gcpy           self.shaftlength = 10.0
745 gcpy           self.toolnumber = "602032"
746 gcpy #         elif (tool_number == 312):#1.524/2, 3.175
747 gcpy #         self.writegc("(TOOL/CRMILL, Diameter1, Diameter2,
748 gcpy #         Radius, Height, Length)")
749 gcpy #         elif (tool_number == 1568):#0.507/2, 4.509 1568 == 603032
750 gcpy ##FIX      self.currenttoolshape = self.toolshapes("roundover",
751 gcpy #         3.175, 6.35, 3.175, 0.396875)
752 gcpy #         self.endmilltype = "roundover"
753 gcpy ##https://www.amanatool.com/45982-carbide-tipped-bowl-tray-1-4-
754 gcpy #         radius-x-3-4-dia-x-5-8-x-1-4-inch-shank.html
755 gcpy #         elif (tool_number == 1570):#0.507/2, 4.509 1570 == 600002
756 gcpy ##         self.currenttoolshape = self.toolshapes("roundover",
757 gcpy #         4.7625, 9.525, 4.7625, 0.396875)
758 gcpy #         self.endmilltype = "roundover"
759 gcpy #         self.diameter = 4.7625
760 gcpy #         self.flute = 9.525
761 gcpy #         self.radius = 4.7625
762 gcpy #         self.tip = 0.396875
763 gcpy #         self.toolnumber = "600002"
764 gcpy ##FIX      elif (tool_number == 1572): #1572 = 604042
765 gcpy ##         self.writegc("(TOOL/CRMILL, 0.17018, 9.525,
766 gcpy #         4.7625, 12.7, 4.7625)")
767 gcpy #         self.currenttoolshape = self.toolshapes("roundover",
768 gcpy #         6.35, 12.7, 6.35, 0.396875)
769 gcpy #         self.endmilltype = "roundover"
770 gcpy #         self.diameter = 6.35
771 gcpy #         self.flute = 12.7
772 gcpy #         self.radius = 6.35
773 gcpy ##FIX      self.writegc("(TOOL/CRMILL, 0.17018, 9.525,
774 gcpy #         4.7625, 12.7, 4.7625)")
775 gcpy #         self.currenttoolshape = self.toolshapes("roundover",
776 gcpy #         9.525, 19.5, 9.515, 0.396875)
777 gcpy #         self.endmilltype = "roundover"
778 gcpy #         self.diameter = 9.525
779 gcpy #         self.flute = 19.5
780 gcpy #         self.radius = 9.515
781 gcpy #         self.tip = 0.396875
782 gcpy #         self.toolnumber = "600062"

```

3.4.1.1.10 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).

```

782 gcpy      elif (tool_number == 814) or (tool_number == 814071): #814
               == 814071
783 gcpy #Item 18J1607, 1/2" 14° Dovetail Bit, 8mm shank
784 gcpy      self.writegc("(TOOL/MILL,12.7,6.367,12.7,0.00)")
785 gcpy      # dt_bottomdiameter, dt_topdiameter, dt_height, dt_angle
               )
786 gcpy      # https://www.leevalley.com/en-us/shop/tools/power-tool-
               accessories/router-bits/30172-dovetail-bits?item=18J1607
787 gcpy #      self.currenttoolshape = self.toolshapes("dovetail",
               12.7, 12.7, 14)
788 gcpy      self.endmilltype = "dovetail"
789 gcpy      self.diameter = 12.7
790 gcpy      self.flute = 12.7
791 gcpy      self.angle = 14
792 gcpy      self.toolnumber = "814071"
793 gcpy #      elif (tool_number == 808079) or (tool_number == 808071): #
               45828 == 808071
794 gcpy      self.writegc("(TOOL/MILL,12.7,6.816,20.95,0.00)")
795 gcpy #      http://www.amanatool.com/45828-carbide-tipped-dovetail
               -8-deg-x-1-2-dia-x-825-x-1-4-inch-shank.html
796 gcpy #      self.currenttoolshape = self.toolshapes("dovetail",
               12.7, 20.955, 8)
797 gcpy      self.endmilltype = "dovetail"
798 gcpy      self.diameter = 12.7
799 gcpy      self.flute = 20.955
800 gcpy      self.angle = 8
801 gcpy      self.toolnumber = "808071"
802 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 gpcscad module toolchange(tool_number, speed){
49 gpcscad     gcp.toolchange(tool_number, speed);
50 gpcscad }

```

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.)

3.4.1.1.11 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.

One possible feature for the G-code for tool changes would be to have the various ratios available and then to apply the appropriate one. Directly applying them in the file generated by the user is sufficiently straight-forward that this expedient option seems a needless complexity unless a compelling reason comes up.

```
803 gcpy      self.writegc("M6T", str(tool_number))
804 gcpy #    if (self.endmilltype == "square"):
805 gcpy #        speed = speed *
806 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... Another possibility is describing tools in terms of outline which will allow using linear/rotate_extrude to be used which requires a description of the tools as profiles/outlines, but which matches the G0/G1 and G2/G3 G-code commands.

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.

3.5.1 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.

3.5.2 Generalized commands and cuts

The first consideration is a naming convention which will allow a generalized set of associated commands to be defined.

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

3.5.3 Movement and color

`toolmovement` The first command which must be defined is `toolmovement` which is used as the core of the other `shaftmovement` 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.

```

808 gcpy      def setcolor(self,
809 gcpy          cutcolor = "green",
810 gcpy          rapidcolor = "orange",
811 gcpy          shaftcolor = "red"):
812 gcpy          self.cutcolor = cutcolor
813 gcpy          self.rapidcolor = rapidcolor
814 gcpy          self.shaftcolor = shaftcolor

```

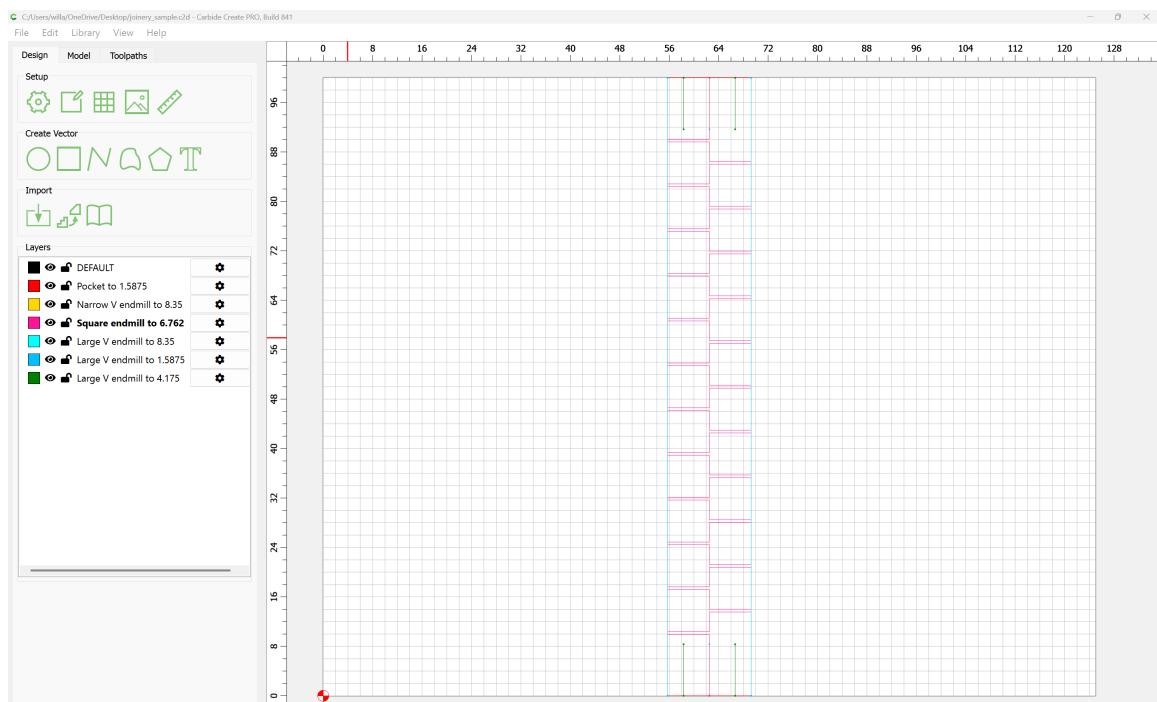
```

52 gpcscad module setcolor(cutcolor, rapidcolor, shaftcolor){
53 gpcscad     gcp.setcolor(cutcolor, rapidcolor, shaftcolor);
54 gpcscad }

```

The possible colors for OpenSCAD 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 Carbide Create's colour-coding of layers adds another set and applying a Venn diagram and removing problematic extremes we arrive at the third column (Both) 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.

A further consideration is that colors *per se* are not a useful characteristic for the typical usage of cutting a single material (it may be that in the future, supporting this for 3D-printing will be an option). Instead, placing all geometry which is associated with a given tool at a specified depth on an appropriately named layer will facilitate associating said elements with a matching toolpath by using the Carbide Create features for assigning layers when importing a DXF and its ability to set a toolpath to be applied to the elements on a specified layer.



A naming convention of Toolpath type (since the tool associated tool is obvious) or Tool type and size, the word “ to ” and a specified depth matches how toolpaths are often named and has the potential to be generated automatically. It will be helpful if the colors used are coordinated between the two usages, but that is not strictly necessary (the assignment in Carbide Create will default to Black when a DXF is opened, or will remain as assigned if imported into a .c2d file with extant layers of the same name(s)) which if serving as a template will have appropriate toolpaths with matching names and settings.

Table 1: Colors in OpenSCAD and DXF with Carbide Create swatches

Web Colors (OpenSCAD)	DXF	Both	Carbide Create
Black	"Black" (0)	Black	
Red	"Red" (1)	Red	
Yellow	"Yellow" (2)	Yellow	
Green	"Green" (3)	Green	
Aqua	"Cyan" (4)	Aqua/Cyan	
Blue	"Blue" (5)	Blue	
Fuchsia	"Magenta" (6)	Fuchsia/Magenta	
Gray	"Dark Gray" (8)	(Dark) Gray	
Silver	"Light Gray" (9)	Silver/Light Gray	
Maroon			
Olive			
Lime			
Teal			
Navy			
Purple			
White (7) is omitted from the above list (note that the names are not case-sensitive)			

Most tools are easily implemented with concise 3D descriptions which may be connected with

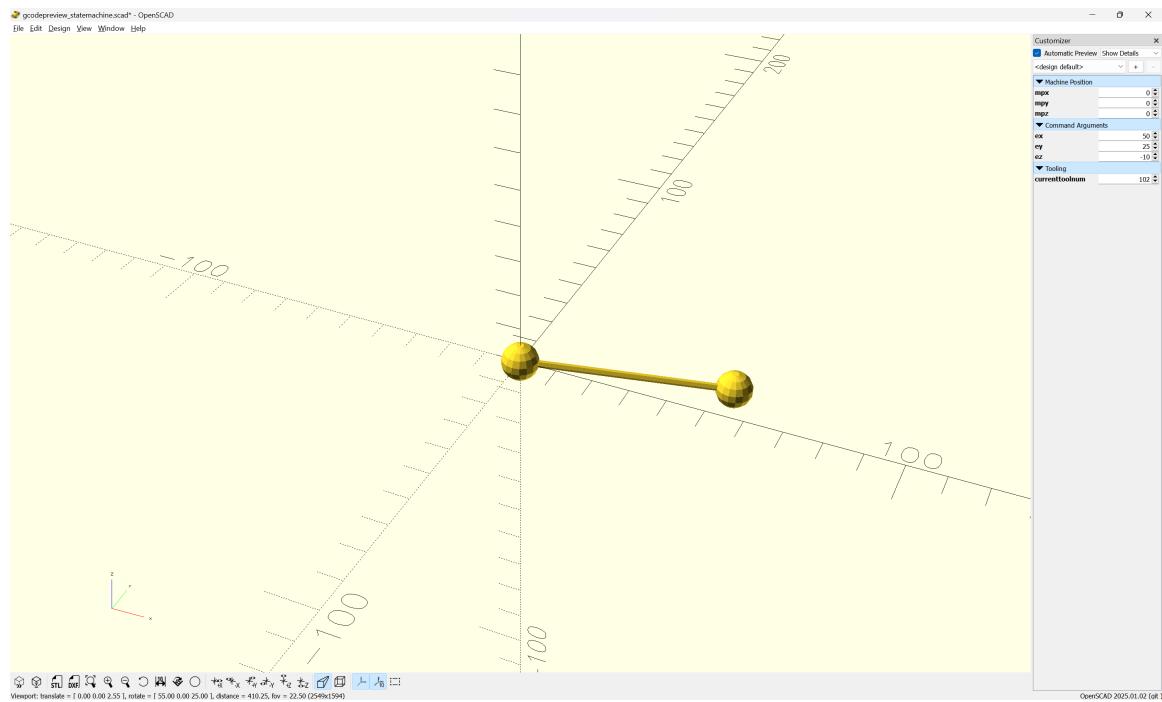
a simple hull operation. Note that extending the normal case to a pair of such operations, one for the shaft, the other for the cutting shape will markedly simplify the code, and will make it possible to color-code the shaft which may afford indication of instances of it rubbing against the stock.

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

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

When diagramming this, note that there are two possibilities — in the simplest, 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 variables will change or not change value depending on whether the current move is a compleat operation or an incremental portion of a move where the exposed variables will be updated at the end.

The code for showing this graphically 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):



```

816 gcpy      def toolmovement(self, bx, by, bz, ex, ey, ez, step = 0):
817 gcpy          tslist = []
818 gcpy          if step > 0:
819 gcpy              steps = step
820 gcpy          else:
821 gcpy              steps = self.steps
822 gcpy #

```

```

56 gpcscad module toolmovement(bx, by, bz, ex, ey, ez, step){
57 gpcscad     gcp.toolmovement(bx, by, bz, ex, ey, ez, step);
58 gpcscad }

```

For the implementation of tool movement, each different sort of tool will need to be handled separately in an `if | then` structure:

endmill square

3.5.3.1.1 Square (including O-flute) The endmill square is a simple cylinder:

```

823 gcpy      if self.endmilltype == "square":
824 gcpy          ts = cylinder(r1=(self.diameter / 2), r2=(self.diameter
825 gcpy              / 2), h=self.flute, center = False)
826 gcpy          tslist.append(hull(ts.translate([bx, by, bz]), ts.
827 gcpy              translate([ex, ey, ez])))
828 gcpy #
829 gcpy #      if self.endmilltype == "O-flute":
829 gcpy #          ts = cylinder(r1=(self.diameter / 2), r2=(self.
829 gcpy #              diameter / 2), h=self.flute, center = False)

```

```

830 gcpy #           tslist.append(hull(ts.translate([bx, by, bz]), ts.
     translate([ex, ey, ez])))
831 gcpy #           return tslist
832 gcpy #

```

ballnose **3.5.3.1.2 Ball nose** The ballnose is modeled as a hemisphere joined with a cylinder:

```

833 gcpy if self.endmilltype == "ball":
834 gcpy     b = sphere(r=(self.diameter / 2))
835 gcpy     s = cylinder(r1=(self.diameter / 2), r2=(self.diameter
     / 2), h=self.flute, center=False)
836 gcpy     bs = union(b, s)
837 gcpy     bs = bs.translate([0, 0, (self.diameter / 2)])
838 gcpy     tslist.append(hull(bs.translate([bx, by, bz]), bs.
     translate([ex, ey, ez])))
839 gcpy     return tslist
840 gcpy #

```

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

```

841 gcpy if self.endmilltype == "bowl":
842 gcpy     inner = cylinder(r1 = self.diameter/2 - self.radius, r2
     = self.diameter/2 - self.radius, h = self.flute)
843 gcpy     outer = cylinder(r1 = self.diameter/2, r2 = self.
     diameter/2, h = self.flute - self.radius)
844 gcpy     outer = outer.translate([0,0, self.radius])
845 gcpy     slices = hull(outer, inner)
846 gcpy #     slices = cylinder(r1 = 0.0001, r2 = 0.0001, h = 0.0001, center
     =False)
847 gcpy     for i in range(1, 90 - self.steps, self.steps):
848 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)
849 gcpy     slices = hull(slices, slice.translate([0, 0, self.
     radius - self.radius * Cos(i+self.steps)]))
850 gcpy     tslist.append(hull(slices.translate([bx, by, bz]),
     slices.translate([ex, ey, ez])))
851 gcpy     return tslist
852 gcpy #

```

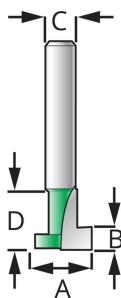
endmill v **3.5.3.1.4 V** The endmill v is modeled as a cylinder with a zero width base and a second cylinder for the shaft (note that Python's math defaults to radians, hence the need to convert from degrees if using it, but fortunately, trigonometric commands have been added to OpenPython-SCAD (Sin, Cos, Tan, Atan)):

```

853 gcpy if self.endmilltype == "V":
854 gcpy     v = cylinder(r1=0, r2=(self.diameter / 2), h=((self.
     diameter / 2) / Tan((self.angle / 2))), center=False
     )
855 gcpy #     s = cylinder(r1=(self.diameter / 2), r2=(self.
     diameter / 2), h=self.flute, center=False)
856 gcpy #     sh = s.translate([0, 0, ((self.diameter / 2) / Tan
     ((self.angle / 2))]))
857 gcpy     tslist.append(hull(v.translate([bx, by, bz]), v.
     translate([ex, ey, ez])))
858 gcpy     return tslist

```

3.5.3.1.5 Keyhole Keyhole toolpaths (see: subsection 3.8.1.1.3 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



```

860 gcpy      if self.endmilltype == "keyhole":
861 gcpy          kh = cylinder(r1=(self.diameter / 2), r2=(self.diameter
862 gcpy                  / 2), h=self.flute, center=False)
863 gcpy          sh = (cylinder(r1=(self.radius / 2), r2=(self.radius /
864 gcpy                  2), h=self.flute*2, center=False))
865 gcpy          tslist.append(hull(kh.translate([bx, by, bz]), kh.
866 gcpy                  translate([ex, ey, ez])))
867 gcpy          tslist.append(hull(sh.translate([bx, by, bz]), sh.
868 gcpy                  translate([ex, ey, ez])))
869 gcpy      return tslist

```

3.5.3.1.6 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.

```

867 gcpy      if self.endmilltype == "tapered_ball":
868 gcpy          b = sphere(r=(self.tip / 2))
869 gcpy          s = cylinder(r1=(self.tip / 2), r2=(self.diameter / 2),
870 gcpy                  h=self.flute, center=False)
871 gcpy          bshape = union(b, s)
872 gcpy          tslist.append(hull(bshape.translate([bx, by, bz]),
873 gcpy                  bshape.translate([ex, ey, ez])))
874 gcpy      return tslist

```

dovetail **3.5.3.1.7 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)

```

874 gcpy      if self.endmilltype == "dovetail":
875 gcpy          dt = cylinder(r1=(self.diameter / 2), r2=(self.diameter
876 gcpy                  / 2) - self.flute * Tan(self.angle), h= self.flute,
877 gcpy                  center=False)
878 gcpy          tslist.append(hull(dt.translate([bx, by, bz]), dt.
879 gcpy                  translate([ex, ey, ez])))
880 gcpy      return tslist
881 gcpy      if self.endmilltype == "other":
882 gcpy          tslist = []
883 gcpy      def dovetail(self, dt_bottomdiameter, dt_topdiameter,
884 gcpy          dt_height, dt_angle):
885 gcpy          return cylinder(r1=(dt_bottomdiameter / 2), r2=(


```

3.5.3.2 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-unupported-tooling-in-carbide-create-roundover-cove-radius-bits/43723](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.

3.5.3.2.1 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 roundover below, see paragraph 3.5.3.2 — roundover tooling will need to generate a list of slices of the tool shape hulled together.

```

883 gcpy      if self.endmilltype == "roundover":
884 gcpy          shaft = cylinder(self.steps, self.tip/2, self.tip/2)
885 gcpy          toolpath = hull(shaft.translate([bx, by, bz]), shaft.
886 gcpy              translate([ex, ey, ez]))
887 gcpy          shaft = cylinder(self.flute, self.diameter/2 + self.tip
888 gcpy              /2, self.diameter/2 + self.tip/2)
889 gcpy          toolpath = toolpath.union(hull(shaft.translate([bx, by,
890 gcpy                  bz + self.radius]), shaft.translate([ex, ey, ez +
891 gcpy                      self.radius])))
892 gcpy          tslist = [toolpath]
893 gcpy          slice = cylinder(0.0001, 0.0001, 0.0001)
894 gcpy          slices = slice
895 gcpy          for i in range(1, 90 - self.steps, self.steps):
896 gcpy              dx = self.radius*cos(i)
897 gcpy              dxx = self.radius*cos(i + self.steps)
898 gcpy              dzz = self.radius*sin(i)
899 gcpy              dz = self.radius*sin(i + self.steps)
900 gcpy              dh = dz - dzz
901 gcpy              slice = cylinder(r1 = self.tip/2+self.radius-dx, r2
902 gcpy                  = self.tip/2+self.radius-dxx, h = dh)
903 gcpy              slices = slices.union(hull(slice.translate([bx, by,
904 gcpy                  bz+dz]), slice.translate([ex, ey, ez+dz])))
905 gcpy          tslist.append(slices)
906 gcpy
907 gcpy      return tslist

```

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 **3.5.3.3 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:

```

902 gcpy      def shaftmovement(self, bx, by, bz, ex, ey, ez):
903 gcpy          tslist = []
904 gcpy          ts = cylinder(r1=(self.shaftdiameter / 2), r2=(self.
905 gcpy              shaftdiameter / 2), h=self.shaftlength, center = False)
906 gcpy          ts = ts.translate([0, 0, self.shaftheight])
907 gcpy          tslist.append(hull(ts.translate([bx, by, bz]), ts.translate
908 gcpy              ([ex, ey, ez])))
909 gcpy
910 gcpcad module shaftmovement(bx, by, bz, ex, ey, ez){
911 gcpcad     gcp.shaftmovement(bx, by, bz, ex, ey, ez);
912 gcpcad }

```

3.5.3.4 tool outlines Defining the tools as outlines which may be scaled to different sizes and rotate_extruded requires a series of modules which must define:

- self.tooloutline — the entire outline of the tool used for rotate_extrude when cutting an arc (or a line if linear_extrude is used)
- self.toolprofile — the profile of one half of the tool suited to creating a 3D model using rotate_extrude
- self.shaftoutline
- self.shaftprofile
- self.currenttoolshape
- self.currenttoolshaft

Note that when defining tooling it is expedient to use a mix of the 2D and 3D systems. The various self.<toolparameters> are defined in toolchange and may be used at need.

An expedient option would seem to be slicing the 3D model and hulling slices from the begin/end positions, but that may result in distortions for certain tool geometries (e.g., keyhole tooling).

There are several possible options for handling outlines and models — a hybrid approach governed by if branches will allow optimization of the resultant CSG commands.

- simple shape and straight move — 3D models of the tool at the begin and end points of the move are hulled
- complex shape and straight move — 3D models of the tool at the begin and end points of the move are connected by a linear_extrude
- any shape and arc move — 3D models of the tool at the begin and end points of the move are connected by a rotate_extrude

Similarly for the tool profiles and outlines and 3D shapes:

- polygon — defining the shape in terms of point positions (note the PythonSCAD has an option for rounding which may be used for some shapes)
- 2D — defining the shape using rectangles or polygons and circles and Boolean operations
- SVG — drawing up the outlines and profiles in a vector drawing tool so that they may be imported as SVG files allows any shape to be imported. Filenames would be mapped to the tool numbering scheme.

3.5.3.4.1 defineshaft A separate command for defining the shaft is expedient, and allows handling the case of the cutting diameter and the shaft diameter being different, and by including both diameters as arguments, allows the transition, if not abrupt, to be modeled. The parameters:

- toolingdiameter
- shaftdiameter
- flute
- transition
- shaft

are obvious except for shaft — rather than the O.A.L., this is the expected length of the tool as measured from the specified flute and transition lengths to the bottom of the collet. In the absence of a specified length, the flute length (assuming no transition) should be a workable approximation.

Frequently, tools will have different diameters for cutting end and shaft — when the former is smaller, the angle typically seems to be 60 degrees — since this should *not* be used for modeling, the expedient solution is to use an easily drawn angle which is obtuse enough to be obvious, so 45 degrees will be used.

```

909 gcpy      def defineshaft(self, toolingdiameter, shaftdiameter, flute,
910 gcpy          transition, shaft):
911 gcpy          if shaftdiameter == 0:
912 gcpy              self.shaftoutline = polygon(points=[[0, flute], [
913 gcpy                  diameter, flute], [diameter, shaft], [0, shaft]])
914 gcpy              self.shaftprofile = polygon(points=[[0, flute], [
915 gcpy                  diameter/2 ,flute], [diameter/2, shaft], [0, shaft
916 gcpy                  ]])
917 gcpy              sh = cylinder(h = shaft, r = diameter/2)
918 gcpy              self.currenttoolshaft = sh.translate([0,0,flute])
919 gcpy          if shaftdiameter > 0:
920 gcpy              self.shaftoutline = polygon(points=[
921 gcpy                  [shaftdiameter / 2 - toolingdiameter / 2, flute],
922 gcpy                  [0, flute + transition],
923 gcpy                  [0, flute + transition + shaft],
924 gcpy                  [shaftdiameter, flute + transition + shaft],
925 gcpy                  [shaftdiameter, flute + transition],
926 gcpy                  [shaftdiameter / 2 + toolingdiameter / 2, flute],
927 gcpy                  ])
928 gcpy              self.shaftprofile = polygon( points= [
929 gcpy                  [0, flute],
930 gcpy                  [0, flute + transition + shaft],
931 gcpy                  [shaftdiameter/2, flute + transition + shaft],
932 gcpy                  [shaftdiameter/2, flute + transition],
933 gcpy                  [toolingdiameter/2, flute]
934 gcpy                  ] )
935 gcpy              self.currenttoolshaft = rotate_extrude(self.
936 gcpy                  shaftprofile)

```

```

64 gpcscad module defineshaft(toolingdiameter, shaftdiameter, flute,
65 gpcscad     transition, shaft){
66 gpcscad     gcp.defineshaft(toolingdiameter, shaftdiameter, flute,
66 gpcscad     transition, shaft);
66 gpcscad }
```

3.5.3.4.2 Square (including O-flute) The simplest sort of tooling, which is easily defined using a polygon and cylinder.

```

933 gcpy     def definesquaretool(self, diameter, flute, shaft, offset = 0):
934 gcpy         self.tooloutline = polygon( points=[[0 + offset,0],[diameter + offset,flute],[0 +
935 gcpy             offset,flute]] )
935 gcpy         self.toolprofile = polygon( points=[[0,0],[diameter/2,0],[diameter/2,flute],[0,flute]] )
936 gcpy         self.currenttoolshape = cylinder(h = flute, r = diameter/2)
937 gcpy         sh = cylinder(h = flute, r = diameter/2)
```

3.5.3.4.3 Ball-nose Defined using 2D and 3D primitives which are unioned together, this allows the shape of the tool to be influenced by the variables fa/fs/fn.

```

939 gcpy     def defineballnosetool(self, diameter, flute, shaft, offset =
940 gcpy         0):
941 gcpy         s = square([diameter,flute - diameter/2])
942 gcpy         sh = s.translate([0 + offset, diameter/2])
943 gcpy         c = circle(d=diameter)
944 gcpy         b = c.translate([diameter/2 + offset, diameter/2])
944 gcpy         self.tooloutline = union(sh, b)
945 gcpy #
946 gcpy         s = square([diameter/2,flute - diameter/2])
947 gcpy         sh = s.translate([0, diameter/2])
948 gcpy         c = circle(d=diameter)
949 gcpy         b = c.translate([0, diameter/2])
950 gcpy         bn = union(sh, b)
951 gcpy #
952 gcpy         bns = bn.translate([0, diameter/2])
953 gcpy #
953 gcpy         thein = square([diameter/2,flute])
954 gcpy         theins = thein.translate([diameter/2, 0])
954 gcpy         self.toolprofile = intersection(thein, bn)
955 gcpy #
956 gcpy         self.shaftprofile = polygon( points=[[0,flute],[diameter/2,
956 gcpy             flute],[diameter/2,shaft],[0,shaft]] )
957 gcpy #
958 gcpy #
959 gcpy #
960 gcpy         b = self.toolprofile
961 gcpy         bn = b.translate([-diameter/2, 0])
960 gcpy         self.currenttoolshape = rotate_extrude(self.toolprofile)
961 gcpy #
962 gcpy         self.currenttoolshaft = sh.translate([0,0,flute])
```

3.5.3.4.4 V tool outline V shaped tooling often has the V cutting flutes attached to a cylindrical shaft.

```

964 gcpy     def defineVtool(self, diameter, flute, shaft, shaftdiameter =
965 gcpy         0):
965 gcpy         self.tooloutline = polygon([[diameter/2, 0], [diameter,
966 gcpy         flute], [0, flute]])
966 gcpy #
967 gcpy
968 gcpy         self.toolprofile = polygon([[0, 0], [diameter/2, flute],
968 gcpy             [0, flute]])
969 gcpy
970 gcpy #
971 gcpy         if shaftdiameter == 0:
972 gcpy             shaftdiameter = diameter
973 gcpy         self.shaftprofile = polygon([[0, flute], [shaftdiameter/2,
973 gcpy             flute], [shaftdiameter/2, flute + shaft], [0, flute +
973 gcpy             shaft]])
974 gcpy
975 gcpy #
976 gcpy         self.currenttoolshape = rotate_extrude(self.toolprofile)
977 gcpy #
978 gcpy         self.currenttoolshaft = rotate_extrude(self.shaftprofile)
```

3.5.3.4.5 Keyhole outline Keyhole outlines will require two cutting surfaces, since it is usual for the shaft to have cutting flutes for clearing the narrow region as part of their functionality.

```

980 gcpy     def defineKeyholetool(self, diameter, flute, narrowdiameter,
981 gcpy         narrowflute, shaftdiameter, shaftlength):
982 gcpy         self.tooloutline = polygon([[0, 0], [diameter, 0], [
983 gcpy             diameter, flute], [diameter/2 + narrowdiameter/2, flute
984 gcpy             ], [diameter/2 + narrowdiameter/2, flute + narrowflute],
985 gcpy             [diameter/2 - narrowdiameter/2, flute + narrowflute], [
986 gcpy             diameter/2 - narrowdiameter/2, flute], [0, flute]])
987 gcpy         self.toolprofile = polygon([[0, 0], [diameter/2, 0], [
988 gcpy             diameter/2, flute], [narrowdiameter/2, flute], [
989 gcpy             narrowdiameter/2, flute + narrowflute], [0, flute +
990 gcpy             narrowflute]])
991 gcpy         self.shaftprofile = polygon([[0, flute + narrowflute], [
992 gcpy             narrowdiameter/2, flute + narrowflute], [shaftdiameter
993 gcpy             /2, flute + narrowflute + shaftlength], [0, flute +
994 gcpy             narrowflute + shaftlength]])
995 gcpy         self.currenttoolshape = rotate_extrude(self.toolprofile)
996 gcpy         self.currenttoolshaft = rotate_extrude(self.shaftprofile)

```

3.5.3.4.6 Bowl outline Bowl tooling is done using polygon() with the third value added so as to cause the rounding of the radius.

```

994 gcpy     def definebowltool(self, diameter, flute, radius, shaftdiameter
995 gcpy         , shaftlength):
996 gcpy         self.tooloutline =
997 gcpy         self.toolprofile = polygon([[0,0], [diameter/2, 0, radius],
998 gcpy             [diameter/2, radius], [diameter/2, flute], [0, flute]])
999 gcpy         self.shaftprofile = polygon([[0,flute], [shaftdiameter/2,
1000 gcpy             flute], [shaftdiameter/2, flute + shaftlength], [0,
1001 gcpy             flute + shaftlength]])
1002 gcpy         self.currenttoolshape = rotate_extrude(self.toolprofile)
1003 gcpy         self.currenttoolshaft = rotate_extrude(self.shaftprofile)

```

3.5.3.4.7 Tapered ball nose Creating outlines for Tapered ball nose tooling will require that the arc and tangent for the angle and rounding be calculated out if programmed, or instead, they may be drawn.

3.5.3.4.8 Roundover (cove tooling) The polygon() command does not afford an option for coves, so it will be necessary to over-draw the geometry, then remove the cove if programming, or, to simply draw the outline.

```

1005 gcpy     def defineRoundovertool(self, diameter, tipdiameter, flute,
1006 gcpy         radius, shaftdiameter, shaftlength):
1007 gcpy         self.tip = 0.508
1008 gcpy         self.diameter = 6.35 - self.tip
1009 gcpy         self.flute = 8 - self.tip
1010 gcpy         self.radius = 3.175 - self.tip/2
1011 gcpy         self.shaftdiameter = 6.35
1012 gcpy         self.shaftheight = 8
1013 gcpy         self.shaftlength = 10.0
1014 gcpy         print(diameter)
1015 gcpy         print(tipdiameter)
1016 gcpy         print(flute)
1017 gcpy         print(radius)
1018 gcpy         print(shiftdiameter)
1019 gcpy         print(shiftlength)
1020 gcpy         self.tooloutline =
1021 gcpy         self.toolprofile = polygon([[0,0], [tipdiameter/2, 0], [
1022 gcpy             diameter/2, flute], [0, flute]])

```

```

1022 gcpy #
1023 gcpy     self.shaftprofile = polygon([[0,flute], [shaftdiameter/2,
1024 gcpy         flute], [shaftdiameter/2, flute + shaftlength], [0,
1025 gcpy         flute + shaftlength]])
1026 gcpy #
1027 gcpy     self.currenttoolshape = rotate_extrude(self.toolprofile)
1028 gcpy     self.currenttoolshaft = rotate_extrude(self.shaftprofile)

```

rapid 3.5.3.5 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 before making a rapid movement — likely further changes will be required.

```

1029 gcpy     def rapid(self, ex, ey, ez, laser = 0):
1030 gcpy #         print(self.rapidcolor)
1031 gcpy         if self.generateprint == True:
1032 gcpy             laser = 1
1033 gcpy         if laser == 0:
1034 gcpy             tm = self.toolmovement(self.xpos(), self.ypos(), self.
1035 gcpy                 zpos(), ex, ey, ez)
1036 gcpy             ts = self.shaftmovement(self.xpos(), self.ypos(), self.
1037 gcpy                 zpos(), ex, ey, ez)
1038 gcpy             ts = color(ts, self.shaftcolor)
1039 gcpy             self.toolpaths.extend([tm, ts])
1040 gcpy         if self.generateprint == True:
1041 gcpy             self.steps.append(self.fgc.Extruder(on=False))
1042 gcpy             self.steps.append(self.fgc.Point(x=ex,y=ey,z=ez))
1043 gcpy             self.setxpos(ex)
1044 gcpy             self.setypos(ey)
1045 gcpy             self.setzpos(ez)
1046 gcpy
1047 gcpy     def cutline(self, ex, ey, ez):
1048 gcpy #         print(self.cutcolor)
1049 gcpy #         print(ex, ey, ez)
1050 gcpy         tm = self.toolmovement(self.xpos(), self.ypos(), self.zpos
1051 gcpy             (), ex, ey, ez)
1052 gcpy         ts = self.shaftmovement(self.xpos(), self.ypos(), self.zpos
1053 gcpy             (), ex, ey, ez)
1054 gcpy         ts = color(ts, self.rapidcolor)
1055 gcpy         self.setxpos(ex)
1056 gcpy         self.setypos(ey)
1057 gcpy         self.setzpos(ez)
1058 gcpy         if self.generatecut == True:
1059 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:

```

1060 gcpy     def movetosafeZ(self):
1061 gcpy         rapid = self.rapid(self.xpos(), self.ypos(), self.
1062 gcpy             retractheight)
1063 gcpy         if self.generatepaths == True:
1064 gcpy             rapid = self.rapid(self.xpos(), self.ypos(), self.
1065 gcpy             retractheight)
1066 gcpy             self.rapids = self.rapids.union(rapid)
1067 gcpy         // writecomment("PREPOSITION FOR RAPID PLUNGE");Z25.650
1068 gcpy         //G1Z24.663F381.0, "F", str(plunge)
1069 gcpy         if self.generatepaths == False:
1070 gcpy             return rapid
1071 gcpy         else:
1072 gcpy             return cube([0.001, 0.001, 0.001])
1073 gcpy
1074 gcpy     def rapidXYZ(self, ex, ey, ez):

```

```

1076 gcpy      rapid = self.rapid(ex, ey, ez)
1077 gcpy #     if self.generatepaths == False:
1078 gcpy       return rapid
1079 gcpy
1080 gcpy   def rapidXY(self, ex, ey):
1081 gcpy     rapid = self.rapid(ex, ey, self.zpos())
1082 gcpy #     if self.generatepaths == True:
1083 gcpy     self.rapids = self.rapids.union(rapid)
1084 gcpy #
1085 gcpy #     if self.generatepaths == False:
1086 gcpy       return rapid
1087 gcpy
1088 gcpy   def rapidXZ(self, ex, ez):
1089 gcpy     rapid = self.rapid(ex, self.ypos(), ez)
1090 gcpy #     if self.generatepaths == False:
1091 gcpy       return rapid
1092 gcpy
1093 gcpy   def rapidYZ(self, ey, ez):
1094 gcpy     rapid = self.rapid(self.xpos(), ey, ez)
1095 gcpy #     if self.generatepaths == False:
1096 gcpy       return rapid
1097 gcpy
1098 gcpy   def rapidX(self, ex):
1099 gcpy     rapid = self.rapid(ex, self.ypos(), self.zpos())
1100 gcpy #     if self.generatepaths == False:
1101 gcpy       return rapid
1102 gcpy
1103 gcpy   def rapidY(self, ey):
1104 gcpy     rapid = self.rapid(self.xpos(), ey, self.zpos())
1105 gcpy #     if self.generatepaths == False:
1106 gcpy       return rapid
1107 gcpy
1108 gcpy   def rapidZ(self, ez):
1109 gcpy     rapid = [self.rapid(self.xpos(), self.ypos(), ez)]
1110 gcpy #     if self.generatepaths == True:
1111 gcpy     self.rapids = self.rapids.union(rapid)
1112 gcpy #
1113 gcpy #     if self.generatepaths == False:
1114 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):

```

68 gpcscad module movetosafeZ(){
69 gpcscad   gcp.rapid(gcp.xpos(), gcp.ypos(), retractheight);
70 gpcscad }
71 gpcscad
72 gpcscad module rapid(ex, ey, ez) {
73 gpcscad   gcp.rapid(ex, ey, ez);
74 gpcscad }
75 gpcscad
76 gpcscad module rapidXY(ex, ey) {
77 gpcscad   gcp.rapid(ex, ey, gcp.zpos());
78 gpcscad }
79 gpcscad
80 gpcscad module rapidXZ(ex, ez) {
81 gpcscad   gcp.rapid(ex, gcp.zpos(), ez);
82 gpcscad }
83 gpcscad
84 gpcscad module rapidZ(ez) {
85 gpcscad   gcp.rapid(gcp.xpos(), gcp.ypos(), ez);
86 gpcscad }

```

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

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

```

1116 gcpy   def moveatfeedrate(self, ex, ey, ez, f):
1117 gcpy     self.writegc("G01\u21D3X", str(ex), "\u21D3Y", str(ey), "\u21D3Z", str(ez)
1118 gcpy     , "\u21D3F", str(f))
1119 gcpy     self.feedrate = f
1120 gcpy     return self.cutline(ex, ey, ez)
1121 gcpy
1122 gcpy   def cutlinedxf(self, ex, ey, ez):
1123 gcpy     self.dxfline(self.xpos(), self.ypos(), ex, ey)

```

```

1123 gcpy           self.cutline(ex, ey, ez)
1124 gcpy
1125 gcpy     def cutlinedxfgc(self, ex, ey, ez):
1126 gcpy         self.dxfline(self.xpos(), self.ypos(), ex, ey)
1127 gcpy         self.writegc("G01\u00d7X", str(ex), "\u00d7Y", str(ey), "\u00d7Z", str(ez)
1128 gcpy             )
1129 gcpy         self.cutline(ex, ey, ez)
1130 gcpy     def cutvertexdxf(self, ex, ey, ez):
1131 gcpy         self.addvertex(ex, ey)
1132 gcpy         self.writegc("G01\u00d7X", str(ex), "\u00d7Y", str(ey), "\u00d7Z", str(ez)
1133 gcpy             )
1134 gcpy         self.cutline(ex, ey, ez)
1135 gcpy     def cutlineXYZwithfeed(self, ex, ey, ez, feed):
1136 gcpy         return self.cutline(ex, ey, ez)
1137 gcpy
1138 gcpy     def cutlineXYZ(self, ex, ey, ez):
1139 gcpy         return self.cutline(ex, ey, ez)
1140 gcpy
1141 gcpy     def cutlineXYwithfeed(self, ex, ey, feed):
1142 gcpy         return self.cutline(ex, ey, self.zpos())
1143 gcpy
1144 gcpy     def cutlineXY(self, ex, ey):
1145 gcpy         return self.cutline(ex, ey, self.zpos())
1146 gcpy
1147 gcpy     def cutlineXZwithfeed(self, ex, ez, feed):
1148 gcpy         return self.cutline(ex, self.ypos(), ez)
1149 gcpy
1150 gcpy     def cutlineXZ(self, ex, ez):
1151 gcpy         return self.cutline(ex, self.ypos(), ez)
1152 gcpy
1153 gcpy     def cutlineXwithfeed(self, ex, feed):
1154 gcpy         return self.cutline(ex, self.ypos(), self.zpos())
1155 gcpy
1156 gcpy     def cutlineX(self, ex):
1157 gcpy         return self.cutline(ex, self.ypos(), self.zpos())
1158 gcpy
1159 gcpy     def cutlineYZ(self, ey, ez):
1160 gcpy         return self.cutline(self.xpos(), ey, ez)
1161 gcpy
1162 gcpy     def cutlineYwithfeed(self, ey, feed):
1163 gcpy         return self.cutline(self.xpos(), ey, self.zpos())
1164 gcpy
1165 gcpy     def cutlineY(self, ey):
1166 gcpy         return self.cutline(self.xpos(), ey, self.zpos())
1167 gcpy
1168 gcpy     def cutlineZgcfeed(self, ez, feed):
1169 gcpy         self.writegc("G01\u00d7Z", str(ez), "F", str(feed))
1170 gcpy         return self.cutline(self.xpos(), self.ypos(), ez)
1171 gcpy
1172 gcpy     def cutlineZwithfeed(self, ez, feed):
1173 gcpy         return self.cutline(self.xpos(), self.ypos(), ez)
1174 gcpy
1175 gcpy     def cutlineZ(self, ez):
1176 gcpy         return self.cutline(self.xpos(), self.ypos(), ez)

```

The matching OpenSCAD command is a descriptor:

```

88 gpcscad module cutline(ex, ey, ez){
89 gpcscad     gcp.cutline(ex, ey, ez);
90 gpcscad }
91 gpcscad
92 gpcscad module cutlinedxfgc(ex, ey, ez){
93 gpcscad     gcp.cutlinedxfgc(ex, ey, ez);
94 gpcscad }
95 gpcscad
96 gpcscad module cutlineZgcfeed(ez, feed){
97 gpcscad     gcp.cutlineZgcfeed(ez, feed);
98 gpcscad }

```

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

- cutarcCW — cut a partial arc described in a clock-wise direction

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

It will be necessary to have two separate representations of arcs — the G-code and DXF may be easily and directly supported with a single command, but representing the matching tool movement in OpenSCAD may be done in two different fashions. Originally, 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 was implemented, but instead representing the tool as an outline and using `rotate_extrude` to model the movement of the tool's outline representation through the arc movement.

- 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:

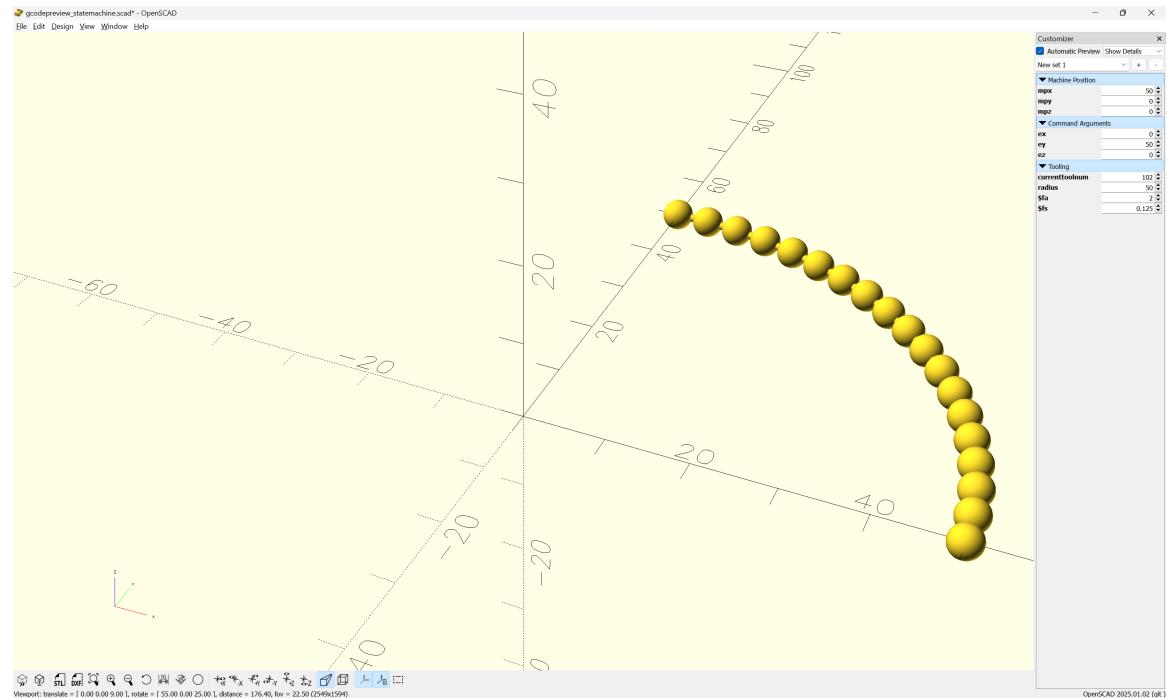
- `ex` — note that the matching origins (`bx, by, bz`) as well as the (current) toolnumber are accessed using the appropriate commands for machine position
- `ey`
- `ez` — allowing a different Z position will make possible threading and similar helical tool-paths
- `xcenter` — the center position will 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

There are two possibilities for arc movement:

- stepping through the arc and approximating with straight line movements
- using `rotate_extrude` to move an outline of the tool through the specified arc — this has the added complexity of being limited to the range of the arc, requiring that the round profile of the tool be instantiated in 3D at each end

`cutarcCW` Stepping through the arc manually is done by iterating through a loop: `cutarcCW` (clockwise) or `cutarcCC` (counterclockwise) to handle the drawing and processing of the `cutline()` toolpaths as short line segments which additionally affords a single point of control for adding additional features such as allowing the depth to vary as one cuts along an arc (the line version is used rather than shape so as to capture the changing machine positions with each step through the loop). Note that the definition matches the DXF definition of defining the center position with a matching radius, but it will be necessary to move the tool to the actual origin, and to calculate the end position when writing out a G2/G3 arc.

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



The code for which makes this obvious:

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

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

/* [Tooling] */
currenttoolnum = 102;

machine_extents();

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

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

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

```

        sphere(r=0.5);
    }
}
translate([acx + cos(i)*radius,
           acy + sin(i)*radius,
           0]){
sphere(r=2);
}
translate([acx + cos(i+inc)*radius,
           acy + sin(i+inc)*radius,
           0]){
sphere(r=2);
}
}

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

```

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.)

```

1178 gcpy      def cutarcCC(self, barc, earc, xcenter, ycenter, radius,
                           tpzreldim, stepsizearc=1):
1179 gcpy          tpzinc = tpzreldim / (earc - barc)
1180 gcpy          i = barc
1181 gcpy          while i < earc:
1182 gcpy              self.cutline(xcenter + radius * Cos(i), ycenter +
                           radius * Sin(i), self.zpos()+tpzinc)
1183 gcpy          i += stepsizearc
1184 gcpy #
1185 gcpy          self.setxpos(xcenter + radius * Cos(earc))
1186 gcpy          self.setypos(ycenter + radius * Sin(earc))
1187 gcpy      def cutarcCW(self, barc, earc, xcenter, ycenter, radius,
                           tpzreldim, stepsizearc=1):
1188 gcpy #
1189 gcpy          print(str(self.zpos()))
1190 gcpy          print(str(ez))
1191 gcpy          print(str(barc - earc))
1192 gcpy          tpzinc = ez - self.zpos() / (barc - earc)
1193 gcpy          print(str(tpzinc))
1194 gcpy          global toolpath
1195 gcpy          print("Entering n toolpath")
1196 gcpy          tpzinc = tpzreldim / (barc - earc)
1197 gcpy          cts = self.currenttoolshape
1198 gcpy          toolpath = cts
1199 gcpy          toolpath = toolpath.translate([self.xpos(), self.ypos(),
                           self.zpos()])
1200 gcpy          i = barc
1201 gcpy          while i > earc:
1202 gcpy              self.cutline(xcenter + radius * Cos(i), ycenter +
                           radius * Sin(i), self.zpos()+tpzinc)
1203 gcpy          self.setxpos(xcenter + radius * Cos(i))
1204 gcpy          self.setypos(ycenter + radius * Sin(i))
1205 gcpy          print(str(self.xpos()), str(self.ypos()), str(self.zpos()
                           ()))
1206 gcpy          self.setzpos(self.zpos()+tpzinc)
1207 gcpy          i += abs(stepsizearc) * -1
1208 gcpy          self.dxfarc(self.currenttoolnumber(), xcenter, ycenter,
                           radius, barc, earc)
1209 gcpy          if self.generatepaths == True:
1210 gcpy              print("Unioning n toolpath")
1211 gcpy              self.toolpaths = self.toolpaths.union(toolpath)
1212 gcpy          else:
1213 gcpy              self.setxpos(xcenter + radius * Cos(earc))
1214 gcpy              self.setypos(ycenter + radius * Sin(earc))
1215 gcpy              self.toolpaths.extend(toolpath)
1216 gcpy          if self.generatepaths == False:
1217 gcpy              return toolpath
1218 gcpy          else:

```

```
1219 gcpy #           return cube([0.01, 0.01, 0.01])
```

Alternately, the command for using rotate_extrude is quite straight-forward:

```
1221 gcpy     def extrudearcCC(self, barc, earc, xcenter, ycenter, radius,
1222 gcpy         tpzreldim, stepsizearc=1):
1223 gcpy         tm = self.toolmovement(self.xpos(), self.ypos(), self.zpos
1224 gcpy         (), ex, ey, ez)
1225 gcpy         tm = union(self.toolshape.translate(self.xpos(), self.ypos
1226 gcpy         (), self.zpos()))
1227 gcpy             self.toolshape.translate(),
1228 gcpy             tooloutline.translate([r-3.175,0,0]).rotate_extrude(angle=ang2-ang1).rotz(ang1) + G3_center
1229 gcpy
1230 gcpy         tm = color(tm, self.cutcolor)
1231 gcpy         ts = self.shaftmovement(self.xpos(), self.ypos(), self.zpos
1232 gcpy         (), ex, ey, ez)
1233 gcpy         ts = color(ts, self.rapidcolor)
1234 gcpy         self.setxpos(ex)
1235 gcpy         self.setypos(ey)
1236 gcpy         self.setzpos(ez)
1237 gcpy         self.toolpaths.extend([tm, ts])
```

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

```
1235 gcpy     def cutarcCWdxf(self, barc, earc, xcenter, ycenter, radius,
1236 gcpy         tpzreldim, stepsizearc=1):
1237 gcpy         self.cutarcCW(barc, earc, xcenter, ycenter, radius,
1238 gcpy         tpzreldim, stepsizearc=1)
1239 gcpy         self.dxfarc(self.currenttoolnumber(), xcenter, ycenter,
1240 gcpy         radius, earc, barc)
1241 gcpy         if self.generatepaths == False:
1242 gcpy             return toolpath
1243 gcpy         else:
1244 gcpy             return cube([0.01, 0.01, 0.01])
1245 gcpy
1246 gcpy     def cutarcCCdxf(self, barc, earc, xcenter, ycenter, radius,
1247 gcpy         tpzreldim, stepsizearc=1):
1248 gcpy         self.cutarcCC(barc, earc, xcenter, ycenter, radius,
1249 gcpy         tpzreldim, stepsizearc=1)
1250 gcpy         self.dxfarc(self.currenttoolnumber(), xcenter, ycenter,
1251 gcpy         radius, barc, earc)
```

Matching OpenSCAD modules are easily made:

```
100 gpcscad module cutarcCC(barc, earc, xcenter, ycenter, radius, tpzreldim){
101 gpcscad     gcp.cutarcCC(barc, earc, xcenter, ycenter, radius, tpzreldim);
102 gpcscad }
103 gpcscad
104 gpcscad module cutarcCW(barc, earc, xcenter, ycenter, radius, tpzreldim){
105 gpcscad     gcp.cutarcCW(barc, earc, xcenter, ycenter, radius, tpzreldim);
106 gpcscad }
```

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

```
1247 gcpy     def cutquarterCCNE(self, ex, ey, ez, radius):
1248 gcpy         if self.zpos() == ez:
1249 gcpy             tpzinc = 0
1250 gcpy         else:
1251 gcpy             tpzinc = (ez - self.zpos()) / 90
1252 gcpy             print("tpzinc ", tpzinc)
1253 gcpy             i = 1
1254 gcpy             while i < 91:
1255 gcpy                 self.cutline(ex + radius * Cos(i), ey - radius + radius
1256 gcpy                     * Sin(i), self.zpos()+tpzinc)
1257 gcpy                 i += 1
1258 gcpy     def cutquarterCCNW(self, ex, ey, ez, radius):
1259 gcpy         if self.zpos() == ez:
1260 gcpy             tpzinc = 0
1261 gcpy         else:
1262 gcpy             tpzinc = (ez - self.zpos()) / 90
1263 gcpy             tpzinc = (self.zpos() + ez) / 90
1264 gcpy             self.debug("tpzinc", tpzinc)
1265 gcpy             i = 91
```

```

1266 gcpy      while i < 181:
1267 gcpy          self.cutline(ex + radius + radius * Cos(i), ey + radius
1268 gcpy              * Sin(i), self.zpos()+tpzinc)
1269 gcpy          i += 1
1270 gcpy      def cutquarterCCSW(self, ex, ey, ez, radius):
1271 gcpy          if self.zpos() == ez:
1272 gcpy              tpzinc = 0
1273 gcpy          else:
1274 gcpy              tpzinc = (ez - self.zpos()) / 90
1275 gcpy      #                  tpzinc = (self.zpos() + ez) / 90
1276 gcpy      #                  print("tpzinc ", tpzinc)
1277 gcpy          i = 181
1278 gcpy      while i < 271:
1279 gcpy          self.cutline(ex + radius * Cos(i), ey + radius + radius
1280 gcpy              * Sin(i), self.zpos()+tpzinc)
1281 gcpy          i += 1
1282 gcpy      def cutquarterCCSE(self, ex, ey, ez, radius):
1283 gcpy          if self.zpos() == ez:
1284 gcpy              tpzinc = 0
1285 gcpy          else:
1286 gcpy              tpzinc = (ez - self.zpos()) / 90
1287 gcpy      #                  tpzinc = (self.zpos() + ez) / 90
1288 gcpy      #                  print("tpzinc ", tpzinc)
1289 gcpy          i = 271
1290 gcpy      while i < 361:
1291 gcpy          self.cutline(ex - radius + radius * Cos(i), ey + radius
1292 gcpy              * Sin(i), self.zpos()+tpzinc)
1293 gcpy          i += 1
1294 gcpy      def cutquarterCCNEdxf(self, ex, ey, ez, radius):
1295 gcpy          self.cutquarterCCNE(ex, ey, ez, radius)
1296 gcpy          self.dxfarc(ex, ey - radius, radius, 0, 90)
1297 gcpy      def cutquarterCCNWdxf(self, ex, ey, ez, radius):
1298 gcpy          self.cutquarterCCNW(ex, ey, ez, radius)
1299 gcpy          self.dxfarc(ex + radius, ey, radius, 90, 180)
1300 gcpy      def cutquarterCCSWdxf(self, ex, ey, ez, radius):
1301 gcpy          self.cutquarterCCSW(ex, ey, ez, radius)
1302 gcpy          self.dxfarc(ex, ey + radius, radius, 180, 270)
1303 gcpy      def cutquarterCCSEdxf(self, ex, ey, ez, radius):
1304 gcpy          self.cutquarterCCSE(ex, ey, ez, radius)
1305 gcpy          self.dxfarc(ex - radius, ey, radius, 270, 360)

```

```

108 gpcscad module cutquarterCCNE(ex, ey, ez, radius){
109 gpcscad     gcp.cutquarterCCNE(ex, ey, ez, radius);
110 gpcscad }
111 gpcscad
112 gpcscad module cutquarterCCNW(ex, ey, ez, radius){
113 gpcscad     gcp.cutquarterCCNW(ex, ey, ez, radius);
114 gpcscad }
115 gpcscad
116 gpcscad module cutquarterCCSW(ex, ey, ez, radius){
117 gpcscad     gcp.cutquarterCCSW(ex, ey, ez, radius);
118 gpcscad }
119 gpcscad
120 gpcscad module cutquarterCCSE(self, ex, ey, ez, radius){
121 gpcscad     gcp.cutquarterCCSE(ex, ey, ez, radius);
122 gpcscad }
123 gpcscad
124 gpcscad module cutquarterCCNEdxf(ex, ey, ez, radius){
125 gpcscad     gcp.cutquarterCCNEdxf(ex, ey, ez, radius);
126 gpcscad }
127 gpcscad
128 gpcscad module cutquarterCCNWdxf(ex, ey, ez, radius){
129 gpcscad     gcp.cutquarterCCNWdxf(ex, ey, ez, radius);
130 gpcscad }
131 gpcscad
132 gpcscad module cutquarterCCSWdxf(ex, ey, ez, radius){
133 gpcscad     gcp.cutquarterCCSWdxf(ex, ey, ez, radius);
134 gpcscad }
135 gpcscad
136 gpcscad module cutquarterCCSEdxf(self, ex, ey, ez, radius){
137 gpcscad     gcp.cutquarterCCSEdxf(ex, ey, ez, radius);

```

```
138 gpcscad }
```

3.5.4 tooldiameter

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

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

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

```
140 gpcscad 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:

```
1310 gcpy      def tool_diameter(self, ptd_tool, ptd_depth):
1311 gcpy # Square 122, 112, 102, 201
1312 gcpy      if ptd_tool == 122:
1313 gcpy          return 0.79375
1314 gcpy      if ptd_tool == 112:
1315 gcpy          return 1.5875
1316 gcpy      if ptd_tool == 102:
1317 gcpy          return 3.175
1318 gcpy      if ptd_tool == 201:
1319 gcpy          return 6.35
1320 gcpy # Ball 121, 111, 101, 202
1321 gcpy      if ptd_tool == 122:
1322 gcpy          if ptd_depth > 0.396875:
1323 gcpy              return 0.79375
1324 gcpy          else:
1325 gcpy              return ptd_tool
1326 gcpy      if ptd_tool == 112:
1327 gcpy          if ptd_depth > 0.79375:
1328 gcpy              return 1.5875
1329 gcpy          else:
1330 gcpy              return ptd_tool
1331 gcpy      if ptd_tool == 101:
1332 gcpy          if ptd_depth > 1.5875:
1333 gcpy              return 3.175
1334 gcpy          else:
1335 gcpy              return ptd_tool
1336 gcpy      if ptd_tool == 202:
1337 gcpy          if ptd_depth > 3.175:
1338 gcpy              return 6.35
1339 gcpy          else:
1340 gcpy              return ptd_tool
1341 gcpy # V 301, 302, 390
1342 gcpy      if ptd_tool == 301:
1343 gcpy          return ptd_tool
1344 gcpy      if ptd_tool == 302:
1345 gcpy          return ptd_tool
1346 gcpy      if ptd_tool == 390:
1347 gcpy          return ptd_tool
1348 gcpy # Keyhole
1349 gcpy      if ptd_tool == 374:
1350 gcpy          if ptd_depth < 3.175:
1351 gcpy              return 9.525
1352 gcpy          else:
1353 gcpy              return 6.35
1354 gcpy      if ptd_tool == 375:
1355 gcpy          if ptd_depth < 3.175:
1356 gcpy              return 9.525
1357 gcpy          else:
1358 gcpy              return 8
1359 gcpy      if ptd_tool == 376:
1360 gcpy          if ptd_depth < 4.7625:
1361 gcpy              return 12.7
1362 gcpy          else:
1363 gcpy              return 6.35
1364 gcpy      if ptd_tool == 378:
1365 gcpy          if ptd_depth < 4.7625:
1366 gcpy              return 12.7
```

```

1367 gcpy           else:
1368 gcpy               return 8
1369 gcpy # Dovetail
1370 gcpy           if ptd_tool == 814:
1371 gcpy               if ptd_depth > 12.7:
1372 gcpy                   return 6.35
1373 gcpy               else:
1374 gcpy                   return ptd_tool
1375 gcpy           if ptd_tool == 808079:
1376 gcpy               if ptd_depth > 20.95:
1377 gcpy                   return 6.816
1378 gcpy               else:
1379 gcpy                   return ptd_tool
1380 gcpy # Bowl Bit
1381 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
1382 gcpy           if ptd_tool == 45982:
1383 gcpy               if ptd_depth > 6.35:
1384 gcpy                   return 15.875
1385 gcpy               else:
1386 gcpy                   return ptd_tool
1387 gcpy # Tapered Ball Nose
1388 gcpy           if ptd_tool == 204:
1389 gcpy               if ptd_depth > 6.35:
1390 gcpy                   return ptd_tool
1391 gcpy           if ptd_tool == 304:
1392 gcpy               if ptd_depth > 6.35:
1393 gcpy                   return ptd_tool
1394 gcpy               else:
1395 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:

```

1397 gcpy     def tool_radius(self, ptd_tool, ptd_depth):
1398 gcpy         tr = self.tool_diameter(ptd_tool, ptd_depth)/2
1399 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.)

3.5.5 Feeds and Speeds

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

The tools which need to be calculated thus are those in addition to the large_square tool:

- small_square_ratio
- small_ball_ratio
- large_ball_ratio
- small_V_ratio
- large_V_ratio
- KH_ratio
- DT_ratio

3.5.6 3D Printing

Support for 3D printing requires that there be G-code commands for non-mill/router aspects such as:

- fan(s) on/off
- extruder(s)
- Heater(s)
- temperature(s)
- accelerometers

- load cells
- Filament Sensor(s)
- Filament Cutter(s)
- Display Status
 - Message
 - Build Percentage
 - (Clear) Message
- any additional commands such as “Clean Nozzle”

Moreover, it will be necessary for all values to be adjusted for specific firmware, printer and filament type combinations. Probably the best beginning will be to create a simple file using a tested set of settings in a compatible slicer as a template and to adjust based on the values from such a file.

3.5.6.1 fullcontrolgcode An extant tool for this is: <https://fullcontrolgcode.com/> which has a Python implementation at: <https://github.com/FullControlXYZ/fullcontrol>. It affords a compleat system for programming a 3D printer. The implementation <https://py2g.com/> as announced at: https://old.reddit.com/r/FullControl/comments/1mjgta3/i_made_an_online_ide_for_fullcontrol_py2gcom/ affords a straight-forward usage from which the following typical example code is pulled:

```
# see https://py2g.com/customize/grid-bins for a bonus interactive UI to use with this sketch

# =====
# PARAMETERS
# =====
layer_height = 0.4
line_width = 1.2
start_x, start_y = 10, 10
grid_unit = 25
units_x, units_y, units_z = 4, 8, 0.5
outer_radius = 5
tolerance = 0.05

flow_rate = 1.02 # fill in the gaps

bin_type_outer = True # set True to create a bin container

print_speed = 40 # highest speed you'd want to go
max_flow = 8 # in mm3/s
max_print_speed = max_flow / (layer_height*line_width) # highest speed you can go
print_speed = min(print_speed,max_print_speed)

printer_name = 'generic'
printer_settings = {
    'primer': 'travel',
    'print_speed': print_speed*60,
    'travel_speed': 20*60,
    'nozzle_temp': 210,
    'bed_temp': 50,
    'fan_percent': 100,
    'extrusion_width': line_width,
    'extrusion_height': layer_height * flow_rate
}

# =====
# DERIVED DIMENSIONS
# =====
len_x = units_x * grid_unit
len_y = units_y * grid_unit
len_z = units_z * grid_unit

lim_left = start_x + line_width/2 + tolerance/2
lim_right = start_x + len_x - line_width/2 - tolerance/2
lim_bottom = start_y + line_width/2 + tolerance/2
lim_top = start_y + len_y - line_width/2 - tolerance/2

# set up outer bin dimensions
if bin_type_outer:
    lim_left -= line_width + tolerance
    lim_right += line_width + tolerance
    lim_bottom -= line_width + tolerance
```

```

lim_top      += line_width + tolerance
outer_radius += line_width + tolerance
# make outer edge come to the same height as inner bins
len_z += layer_height*2 + tolerance

ilim_left = lim_left + line_width*2
ilim_right = lim_right - line_width*2
ilim_bottom = lim_bottom + line_width*2
ilim_top = lim_top - line_width*2

outer_left   = lim_left
outer_right  = lim_right
outer_bottom = lim_bottom
outer_top    = lim_top

# =====#
# HELPERS: Roundedrectangle boundaryfinders
# =====#
def find_boundary_x(y, going_right=True):
    if ilim_bottom + outer_radius <= y <= ilim_top - outer_radius:
        return ilim_right if going_right else ilim_left
    # bottom arc
    if y < ilim_bottom + outer_radius:
        cy = ilim_bottom + outer_radius
        dy = abs(y - cy)
        dx = math.sqrt(max(0, outer_radius**2 - dy**2))
        cx = (ilim_right - outer_radius) if going_right else (ilim_left + outer_radius)
        return cx + ( dx if going_right else -dx )
    # top arc
    if y > ilim_top - outer_radius:
        cy = ilim_top - outer_radius
        dy = abs(y - cy)
        dx = math.sqrt(max(0, outer_radius**2 - dy**2))
        cx = (ilim_right - outer_radius) if going_right else (ilim_left + outer_radius)
        return cx + ( dx if going_right else -dx )
    return ilim_right if going_right else ilim_left

def find_boundary_y(x, going_up=True):
    if ilim_left + outer_radius <= x <= ilim_right - outer_radius:
        return ilim_top if going_up else ilim_bottom
    # left arc
    if x < ilim_left + outer_radius:
        cx = ilim_left + outer_radius
        dx = abs(x - cx)
        dy = math.sqrt(max(0, outer_radius**2 - dx**2))
        cy = (ilim_top - outer_radius) if going_up else (ilim_bottom + outer_radius)
        return cy + ( dy if going_up else -dy )
    # right arc
    if x > ilim_right - outer_radius:
        cx = ilim_right - outer_radius
        dx = abs(x - cx)
        dy = math.sqrt(max(0, outer_radius**2 - dx**2))
        cy = (ilim_top - outer_radius) if going_up else (ilim_bottom + outer_radius)
        return cy + ( dy if going_up else -dy )
    return ilim_top if going_up else ilim_bottom

# =====#
# BUILD STEPS
# =====#
steps      = []
arc_segs  = 16
r          = line_width/2

wall_taper = 1.4
if bin_type_outer:
    wall_taper = 0.4

# helper function to draw an outer wall
def add_rounded_rectangle_wall(zh, r, inset = 0):
    rect_left   = outer_left + inset
    rect_right  = outer_right - inset
    rect_bottom = outer_bottom + inset
    rect_top    = outer_top - inset
    corners = [
        fc.Point(x=rect_right - r, y=rect_bottom + r, z=zh), # br
        fc.Point(x=rect_right - r, y=rect_top      - r, z=zh), # tr
        fc.Point(x=rect_left   + r, y=rect_top      - r, z=zh), # tl

```

```

        fc.Point(x=rect_left + r, y=rect_bottom + r, z=zh)    # bl
    ]
steps.append(fc.Point(x=rect_right - r, y=rect_bottom, z=zh))
steps.extend(fc.arcXY(corners[0], r, -math.pi/2, +math.pi/2, arc_segs))
steps.append(fc.Point(x=rect_right, y=rect_top - r, z=zh))
steps.extend(fc.arcXY(corners[1], r, 0, math.pi/2, arc_segs))
steps.append(fc.Point(x=rect_left + r, y=rect_top, z=zh))
steps.extend(fc.arcXY(corners[2], r, math.pi/2, math.pi/2, arc_segs))
steps.append(fc.Point(x=rect_left, y=rect_bottom + r, z=zh))
steps.extend(fc.arcXY(corners[3], r, math.pi, math.pi/2, arc_segs))

# turn extruder on
steps.append(fc.Extruder(on=True))

# -----
# LAYER 1: HORIZONTAL ZIG-ZAG
# -----
z = layer_height
y = ilim_bottom
dir_h = +1    # +1 = leftright, -1 = rightleft

# prime at first point
x0 = find_boundary_x(y, going_right=(dir_h>0))
steps.append(fc.Point(x=x0, y=y, z=z))

while True:
    # travel to boundary
    xt = find_boundary_x(y, going_right=(dir_h>0))
    steps.append(fc.Point(x=xt, y=y, z=z))
    current_x = xt

    # next scan-line
    next_y = y + line_width
    if next_y > ilim_top:
        break

    # U-turn semicircle of radius r
    center = fc.Point(x=current_x, y=y + r, z=z)
    if dir_h > 0:
        # right edge: CCW half-circle from bottom to top
        steps.extend(fc.arcXY(center, r, -math.pi/2, +math.pi, arc_segs))
    else:
        # left edge: CW half-circle from bottom to top
        steps.extend(fc.arcXY(center, r, -math.pi/2, -math.pi, arc_segs))

    y      = next_y
    dir_h = -dir_h

    # outline the first layer
    weld_offset = (wall_taper+0.5)*line_width
    add_rounded_rectangle_wall(z, outer_radius - weld_offset, weld_offset)

# -----
# LAYER 2: VERTICAL ZIG-ZAG
# -----
z += layer_height
x = ilim_left
dir_v = +1    # +1 = bottomtop, -1 = topbottom

# prime at first point
y0 = find_boundary_y(x, going_up=(dir_v>0))
steps.append(fc.Point(x=x, y=y0, z=z))

while True:
    # travel to boundary
    yt = find_boundary_y(x, going_up=(dir_v>0))
    steps.append(fc.Point(x=x, y=yt, z=z))
    current_y = yt

    # next scan-line
    next_x = x + line_width
    if next_x > ilim_right:
        break

    # U-turn semicircle of radius r
    center = fc.Point(x=x + r, y=current_y, z=z)
    if dir_v > 0:
        # top edge: CCW half-circle from left to right

```

```

        steps.extend(fc.arcXY(center, r, math.pi, -math.pi, arc_segs))
    else:
        # bottom edge: CW half-circle from left to right
        steps.extend(fc.arcXY(center, r, math.pi, +math.pi, arc_segs))

    x      = next_x
    dir_v = -dir_v

# =====
# WALLS WITH ROUNDED CORNERS (remaining layers)
# =====

weld_offset = (wall_taper+1.5)*line_width
add_rounded_rectangle_wall(z, outer_radius - weld_offset, weld_offset)
weld_offset = (wall_taper+0.75)*line_width
add_rounded_rectangle_wall(z, outer_radius - weld_offset, weld_offset)

while z < len_z:
    if wall_taper > 0:
        wall_taper -= layer_height/2
        wall_taper = max(wall_taper,0)
    add_rounded_rectangle_wall(z, outer_radius, wall_taper*line_width)
    z += layer_height

# repeat final wall and then quick ironing pass to smooth the top
add_rounded_rectangle_wall(z, outer_radius)
add_rounded_rectangle_wall(z, outer_radius)
steps.append(fc.Extruder(on=False))
z += layer_height/10 # lift a bit
add_rounded_rectangle_wall(z, outer_radius)
z += layer_height/10 # lift a bit
add_rounded_rectangle_wall(z, outer_radius)
z += layer_height # lift off
add_rounded_rectangle_wall(z, outer_radius) # maybe unnecessary
steps.append(fc.Point(z=z+20)) # lift after complete

```

A working sample file (from https://github.com/FullControlXYZ/fullcontrol/blob/master/l1m_ref.md) is:

```

import fullcontrol as fc

# Define design parameters
layer_height = 0.2

# Create a list of steps
steps = []
steps.append(fc.Point(x=0, y=0, z=0))
steps.append(fc.Point(x=10, y=0, z=0))
steps.append(fc.Point(x=10, y=10, z=0))
steps.append(fc.Point(x=0, y=10, z=0))
steps.append(fc.Point(x=0, y=0, z=layer_height))

# For visualization
fc.transform(steps, 'plot', fc.PlotControls(style='line'))

# For G-code
gcode = fc.transform(steps, 'gcode', fc.GcodeControls(
    printer_name='prusa_i3',
    save_as='my_design',
    initialization_data={
        'print_speed': 1000,
        'nozzle_temp': 210,
        'bed_temp': 60
    }
))

```

As was discussed at: https://old.reddit.com/r/FullControl/comments/1pr0o21/problems_installing_in_new_libraries_folder_in/ running this requires a fairly clean Python installation (if need be, delete and reinstall *everything*), and using code to remove two library folders from the path: <https://pastebin.com/LZFeCvVT> — the relevant code from that:

```

import sys, os
from openscad import *

def sys_path_site_pkg():
    """
    Make pip installs from OS level python accessible to PythonScad. Requires matching version (3.12.9)
    """

```

```

SITE_PKG = rf"C:\Users\{os.getlogin()}\AppData\Local\Programs\Python\Python312\Lib\site-
packages"

if SITE_PKG not in sys.path:
    sys.path.append(SITE_PKG)

# Unwind some default folder adds by PythonScad that seem to conflict!!
# Specifically: ctypes.
unwinds = set([
    rf"C:\Users\{os.getlogin()}\AppData\Local\Programs\Python\Python312\Lib",
    rf"C:\Users\{os.getlogin()}\AppData\Local\Programs\Python\Python312\DLLs"
])

sys.path = [path for path in sys.path if path not in unwinds]

sys.path_site_pkg()
print('sys.path', sys.path)

import fullcontrol as fc

# Define design parameters
layer_height = 0.2

# Create a list of steps
steps = []
steps.append(fc.Point(x=0, y=0, z=0))
steps.append(fc.Point(x=10, y=0, z=0))
steps.append(fc.Point(x=10, y=10, z=0))
steps.append(fc.Point(x=0, y=10, z=0))
steps.append(fc.Point(x=0, y=0, z=layer_height))

# For visualization
fc.transform(steps, 'plot', fc.PlotControls(style='line'))

# For G-code
gcode = fc.transform(steps, 'gcode', fc.GcodeControls(
    printer_name='prusa_i3',
    save_as='my_design',
    initialization_data={
        'print_speed': 1000,
        'nozzle_temp': 210,
        'bed_temp': 60
    }
))

```

3.5.6.2 Previewing/verifying G-code for 3D printers A 3rd-party tool for this is: https://help.prusa3d.com/article/prusaslicer-g-code-viewer_193152

3.5.6.3 Time and Firmware for 3D printers The various G-code commands are specific to firmware implementations such as <https://www.klipper3d.org/G-Codes.html>

Where CNC operations normally only are concerned about time in the moment, and pausing until a given time has elapsed, 3D operations, with their control of heating up filament, melting it, and extruding thin ribbons of it require a greater control over time and duration.

3.5.6.4 Sample 3D printing file

```

M106 S0
M106 P2 S0
;TYPE:Custom
;===== date: 20240520 =====
;printer_model:Elegoo Centauri Carbon
;initial_filament:PLA
;curr_bed_type:Textured PEI Plate
M400 ; wait for buffer to clear
M220 S100 ;Set the feed speed to 100%
M221 S100 ;Set the flow rate to 100%
M104 S140
M140 S60
G90
G28 ;home
M729 ;Clean Nozzle
M190 S60

```

```
;=====turn on fans to prevent PLA jamming=====
```

```
M106 P3 S255
;Prevent PLA from jamming

;enable_pressure_advance:false
;This value is called if pressure advance is enabled

M204 S5000 ;Call exterior wall print acceleration

G1 X128.5 Y-1.2 F20000
G1 Z0.3 F900
M73 P1 R0
M109 S210
M83
G92 E0 ;Reset Extruder
G1 F6000
G1 X-1.2 E10.156 ;Draw the first line
G1 Y98.8 E7.934
M73 P7 R0
G1 X-0.5 Y100 E0.1
M73 P11 R0
G1 Y-0.3 E7.934
G1 X78.5 E6.284
M73 P15 R0
G1 F1680
M73 P18 R0
G1 X98.5 E2
G1 F8400
M73 P21 R0
G1 X118.5 E2
G1 F1680
G1 X138.5 E2
G1 F8400
M73 P24 R0
G1 X158.5 E2
G1 F8400
M73 P25 R0
G1 X178.5 E2
;End PA test.
```

```
G3 I-1 J0 Z0.6 F1200.0 ;Move to side a little
M73 P27 R0
G1 F20000
G92 E0 ;Reset Extruder
;LAYER_COUNT:1
;LAYER:0
G90
G21
M83 ; use relative distances for extrusion
; filament start gcode
M106 P3 S200
```

```
;LAYER_CHANGE
;Z:0.2
;HEIGHT:0.2
;BEFORE_LAYER_CHANGE
;0.2
G92 E0

G1 E-.8 F1800
;LAYER:1

;_SET_FAN_SPEED_CHANGING_LAYER
SET_VELOCITY_LIMIT ACCEL=500
EXCLUDE_OBJECT_START NAME=Disc_id_0_copy_0
G1 X135.645 Y128.74 F30000
M73 P31 R0
G1 Z.6
G1 Z.2
G1 E.8 F1800
;TYPE:Outer wall
;WIDTH:0.499999
```

```
G1 F3000
G3 X128.198 Y121.357 I-7.146 J-.24 E1.19765
M73 P34 R0
G3 X130.232 Y121.573 I.058 J9.145 E.07407
G3 X135.591 Y127.663 I-1.733 J6.927 E.31169
M73 P35 R0
G1 X135.643 Y128.7 E.03754
G1 E-.728 F1800
;WIPE_START
G1 F30000
G1 X135.585 Y129.458 E-.0456
G1 X135.504 Y129.891 E-.0264
;WIPE_END
G1 X132.262 Y122.981 Z.6
M73 P36 R0
G1 X132.077 Y122.586 Z.6
G1 Z.2
M73 P37 R0
G1 E.8 F1800
;TYPE:Bottom surface
;WIDTH:0.505817
G1 F6300
G1 X133.335 Y123.844 E.06511
G3 X134.64 Y125.803 I-4.602 J4.479 E.08662
G1 X131.189 Y122.353 E.17854
M73 P38 R0
G1 X130.445 Y122.073 E.02909
G1 X130.192 Y122.01 E.00954
G1 X134.995 Y126.813 E.24849
M73 P39 R0
G3 X135.149 Y127.621 I-3.921 J1.166 E.03018
G1 X129.378 Y121.851 E.29858
M73 P40 R0
G2 X128.676 Y121.803 I-.554 J2.949 E.02582
G1 X135.204 Y128.331 E.33779
M73 P41 R0
G3 X135.19 Y128.972 I-3.173 J.251 E.02348
G1 X128.027 Y121.809 E.37065
M73 P42 R0
G2 X127.438 Y121.874 I.029 J2.945 E.02172
M73 P43 R0
G1 X135.124 Y129.56 E.39772
M73 P44 R0
G3 X135.017 Y130.108 I-2.76 J-.255 E.02045
G1 X126.89 Y121.981 E.42051
M73 P45 R0
G1 X126.387 Y122.133 E.01923
G1 X134.868 Y130.614 E.43887
M73 P46 R0
G3 X134.687 Y131.087 I-2.431 J-.66 E.01858
G1 X125.912 Y122.313 E.45404
M73 P47 R0
G2 X125.463 Y122.518 I.79 J2.324 E.01811
M73 P48 R0
G1 X134.481 Y131.536 E.46662
M73 P49 R0
G3 X134.252 Y131.962 I-2.22 J-.918 E.01772
G1 X125.038 Y122.748 E.47677
M73 P50 R0
G2 X124.646 Y123.01 I1.102 J2.07 E.01729
G1 X133.99 Y132.354 E.4835
M73 P52 R0
G3 X133.707 Y132.726 I-1.979 J-1.213 E.01712
G1 X124.273 Y123.292 E.48816
M73 P53 R0
G2 X123.918 Y123.592 I1.305 J1.903 E.01702
G1 X133.406 Y133.079 E.49092
M73 P54 R0
G1 X133.077 Y133.405 E.01694
G1 X123.595 Y123.923 E.49064
M73 P56 R0
G2 X123.291 Y124.274 I1.583 J1.677 E.01701
G1 X132.725 Y133.708 E.48813
M73 P57 R0
G3 X132.354 Y133.992 I-1.59 J-1.689 E.01711
G1 X123.006 Y124.643 E.48373
M73 P58 R0
G1 X122.75 Y125.042 E.01733
```

```
M73 P59 R0
G1 X131.959 Y134.251 E.47651
M73 P60 R0
G3 X131.534 Y134.481 I-1.349 J-1.984 E.0177
G1 X122.519 Y125.466 E.46649
M73 P61 R0
G2 X122.31 Y125.912 I2.1 J1.254 E.01805
G1 X131.087 Y134.688 E.45415
M73 P62 R0
G3 X130.615 Y134.871 I-1.138 J-2.244 E.01855
M73 P63 R0
G1 X122.127 Y126.383 E.43917
M73 P64 R0
G1 X121.985 Y126.896 E.01946
G1 X130.105 Y135.016 E.42016
M73 P65 R0
G3 X129.558 Y135.123 I-.806 J-2.651 E.02043
G1 X121.877 Y127.442 E.39747
M73 P66 R0
G2 X121.81 Y128.03 I2.87 J.626 E.02167
G1 X128.97 Y135.19 E.37051
M73 P68 R0
G3 X128.33 Y135.204 I-.391 J-3.158 E.02348
G1 X121.795 Y128.67 E.33813
M73 P69 R0
G2 X121.851 Y129.38 I3.542 J.078 E.02613
G1 X127.619 Y135.149 E.29847
M73 P70 R0
G3 X126.809 Y134.992 I.366 J-4.085 E.03026
G1 X122.009 Y130.193 E.24836
M73 P71 R0
G1 X122.057 Y130.392 E.00749
G1 X122.28 Y131.031 E.02476
G1 X122.356 Y131.195 E.00663
G1 X125.802 Y134.641 E.17832
M73 P72 R0
G3 X123.807 Y133.3 I2.526 J-5.915 E.0885
G1 X122.586 Y132.079 E.06316
M73 P73 R0
G1 E-.728 F1800
;WIPE_START
G1 F30000
G1 X123.435 Y132.928 E-.072
;WIPE_END
EXCLUDE_OBJECT_END NAME=Disc_id_0_copy_0
M106 S0
M106 P2 S0
;TYPE:Custom
; filament end gcode
===== date: 20250109 =====
M400 ; wait for buffer to clear
M140 S0 ;Turn-off bed
M106 S255 ;Cooling nozzle
M83
G92 E0 ; zero the extruder
G2 I1 J0 Z0.7 E-1 F3000 ; lower z a little
M73 P74 R0
G90
G1 Z100 F20000 ; Move print head up
M73 P94 R0
M204 S5000
M400
M83
G1 X202 F20000
M73 P95 R0
M400
G1 Y250 F20000
M73 P97 R0
G1 Y264.5 F1200
M73 P100 R0
M400
G92 E0
M104 S0 ;Turn-off hotend
M140 S0 ;Turn-off bed
M106 S0 ; turn off fan
M106 P2 S0 ; turn off remote part cooling fan
M106 P3 S0 ; turn off chamber cooling fan
M84 ;Disable all steppers
```

3.5.6.5 Initialize Certain commands are only needed for initialization, so may be grouped together in a single command:

```

1401 gcpy      def initializeforprinting(self, nozzlediameter = 0.4,
1402 gcpy          filamentdiameter = 1.75, extrusionwidth = 0.6, layerheight =
1403 gcpy              0.2, extrusiontype = "relative", extruder_temperature =
1404 gcpy              260, bed_temperature = 60, printer_name = "generic",
1405 gcpy              Base_filename = "export"):
1406 gcpy          self.nozzlediameter = nozzlediameter
1407 gcpy          self.filamentdiameter = filamentdiameter
1408 gcpy          self.extrusionwidth = extrusionwidth
1409 gcpy          self.layerheight = layerheight
1410 gcpy          self.extrusiontype = extrusiontype
1411 gcpy          self.extruder_temperature = extruder_temperature
1412 gcpy          self.bed_temperature = bed_temperature
1413 gcpy          self.printer_name = printer_name
1414 gcpy          self.Base_filename= Base_filename
1415 gcpy
1416 gcpy #      def sys_path_site_pkg():
1417 gcpy      '''
1418 gcpy          Make pip installs from OS level python accessible to
1419 gcpy          PythonScad. Requires matching version (3.12.9)
1420 gcpy      '''
1421 gcpy      SITE_PKG = rf"C:\Users\{os.getlogin()}\AppData\Local\
1422 gcpy          Programs\Python\Python312\Lib\site-packages"
1423 gcpy
1424 gcpy
1425 gcpy      # Unwind some default folder adds by PythonScad that seem
1426 gcpy          to conflict!!
1427 gcpy
1428 gcpy      # Specifically: ctypes.
1429 gcpy      unwinds = set([
1430 gcpy          rf"C:\Users\{os.getlogin()}\AppData\Local\Programs\
1431 gcpy          Python\Python312\Lib",
1432 gcpy          rf"C:\Users\{os.getlogin()}\AppData\Local\Programs\
1433 gcpy          Python\Python312\DLLs"
1434 gcpy
1435 gcpy
1436 gcpy      sys.path = [path for path in sys.path if path not in
1437 gcpy          unwinds]
1438 gcpy
1439 gcpy
1440 gcpy #  initialization/prime procedure
1441 gcpy      self.rapid(10,10,0.3)                                # G0
1442 gcpy          F8000 X10 Y10 Z0.3
1443 gcpy          self.rapid(self.xpos(),12,0.2)                  # G0
1444 gcpy          F8000 Y12 Z0.2
1445 gcpy          self.extrude(110, self.ypos(),self.zpos(), True)  # G1
1446 gcpy          F1000 X110 E3.326014
1447 gcpy          self.extrude(self.xpos(), 14, self.zpos(), True) # G1 Y14
1448 gcpy          E0.06652
1449 gcpy          self.extrude(10,self.ypos(), self.zpos(), True)  # G1 X10
1450 gcpy          E3.326014
1451 gcpy          self.extrude(self.xpos(), 16, self.zpos(), True) # G1 Y16
1452 gcpy          E0.06652
1453 gcpy          self.extrude(self.xpos(), 10, self.zpos(), True) # G1 Y10
1454 gcpy          E0.199561
1455 gcpy          self.extrude(20, self.ypos(), self.zpos(), True)  # G1 X20
1456 gcpy          E0.332601
1457 gcpy          self.extrude(self.xpos(), 20,self.zpos(), True)  # G1 Y20
1458 gcpy          E0.133041
1459 gcpy          self.rapid(self.xpos(),12,0.2)                  # G0
1460 gcpy          F8000 Y12 Z0.2
1461 gcpy
1462 gcpy      # end position X20, Y20, Z0.2

```

The program https://github.com/FullControlXYZ/fullcontrol/blob/master/models/hex_adapter.ipynb suggests certain variables:

```
# printer/gcode parameters

design_name = 'hex_adapter'
nozzle_temp = 210
bed_temp = 40
print_speed = 1000
fan_percent = 100
printer_name='prusa_i3' # generic / ultimaker2plus / prusa_i3 / ender_3 / cr_10 / bambulab_x1 / toolchan
```

Movement commands add an E position aspect to the command which results in the Extruder advancing to that position so as to extrude a sufficient volume of filament to match the movement and the space which is intended to be filled. Modeling these in 3D without the complexity of managing the entire 3D model and tracking the elevation of the current position relative to the model at a given point in time will require that the user maintain the current layer thickness and ensure that if unsupported, the extruded plastic will be extruded with a fan speed and flow rate which will allow bridging from/to supported areas of the model.

Calculating the volume necessary/the amount extruded will require the nozzle size, the layer height, an estimate for how much the extruded filament will spread out/deform, and the diameter of the filament. Further potential complications include whether the first layer is being extruded (normally this is done at a quite slow speed to facilitate adhesion, which also serves as a chance to catch a problem at an early stage), or if a strand is an inside or outside wall or infill or bridging open space, if it is crossing an already extruded segment(?) and so forth.

```
; --- Start of G-code: Demonstration of Layer and Extrusion Concepts ---
G21 ; Set units to millimeters
G90 ; Use absolute positioning
M82 ; Set extruder to absolute mode
M104 S200 ; Set extruder temperature to 200°C
M140 S60 ; Set bed temperature to 60°C
M190 S60 ; Wait for bed to reach target temp
M109 S200 ; Wait for extruder to reach target temp
G28 ; Home all axes

; --- Initial test extrusion ---
G92 E0 ; Reset extruder position
G1 F100 E5 ; Extrude 5 mm of filament at low speed to prime the nozzle
; Purpose: Ensures clean flow and purges any residual filament

; --- First layer adhesion test ---
G1 Z0.2 ; Move nozzle to first layer height
G1 X10 Y10 F3000 ; Move to starting position
G1 F1800 ; Set slower speed for first layer
G1 E0.8 ; Slight retraction before starting
G1 X100 E10 ; Draw a line along X to test bed adhesion
; Comment: This line helps verify that the first layer sticks properly

; --- Outer wall generation ---
G1 Z0.2 ; Maintain layer height
G1 X100 Y100 E10 ; Move and extrude to start outer square
G1 X10 Y100 E10 ;
G1 X10 Y10 E10 ;
G1 X100 Y10 E10 ;
; Outer walls: Typically printed first to preserve dimensional accuracy

; --- Cornering adjustment ---
G1 F1200 ; Reduce speed at corners
G1 X100 Y100 E0.5 ;
; Comment: Slower cornering helps prevent blobbing and maintains sharp edges

; --- Inner wall generation ---
G1 F1800 ; Resume regular speed
G1 X95 Y95 E8 ;
G1 X15 Y95 E8 ;
G1 X15 Y15 E8 ;
G1 X95 Y15 E8 ;
; Comment: Inner walls follow outer walls to enhance structural strength

; --- Understanding extrusion width ---
; Parameters:
; - Nozzle = 0.4 mm
; - Layer height = 0.2 mm
; - Filament diameter = 1.75 mm

; Flow rate ~ (extrusion_width * layer_height) / ( * (filament_diameter/2)^2)
; Example calculation: (0.4 * 0.2) / ($\pi$ * (0.875)^2) 0.033 mm³/mm
```

```

; --- Smooth top layer strategy ---
G1 Z0.4 ; Move to top layer height
G1 X20 Y20 ;
G1 F1500 ;
G1 X90 E3 ; Lay down parallel top layer strokes
G1 X90 Y90 E3 ;
G1 X20 Y90 E3 ;
G1 X20 Y20 E3 ;
G1 F3000 ;
G1 X20 Y20 ;
G1 F1500 ;
G1 X90 E3 ; Repeat for second pass for smoothing
; Tip: Overlapping infill with slightly lower extrusion helps achieve a smooth finish

; --- Wrap up ---
G92 E0 ; Reset extruder
G1 E-2 F1800 ; Retract filament to prevent stringing
M104 S0 ; Turn off hotend
M140 S0 ; Turn off bed
G28 X0 ; Home X-axis
M84 ; Disable motors
; --- End of G-code demonstration ---

```

3.5.6.6 extrude 3D printing requires control of the extruder, and matching volumetric calculations (or, more accurately, volumetric calculations which then determine the rate of extrusion).

Previewing in 3D/programming for 3D extrusion will likely want previewing not just the extruded shape, but also tracking the volume of material extruded and how it relates to the volume of the object being filled/the intersection of a just-extruded region with previously extruded material, and how large a void is left (presumably those two volumes would match up).

One concern is that G2/G3 support apparently is not common/guaranteed in 3D printer firmwares:

available if a gcode_arcs config section is enabled

<https://www.klipper3d.org/G-Codes.html> While it is possible to separately control the feed rate of the extrusion, and the length of material extruded:

```
G1 F100 E5 ; Extrude 5 mm of filament at low speed to prime the nozzle
```

The normal usage is to move at a preset Feed rate in terms of motion, and while that movement is being made, extrude a given length of material:

```

; --- First layer adhesion test ---
G1 Z0.2 ; Move nozzle to first layer height
G1 X10 Y10 F3000 ; Move to starting position
G1 F1800 ; Set slower speed for first layer
G1 E0.8 ; Slight retraction before starting
G1 X100 E10 ; Draw a line along X to test bed adhesion
; Comment: This line helps verify that the first layer sticks properly

```

In theory, if one had a layer height equal to the diameter of the filament, and wanted to extrude a circular cross-section of filament, the value for E would be equal to the distance traveled.

Apparently, the firmware control is limited so that the extrusion rate cannot be varied relative to the feed rate so that it is not possible to for example, decrease the speed/increase the extrusion rate, resulting in a trapezoidal extrusion.

Given all that, the idealized (normalized?) shape and dimensions of the extrusion would be controlled by:

- layer height (for height along Z)
- extrusion rate (for width in X/Y)

which would be previewed as a rounded cross section, so it should work to create a preview by calculating the volume of material which is being extruded, then determining the volume of a circle of radius layer height/2, subtract that from the extruded volume, then determine what width of rectangle cross section would be necessary at the specified length to make up the difference.

```

1454 gcpy      def extrude(self, ex, ey, ez, extrudeonly = False):
1455 gcpy          if extrudeonly == False:
1456 gcpy              self.steps.append(self.fgc.Point(x=ex, y=ey, z=ez))
1457 gcpy              ew = self.extrusionwidth
1458 gcpy              lh = self.layerheight
1459 gcpy              i = circle(lh/2)
1460 gcpy              j = i.translate([0, lh/2, 0])
1461 gcpy              k = intersection(j, square([lh, lh]))
1462 gcpy              l = k.translate([ew/2-lh/2, 0, 0])

```

```

1463 gcpy      m = union(l, square([ew/2-lh/2, lh]))
1464 gcpy      c = rotate_extrude(m)
1465 gcpy      c = c.translate([0,0,-self.layerheight])
1466 gcpy      tslist = hull(c.translate([self.xpos(), self.ypos(), self.
1467 gcpy          zpos()]), c.translate([ex, ey, ez]))
1468 gcpy      self.toolpaths.append(tslist)
1469 gcpy      self.mpx = ex
1470 gcpy      self.mpy = ey
1470 gcpy      self.mpz = ez

```

3.5.6.7 Shutdown Shutting the machine down at the end of a print affords the chance to also write out the G-code using FullControl (as opposed to having a separate command for this)

```

1472 gcpy      def shutdownafterprinting(self, print_speed = 1000):
1473 gcpy          print(self.steps)
1474 gcpy # For G-code
1475 gcpy          gcode = self.fgc.transform(self.steps, 'gcode',
1476 gcpy                      self.fgc.GcodeControls(printer_name =
1477 gcpy                          self.printer_name,
1478 gcpy                          save_as = self.Base_filename,
1479 gcpy                          initialization_data={
1480 gcpy                              'print_speed': str(print_speed),
1481 gcpy                              'nozzle_temp': str(self.
1482 gcpy                                  extruder_temperature),
1483 gcpy                              'bed_temp': str(self.
1483 gcpy                                  bed_temperature)
1482 gcpy          }
1483 gcpy      )

```

3.5.6.8 fullcontrolgcode commands At https://github.com/FullControlXYZ/fullcontrol/blob/master/l1m_ref.md there are a number of commands beyond the basic Point movement implemented above asextrude().

Things which will need to be looked into include:

- printer models for initialization — an if-then structure for the specific implementations may be needed, but if implemented will need to be kept in synch
- rectangle: Requires width and height
 - rectangleXY(start_point, x_size, y_size, cw=False): Generate a 2D XY rectangle, returns a list of FullControl Point objects
 - stadium: Rectangle with semi-circle at each end, requires width and height
- circle: Requires diameter
 - circleXY(centre, radius, start_angle, segments=100, cw=False): Generate a 2D XY circle, returns a list of FullControl Point objects
 - circleXY_3pt(pt1, pt2, pt3, start_angle=None, start_at_first_point=None, segments=100, cw=False): Generate a circle passing through three points, returns a list of FullControl Point objects
- arcXY(centre, radius, start_angle, arc_angle, segments): Generate an arc
- variable_arcXY(centre, start_radius, start_angle, arc_angle, segments, radius_change=0, z_change=0): Generate an arc with variable radius and z-height
- ellipseXY(centre, a, b, start_angle, segments=100, cw=False): Generate a 2D XY ellipse, returns a list of FullControl Point objects
- polygonXY(centre, enclosing_radius, start_angle, sides, cw=False): Generate a 2D XY polygon, returns a list of FullControl Point objects
- Complex Shapes
 - spiralXY(centre, start_radius, end_radius, start_angle, n_turns, segments, cw=False): Generate a 2D XY spiral
 - helixZ(centre, start_radius, end_radius, start_angle, n_turns, pitch_z, segments, cw=False): Generate a helix in the Z direction
- Wave Functions (fullcontrol/geometry/waves.py)
 - squarewaveXY(start_point, direction_vector, amplitude, line_spacing, periods, extra_half_period=False, extra_end_line=False): Generate a square wave

- squarewaveXYpolar(start_point, direction_polar, amplitude, line_spacing, periods, extra_half_period=False, extra_end_line=False): Generate a square wave using polar coordinates
- trianglewaveXYpolar(start_point, direction_polar, amplitude, tip_separation, periods, extra_half_period=False): Generate a triangle wave
- sinewaveXYpolar(start_point, direction_polar, amplitude, period_length, periods, segments_per_period=16, extra_half_period=False, phase_shift=0): Generate a sine wave
- segmented_line(start_point, end_point, segments): Create a line with multiple segments that can be modified after creation

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:

```

1485 gcpy      def stockandtoolpaths(self, option = "stockandtoolpaths"):
1486 gcpy          if option == "stock":
1487 gcpy              show(self.stock)
1488 gcpy          elif option == "toolpaths":
1489 gcpy              show(self.toolpaths)
1490 gcpy          elif option == "rapids":
1491 gcpy              show(self.rapids)
1492 gcpy      else:
1493 gcpy          part = self.stock.difference(self.rapids)
1494 gcpy          part = self.stock.difference(self.toolpaths)
1495 gcpy          show(part)

```

A separate set of commands for showing the outline of the currently selected tool and/or its shaft is useful for checking that a tool outline definition is correctly formed.

```

1497 gcpy      def showtooloutline(self):
1498 gcpy          to = union(self.tooloutline, self.shaftoutline)
1499 gcpy          show(to)
1500 gcpy
1501 gcpy      def showtoolprofile(self):
1502 gcpy          to = union(self.toolprofile, self.shaftprofile)
1503 gcpy          show(to)
1504 gcpy
1505 gcpy      def showtoolshape(self):
1506 gcpy          to = union(self.currenttoolshape, self.currenttoolshaft)
1507 gcpy          show(to)

```

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 which is returned is displayed), the most expedient mechanism is to have an explicit Python command which returns the 3D model:

```

1509 gcpy      def returnstockandtoolpaths(self):
1510 gcpy          part = self.stock.difference(self.toolpaths)
1511 gcpy          return part

```

and then make use of that specific command for OpenSCAD:

```

142 gpcscad module stockandtoolpaths(){
143 gpcscad     gcp.returnstockandtoolpaths();
144 gpcscad }

```

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. Note that G-code support is handled in two separate fashions, using Full Control G-code for 3D Printing as described above, and directly managing the file as described below.

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 definition for each sort of file which may be opened.

```

1513 gcpy     def writegc(self, *arguments):
1514 gcpy         if self.generategc == True:
1515 gcpy             line_to_write = ""
1516 gcpy             for element in arguments:
1517 gcpy                 line_to_write += element
1518 gcpy                 self.gc.write(line_to_write)
1519 gcpy                 self.gc.write("\n")
1520 gcpy
1521 gcpy     def writedxf(self, *arguments):
1522 gcpy #         global dxfclosed
1523 gcpy         line_to_write = ""
1524 gcpy         for element in arguments:
1525 gcpy             line_to_write += element
1526 gcpy         if self.generatedxf == True:
1527 gcpy             if self.dxfclosed == False:
1528 gcpy                 self.dxf.write(line_to_write)
1529 gcpy                 self.dxf.write("\n")

```

which commands will accept a series of arguments and then write them out to a file object for the appropriate file.

`opengcodefile` For writing to files it will be necessary to have commands for opening the files: `opengcodefile` `opendxfile` and `opendxffile` which will set the associated defaults. There is a separate function for each type of file.

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

```

146 gpcscad module opendxffile(basefilename){
147 gpcscad     gcp.opendxffile(basefilename);
148 gpcscad }

```

`opengcodefile` With matching OpenSCAD commands: `opengcodefile` for OpenSCAD:

```

154 gpcscad module opengcodefile(basefilename, currenttoolnum, toolradius,
155 gpcscad     plunge, feed, speed) {
155 gpcscad     gcp.opengcodefile(basefilename, currenttoolnum, toolradius,
156 gpcscad     plunge, feed, speed);
156 gpcscad }

```

and Python:

```

1531 gcpy     def opengcodefile(self, basefilename = "export",
1532 gcpy         currenttoolnum = 102,
1533 gcpy         toolradius = 3.175,
1534 gcpy         plunge = 400,
1535 gcpy         feed = 1600,
1536 gcpy         speed = 10000
1537 gcpy         ):
1538 gcpy         self.basefilename = basefilename
1539 gcpy         self.currenttoolnum = currenttoolnum
1540 gcpy         self.toolradius = toolradius
1541 gcpy         self.plunge = plunge
1542 gcpy         self.feed = feed
1543 gcpy         self.speed = speed
1544 gcpy         if self.generategc == True:
1545 gcpy             self.gcodename = basefilename + self.gcodext
1546 gcpy             self.gc = open(self.gcodename, "w")
1547 gcpy             self.writegc("(DesignFile: " + self.basefilename + ")"
1548 gcpy
1549 gcpy     def opendxfile(self, basefilename = "export"):
1550 gcpy         self.basefilename = basefilename
1551 gcpy #         global generateddxfs
1552 gcpy #         global dxfclosed
1553 gcpy         self.dxfclosed = False
1554 gcpy         self.dxfcolor = "Black"
1555 gcpy         self.dxflayer = "DEFAULT"
1556 gcpy         if self.generatedxf == True:
1557 gcpy             self.generateddxfs = False
1558 gcpy             self.dxffilename = basefilename + ".dxf"
1559 gcpy             self.dxf = open(self.dxffilename, "w")
1560 gcpy             self.dxfpreamble()

```

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

Drawing Exchange Format (DXF) is a file format originally developed by AutoDesk in 1982 for using plain text to describe CAD files. The format specification is available: https://images.autodesk.com/adsk/files/autocad_2012_pdf_dxf-reference_enu.pdf. A different Python project in this space, *ezdxf* also affords extensive documentation: <https://ezdxf.readthedocs.io/en/stable/introduction.html>. Applications may write out or import it as a mechanism for moving a CAD file from one system to another, or use it as a file format for saving and opening.

Elements in DXFs are represented as lines or arcs. A minimal file showing both as well as setting units to metric and placing each element on a separate layer and assigning a different colour:

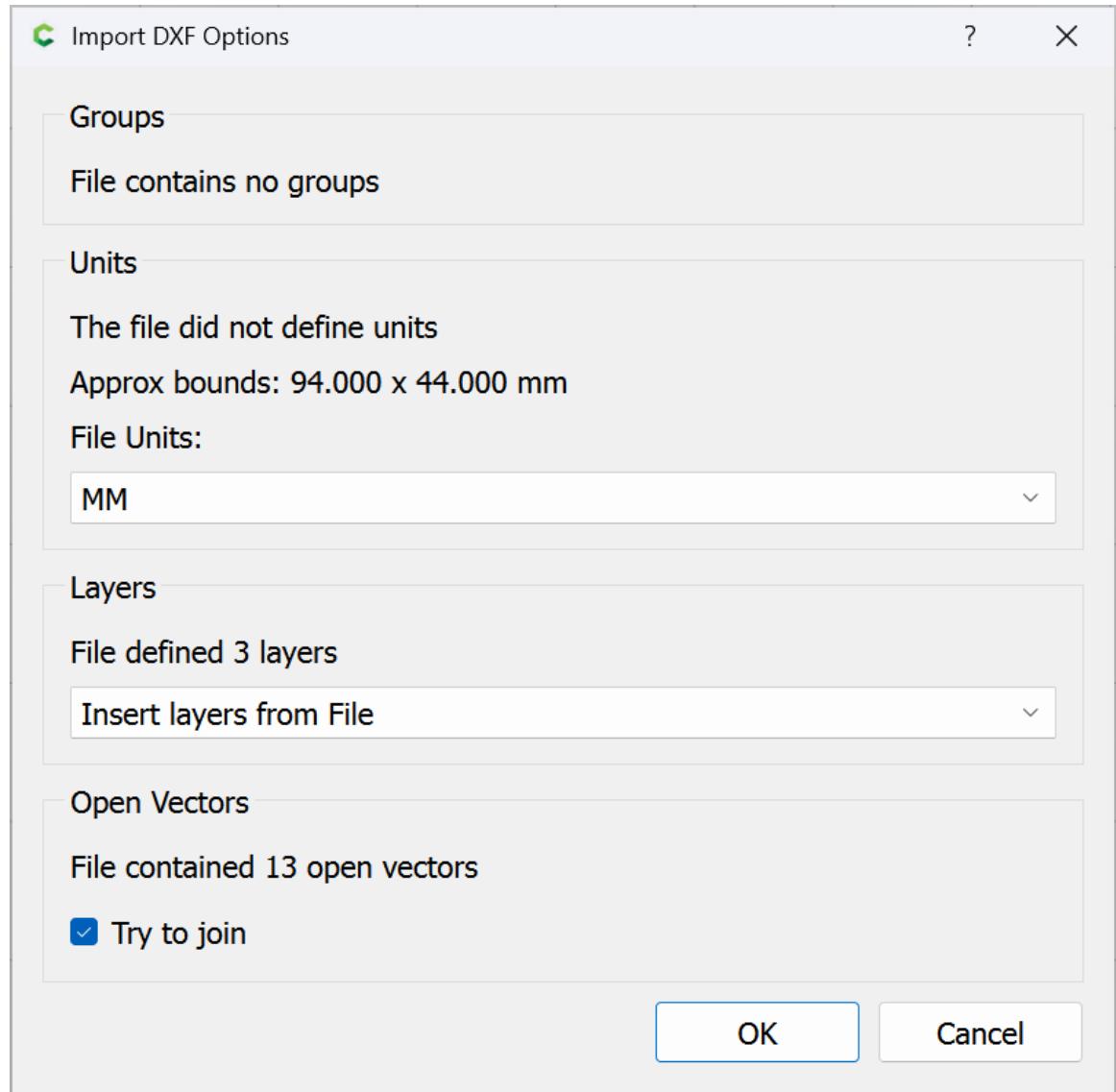
```

0
SECTION
2
HEADER
9
$INSUNITS
70
4
9
$ACADVER
1
AC1014
0
ENDSEC
0
SECTION
2
ENTITIES
0
LINE
8
Layer_Red
62
1
10
0
20
0
30
0.0
11
100
21
0
31
0.0
0
ARC
8
Layer_Blue
62
5
10
3
20
47
40
6
50
270
51
360
0
ENDSEC
0
EOF

```

Gcodepreview will write to DXF files, so a primary consideration is which features of the file format are supported by the applications which the files will be imported into. As of Carbide Create build 839 <https://community.carbide3d.com/t/carbide-create-beta-839/101187>, it is

now possible to import groups and layers, apply/use units, and have a report on open (unjoined) vectors with an option to close them:



With Carbide Create's facility to use Variables for Stock Dimensions <https://willadams.gitbook.io/design-into-3d/2d-drawing#a-note-on-dimensions> it then becomes possible to realize a fully parametric workflow where toolpaths are defined in terms of Stock Thickness (T) for depth of cut, while the part dimensions are derived from the imported geometry as placed on the associated layer.

An example file which supports units (mm) and a layer (o) and has accurate dimensions is available at: <https://github.com/keebio/BDN9-case/blob/master/rev2/bdn9-rev2-mid-enlarged.dxf>. Note that layer o is required by most DXF implementations, however, Carbide Create does not enforce this requirement.

dxfpreamble 3.7.2.1 Writing a preamble to DXF files The dxfpreamble command writes out a header file which sets units to metric and defines the file version and sets up and ends SECTIONs as necessary.

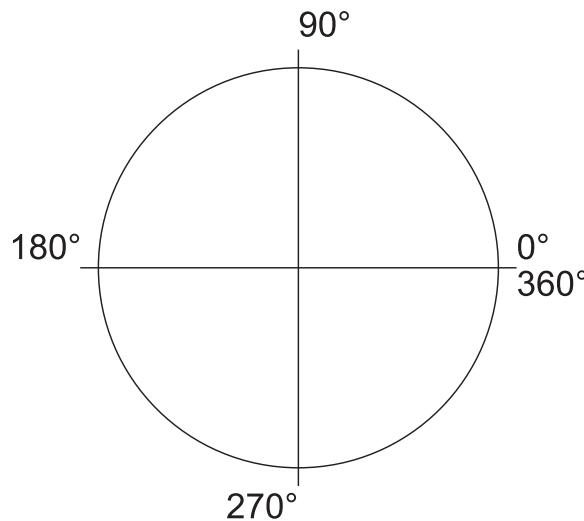
```

1561 gcpy      def dxfpreamble(self):
1562 gcpy          self.writedxf("0")
1563 gcpy          self.writedxf("SECTION")
1564 gcpy          self.writedxf("2")
1565 gcpy          self.writedxf("HEADER")
1566 gcpy          self.writedxf("9")
1567 gcpy          self.writedxf("$INSUNITS")
1568 gcpy          self.writedxf("70")
1569 gcpy          self.writedxf("4")
1570 gcpy          self.writedxf("9")
1571 gcpy          self.writedxf("$ACADVER")
1572 gcpy          self.writedxf("1")
1573 gcpy          self.writedxf("AC1014")
1574 gcpy          self.writedxf("0")
1575 gcpy          self.writedxf("ENDSEC")
1576 gcpy          self.writedxf("0")
1577 gcpy          self.writedxf("SECTION")
1578 gcpy          self.writedxf("2")
1579 gcpy          self.writedxf("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
- addvertex
- closepolyline
- dxfarcs
- 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:

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

dxfcolor Layers and colours are controlled by separate variables: dxfcology and dxflayer respectively.
 dxflayer The latter is initially set to "DEFAULT" and the colour to "Black". Layer names will normally be set to be descriptive (it is possible that this could be done automatically based on toolpath and tool and depth of cut) and to match the name set in the associated Carbide Create template file.

```
1581 gcpy      def setdxfcolor(self, color):
1582 gcpy          self.dxfcolor = color
1583 gcpy          self.cutcolor = color
1584 gcpy
1585 gcpy      def setdxflayer(self, layer):
1586 gcpy          self.dxflayer = layer
1587 gcpy
1588 gcpy      def writedxfc(self):
1589 gcpy          self.writedxf("8")
1590 gcpy          self.writedxf(self.dxflayer)
1591 gcpy
1592 gcpy          self.writedxf("62")
1593 gcpy          if (self.dxfcolor == "Black"):
1594 gcpy              self.writedxf("0")
1595 gcpy          if (self.dxfcolor == "Red"):
1596 gcpy              self.writedxf("1")
1597 gcpy          if (self.dxfcolor == "Yellow"):
```

```

1598 gcpy           self.writedxf("2")
1599 gcpy           if (self.dxfcolor == "Green"):
1600 gcpy             self.writedxf("3")
1601 gcpy           if (self.dxfcolor == "Cyan"):
1602 gcpy             self.writedxf("4")
1603 gcpy           if (self.dxfcolor == "Blue"):
1604 gcpy             self.writedxf("5")
1605 gcpy           if (self.dxfcolor == "Magenta"):
1606 gcpy             self.writedxf("6")
1607 gcpy           if (self.dxfcolor == "White"):
1608 gcpy             self.writedxf("7")
1609 gcpy           if (self.dxfcolor == "DarkGray"):
1610 gcpy             self.writedxf("8")
1611 gcpy           if (self.dxfcolor == "LightGray"):
1612 gcpy             self.writedxf("9")

```

```

158 gpcscad module setdxfcolor(color){
159 gpcscad     gcp.setdxfcolor(color);
160 gpcscad }
161 gpcscad
162 gpcscad module setdxflayer(layer){
163 gpcscad     gcp.setdxflayer(layer);
164 gpcscad }

```

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 or afford an option to do so (Carbide Create does the latter), 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.

Note the addition of variables for adding beginning (zbegin) and ending (zend) variables for setting Z-height — while not convenient to access when using the command directly, it should be workable when using a polyline and addvertex.

```

1614 gcpy   def dxfline(self, xbegin, ybegin, xend, yend, zbegin = 0.0,
1615 gcpy       zend = 0.0):
1616 gcpy       self.writedxf("0")
1617 gcpy       self.writedxf("LINE")
1618 gcpy       self.writedxfcolor()
1619 gcpy       self.writedxf("10")
1620 gcpy       self.writedxf(str(xbegin))
1621 gcpy       self.writedxf("20")
1622 gcpy       self.writedxf(str(ybegin))
1623 gcpy       self.writedxf("30")
1624 gcpy       self.writedxf(str(zbegin))
1625 gcpy       self.writedxf("11")
1626 gcpy       self.writedxf(str(xend))
1627 gcpy       self.writedxf(str(yend))
1628 gcpy       self.writedxf("21")
1629 gcpy       self.writedxf(str(zend))
1630 gcpy       self.writedxf("31")
1631 gcpy       self.writedxf(str(zend))

```

In addition to dxfline which allows creating a line without consideration of context, there is also a dxfpolyline which ideally would create a continuous/joined sequence of line segments — instead, this is used as a shortcut for the dxfline() command, but still requires beginning it, adding vertexes, and then when done, ending the sequence — this implementation simplifies the programming and relies on the software which imports the DXF to join lines where necessary.

Note that the input of a single coordinate makes adding support for the Z-axis and accessing it straight-forward.

First, rather than actually beginning the polyline, store the arguments as values:

```

1633 gcpy   def beginpolyline(self, xbegin, ybegin, zbegin = 0.0):
1634 gcpy       self.bpx = xbegin

```

```
1635 gcpy      self.bpy = ybegin
1636 gcpy      self.bpz = zbegin
```

then add as many vertexes as are wanted (note that it will be necessary to add at least one so as to have any output, however, not having one will only result in the initial position values being discarded):

```
1638 gcpy      def addvertex(self, xend, yend, zend = 0.0):
1639 gcpy          self.dxfline(self.bpx, self.bpy, xend, yend, self.bpz, zend
1640 gcpy          )
1641 gcpy          self.bpx = xend
1642 gcpy          self.bpy = yend
1643 gcpy          self.bpz = zend
```

there is no need to end the sequence since the `dxfline()` command is compleat unto itself. While not strictly necessary, having a `closepolyline()` adds symmetry and affords an option for error messages and so forth.

```
1643 gcpy      def closepolyline(self):
1644 gcpy          self.bpx = 0.0
1645 gcpy          self.bpy = 0.0
```

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).

```
1647 gcpy      def dxfarc(self, xcenter, ycenter, radius, anglebegin, endangle
1648 gcpy          ):
1649 gcpy          if (self.generatedxf == True):
1650 gcpy              self.writedxf("0")
1651 gcpy          #
1652 gcpy          self.writedxfcolor()
1653 gcpy          #
1654 gcpy          self.writedxf("10")
1655 gcpy          self.writedxf(str(xcenter))
1656 gcpy          self.writedxf("20")
1657 gcpy          self.writedxf(str(ycenter))
1658 gcpy          self.writedxf("40")
1659 gcpy          self.writedxf(str(radius))
1660 gcpy          self.writedxf("50")
1661 gcpy          self.writedxf(str(anglebegin))
1662 gcpy          self.writedxf("51")
1663 gcpy          self.writedxf(str(endangle))
```

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.

```
1665 gcpy      def cutarcNECCdxif(self, ex, ey, ez, xcenter, ycenter, radius):
1666 gcpy      global toolpath
1667 gcpy      toolpath = self.currenttool()
1668 gcpy      toolpath = toolpath.translate([self.xpos(), self.ypos(),
1669 gcpy          self.zpos()])
1670 gcpy          self.dxfarc(xcenter, ycenter, radius, 0, 90)
1671 gcpy          if (self.zpos == ez):
1672 gcpy              self.settzpos(0)
1673 gcpy          else:
1674 gcpy              self.settzpos((self.zpos()-ez)/90)
1675 gcpy          self.setxpos(ex)
1676 gcpy          self.setypos(ey)
1677 gcpy          self.setzpos(ez)
1678 gcpy          if self.generatepaths == True:
1679 gcpy              print("Unioning cutarcNECCdxif toolpath")
1680 gcpy              self.arcloop(1, 90, xcenter, ycenter, radius)
1681 gcpy              self.toolpaths = self.toolpaths.union(toolpath)
1682 gcpy          else:
1683 gcpy              toolpath = self.arcloop(1, 90, xcenter, ycenter,
1684 gcpy          radius)
1685 gcpy          print("Returning cutarcNECCdxif toolpath")
1686 gcpy      return toolpath
1687 gcpy      def cutarcNWCCdxif(self, ex, ey, ez, xcenter, ycenter, radius):
1688 gcpy      global toolpath
1689 gcpy      toolpath = self.currenttool()
```

```

                self.zpos()])
1690 gcpy      self.dxfarc(xcenter, ycenter, radius, 90, 180)
1691 gcpy      if (self.zpos == ez):
1692 gcpy          self.setzpos(0)
1693 gcpy      else:
1694 gcpy          self.setzpos((self.zpos()-ez)/90)
1695 gcpy #     self.setxpos(ex)
1696 gcpy #     self.setypos(ey)
1697 gcpy #
1698 gcpy #
1699 gcpy #
1700 gcpy #
1701 gcpy #
1702 gcpy
1703 gcpy
1704 gcpy
1705 gcpy     def cutarcSWCCdxdf(self, ex, ey, ez, xcenter, ycenter, radius):
1706 gcpy #
1707 gcpy #
1708 gcpy #
1709 gcpy     global toolpath
1710 gcpy     toolpath = self.currenttool()
1711 gcpy     toolpath = toolpath.translate([self.xpos(), self.ypos(),
1712 gcpy     self.zpos()])
1713 gcpy     self.dxfarc(xcenter, ycenter, radius, 180, 270)
1714 gcpy #
1715 gcpy #
1716 gcpy #
1717 gcpy     if self.generatepaths == True:
1718 gcpy         self.arcloop(91, 180, xcenter, ycenter, radius)
1719 gcpy #
1720 gcpy     self.toolpaths = self.toolpaths.union(toolpath)
1721 gcpy     else:
1722 gcpy         toolpath = self.arcloop(91, 180, xcenter, ycenter,
1723 gcpy         radius)
1724 gcpy     return toolpath
1725 gcpy #
1726 gcpy #
1727 gcpy #
1728 gcpy     def cutarcSECCdxdf(self, ex, ey, ez, xcenter, ycenter, radius):
1729 gcpy     global toolpath
1730 gcpy     toolpath = self.currenttool()
1731 gcpy     toolpath = toolpath.translate([self.xpos(), self.ypos(),
1732 gcpy     self.zpos()])
1733 gcpy     self.dxfarc(xcenter, ycenter, radius, 270, 360)
1734 gcpy #
1735 gcpy #
1736 gcpy     if self.generatepaths == True:
1737 gcpy         self.arcloop(181, 270, xcenter, ycenter, radius)
1738 gcpy #
1739 gcpy     self.toolpaths = self.toolpaths.union(toolpath)
1740 gcpy     else:
1741 gcpy         toolpath = self.arcloop(181, 270, xcenter, ycenter,
1742 gcpy         radius)
1743 gcpy     return toolpath
1744 gcpy #
1745 gcpy #
1746 gcpy #
1747 gcpy     def cutarcNECWdxdf(self, ex, ey, ez, xcenter, ycenter, radius):
1748 gcpy     global toolpath
1749 gcpy     toolpath = self.currenttool()
1750 gcpy     toolpath = toolpath.translate([self.xpos(), self.ypos(),
1751 gcpy     self.zpos()])
1752 gcpy     self.dxfarc(xcenter, ycenter, radius, 0, 90)
1753 gcpy #
1754 gcpy #
1755 gcpy #
1756 gcpy #
1757 gcpy #
1758 gcpy #
1759 gcpy #
1760 gcpy

```

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

```

cutarcNECCdx(-stockXwidth/4, stockYheight/4+stockYheight/16, -stockZthickness, -stockXwidth/4, stockYheight/4+stockYheight/16)
cutarcNWCCdx(-(stockXwidth/4+stockYheight/16), stockYheight/4, -stockZthickness, -stockXwidth/4, stockYheight/4+stockYheight/16)
cutarcSWCCdx(-stockXwidth/4, stockYheight/4-stockYheight/16, -stockZthickness, -stockXwidth/4, stockYheight/4)
cutarcSECCdx(-(stockXwidth/4-stockYheight/16), stockYheight/4, -stockZthickness, -stockXwidth/4, stockYheight/4)

```

```
1819 gcpy     def arcCCgc(self, ex, ey, ez, xcenter, ycenter, radius):
1820 gcpy         self.writegc("G03\u2022X", str(ex), "\u2022Y", str(ey), "\u2022Z", str(ez)
1821 gcpy             , "\u2022R", str(radius))
1822 gcpy     def arcCWgc(self, ex, ey, ez, xcenter, ycenter, radius):
1823 gcpy         self.writegc("G02\u2022X", str(ex), "\u2022Y", str(ey), "\u2022Z", str(ez)
1824 gcpy             , "\u2022R", str(radius))
```

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

```

1825 gcpy      def cutarcNECCdxfgc(self, ex, ey, ez, xcenter, ycenter, radius):
1826 gcpy          :
1827 gcpy          self.arcCCgc(ex, ey, ez, xcenter, ycenter, radius)
1828 gcpy          if self.generatepaths == True:
1829 gcpy              self.cutarcNECCdxf(ex, ey, ez, xcenter, ycenter, radius)
1830 gcpy          )
1831 gcpy          else:
1832 gcpy              return self.cutarcNECCdxf(ex, ey, ez, xcenter, ycenter,
1833 gcpy                  radius)
1834 gcpy
1835 gcpy      def cutarcNWCCdxfgc(self, ex, ey, ez, xcenter, ycenter, radius):
1836 gcpy          :
1837 gcpy          self.arcCCgc(ex, ey, ez, xcenter, ycenter, radius)
1838 gcpy          if self.generatepaths == False:
1839 gcpy              return self.cutarcNWCCdxf(ex, ey, ez, xcenter, ycenter,
1840 gcpy                  radius)
1841 gcpy
1842 gcpy      def cutarcSWCCdxfgc(self, ex, ey, ez, xcenter, ycenter, radius):
1843 gcpy          :
1844 gcpy          self.arcCCgc(ex, ey, ez, xcenter, ycenter, radius)
1845 gcpy          if self.generatepaths == False:
1846 gcpy              return self.cutarcSWCCdxf(ex, ey, ez, xcenter, ycenter,
1847 gcpy                  radius)
1848 gcpy
1849 gcpy      def cutarcNECWdxfgc(self, ex, ey, ez, xcenter, ycenter, radius):
1850 gcpy          :
1851 gcpy          self.arcCWgc(ex, ey, ez, xcenter, ycenter, radius)
1852 gcpy          if self.generatepaths == False:
1853 gcpy              return self.cutarcNECWdxf(ex, ey, ez, xcenter, ycenter,
1854 gcpy                  radius)
1855 gcpy
1856 gcpy
1857 gcpy      def cutarcSECWdxfgc(self, ex, ey, ez, xcenter, ycenter, radius):
1858 gcpy          :
1859 gcpy          self.arcCWgc(ex, ey, ez, xcenter, ycenter, radius)
1860 gcpy          if self.generatepaths == False:
1861 gcpy              return self.cutarcSECWdxf(ex, ey, ez, xcenter, ycenter,
1862 gcpy                  radius)
1863 gcpy
1864 gcpy      def cutarcNWCWdxfgc(self, ex, ey, ez, xcenter, ycenter, radius):
1865 gcpy          :
1866 gcpy          self.arcCWgc(ex, ey, ez, xcenter, ycenter, radius)
1867 gcpy          if self.generatepaths == False:
1868 gcpy              return self.cutarcNWCWdxf(ex, ey, ez, xcenter, ycenter,
1869 gcpy                  radius)
1870 gcpscad module cutarcNECCdxfgc(ex, ey, ez, xcenter, ycenter, radius){
1871 gcpscad     gcp.cutarcNECCdxfgc(ex, ey, ez, xcenter, ycenter, radius);
1872 gcpscad }
1873 gcpscad
1874 gcpscad module cutarcNWCCdxfgc(ex, ey, ez, xcenter, ycenter, radius){
1875 gcpscad     gcp.cutarcNWCCdxfgc(ex, ey, ez, xcenter, ycenter, radius);
1876 gcpscad }
1877 gcpscad
1878 gcpscad module cutarcSWCCdxfgc(ex, ey, ez, xcenter, ycenter, radius){
1879 gcpscad     gcp.cutarcSWCCdxfgc(ex, ey, ez, xcenter, ycenter, radius);
1880 gcpscad }
```

```

166 gcpscad module cutarcNECCdxfgc(ex, ey, ez, xcenter, ycenter, radius){
167 gcpscad     gcp.cutarcNECCdxfgc(ex, ey, ez, xcenter, ycenter, radius);
168 gcpscad }
169 gcpscad
170 gcpscad module cutarcNWCCdxfgc(ex, ey, ez, xcenter, ycenter, radius){
171 gcpscad     gcp.cutarcNWCCdxfgc(ex, ey, ez, xcenter, ycenter, radius);
172 gcpscad }
173 gcpscad
174 gcpscad module cutarcSWCCdxfgc(ex, ey, ez, xcenter, ycenter, radius){
175 gcpscad     gcp.cutarcSWCCdxfgc(ex, ey, ez, xcenter, ycenter, radius);
176 gcpscad }
177 gcpscad
178 gcpscad module cutarcSECCdxfgc(ex, ey, ez, xcenter, ycenter, radius){
179 gcpscad     gcp.cutarcSECCdxfgc(ex, ey, ez, xcenter, ycenter, radius);
180 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) GOXOYO 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)	writecomment(...)
GOX0.000Y-152.400 Z0.250	rapid(...) rapid(...)
G1Z-1.000F203.2 X109.500Y-77.400F508.0 X57.918Y16.302Z-0.726 Y22.023Z-1.023 X61.190Z-0.681 Y21.643 X57.681 Z12.700	cutwithfeed(...); cutwithfeed(...);
M05 M02	closegcodefile();

The implication here is that it should be possible to read in a G-code file, and for each line/command instantiate a matching command so as to create a 3D model/preview of the file. This is addressed by making specialized commands for movement which correspond to the various axis combinations (xyz, xy, xz, yz, x, y, z).

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

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

- onOpen

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

3.7.3.1 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.

```

1867 gcpy      def dxfpostamble(self):
1868 gcpy #       self.writedxf(str(tn))
1869 gcpy       self.writedxf("0")
1870 gcpy       self.writedxf("ENDSEC")
1871 gcpy       self.writedxf("0")
1872 gcpy       self.writedxf("EOF")


---


1874 gcpy      def gcodepostamble(self):
1875 gcpy         if self.generatecut == True:
1876 gcpy             self.writegc("Z12.700")
1877 gcpy             self.writegc("M05")
1878 gcpy             self.writegc("M02")
1879 gcpy         if self.generateprint == True:
1880 gcpy             self.writegc("G92\u00E0")
1881 gcpy             self.writegc("M107\u00u00;uturn\u00off\u00cooling\u00fans")
1882 gcpy             self.writegc("M104\u00S0\u00;uturn\u00off\u00temperature")
1883 gcpy             self.writegc("G28\u00X0\u00;uhome\u00Xaxis")
1884 gcpy             self.writegc("M84\u00u000;udisable\u00motors")


---


1886 gcpy      def closegcodefile(self):
1887 gcpy         if self.generategcode == True:
1888 gcpy             self.gcodepostamble()
1889 gcpy             self.gc.close()
1890 gcpy
1891 gcpy         def closedxfile(self):
1892 gcpy             if self.generatedxf == True:
1893 gcpy                 global dxfclosed
1894 gcpy                 self.dxfpostamble()
1895 gcpy                 self.dxfclosed = True
1896 gcpy                 self.dxf.close()

```

closegcodefile The commands: closegcodefile, and closedxfile are used to close the files at the end of a closedxfile program. For efficiency, each references the command: dxfpostamble which when called provides dxfpostamble the boilerplate needed at the end of their respective files.

```

182 gpcscad module closegcodefile(){
183 gpcscad     gcp.closegcodefile();
184 gpcscad }
185 gpcscad
186 gpcscad module closedxfile(){
187 gpcscad     gcp.closedxfile();
188 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.

3.8.1 Building blocks

The outlines of shapes will be defined using:

- lines — dxfline

- arcs — dxarfarc

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

3.8.1.1 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 (digons and similar shapes)
 - **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 (triangles and other 3 pointed figures)
 - **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 sector⁷
 - two convex, one concave arc
- 4 (quadrilaterals and so forth)
 - **rectangle (including square)** — dxfractangle, dxfractangleround
 - **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 dxfractangleround.

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

⁷en.wikipedia.org/wiki/Hyperbolic_sector and www.reddit.com/r/Geometry/comments/bkbzgh/is_there_a_name_for_a_3_pointed_figure_with_two/

- 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 asteroid.) 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., dxrectangleround
- 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?

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

```

1898 gcpy      def dxfcircle(self, xcenter, ycenter, radius):
1899 gcpy          self.dxfarc(xcenter, ycenter, radius, 0, 90)
1900 gcpy          self.dxfarc(xcenter, ycenter, radius, 90, 180)
1901 gcpy          self.dxfarc(xcenter, ycenter, radius, 180, 270)
1902 gcpy          self.dxfarc(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.

```

1904 gcpy      def cutcircleCC(self, xcenter, ycenter, bz, ez, radius):
1905 gcpy          self.setzpos(bz)
1906 gcpy          self.cutquarterCCNE(xcenter, ycenter + radius, self.zpos()
1907 gcpy                  + ez/4, radius)
1908 gcpy          self.cutquarterCCNW(xcenter - radius, ycenter, self.zpos()
1909 gcpy                  + ez/4, radius)
1910 gcpy          self.cutquarterCCSW(xcenter, ycenter - radius, self.zpos()
1911 gcpy                  + ez/4, radius)
1912 gcpy          self.cutquarterCCSE(xcenter + radius, ycenter, self.zpos()
1913 gcpy                  + ez/4, radius)

```

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

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

```

1915 gcpy     def dxfractangle(self, xorigin, yorigin, xwidth, yheight,
1916 gcpy         corners = "Square", radius = 6):
1917 gcpy             if corners == "Square":
1918 gcpy                 self.dxfline(xorigin, yorigin, xorigin + xwidth,
1919 gcpy                     yorigin)
1920 gcpy                     self.dxfline(xorigin + xwidth, yorigin, xorigin +
1921 gcpy                         xwidth, yorigin + yheight)
1922 gcpy                     self.dxfline(xorigin + xwidth, yorigin + yheight,
1923 gcpy                         xorigin, yorigin + yheight)
1924 gcpy                     self.dxfline(xorigin, yorigin + yheight, xorigin,
1925 gcpy                         yorigin)
1926 gcpy             elif corners == "Fillet":
1927 gcpy                 self.dxfractangleround(xorigin, yorigin, xwidth,
1928 gcpy                     yheight, radius)
1929 gcpy             elif corners == "Chamfer":
1930 gcpy                 self.dxfractanglechamfer(xorigin, yorigin, xwidth,
1931 gcpy                     yheight, radius)
1932 gcpy             elif corners == "Flipped Fillet":
1933 gcpy                 self.dxfractangleflippedfillet(xorigin, yorigin, xwidth
1934 gcpy                     , yheight, radius)

```

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

```

1928 gcpy     def dxfractangleround(self, xorigin, yorigin, xwidth, yheight,
1929 gcpy         radius):
1930 gcpy             self.dxfarc(xorigin + radius, yorigin + radius, radius,
1931 gcpy                 180, 270)
1932 gcpy             self.dxfarc(xorigin + xwidth - radius, yorigin + radius,
1933 gcpy                 radius, 270, 360)
1934 gcpy             self.dxfarc(xorigin + xwidth - radius, yorigin + yheight -
1935 gcpy                 radius, radius, 0, 90)
1936 gcpy             self.dxfarc(xorigin + radius, yorigin + yheight - radius,
1937 gcpy                 radius, 90, 180)
1938 gcpy             self.dxfline(xorigin + radius, yorigin, xorigin + xwidth -
1939 gcpy                 radius, yorigin)
1940 gcpy             self.dxfline(xorigin + xwidth, yorigin + radius, xorigin +
1941 gcpy                 xwidth, yorigin + yheight - radius)
1942 gcpy             self.dxfline(xorigin + xwidth - radius, yorigin + yheight,
1943 gcpy                 xorigin + radius, yorigin + yheight)
1944 gcpy             self.dxfline(xorigin, yorigin + yheight - radius, xorigin,
1945 gcpy                 yorigin + radius)

```

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

```

2009 gcpy     def dxfractanglechamfer(self, tool_num, xorigin, yorigin,
2010 gcpy         xwidth, yheight, radius):
2011 gcpy             self.dxfline(tool_num, xorigin + radius, yorigin, xorigin,
2012 gcpy                 yorigin + radius)
2013 gcpy             self.dxfline(tool_num, xorigin, yorigin + yheight - radius,
2014 gcpy                 xorigin + radius, yorigin + yheight)
2015 gcpy             self.dxfline(tool_num, xorigin + xwidth - radius, yorigin +
2016 gcpy                 yheight, xorigin + xwidth, yorigin + yheight - radius)
2017 gcpy             self.dxfline(tool_num, xorigin + xwidth - radius, yorigin +
2018 gcpy                 yheight, xorigin + radius, yorigin + yheight)
2019 gcpy             self.dxfline(tool_num, xorigin, yorigin + yheight - radius,
2020 gcpy                 xorigin, yorigin + radius)

```

Flipped Fillet:

```

2020 gcpy     def dxfractangleflippedfillet(self, xorigin, yorigin, xwidth,
2021 gcpy         yheight, radius):
2022 gcpy             self.dxfarc(xorigin, yorigin, radius, 0, 90)
2023 gcpy             self.dxfarc(xorigin + xwidth, yorigin, radius, 90, 180)
2024 gcpy             self.dxfarc(xorigin + xwidth, yorigin + yheight, radius,
2025 gcpy                 90, 180)

```

```

    180, 270)
2024 gcpy self.dxfarc(xorigin, yorigin + yheight, radius, 270, 360)
2025 gcpy
2026 gcpy self.dxfline(xorigin + radius, yorigin, xorigin + xwidth -
    radius, yorigin)
2027 gcpy self.dxfline(xorigin + xwidth, yorigin + radius, xorigin +
    xwidth, yorigin + yheight - radius)
2028 gcpy self.dxfline(xorigin + xwidth - radius, yorigin + yheight,
    xorigin + radius, yorigin + yheight)
2029 gcpy self.dxfline(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.

```

2031 gcpy     def cutrectangle(self, bx, by, bz, xwidth, yheight, zdepth):
2032 gcpy         self.cutline(bx, by, bz)
2033 gcpy         self.cutline(bx, by, bz - zdepth)
2034 gcpy         self.cutline(bx + xwidth, by, bz - zdepth)
2035 gcpy         self.cutline(bx + xwidth, by + yheight, bz - zdepth)
2036 gcpy         self.cutline(bx, by + yheight, bz - zdepth)
2037 gcpy         self.cutline(bx, by, bz - zdepth)
2038 gcpy
2039 gcpy     def cutrectangledxf(self, bx, by, bz, xwidth, yheight, zdepth):
2040 gcpy         self.cutrectangle(bx, by, bz, xwidth, yheight, zdepth)
2041 gcpy         self.dxfrctangle(bx, by, xwidth, yheight, "Square")

```

The rounded forms instantiate a radius:

```

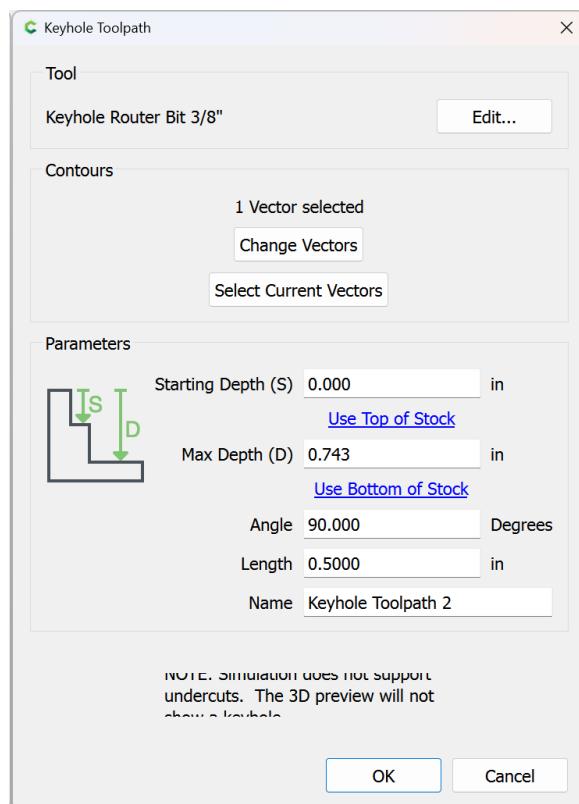
2043 gcpy     def cutrectangleround(self, bx, by, bz, xwidth, yheight, zdepth
    , radius):
2044 gcpy #         self.rapid(bx + radius, by, bz)
2045 gcpy         self.cutline(bx + radius, by, bz + zdepth)
2046 gcpy         self.cutline(bx + xwidth - radius, by, bz + zdepth)
2047 gcpy         self.cutquarterCCSE(bx + xwidth, by + radius, bz + zdepth,
    radius)
2048 gcpy         self.cutline(bx + xwidth, by + yheight - radius, bz +
    zdepth)
2049 gcpy         self.cutquarterCCNE(bx + xwidth - radius, by + yheight, bz
    + zdepth, radius)
2050 gcpy         self.cutline(bx + radius, by + yheight, bz + zdepth)
2051 gcpy         self.cutquarterCCNW(bx, by + yheight - radius, bz + zdepth,
    radius)
2052 gcpy         self.cutline(bx, by + radius, bz + zdepth)
2053 gcpy         self.cutquarterCCSW(bx + radius, by, bz + zdepth, radius)
2054 gcpy
2055 gcpy     def cutrectangleroundddxf(self, bx, by, bz, xwidth, yheight,
    zdepth, radius):
2056 gcpy         self.cutrectangleround(bx, by, bz, xwidth, yheight, zdepth,
    radius)
2057 gcpy         self.dxfrctangleround(bx, by, xwidth, yheight, radius)

```

3.8.1.1.3 Keyhole toolpath and undercut tooling The first topologically unusual toolpath is `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 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 [3.5.1](#)

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:

```

2059 gcpy      def cutkeyholegcdxf(self, kh_tool_num, kh_start_depth,
2060 gcpy          kh_max_depth, kht_direction, kh_distance):
2061 gcpy          if (kht_direction == "N"):
2062 gcpy              toolpath = self.cutKHgcdxf(kh_tool_num, kh_start_depth,
2063 gcpy                  kh_max_depth, 90, kh_distance)
2064 gcpy          elif (kht_direction == "S"):
2065 gcpy              toolpath = self.cutKHgcdxf(kh_tool_num, kh_start_depth,
2066 gcpy                  kh_max_depth, 270, kh_distance)
2067 gcpy          elif (kht_direction == "E"):
2068 gcpy              toolpath = self.cutKHgcdxf(kh_tool_num, kh_start_depth,
2069 gcpy                  kh_max_depth, 0, kh_distance)
2070 gcpy          elif (kht_direction == "W"):
2071 gcpy              toolpath = self.cutKHgcdxf(kh_tool_num, kh_start_depth,
2072 gcpy                  kh_max_depth, 180, kh_distance)
2073 gcpy      if self.generatepaths == True:
2074 gcpy          self.toolpaths = union([self.toolpaths, toolpath])
2075 gcpy      return toolpath
2076 gcpy      else:
2077 gcpy          return cube([0.01, 0.01, 0.01])

```

```

190 gpcscad module cutkeyholegcdxf(kh_tool_num, kh_start_depth, kh_max_depth,
191 gpcscad     kht_direction, kh_distance){
192 gpcscad     gcp.cutkeyholegcdxf(kh_tool_num, kh_start_depth, kh_max_depth,
193 gpcscad         kht_direction, kh_distance);
194 gpcscad }
```

`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).

This command makes use of `self.currenttool()` and will require that the appropriate tool is selected.

The first task is to place a circle at the origin which is invariant of angle:

```

2074 gcpy     def cutKHgcdxf(self, kh_tool_num, kh_start_depth, kh_max_depth,
2075 gcpy         kh_angle, kh_distance):
2076 gcpy         oXpos = self.xpos()
2077 gcpy         oYpos = self.ypos()
2078 gcpy         self.dxfKH(kh_tool_num, self.xpos(), self.ypos(),
2079 gcpy             kh_start_depth, kh_max_depth, kh_angle, kh_distance)
2080 gcpy         toolpath = self.cutline(self.xpos(), self.ypos(), -
2081 gcpy             kh_max_depth)
2082 gcpy         self.setxpos(oXpos)
2083 gcpy         self.setypos(oYpos)
2084 gcpy #       if self.generatepaths == False:
2085 gcpy #           return toolpath
2086 gcpy #       else:
2087 gcpy #           return cube([0.001, 0.001, 0.001])
```

```

2086 gcpy     def dxfKH(self, oXpos, oYpos, kh_start_depth, kh_max_depth,
2087 gcpy         kh_angle, kh_distance):
2088 gcpy #       oXpos = self.xpos()
2089 gcpy #       oYpos = self.ypos()
2090 gcpy #Circle at entry hole
2091 gcpy         self.dxfarc(oXpos, oYpos, self.tool_radius(self.
2092 gcpy             currenttoolnumber(), 7), 0, 90)
2093 gcpy         self.dxfarc(oXpos, oYpos, self.tool_radius(self.
2094 gcpy             currenttoolnumber(), 7), 90, 180)
2095 gcpy         self.dxfarc(oXpos, oYpos, self.tool_radius(self.
2096 gcpy             currenttoolnumber(), 7), 180, 270)
2097 gcpy         self.dxfarc(oXpos, oYpos, self.tool_radius(self.
2098 gcpy             currenttoolnumber(), 7), 270, 360)
```

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

```

2094 gcpy #pre-calculate needed values
2095 gcpy         r = self.tool_radius(self.currenttoolnumber(), 7)
2096 gcpy #       print(r)
2097 gcpy         rt = self.tool_radius(self.currenttoolnumber(), 1)
2098 gcpy #       print(rt)
2099 gcpy         ro = math.sqrt((self.tool_radius(self.currenttoolnumber(),
2100 gcpy             1))**2-(self.tool_radius(self.currenttoolnumber(), 7))
2101 gcpy             **2)
2102 gcpy #Outlines of entry hole and slot
2103 gcpy         if (kh_angle == 0):
2104 gcpy #Lower left of entry hole
2105 gcpy             self.dxfarc(self.xpos(), self.ypos(), self.tool_radius(
2106 gcpy                 self.currenttoolnumber(), 1), 180, 270)
2107 gcpy #Upper left of entry hole
2108 gcpy             self.dxfarc(self.xpos(), self.ypos(), self.tool_radius(
2109 gcpy                 self.currenttoolnumber(), 1), 90, 180)
2110 gcpy #Upper right of entry hole
2111 gcpy             self.dxfarc(self.xpos(), self.ypos(), rt, 41.810, 90)
2112 gcpy #Lower right of entry hole
2113 gcpy             self.dxfarc(self.xpos(), self.ypos(), rt, 270, 360-
2114 gcpy                 angle)
2115 gcpy #       self.dxfarc(self.xpos(), self.ypos(), self.tool_radius(
2116 gcpy                 (self.currenttoolnumber(), 1), 270, 270+math.acos(self.
2117 gcpy                     tool_diameter(self.currenttoolnumber(), 5)/self.tool_diameter(
2118 gcpy                     self.currenttoolnumber(), 1)))
2119 gcpy #Actual line of cut
2120 gcpy             self.dxfline(self.xpos(), self.ypos(), self.xpos()+
2121 gcpy                 kh_distance, self.ypos())
```

```

2116 gcpy #upper right of end of slot (kh_max_depth+4.36))/2
2117 gcpy           self.dxfarc(self.xpos()+kh_distance, self.ypos(), self.
                           tool_diameter(self.currenttoolnumber(), (
                           kh_max_depth+4.36))/2, 0, 90)
2118 gcpy #lower right of end of slot
2119 gcpy           self.dxfarc(self.xpos()+kh_distance, self.ypos(), self.
                           tool_diameter(self.currenttoolnumber(), (
                           kh_max_depth+4.36))/2, 270, 360)
2120 gcpy #upper right slot
2121 gcpy           self.dxfline(self.xpos()+ro, self.ypos()-(self.
                           tool_diameter(self.currenttoolnumber(), 7)/2), self.
                           xpos()+kh_distance, self.ypos()-(self.tool_diameter(
                           self.currenttoolnumber(), 7)/2))
2122 gcpy #
           self.dxfline(self.xpos())+(math.sqrt((self.
                           tool_diameter(self.currenttoolnumber(), 1)^2)-(self.
                           tool_diameter(self.currenttoolnumber(), 5)^2))/2, self.ypos()+
                           self.tool_diameter(self.currenttoolnumber(), (kh_max_depth))/2,
                           ((kh_max_depth-6.34))/2)^2-(self.tool_diameter(self.
                           currenttoolnumber(), (kh_max_depth-6.34))/2)^2, self.xpos()+
                           kh_distance, self.ypos()+(self.tool_diameter(self.
                           currenttoolnumber(), (kh_max_depth))/2, self.currenttoolnumber())
)
2123 gcpy #end position at top of slot
2124 gcpy #lower right slot
2125 gcpy           self.dxfline(self.xpos()+ro, self.ypos()+(self.
                           tool_diameter(self.currenttoolnumber(), 7)/2), self.
                           xpos()+kh_distance, self.ypos()+(self.tool_diameter(
                           self.currenttoolnumber(), 7)/2))
2126 gcpy #
           dxfline(self.xpos())+(math.sqrt((self.tool_diameter(self.
                           currenttoolnumber(), 1)^2)-(self.tool_diameter(self.
                           currenttoolnumber(), 5)^2))/2, self.ypos()-self.tool_diameter(
                           self.currenttoolnumber(), (kh_max_depth))/2, ((kh_max_depth-
                           6.34))/2)^2-(self.tool_diameter(self.currenttoolnumber(), (
                           kh_max_depth-6.34))/2)^2, self.xpos()+kh_distance, self.ypos()-
                           self.tool_diameter(self.currenttoolnumber(), (kh_max_depth))/2,
                           self.currenttoolnumber())
2127 gcpy #end position at top of slot
2128 gcpy #   hull(){
2129 gcpy #     translate([xpos(), ypos(), zpos()]){
2130 gcpy #       keyhole_shaft(6.35, 9.525);
2131 gcpy #     }
2132 gcpy #     translate([xpos(), ypos(), zpos()-kh_max_depth]){
2133 gcpy #       keyhole_shaft(6.35, 9.525);
2134 gcpy #     }
2135 gcpy #
2136 gcpy #   hull(){
2137 gcpy #     translate([xpos(), ypos(), zpos()-kh_max_depth]){
2138 gcpy #       keyhole_shaft(6.35, 9.525);
2139 gcpy #
2140 gcpy #     translate([xpos()+kh_distance, ypos(), zpos()-kh_max_depth])
{
2141 gcpy #       keyhole_shaft(6.35, 9.525);
2142 gcpy #
2143 gcpy #
2144 gcpy #     cutwithfeed(getxpos(), getypos(), -kh_max_depth, feed);
2145 gcpy #     cutwithfeed(getxpos()+kh_distance, getypos(), -kh_max_depth,
                           feed);
2146 gcpy #     setxpos(getxpos()-kh_distance);
2147 gcpy #   } else if (kh_angle > 0 && kh_angle < 90) {
2148 gcpy //echo(kh_angle);
2149 gcpy #   dxfarc(getxpos(), getypos(), tool_diameter(self.
                           currenttoolnumber(), (kh_max_depth))/2, 90+kh_angle, 180+
                           kh_angle, self.currenttoolnumber());
2150 gcpy #   dxfarc(getxpos(), getypos(), tool_diameter(self.
                           currenttoolnumber(), (kh_max_depth))/2, 180+kh_angle, 270+
                           kh_angle, self.currenttoolnumber());
2151 gcpy #dxfarc(getxpos(), getypos(), tool_diameter(self.currenttoolnumber
                           (), (kh_max_depth))/2, kh_angle+asin((tool_diameter(self.
                           currenttoolnumber(), (kh_max_depth+4.36))/2)/(tool_diameter(self.
                           .currenttoolnumber(), (kh_max_depth))/2)), 90+kh_angle, self.
                           currenttoolnumber());
2152 gcpy #dxfarc(getxpos(), getypos(), tool_diameter(self.currenttoolnumber
                           (), (kh_max_depth))/2, 270+kh_angle, 360+kh_angle-asin((
                           tool_diameter(self.currenttoolnumber(), (kh_max_depth+4.36))/2)/
                           (tool_diameter(self.currenttoolnumber(), (kh_max_depth))/2)),
                           self.currenttoolnumber());
2153 gcpy #dxfarc(getxpos()+(kh_distance*cos(kh_angle)),
2154 gcpy #   getypos()+(kh_distance*sin(kh_angle)), tool_diameter(self.

```

```

        currenttoolnumber(), (kh_max_depth+4.36))/2, 0+kh_angle, 90+
        kh_angle, self.currenttoolnumber());
2155 gcpy #dxfarl(getxpos()+(kh_distance*cos(kh_angle)), getypos()+(

        kh_distance*sin(kh_angle)), tool_diameter(self.currenttoolnumber()
        (), (kh_max_depth+4.36))/2, 270+kh_angle, 360+kh_angle, self.

        currenttoolnumber());
2156 gcpy #dxfline( getxpos() + tool_diameter(self.currenttoolnumber()), (
        kh_max_depth)/2*cos(kh_angle)+asin((tool_diameter(self.

        currenttoolnumber(), (kh_max_depth+4.36))/2)/(tool_diameter(self.

        .currenttoolnumber(), (kh_max_depth))/2))),
2157 gcpy # getypos() + tool_diameter(self.currenttoolnumber(), (kh_max_depth))

        /2*sin(kh_angle)+asin((tool_diameter(self.currenttoolnumber(), (

        kh_max_depth+4.36))/2)/(tool_diameter(self.currenttoolnumber(),

        (kh_max_depth))/2))),
2158 gcpy # getxpos() +(kh_distance*cos(kh_angle))-((tool_diameter(self.

        currenttoolnumber(), (kh_max_depth+4.36))/2)*sin(kh_angle)),
2159 gcpy # getypos() +(kh_distance*sin(kh_angle))+((tool_diameter(self.

        currenttoolnumber(), (kh_max_depth+4.36))/2)*cos(kh_angle)),
        self.currenttoolnumber());
2160 gcpy #//echo("a", tool_diameter(self.currenttoolnumber(), (kh_max_depth

        +4.36))/2);
2161 gcpy #//echo("c", tool_diameter(self.currenttoolnumber(), (kh_max_depth)

        )/2);
2162 gcpy #echo("Aangle", asin((tool_diameter(self.currenttoolnumber(), (


        kh_max_depth+4.36))/2)/(tool_diameter(self.currenttoolnumber(),

        (kh_max_depth))/2)));
2163 gcpy #//echo(kh_angle);
2164 gcpy # cutwithfeed(getxpos()+(kh_distance*cos(kh_angle)), getypos()+(

        kh_distance*sin(kh_angle)), -kh_max_depth, feed);
2165 gcpy #           toolpath = toolpath.union(self.cutline(self.xpos() +

        kh_distance, self.ypos(), -kh_max_depth))
2166 gcpy     elif (kh_angle == 90):
2167 gcpy #Lower left of entry hole
2168 gcpy           self.dxfarc(oXpos, oYpos, self.tool_radius(self.

        currenttoolnumber(), 1), 180, 270)
2169 gcpy #Lower right of entry hole
2170 gcpy           self.dxfarc(oXpos, oYpos, self.tool_radius(self.

        currenttoolnumber(), 1), 270, 360)
2171 gcpy #left slot
2172 gcpy           self.dxfline(oXpos-r, oYpos+ro, oXpos-r, oYpos+

        kh_distance)
2173 gcpy #right slot
2174 gcpy           self.dxfline(oXpos+r, oYpos+ro, oXpos+r, oYpos+


        kh_distance)
2175 gcpy #upper left of end of slot
2176 gcpy           self.dxfarc(oXpos, oYpos+kh_distance, r, 90, 180)
2177 gcpy #upper right of end of slot
2178 gcpy           self.dxfarc(oXpos, oYpos+kh_distance, r, 0, 90)
2179 gcpy #Upper right of entry hole
2180 gcpy           self.dxfarc(oXpos, oYpos, rt, 0, 90-angle)
2181 gcpy #Upper left of entry hole
2182 gcpy           self.dxfarc(oXpos, oYpos, rt, 90+angle, 180)
2183 gcpy #           toolpath = toolpath.union(self.cutline(oXpos, oYpos+


        kh_distance, -kh_max_depth))
2184 gcpy     elif (kh_angle == 180):
2185 gcpy #Lower right of entry hole
2186 gcpy           self.dxfarc(oXpos, oYpos, self.tool_radius(self.

        currenttoolnumber(), 1), 270, 360)
2187 gcpy #Upper right of entry hole
2188 gcpy           self.dxfarc(oXpos, oYpos, self.tool_radius(self.

        currenttoolnumber(), 1), 0, 90)
2189 gcpy #Upper left of entry hole
2190 gcpy           self.dxfarc(oXpos, oYpos, rt, 90, 180-angle)
2191 gcpy #Lower left of entry hole
2192 gcpy           self.dxfarc(oXpos, oYpos, rt, 180+angle, 270)
2193 gcpy #upper slot
2194 gcpy           self.dxfline(oXpos-ro, oYpos-r, oXpos-kh_distance,


        oYpos-r)
2195 gcpy #lower slot
2196 gcpy           self.dxfline(oXpos-ro, oYpos+r, oXpos-kh_distance,


        oYpos+r)
2197 gcpy #upper left of end of slot
2198 gcpy           self.dxfarc(oXpos-kh_distance, oYpos, r, 90, 180)
2199 gcpy #lower left of end of slot
2200 gcpy           self.dxfarc(oXpos-kh_distance, oYpos, r, 180, 270)
2201 gcpy #           toolpath = toolpath.union(self.cutline(oXpos-


        kh_distance, oYpos, -kh_max_depth))
2202 gcpy     elif (kh_angle == 270):

```

```

2203 gcpy #Upper left of entry hole
2204 gcpy           self.dxfarc(oXpos, oYpos, self.tool_radius(self.
                           currenttoolnumber(), 1), 90, 180)
2205 gcpy #Upper right of entry hole
2206 gcpy           self.dxfarc(oXpos, oYpos, self.tool_radius(self.
                           currenttoolnumber(), 1), 0, 90)
2207 gcpy #left slot
2208 gcpy           self.dxfline(oXpos-r, oYpos-ro, oXpos-r, oYpos-
                           kh_distance)
2209 gcpy #right slot
2210 gcpy           self.dxfline(oXpos+r, oYpos-ro, oXpos+r, oYpos-
                           kh_distance)
2211 gcpy #lower left of end of slot
2212 gcpy           self.dxfarc(oXpos, oYpos-kh_distance, r, 180, 270)
2213 gcpy #lower right of end of slot
2214 gcpy           self.dxfarc(oXpos, oYpos-kh_distance, r, 270, 360)
2215 gcpy #lower right of entry hole
2216 gcpy           self.dxfarc(oXpos, oYpos, rt, 180, 270-angle)
2217 gcpy #lower left of entry hole
2218 gcpy           self.dxfarc(oXpos, oYpos, rt, 270+angle, 360)
2219 gcpy #
           toolpath = toolpath.union(self.cutline(oXpos, oYpos-
                           kh_distance, -kh_max_depth))
2220 gcpy #
           print(self.zpos())
2221 gcpy #
           self.setxpos(oXpos)
2222 gcpy #
           self.setypos(oYpos)
2223 gcpy #
           if self.generatepaths == False:
2224 gcpy #
           return toolpath
2225 gcpy
2226 gcpy # } else if (kh_angle == 90) {
2227 gcpy #   //Lower left of entry hole
2228 gcpy #   dxfarc(getxpos(), getypos(), 9.525/2, 180, 270, self.
                           currenttoolnumber());
2229 gcpy #   //Lower right of entry hole
2230 gcpy #   dxfarc(getxpos(), getypos(), 9.525/2, 270, 360, self.
                           currenttoolnumber());
2231 gcpy #   //Upper right of entry hole
2232 gcpy #   dxfarc(getxpos(), getypos(), 9.525/2, 0, acos(tool_diameter(
                           self.currenttoolnumber(), 5)/tool_diameter(self.
                           currenttoolnumber(), 1)), self.currenttoolnumber());
2233 gcpy #   //Upper left of entry hole
2234 gcpy #   dxfarc(getxpos(), getypos(), 9.525/2, 180-acos(tool_diameter(
                           self.currenttoolnumber(), 5)/tool_diameter(self.
                           currenttoolnumber(), 1)), 180, self.currenttoolnumber());
2235 gcpy #   //Actual line of cut
2236 gcpy #   dxfline(getxpos(), getypos(), getxpos(), getypos()+kh_distance
                           );
2237 gcpy #   //upper right of slot
2238 gcpy #   dxfarc(getxpos(), getypos()+kh_distance, tool_diameter(self.
                           currenttoolnumber(), (kh_max_depth+4.36))/2, 0, 90, self.
                           currenttoolnumber());
2239 gcpy #   //upper left of slot
2240 gcpy #   dxfarc(getxpos(), getypos()+kh_distance, tool_diameter(self.
                           currenttoolnumber(), (kh_max_depth+6.35))/2, 90, 180, self.
                           currenttoolnumber());
2241 gcpy #   //right of slot
2242 gcpy #
           dxfline(
2243 gcpy #           getxpos()+tool_diameter(self.currenttoolnumber(), (
                           kh_max_depth))/2,
2244 gcpy #           getypos()+(math.sqrt((tool_diameter(self.currenttoolnumber
                           (), 1)^2)-(tool_diameter(self.currenttoolnumber(), 5)^2))/2,
                           //((kh_max_depth-6.34))/2)^2-(tool_diameter(self.
                           currenttoolnumber(), (kh_max_depth-6.34))/2)^2,
2245 gcpy #           getxpos()+tool_diameter(self.currenttoolnumber(), (
                           kh_max_depth))/2,
2246 gcpy #           //end position at top of slot
2247 gcpy #           getypos()+kh_distance,
2248 gcpy #
           self.currenttoolnumber());
2249 gcpy #
           dxfline(getxpos()-tool_diameter(self.currenttoolnumber(), (
                           kh_max_depth))/2, getypos()+(math.sqrt((tool_diameter(self.
                           currenttoolnumber(), 1)^2)-(tool_diameter(self.currenttoolnumber
                           (), 5)^2))/2), getxpos()-tool_diameter(self.currenttoolnumber(),
                           (kh_max_depth+6.35))/2, getypos()+kh_distance, self.
                           currenttoolnumber());
2250 gcpy #
           hull(){
2251 gcpy #
           translate([xpos(), ypos(), zpos()]){
2252 gcpy #
           keyhole_shaft(6.35, 9.525);
2253 gcpy #
           }
2254 gcpy #
           translate([xpos(), ypos(), zpos()-kh_max_depth]){


```

```

2255 gcpy #      keyhole_shaft(6.35, 9.525);
2256 gcpy #
2257 gcpy #
2258 gcpy #      hull(){
2259 gcpy #          translate([xpos(), ypos(), zpos()-kh_max_depth]){
2260 gcpy #              keyhole_shaft(6.35, 9.525);
2261 gcpy #
2262 gcpy #          translate([xpos(), ypos()+kh_distance, zpos()-kh_max_depth])
2263 gcpy #
2264 gcpy #
2265 gcpy #
2266 gcpy #          cutwithfeed(getxpos(), getypos(), -kh_max_depth, feed);
2267 gcpy #          cutwithfeed(getxpos(), getypos()+kh_distance, -kh_max_depth,
2268 gcpy #              feed);
2269 gcpy #          setypos(getypos()-kh_distance);
2270 gcpy #      } else if (kh_angle == 180) {
2271 gcpy #          //Lower right of entry hole
2272 gcpy #          dxfarc(getxpos(), getypos(), 9.525/2, 270, 360, self.
2273 gcpy #              currenttoolnumber());
2274 gcpy #          //Upper right of entry hole
2275 gcpy #          dxfarc(getxpos(), getypos(), 9.525/2, 0, 90, self.
2276 gcpy #              currenttoolnumber());
2277 gcpy #          //Upper left of entry hole
2278 gcpy #          dxfarc(getxpos(), getypos(), 9.525/2, 90, 90+acos(
2279 gcpy #              tool_diameter(self.currenttoolnumber(), 5)/tool_diameter(self.
2280 gcpy #                  currenttoolnumber(), 1)), self.currenttoolnumber());
2281 gcpy #          //Lower left of entry hole
2282 gcpy #          dxfarc(getxpos(), getypos(), 9.525/2, 270-acos(tool_diameter(
2283 gcpy #              self.currenttoolnumber(), 5)/tool_diameter(self.
2284 gcpy #                  currenttoolnumber(), 1)), 270, self.currenttoolnumber());
2285 gcpy #          //lower left of slot
2286 gcpy #          dxfarc(getxpos()-kh_distance, getypos(), tool_diameter(self.
2287 gcpy #              currenttoolnumber(), (kh_max_depth+6.35))/2, 90, 180, self.
2288 gcpy #          currenttoolnumber());
2289 gcpy #          //upper left slot
2290 gcpy #          dxfline(
2291 gcpy #              getxpos()-(math.sqrt((tool_diameter(self.currenttoolnumber()
2292 gcpy #                  , 1)^2)-(tool_diameter(self.currenttoolnumber(), 5)^2))/2),
2293 gcpy #              getypos()+tool_diameter(self.currenttoolnumber(), (
2294 gcpy #                  kh_max_depth))/2, //((kh_max_depth-6.34))/2)^2-(tool_diameter(
2295 gcpy #                      self.currenttoolnumber(), (kh_max_depth-6.34))/2)^2,
2296 gcpy #              getxpos()-kh_distance,
2297 gcpy #              //end position at top of slot
2298 gcpy #              getypos()-tool_diameter(self.currenttoolnumber(), (
2299 gcpy #                  kh_max_depth))/2,
2300 gcpy #              self.currenttoolnumber());
2301 gcpy #      hull(){
2302 gcpy #          translate([xpos(), ypos(), zpos()]){
2303 gcpy #              keyhole_shaft(6.35, 9.525);
2304 gcpy #
2305 gcpy #          translate([xpos(), ypos(), zpos()-kh_max_depth]){
2306 gcpy #              keyhole_shaft(6.35, 9.525);
2307 gcpy #
2308 gcpy #          hull(){
2309 gcpy #              translate([xpos(), ypos(), zpos()-kh_max_depth]){
2310 gcpy #                  keyhole_shaft(6.35, 9.525);
2311 gcpy #

```

```

2312 gcpy #      translate([xpos()-kh_distance, ypos(), zpos()-kh_max_depth])
{
    keyhole_shaft(6.35, 9.525);
}
2314 gcpy #
2315 gcpy #
2316 gcpy #      cutwithfeed(getxpos(), getypos(), -kh_max_depth, feed);
2317 gcpy #      cutwithfeed(getxpos()-kh_distance, getypos(), -kh_max_depth,
    feed);
2318 gcpy #      setxpos(getxpos()+kh_distance);
2319 gcpy # } else if (kh_angle == 270) {
2320 gcpy # //Upper right of entry hole
2321 gcpy #      dxfarc(getxpos(), getypos(), 9.525/2, 0, 90, self.
    currenttoolnumber());
2322 gcpy # //Upper left of entry hole
2323 gcpy #      dxfarc(getxpos(), getypos(), 9.525/2, 90, 180, self.
    currenttoolnumber());
2324 gcpy # //lower right of slot
2325 gcpy #      dxfarc(getxpos(), getypos(), -kh_distance, tool_diameter(self.
    currenttoolnumber(), (kh_max_depth+4.36))/2, 270, 360, self.
    currenttoolnumber());
2326 gcpy # //lower left of slot
2327 gcpy #      dxfarc(getxpos(), getypos(), -kh_distance, tool_diameter(self.
    currenttoolnumber(), (kh_max_depth+4.36))/2, 180, 270, self.
    currenttoolnumber());
2328 gcpy # //Actual line of cut
2329 gcpy #      dxfline(getxpos(), getypos(), getxpos(), getypos()-kh_distance
);
2330 gcpy # //right of slot
2331 gcpy #      dxfline(
2332 gcpy #          getxpos()+tool_diameter(self.currenttoolnumber(), (
    kh_max_depth))/2,
2333 gcpy #          getypos()-(math.sqrt((tool_diameter(self.currenttoolnumber
    (), 1)^2)-(tool_diameter(self.currenttoolnumber(), 5)^2))/2),
//((kh_max_depth-6.34))/2)^2-(tool_diameter(self.
    currenttoolnumber(), (kh_max_depth-6.34))/2)^2,
2334 gcpy #          getxpos()+tool_diameter(self.currenttoolnumber(), (
    kh_max_depth))/2,
2335 gcpy # //end position at top of slot
2336 gcpy #          getypos()-kh_distance,
2337 gcpy #          self.currenttoolnumber());
2338 gcpy # //left of slot
2339 gcpy #      dxfline(
2340 gcpy #          getxpos()-tool_diameter(self.currenttoolnumber(), (
    kh_max_depth))/2,
2341 gcpy #          getypos()-(math.sqrt((tool_diameter(self.currenttoolnumber
    (), 1)^2)-(tool_diameter(self.currenttoolnumber(), 5)^2))/2),
//((kh_max_depth-6.34))/2)^2-(tool_diameter(self.
    currenttoolnumber(), (kh_max_depth-6.34))/2)^2,
2342 gcpy #          getxpos()-tool_diameter(self.currenttoolnumber(), (
    kh_max_depth))/2,
2343 gcpy # //end position at top of slot
2344 gcpy #          getypos()-kh_distance,
2345 gcpy #          self.currenttoolnumber());
2346 gcpy # //Lower right of entry hole
2347 gcpy #      dxfarc(getxpos(), getypos(), 9.525/2, 360-acos(tool_diameter(
    self.currenttoolnumber(), 5)/tool_diameter(self.
    currenttoolnumber(), 1)), 360, self.currenttoolnumber());
2348 gcpy # //Lower left of entry hole
2349 gcpy #      dxfarc(getxpos(), getypos(), 9.525/2, 180, 180+acos(
    tool_diameter(self.currenttoolnumber(), 5)/tool_diameter(self.
    currenttoolnumber(), 1)), self.currenttoolnumber());
2350 gcpy #      hull(){
2351 gcpy #          translate([xpos(), ypos(), zpos()]){
2352 gcpy #              keyhole_shaft(6.35, 9.525);
2353 gcpy #          }
2354 gcpy #          translate([xpos(), ypos(), zpos()-kh_max_depth]){
2355 gcpy #              keyhole_shaft(6.35, 9.525);
2356 gcpy #          }
2357 gcpy #
2358 gcpy #      hull(){
2359 gcpy #          translate([xpos(), ypos(), zpos()-kh_max_depth]){
2360 gcpy #              keyhole_shaft(6.35, 9.525);
2361 gcpy #
2362 gcpy #          translate([xpos(), ypos()-kh_distance, zpos()-kh_max_depth])
{
2363 gcpy #              keyhole_shaft(6.35, 9.525);
2364 gcpy #
2365 gcpy #
}

```

```

2366 gcpy #      cutwithfeed(getxpos(), getypos(), -kh_max_depth, feed);
2367 gcpy #      cutwithfeed(getxpos(), getypos()-kh_distance, -kh_max_depth,
2368 gcpy #          feed);
2369 gcpy #      setypos(getypos()+kh_distance);
2370 gcpy #}

```

3.8.1.1.4 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.blockscad3d.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:

```

2372 gcpy     def cut_pins(self, Joint_Width, stockZthickness,
2373 gcpy         Number_of_Dovetails, Spacing, Proportion, DTT_diameter,
2374 gcpy         DTT_angle):
2375 gcpy         DTO = Tan(DTT_angle) * (stockZthickness * Proportion)
2376 gcpy         DTR = DTT_diameter/2 - DTO
2377 gcpy         cpr = self.rapidXY(0, stockZthickness + Spacing/2)
2378 gcpy         ctp = self.cutlinedxfgc(self.xpos(), self.ypos(), -
2379 gcpy             stockZthickness * Proportion)
2380 gcpy         ctp = ctp.union(self.cutlinedxfgc(Joint_Width / (
2381 gcpy             Number_of_Dovetails * 2), self.ypos(), -stockZthickness * 
2382 gcpy             Proportion))
2383 gcpy         i = 1
2384 gcpy         while i < Number_of_Dovetails * 2:
2385 gcpy             print(i)
2386 gcpy             ctp = ctp.union(self.cutlinedxfgc(i * (Joint_Width / (
2387 gcpy                 Number_of_Dovetails * 2)), self.ypos(), -
2388 gcpy                     stockZthickness * Proportion))
2389 gcpy             ctp = ctp.union(self.cutlinedxfgc(i * (Joint_Width / (
2390 gcpy                 Number_of_Dovetails * 2)), (stockZthickness +
2391 gcpy                     Spacing) + (stockZthickness * Proportion) - (
2392 gcpy                         DTT_diameter/2), -(stockZthickness * Proportion)))
2393 gcpy             ctp = ctp.union(self.cutlinedxfgc(i * (Joint_Width / (
2394 gcpy                 Number_of_Dovetails * 2)), stockZthickness + Spacing
2395 gcpy                     /2, -(stockZthickness * Proportion)))
2396 gcpy             ctp = ctp.union(self.cutlinedxfgc((i + 1) * (
2397 gcpy                 Joint_Width / (Number_of_Dovetails * 2)),
2398 gcpy                     stockZthickness + Spacing/2, -(stockZthickness *
2399 gcpy                     Proportion)))
2400 gcpy             self.dxfrectangleround(self.currenttoolnumber(),
2401 gcpy                 i * (Joint_Width / (Number_of_Dovetails * 2))-DTR,
2402 gcpy                 stockZthickness + (Spacing/2) - DTR,
2403 gcpy                 DTR * 2,
2404 gcpy                 (stockZthickness * Proportion) + Spacing/2 + DTR *
2405 gcpy                     2 - (DTT_diameter/2),
2406 gcpy                     DTR)
2407 gcpy             i += 2
2408 gcpy             self.rapidZ(0)
2409 gcpy             return ctp

```

and

```

2395 gcpy     def cut_tails(self, Joint_Width, stockZthickness,
2396 gcpy         Number_of_Dovetails, Spacing, Proportion, DTT_diameter,
2397 gcpy         DTT_angle):
2398 gcpy         DTO = Tan(DTT_angle) * (stockZthickness * Proportion)
2399 gcpy         DTR = DTT_diameter/2 - DTO
2400 gcpy         cpr = self.rapidXY(0, 0)
2401 gcpy         ctp = self.cutlinedxfgc(self.xpos(), self.ypos(), -
2402 gcpy             stockZthickness * Proportion)
2403 gcpy         ctp = ctp.union(self.cutlinedxfgc(
2404 gcpy             Joint_Width / (Number_of_Dovetails * 2) - (DTT_diameter
2405 gcpy                 - DTO),
2406 gcpy             self.ypos(),

```

```

2403 gcpy           -stockZthickness * Proportion))
2404 gcpy           i = 1
2405 gcpy           while i < Number_of_Dovetails * 2:
2406 gcpy           ctp = ctp.union(self.cutlinedxfgc(
2407 gcpy           i * (Joint_Width / (Number_of_Dovetails * 2)) - (
2408 gcpy           DTT_diameter - DTO),
2409 gcpy           stockZthickness * Proportion - DTT_diameter / 2,
2410 gcpy           -(stockZthickness * Proportion)))
2411 gcpy           ctp = ctp.union(self.cutarcCWdx(180, 90,
2412 gcpy           i * (Joint_Width / (Number_of_Dovetails * 2)),
2413 gcpy           stockZthickness * Proportion - DTT_diameter / 2,
2414 gcpy           self.ypos(),
2415 gcpy           DTT_diameter - DTO, 0, 1))
2416 gcpy           ctp = ctp.union(self.cutarcCWdx(90, 0,
2417 gcpy           i * (Joint_Width / (Number_of_Dovetails * 2)),
2418 gcpy           stockZthickness * Proportion - DTT_diameter / 2,
2419 gcpy           DTT_diameter - DTO, 0, 1))
2420 gcpy           ctp = ctp.union(self.cutlinedxfgc(
2421 gcpy           i * (Joint_Width / (Number_of_Dovetails * 2)) + (
2422 gcpy           DTT_diameter - DTO),
2423 gcpy           0,
2424 gcpy           -(stockZthickness * Proportion)))
2425 gcpy           ctp = ctp.union(self.cutlinedxfgc(
2426 gcpy           (i + 2) * (Joint_Width / (Number_of_Dovetails * 2))
2427 gcpy           - (DTT_diameter - DTO),
2428 gcpy           0,
2429 gcpy           -(stockZthickness * Proportion)))
2430 gcpy           self.rapidZ(0)
2431 gcpy           self.rapidXY(0, 0)
2432 gcpy           ctp = ctp.union(self.cutlinedxfgc(self.xpos(), self.ypos(),
2433 gcpy           -stockZthickness * Proportion))
2434 gcpy           self.dxfarc(0, 0, DTR, 180, 270)
2435 gcpy           self.dxfline(-DTR, 0, -DTR, stockZthickness + DTR)
2436 gcpy           self.dxfarc(0, stockZthickness + DTR, DTR, 90, 180)
2437 gcpy           self.dxfline(0, stockZthickness + DTR * 2, Joint_Width,
2438 gcpy           stockZthickness + DTR * 2)
2439 gcpy           i = 0
2440 gcpy           while i < Number_of_Dovetails * 2:
2441 gcpy           ctp = ctp.union(self.cutline(i * (Joint_Width /
2442 gcpy           Number_of_Dovetails * 2)), stockZthickness + DTO, -(

2443 gcpy           stockZthickness * Proportion)))
2444 gcpy           ctp = ctp.union(self.cutline((i+2) * (Joint_Width /
2445 gcpy           Number_of_Dovetails * 2)), stockZthickness + DTO, -(

2446 gcpy           stockZthickness * Proportion)))
2447 gcpy           ctp = ctp.union(self.cutline((i+2) * (Joint_Width /
2448 gcpy           Number_of_Dovetails * 2)), 0, -(stockZthickness *
2449 gcpy           Proportion)))
2450 gcpy           self.dxfarc(i * (Joint_Width / (Number_of_Dovetails * 2)),
2451 gcpy           0, DTR, 270, 360)
2452 gcpy           self.dxfline(
2453 gcpy           i * (Joint_Width / (Number_of_Dovetails * 2)) + DTR
2454 gcpy           ,
2455 gcpy           0,
2456 gcpy           i * (Joint_Width / (Number_of_Dovetails * 2)) + DTR
2457 gcpy           , stockZthickness * Proportion - DTT_diameter /
2458 gcpy           2)
2459 gcpy           self.dxfarc((i + 1) * (Joint_Width / (
2460 gcpy           Number_of_Dovetails * 2)), stockZthickness *
2461 gcpy           Proportion - DTT_diameter / 2, (Joint_Width / (
2462 gcpy           Number_of_Dovetails * 2)) - DTR, 90, 180)
2463 gcpy           self.dxfarc((i + 1) * (Joint_Width / (
2464 gcpy           Number_of_Dovetails * 2)), stockZthickness *
2465 gcpy           Proportion - DTT_diameter / 2, (Joint_Width / (
2466 gcpy           Number_of_Dovetails * 2)) - DTR, 0, 90)
2467 gcpy           self.dxfline(
2468 gcpy           (i + 2) * (Joint_Width / (Number_of_Dovetails * 2))
2469 gcpy           - DTR,
2470 gcpy           0,
2471 gcpy           (i + 2) * (Joint_Width / (Number_of_Dovetails * 2))
2472 gcpy           - DTR, stockZthickness * Proportion -
2473 gcpy           DTT_diameter / 2)
2474 gcpy           self.dxfarc((i + 2) * (Joint_Width / (
2475 gcpy           Number_of_Dovetails * 2)), 0, DTR, 180, 270)
2476 gcpy           i += 2
2477 gcpy           self.dxfarc(Joint_Width, stockZthickness + DTR, DTR, 0, 90)
2478 gcpy           self.dxfline(Joint_Width + DTR, stockZthickness + DTR,
2479 gcpy           Joint_Width + DTR, 0)

```

```
2455 gcpy      self.dxfarc(Joint_Width, 0, DTR, 270, 360)
2456 gcpy      return ctp
```

which are used as:

```
toolpaths = gcp.cut_pins(stockXwidth, stockZthickness, Number_of_Dovetails, Spacing, Proportion, DTT_dia)
toolpaths.append(gcp.cut_tails(stockXwidth, stockZthickness, Number_of_Dovetails, Spacing, Proportion, DTT_dia)
```

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

3.8.1.1.5 Full-blind box joints BlocksCAD project page at: <https://www.blockscad3d.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:

```
2458 gcpy      def Full_Blind_Finger_Joint_square(self, bx, by, orientation,
2459 gcpy          side, width, thickness, Number_of_Pins, largeVdiameter,
2460 gcpy          smallDiameter, normalormirror = "Default"):
2461 gcpy          # Joint_Orientation = "Horizontal" "Even" == "Lower", "Odd" ==
2462 gcpy          "Upper"
2463 gcpy          # Joint_Orientation = "Vertical" "Even" == "Left", "Odd" == "
2464 gcpy          "Right"
2465 gcpy          if (orientation == "Vertical"):
2466 gcpy              if (normalormirror == "Default" and side != "Both"):
2467 gcpy                  if (side == "Left"):
2468 gcpy                      normalormirror = "Even"
2469 gcpy                  if (side == "Right"):
2470 gcy                      normalormirror = "Odd"
2471 gcy          if (orientation == "Horizontal"):
2472 gcy              if (normalormirror == "Default" and side != "Both"):
2473 gcy                  if (side == "Lower"):
2474 gcy                      normalormirror = "Even"
2475 gcy                  if (side == "Upper"):
2476 gcy                      normalormirror = "Odd"
2477 gcy          Finger_Width = ((Number_of_Pins * 2) - 1) * smallDiameter *
2478 gcy              1.1
2479 gcy          Finger_Origin = width/2 - Finger_Width/2
2480 gcy          rapid = self.rapidZ(0)
2481 gcy          self.setdxfcolor("Cyan")
2482 gcy          rapid = rapid.union(self.rapidXY(bx, by))
2483 gcy          toolpath = (self.Finger_Joint_square(bx, by, orientation,
2484 gcy              side, width, thickness, Number_of_Pins, Finger_Origin,
2485 gcy              smallDiameter))
2486 gcy          if (orientation == "Vertical"):
2487 gcy              if (side == "Both"):
2488 gcy                  toolpath = self.cutrectangleroundddxf(self.
2489 gcy                      currenttoolnum, bx - (thickness - smallDiameter
2490 gcy                          /2), by-smallDiameter/2, 0, (thickness * 2) -
2491 gcy                          smallDiameter, width+smallDiameter,
2492 gcy                          (smallDiameter / 2) / Tan(45), smallDiameter/2)
2493 gcy              if (side == "Left"):
2494 gcy                  toolpath = self.cutrectangleroundddxf(self.
2495 gcy                      currenttoolnum, bx - (smallDiameter/2), by-
2496 gcy                      smallDiameter/2, 0, thickness, width+
2497 gcy                      smallDiameter, ((smallDiameter / 2) / Tan(45)),
2498 gcy                      smallDiameter/2)
2499 gcy              if (side == "Right"):
2500 gcy                  toolpath = self.cutrectangleroundddxf(self.
2501 gcy                      currenttoolnum, bx - (thickness - smallDiameter
2502 gcy                          /2), by-smallDiameter/2, 0, thickness, width+
2503 gcy                          smallDiameter, ((smallDiameter / 2) / Tan(45)),
2504 gcy                          smallDiameter/2)
```

```

2486 gcpy          toolpath = toolpath.union(self.Finger_Joint_square(bx, by,
2487 gcpy          orientation, side, width, thickness, Number_of_Pins,
2488 gcpy          Finger_Origin, smallDiameter))
2489 gcpy          if (orientation == "Horizontal"):
2490 gcpy          if (side == "Both"):
2491 gcpy          toolpath = self.cutrectanglerounddxf(
2492 gcpy          self.currenttoolnum,
2493 gcpy          bx-smallDiameter/2,
2494 gcpy          by - (thickness - smallDiameter/2),
2495 gcpy          0,
2496 gcpy          width+smallDiameter,
2497 gcpy          (thickness * 2) - smallDiameter,
2498 gcpy          (smallDiameter / 2) / Tan(45),
2499 gcpy          smallDiameter/2)
2500 gcpy          if (side == "Lower"):
2501 gcpy          toolpath = self.cutrectanglerounddxf(
2502 gcpy          self.currenttoolnum,
2503 gcpy          bx - (smallDiameter/2),
2504 gcpy          by - smallDiameter/2,
2505 gcpy          0,
2506 gcpy          width+smallDiameter,
2507 gcpy          thickness,
2508 gcpy          ((smallDiameter / 2) / Tan(45)),
2509 gcpy          smallDiameter/2)
2510 gcpy          if (side == "Upper"):
2511 gcpy          toolpath = self.cutrectanglerounddxf(
2512 gcpy          self.currenttoolnum,
2513 gcpy          bx - (thickness - smallDiameter/2),
2514 gcpy          0,
2515 gcpy          width+smallDiameter,
2516 gcpy          thickness,
2517 gcpy          ((smallDiameter / 2) / Tan(45)),
2518 gcpy          smallDiameter/2)
2519 gcpy          toolpath = toolpath.union(self.Finger_Joint_square(bx, by,
2520 gcpy          orientation, side, width, thickness, Number_of_Pins,
2521 gcpy          Finger_Origin, smallDiameter))
2522 gcpy          return toolpath
2523 gcpy          def Finger_Joint_square(self, bx, by, orientation, side, width,
2524 gcpy          thickness, Number_of_Pins, Finger_Origin, smallDiameter,
2525 gcpy          normalormirror = "Default"):
2526 gcpy          jointdepth = -(thickness - (smallDiameter / 2) / Tan(45))
2527 gcpy          # Joint_Orientation = "Horizontal" "Even" == "Lower", "Odd" ==
2528 gcpy          # "Upper"
2529 gcpy          # Joint_Orientation = "Vertical" "Even" == "Left", "Odd" ==
2530 gcpy          # "Right"
2531 gcpy          if (orientation == "Vertical"):
2532 gcpy          if (normalormirror == "Default" and side != "Both"):
2533 gcpy          if (side == "Left"):
2534 gcpy          normalormirror = "Even"
2535 gcpy          if (side == "Right"):
2536 gcpy          normalormirror = "Odd"
2537 gcpy          radius = smallDiameter/2
2538 gcpy          jointwidth = thickness - smallDiameter
2539 gcpy          toolpath = self.currenttool()
2540 gcpy          rapid = self.rapidZ(0)
2541 gcpy          self.setdxfcolor("Blue")
2542 gcpy          toolpath = toolpath.union(self.cutlineZgcfeed(jointdepth
2543 gcpy          ,1000))
2544 gcpy          self.beginpolyline(self.currenttool())
2545 gcpy          if (orientation == "Vertical"):
2546 gcpy          rapid = rapid.union(self.rapidXY(bx, by + Finger_Origin
2547 gcpy          ))
2548 gcpy          self.addvertex(self.currenttoolnumber(), self.xpos(),
2549 gcpy          self.ypos())
2550 gcpy          toolpath = toolpath.union(self.cutlineZgcfeed(
2551 gcpy          jointdepth,1000))
2552 gcpy          i = 0
2553 gcpy          while i <= Number_of_Pins - 1:
2554 gcpy          if (side == "Right"):
2555 gcpy          toolpath = toolpath.union(self.cutvertexdxf(

```

```

2552 gcpy
2553 gcpy

2554 gcpy
2555 gcpy
2556 gcpy
2557 gcpy
2558 gcpy
2559 gcpy
2560 gcpy
2561 gcpy
2562 gcpy
2563 gcpy
2564 gcpy
2565 gcpy
2566 gcpy
2567 gcpy
2568 gcpy
2569 gcpy
# Joint_Orientation = "Horizontal" "Even" == "Lower", "Odd" ==
# "Upper"
2570 gcpy
2571 gcpy
2572 gcpy
2573 gcpy
2574 gcpy
2575 gcpy
2576 gcpy
2577 gcpy
2578 gcpy
2579 gcpy
2580 gcpy
2581 gcpy
2582 gcpy
2583 gcpy
2584 gcpy
2585 gcpy
2586 gcpy
2587 gcpy

```

```

2588 gcpy      #
2589 gcpy      print(i)
2590 gcpy      toolpath = toolpath.union(self.cutvertexdxdf
2591 gcpy      (self.xpos() + radius, self.ypos(),
2592 gcpy      jointdepth))
2593 gcpy      toolpath = toolpath.union(self.cutvertexdxdf
2594 gcpy      (self.xpos() + radius/5, self.ypos(),
2595 gcpy      jointdepth))
2596 gcpy      toolpath = toolpath.union(self.cutvertexdxdf
2597 gcpy      (self.xpos(), self.ypos() + jointwidth,
2598 gcpy      jointdepth))
2599 gcpy      toolpath = toolpath.union(self.cutvertexdxdf
2600 gcpy      (self.xpos() + radius, self.ypos(),
2601 gcpy      jointdepth))
2602 gcpy      i += 1
2603 gcpy      self.closepolyline(self.currenttoolnumber())
2604 gcpy      return toolpath

2605 gcpy      def Full_Blind_Finger_Joint_smallV(self, bx, by, orientation,
2606 gcpy      side, width, thickness, Number_of_Pins, largeVdiameter,
2607 gcpy      smallDiameter):
2608 gcpy      rapid = self.rapidZ(0)
2609 gcpy      # rapid = rapid.union(self.rapidXY(bx, by))
2610 gcpy      self.setdxfcolor("Red")
2611 gcpy      if (orientation == "Vertical"):
2612 gcpy      rapid = rapid.union(self.rapidXY(bx, by - smallDiameter
2613 gcpy      /6))
2614 gcpy      toolpath = self.cutlineZgcfeed(-thickness,1000)
2615 gcpy      toolpath = self.cutlinedxfgc(bx, by + width +
2616 gcpy      smallDiameter/6, - thickness)
2617 gcpy      if (orientation == "Horizontal"):
2618 gcpy      rapid = rapid.union(self.rapidXY(bx - smallDiameter/6,
2619 gcpy      by))
2620 gcpy      toolpath = self.cutlineZgcfeed(-thickness,1000)
2621 gcpy      toolpath = self.cutlinedxfgc(bx + width + smallDiameter
2622 gcpy      /6, by, -thickness)
2623 gcpy      # rapid = self.rapidZ(0)

2624 gcpy      return toolpath

2625 gcpy      def Full_Blind_Finger_Joint_largeV(self, bx, by, orientation,
2626 gcpy      side, width, thickness, Number_of_Pins, largeVdiameter,
2627 gcpy      smallDiameter):
2628 gcpy      radius = smallDiameter/2
2629 gcpy      rapid = self.rapidZ(0)
2630 gcpy      Finger_Width = ((Number_of_Pins * 2) - 1) * smallDiameter *
2631 gcpy      1.1
2632 gcpy      Finger_Origin = width/2 - Finger_Width/2
2633 gcpy      # rapid = rapid.union(self.rapidXY(bx, by))
2634 gcpy      # Joint_Orientation = "Horizontal" "Even" == "Lower", "Odd" ==
2635 gcpy      "Upper"
2636 gcpy      # Joint_Orientation = "Vertical" "Even" == "Left", "Odd" == "
2637 gcpy      Right"
2638 gcpy      if (orientation == "Vertical"):
2639 gcpy      rapid = rapid.union(self.rapidXY(bx, by))
2640 gcpy      toolpath = self.cutlineZgcfeed(-thickness,1000)
2641 gcpy      toolpath = toolpath.union(self.cutlinedxfgc(bx, by +
2642 gcpy      Finger_Origin, -thickness))
2643 gcpy      rapid = self.rapidZ(0)
2644 gcpy      rapid = rapid.union(self.rapidXY(bx, by + width -
2645 gcpy      Finger_Origin))
2646 gcpy      self.setdxfcolor("Blue")
2647 gcpy      toolpath = toolpath.union(self.cutlineZgcfeed(-
2648 gcpy      thickness,1000))
2649 gcpy      toolpath = toolpath.union(self.cutlinedxfgc(bx, by +
2650 gcpy      width, -thickness))
2651 gcpy      if (side == "Left" or side == "Both"):
2652 gcpy      rapid = self.rapidZ(0)
2653 gcpy      self.setdxfcolor("DarkGray")
2654 gcpy      rapid = rapid.union(self.rapidXY(bx+thickness -(
2655 gcpy      smallDiameter / 2) / Tan(45), by - radius/2))
2656 gcpy      toolpath = toolpath.union(self.cutlineZgcfeed(-(
2657 gcpy      smallDiameter / 2) / Tan(45),10000))
2658 gcpy      toolpath = toolpath.union(self.cutlinedxfgc(bx+
2659 gcpy      thickness-(smallDiameter / 2) / Tan(45), by +
2660 gcpy      width + radius/2 -(smallDiameter / 2) / Tan(45))

```

```

        ))
2637 gcpy
2638 gcpy
2639 gcpy
2640 gcpy
2641 gcpy
2642 gcpy
2643 gcpy
2644 gcpy
2645 gcpy
2646 gcpy
2647 gcpy
2648 gcpy
2649 gcpy
2650 gcpy
2651 gcpy
2652 gcpy
2653 gcpy
2654 gcpy
2655 gcpy
2656 gcpy
2657 gcpy
2658 gcpy
2659 gcpy
2660 gcpy
2661 gcpy # Joint_Orientation = "Horizontal" "Even" == "Lower", "Odd" ==
           "Upper"
2662 gcpy
2663 gcpy
2664 gcpy
2665 gcpy
2666 gcpy
2667 gcpy
2668 gcpy
2669 gcpy
2670 gcpy
2671 gcpy
2672 gcpy
2673 gcpy
2674 gcpy
2675 gcpy
2676 gcpy
2677 gcpy
2678 gcpy
2679 gcpy
2680 gcpy
2681 gcpy
2682 gcpy
        )
        rapid = self.rapidZ(0)
        self.setdxfcolor("Green")
        rapid = rapid.union(self.rapidXY(bx+thickness/2, by
                                         +width))
        toolpath = toolpath.union(self.cutlineZgcfeed(-
                                         thickness/2,1000))
        toolpath = toolpath.union(self.cutlinedxfgc(bx+
                                         thickness/2, by + width -thickness ,
                                         -thickness /2))
        rapid = self.rapidZ(0)
        rapid = rapid.union(self.rapidXY(bx+thickness/2, by
                                         ))
        toolpath = toolpath.union(self.cutlineZgcfeed(-
                                         thickness/2,1000))
        toolpath = toolpath.union(self.cutlinedxfgc(bx+
                                         thickness/2, by +thickness , -thickness /2))
        if (side == "Right" or side == "Both"):
            rapid = self.rapidZ(0)
            self.setdxfcolor("DarkGray")
            rapid = rapid.union(self.rapidXY(bx-(thickness-
                                         smallDiameter / 2) / Tan(45)), by - radius/2))
            toolpath = toolpath.union(self.cutlineZgcfeed(-(smallDiameter / 2) / Tan(45),10000))
            toolpath = toolpath.union(self.cutlinedxfgc(bx-(thickness-(smallDiameter / 2) / Tan(45)), by +
                                         width + radius/2, -(smallDiameter / 2) / Tan(45)
                                         ))
            rapid = self.rapidZ(0)
            self.setdxfcolor("Green")
            rapid = rapid.union(self.rapidXY(bx-thickness/2, by
                                         +width))
            toolpath = toolpath.union(self.cutlineZgcfeed(-
                                         thickness/2,1000))
            toolpath = toolpath.union(self.cutlinedxfgc(bx-
                                         thickness/2, by + width -thickness , -thickness /2))
            rapid = self.rapidZ(0)
            rapid = rapid.union(self.rapidXY(bx-thickness/2, by
                                         ))
            toolpath = toolpath.union(self.cutlineZgcfeed(-
                                         thickness/2,1000))
            toolpath = toolpath.union(self.cutlinedxfgc(bx-
                                         thickness/2, by + thickness , -thickness /2))
            # Joint_Orientation = "Horizontal" "Even" == "Lower", "Odd" ==
                           "Upper"
            if (orientation == "Horizontal"):
                rapid = rapid.union(self.rapidXY(bx, by))
                self.setdxfcolor("Blue")
                toolpath = self.cutlineZgcfeed(-thickness ,1000)
                toolpath = toolpath.union(self.cutlinedxfgc(bx +
                                         Finger_Origin, by, -thickness))
                rapid = rapid.union(self.rapidZ(0))
                rapid = rapid.union(self.rapidXY(bx + width -
                                         Finger_Origin, by))
                toolpath = toolpath.union(self.cutlineZgcfeed(-
                                         thickness ,1000))
                toolpath = toolpath.union(self.cutlinedxfgc(bx + width,
                                         by, -thickness))
            if (side == "Lower" or side == "Both"):
                rapid = self.rapidZ(0)
                self.setdxfcolor("DarkGray")
                rapid = rapid.union(self.rapidXY(bx - radius, by+
                                         thickness-(smallDiameter / 2) / Tan(45)))
                toolpath = toolpath.union(self.cutlineZgcfeed(-(smallDiameter / 2) / Tan(45),10000))
                toolpath = toolpath.union(self.cutlinedxfgc(bx + width + radius, by+thickness-(smallDiameter / 2) / Tan(45), -(smallDiameter / 2) / Tan(45)))
                rapid = self.rapidZ(0)
                self.setdxfcolor("Green")
                rapid = rapid.union(self.rapidXY(bx+width, by+
                                         thickness/2))
                toolpath = toolpath.union(self.cutlineZgcfeed(-
                                         thickness/2,1000))
                toolpath = toolpath.union(self.cutlinedxfgc(bx +
                                         width -thickness , by+thickness/2, -thickness /2))
                rapid = self.rapidZ(0)

```

```

2683 gcpy         rapid = rapid.union(self.rapidXY(bx, by+thickness
2684 gcpy             /2))
2685 gcpy         toolpath = toolpath.union(self.cutlineZgcfeed(-
2686 gcpy                 thickness/2,1000))
2687 gcpy         toolpath = toolpath.union(self.cutlinedxfgc(bx +
2688 gcpy                 thickness, by+thickness/2, -thickness/2))
2689 gcpy         if (side == "Upper" or side == "Both"):
2690 gcpy             rapid = self.rapidZ(0)
2691 gcpy             self.setdxfcolor("DarkGray")
2692 gcpy             rapid = rapid.union(self.rapidXY(bx - radius, by-
2693 gcpy                     thickness-(smallDiameter / 2) / Tan(45)))
2694 gcpy             toolpath = toolpath.union(self.cutlineZgcfeed(-(-
2695 gcpy                     smallDiameter / 2) / Tan(45),10000))
2696 gcpy             toolpath = toolpath.union(self.cutlinedxfgc(bx +
2697 gcpy                     width + radius, by-(thickness-(smallDiameter /
2698 gcpy                         2) / Tan(45)), -(smallDiameter / 2) / Tan(45)))
2699 gcpy             rapid = self.rapidZ(0)
2700 gcpy             self.setdxfcolor("Green")
2701 gcpy             rapid = rapid.union(self.rapidXY(bx+width, by-
2702 gcpy                     thickness/2))
2703 gcpy             toolpath = toolpath.union(self.cutlineZgcfeed(-
2704 gcpy                     thickness/2,1000))
2705 gcpy             toolpath = toolpath.union(self.cutlinedxfgc(bx +
2706 gcpy                     width -thickness, by-thickness/2, -thickness/2))
2707 gcpy             rapid = self.rapidZ(0)
2708 gcpy             rapid = rapid.union(self.rapidXY(bx, by-thickness
2709 gcpy                     /2))
2710 gcpy             toolpath = toolpath.union(self.cutlineZgcfeed(-
2711 gcpy                     thickness/2,1000))
2712 gcpy             rapid = self.rapidZ(0)
2713 gcpy             return toolpath
2714 gcpy         def Full_Blind_Finger_Joint(self, bx, by, orientation, side,
2715 gcpy             width, thickness, largeVdiameter, smallDiameter,
2716 gcpy             normalormirror = "Default", squaretool = 102, smallV = 390,
2717 gcpy             largeV = 301):
2718 gcpy             Number_of_Pins = int(((width - thickness * 2) / (
2719 gcpy                 smallDiameter * 2.2) / 2) + 0.0) * 2 + 1
2720 gcpy             print("Number of Pins: ",Number_of_Pins)
2721 gcpy             self.movetosafeZ()
2722 gcpy             self.toolchange(squaretool, 17000)
2723 gcpy             toolpath = self.Full_Blind_Finger_Joint_square(bx, by,
2724 gcpy                 orientation, side, width, thickness, Number_of_Pins,
2725 gcpy                 largeVdiameter, smallDiameter)
2726 gcpy             self.movetosafeZ()
2727 gcpy             self.toolchange(smallV, 17000)
2728 gcpy             toolpath = toolpath.union(self.
2729 gcpy                 Full_Blind_Finger_Joint_smallV(bx, by, orientation, side
2730 gcpy                 , width, thickness, Number_of_Pins, largeVdiameter,
2731 gcpy                 smallDiameter))
2732 gcpy             self.toolchange(largeV, 17000)
2733 gcpy             toolpath = toolpath.union(self.
2734 gcpy                 Full_Blind_Finger_Joint_largeV(bx, by, orientation, side
2735 gcpy                 , width, thickness, Number_of_Pins, largeVdiameter,
2736 gcpy                 smallDiameter))
2737 gcpy             return toolpath

```

3.8.1.2 Fonts While OpenSCAD, and by extension PythonSCAD support the use of fonts, the requirements of typography and OpenType/TrueType fonts do not align with the limitations of CNC, making the development of a mechanism for directly creating the stroke geometry of single line fonts desirable.

An initial version of only capital letters and numerals comprised of lines and arcs which a circle (representing the tool or extruded plastic) is either hull()ed or directly set and difference()d (making the same weight stroke) is a suitable trial run for later more complex designs using curves. Such a font would have the following parameters:

- Size — the height of glyphs
- Width — this would adjust from the narrowest possible design through a “normal” version where the O was a circle terminating in an expanded version with glyphs extended horizontally
- Weight — the radius/diameter of the circle used for hull() operations, or the dimension arrived at when difference()ing circles

- Spacing — adjustment for the space between glyphs
- overshoot — optical adjustment for narrow strokes/elements
- roundover — further adjustment for applying overshoot to larger rounded forms

Rather than directly program these, drawing them on a grid in a vector design program which permits changing stroke weight globally is the most expedient option, then the dimensions may be copied to create the letterforms.

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 gcodereview 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/gcodereview/
          refs/heads/main/gcodereview.py")
6 gcpncpy from gcodereview import *
7 gcpncpy
8 gcpncpy #The file to be loaded must be specified:
9 gcpncpy #gc_file = "filename_of_G-code_file_to_process.gcodefileext"
10 gcpncpy #
11 gcpncpy #if using windows the full filepath should be provided with
          backslashes replaced with double backslashes and wrapped in
          quotes since it is provided as a string:
12 gcpncpy gc_file = "C:\\\\Users\\\\willia\\\\OneDrive\\\\Desktop\\\\19mm_1_32_depth.nc"
13 gcpncpy
14 gcpncpy #Create the gcodereview object:
15 gcpncpy gcp = gcodereview("cut", False, False)
16 gcpncpy
17 gcpncpy #Process the file
18 gcpncpy gcp.previewgcodefile(gc_file)

```

previewgcodefile Which simply needs to call the previewgcodefile command:

```

2717 gcpy      def previewgcodefile(self, gc_file):
2718 gcpy          gc_file = open(gc_file, 'r')
2719 gcpy          gcfilecontents = []
2720 gcpy          with gc_file as file:
2721 gcpy              for line in file:
2722 gcpy                  command = line
2723 gcpy                  gcfilecontents.append(line)
2724 gcpy
2725 gcpy          numlinesfound = 0
2726 gcpy          for line in gcfilecontents:
2727 gcpy              print(line)
2728 gcpy              if line[:10] == "(stockMin:":
2729 gcpy                  subdivisions = line.split()
2730 gcpy                  extentleft = float(subdivisions[0][10:-3])
2731 gcpy                  extentfb = float(subdivisions[1][-3:])
2732 gcpy                  extentd = float(subdivisions[2][-3:])
2733 gcpy                  numlinesfound = numlinesfound + 1
2734 gcpy              if line[:13] == "(STOCK/BLOCK,":
2735 gcpy                  subdivisions = line.split()
2736 gcpy                  sizeX = float(subdivisions[0][13:-1])
2737 gcpy                  sizeY = float(subdivisions[1][-1])
2738 gcpy                  sizeZ = float(subdivisions[4][-1])
2739 gcpy                  numlinesfound = numlinesfound + 1
2740 gcpy              if line[:3] == "G21":
2741 gcpy                  units = "mm"
2742 gcpy                  numlinesfound = numlinesfound + 1
2743 gcpy              if numlinesfound >=3:
2744 gcpy                  break
2745 gcpy      print(numlinesfound)

```

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

```
2747 gcpy           self.setupcuttingarea(sizeX, sizeY, sizeZ, extentleft,
                                         extentfb, extentd)
2748 gcpy
2749 gcpy           commands = []
2750 gcpy           for line in gcfilecontents:
2751 gcpy               Xc = 0
2752 gcpy               Yc = 0
2753 gcpy               Zc = 0
2754 gcpy               Fc = 0
2755 gcpy               Xp = 0.0
2756 gcpy               Yp = 0.0
2757 gcpy               Zp = 0.0
2758 gcpy               if line == "G53G0Z-5.000\n":
2759 gcpy                   self.movetosafeZ()
2760 gcpy               if line[:3] == "M6T":
2761 gcpy                   tool = int(line[3:])
2762 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)
```

```
2763 gcpy
2764 gcpy
2765 gcpy
2766 gcpy
2767 gcpy
2768 gcpy
2769 gcpy
2770 gcpy
2771 gcpy
2772 gcpy
2773 gcpy
2774 gcpy
2775 gcpy
2776 gcpy
2777 gcpy
2778 gcpy
2779 gcpy
2780 gcpy
2781 gcpy
2782 gcpy
2783 gcpy
2784 gcpy
2785 gcpy
2786 gcpy
2787 gcpy
2788 gcpy
2789 gcpy
2790 gcpy
2791 gcpy
    if line[:2] == "G0":
        machinestate = "rapid"
    if line[:2] == "G1":
        machinestate = "cutline"
    if line[:2] == "G0" or line[:2] == "G1" or line[:1] ==
        "X" or line[:1] == "Y" or line[:1] == "Z":
        if "F" in line:
            Fplus = line.split("F")
            Fc = 1
            fr = float(Fplus[1])
            line = Fplus[0]
        if "Z" in line:
            Zplus = line.split("Z")
            Zc = 1
            Zp = float(Zplus[1])
            line = Zplus[0]
        if "Y" in line:
            Yplus = line.split("Y")
            Yc = 1
            Yp = float(Yplus[1])
            line = Yplus[0]
        if "X" in line:
            Xplus = line.split("X")
            Xc = 1
            Xp = float(Xplus[1])
    if Zc == 1:
        if Yc == 1:
            if Xc == 1:
                if machinestate == "rapid":
                    command = "rapidXYZ(" + str(Xp) + "
, " + str(Yp) + ", " + str(Zp) +
")"
```

```

2792 gcpy                                self.rapidXYZ(Xp, Yp, Zp)
2793 gcpy
2794 gcpy
2795 gcpy
2796 gcpy
2797 gcpy
2798 gcpy
2799 gcpy
2800 gcpy
2801 gcpy
2802 gcpy
2803 gcpy
2804 gcpy
2805 gcpy
2806 gcpy
2807 gcpy
2808 gcpy
2809 gcpy
2810 gcpy
2811 gcpy
2812 gcpy
2813 gcpy
2814 gcpy
2815 gcpy
2816 gcpy
2817 gcpy
2818 gcpy
2819 gcpy
2820 gcpy
2821 gcpy
2822 gcpy
2823 gcpy
2824 gcpy
2825 gcpy
2826 gcpy
2827 gcpy
2828 gcpy
2829 gcpy
2830 gcpy
2831 gcpy
2832 gcpy
2833 gcpy
2834 gcpy
2835 gcpy
2836 gcpy
2837 gcpy
2838 gcpy
2839 gcpy
2840 gcpy
2841 gcpy
2842 gcpy #
2843 gcpy #
2844 gcpy #
2845 gcpy #
2846 gcpy #
2847 gcpy #
2848 gcpy
2849 gcpy #
2850 gcpy #
2851 gcpy
2852 gcpy #
2853 gcpy
                                self.cutlineXYZ(Xp, Yp, Zp)
else:
    command = "cutlineXYZ(" + str(Xp) +
               ", " + str(Yp) + ", " + str(Zp) +
               ")"
    self.cutlineXYZ(Xp, Yp, Zp)
else:
    if machinestate == "rapid":
        command = "rapidYZ(" + str(Yp) + ", " +
                   str(Zp) + ")"
        self.rapidYZ(Yp, Zp)
    else:
        command = "cutlineYZ(" + str(Yp) +
                   ", " + str(Zp) + ")"
        self.cutlineYZ(Yp, Zp)
else:
    if Xc == 1:
        if machinestate == "rapid":
            command = "rapidXZ(" + str(Xp) + ", " +
                       str(Zp) + ")"
            self.rapidXZ(Xp, Zp)
        else:
            command = "cutlineXZ(" + str(Xp) +
                       ", " + str(Zp) + ")"
            self.cutlineXZ(Xp, Zp)
    else:
        if machinestate == "rapid":
            command = "rapidZ(" + str(Zp) + ")"
            self.rapidZ(Zp)
        else:
            command = "cutlineZ(" + str(Zp) +
                       ")"
            self.cutlineZ(Zp)
else:
    if Yc == 1:
        if Xc == 1:
            if machinestate == "rapid":
                command = "rapidXY(" + str(Xp) + ", " +
                           str(Yp) + ")"
                self.rapidXY(Xp, Yp)
            else:
                command = "cutlineXY(" + str(Xp) +
                           ", " + str(Yp) + ")"
                self.cutlineXY(Xp, Yp)
        else:
            if machinestate == "rapid":
                command = "rapidY(" + str(Yp) + ")"
                self.rapidY(Yp)
            else:
                command = "cutlineY(" + str(Yp) +
                           ")"
                self.cutlineY(Yp)
    else:
        if Xc == 1:
            if machinestate == "rapid":
                command = "rapidX(" + str(Xp) + ")"
                self.rapidX(Xp)
            else:
                command = "cutlineX(" + str(Xp) +
                           ")"
                self.cutlineX(Xp)
        commands.append(command)
        print(line)
        print(command)
        print(machinestate, Xc, Yc, Zc)
        print(Xp, Yp, Zp)
        print("/")
for command in commands:
    print(command)
show(self.stockandtoolpaths())
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

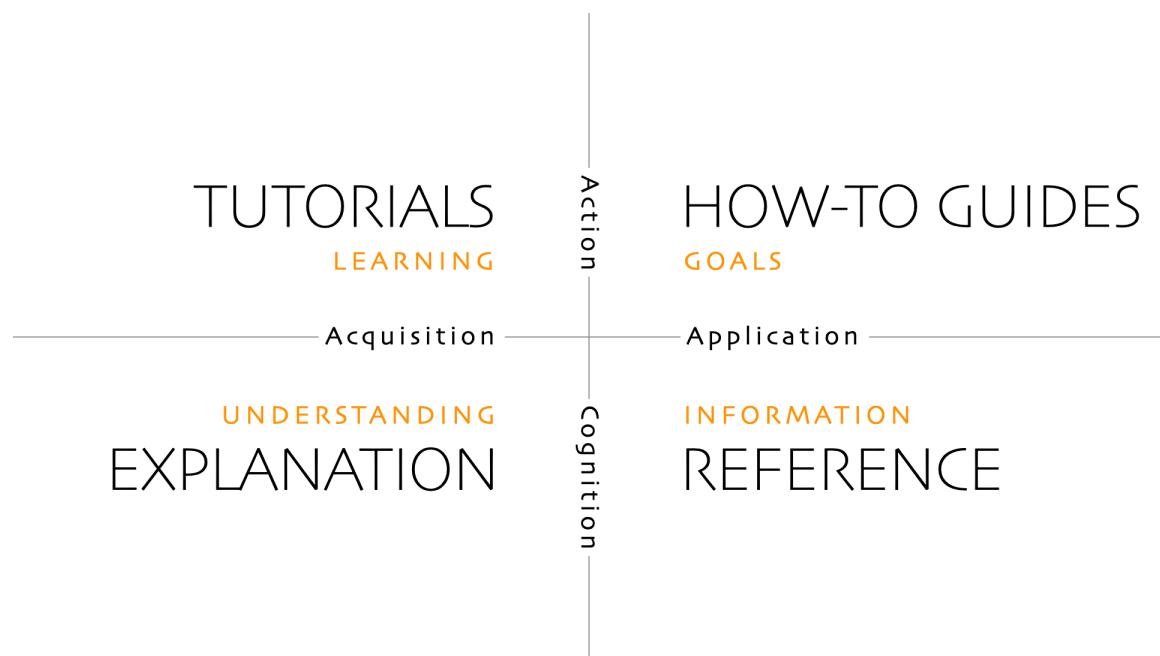
<https://thewhitetulip.gitbook.io/py/06-file-handling>

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/>

Holidays

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

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:

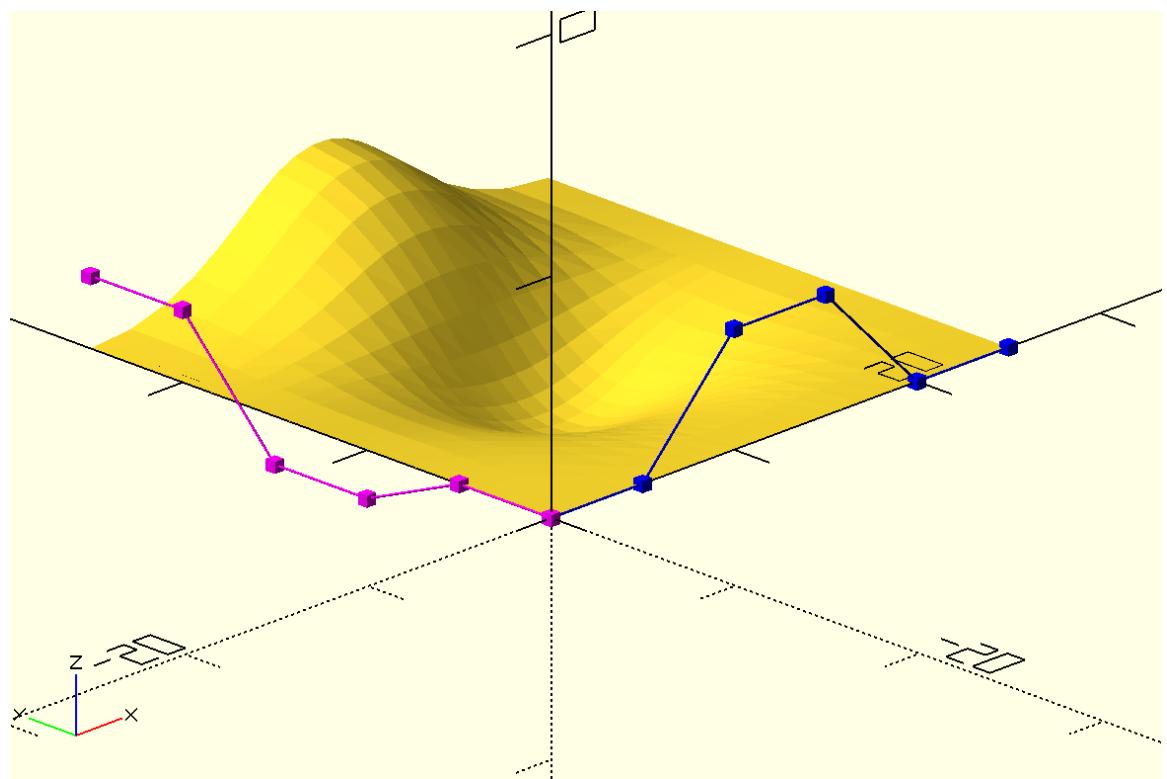
3 planes * 3 Béziers * (2 on-curve + 2 off-curve points) == 36 coordinate pairs

which is a marked contrast to representations such as:

<https://github.com/DavidPhillip0ster/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, 88
ballnose, 56
beginpolyline, 88
closedxffile, 95
closegcodefile, 95
close polyline, 88
currenttool, 33
currenttoolnum, 33
cut..., 52, 63
cutarcCC, 66
cutarcCW, 66
cutkeyhole toolpath, 99
cutKHgcdxf, 101
cutline, 63
cutrectangle, 99
diameter, 34
dovetail, 57
dxfarl, 88
dxfcircle, 88
dxfcolor, 88
dxflayer, 88
dxfline, 88
dxffpostamble, 95
dxfpreamble, 87
endmill square, 55
endmill v, 56
endmilltype, 34
feed, 71
flute, 34
gcodepreview, 31
 writeln, 84
gcp.setupstock, 34
generatecut, 28
generatedxf, 28
generategcode, 28
init, 31
mpx, 33
mpy, 33
mpz, 33
opendxffile, 85
opengcodefile, 85
plunge, 71
previewgcodefile, 115
ra, 34
rapid, 62
rapid..., 52
rapids, 34, 37
roundover, 58
settoolparameters, 41
setupstock, 34
 gcodepreview, 34
setxpos, 33
setypos, 33
setzpos, 33
shaftmovement, 53, 58
speed, 71
stockzero, 34
subroutine
 gcodepreview, 34
 writeln, 84
tip, 34
tool diameter, 70
tool number, 41
tool radius, 71
toolchange, 41, 42
toolmovement, 33, 34, 37, 42, 53
toolpaths, 34, 37
tpzinc, 33
xpos, 33
ypos, 33
zeroheight, 34
zpos, 33

Routines

addvertex, 88
ballnose, 56
beginpolyline, 88
closedxfile, 95
closegcodefile, 95
close polyline, 88
cut..., 52, 63
cutarcCC, 66
cutarcCW, 66
cutkeyhole toolpath, 99
cutKHgcodxf, 101
cutline, 63
cutrectangle, 99
dovetail, 57
dxfar, 88
dxfcircle, 88
dxfline, 88
dxffpostamble, 95
dxfpreamble, 87
endmill square, 55
endmill v, 56
gcodepreview, 31, 34
gcp.setupstock, 34
init, 31
opendxfile, 85
opengcodefile, 85
previewgcodefile, 115
rapid, 62
rapid..., 52
roundover, 58
setupstock, 34
setxpos, 33
setypos, 33
setzpos, 33
shaftmovement, 53, 58
tool diameter, 70
tool radius, 71
toolchange, 41, 42
toolmovement, 33, 34, 37, 42, 53
writeln, 84
xpos, 33
ypos, 33
zpos, 33

Variables

currenttool, 33
currenttoolnum, 33
diameter, 34
dxfcolor, 88
dxflayer, 88
endmilltype, 34
feed, 71
flute, 34
generatecut, 28
generatedxf, 28
generategcode, 28
mpx, 33
mpy, 33
mpz, 33
plunge, 71
ra, 34
rapids, 34, 37
settoolparameters, 41
speed, 71
stockzero, 34
tip, 34
tool number, 41
toolchange, 41
toolpaths, 34, 37
tpzinc, 33
zeroheight, 34