

# The gcodepreview PythonSCAD library\*

Author: William F. Adams  
willadams at aol dot com

2025/11/30

## 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

<b>1</b>	<b>readme.md</b>	<b>4</b>
<b>2</b>	<b>Usage and Templates</b>	<b>8</b>
2.1	gcpdxf.py . . . . .	8
2.2	gpcutdxf.py . . . . .	12
2.3	gcodepreviewtemplate.py . . . . .	14
2.4	gcodepreviewtemplate.scad . . . . .	19
2.5	gpcthreedp.py . . . . .	23
2.6	gcodepreviewtemplate.txt . . . . .	24
<b>3</b>	<b>gcodepreview</b>	<b>24</b>
3.1	Cutviewer . . . . .	25
3.1.1	Stock size and placement . . . . .	25
3.1.2	Tool Shapes . . . . .	25
3.2	Module Naming Convention . . . . .	26
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 . . . . .	32
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 . . . . .	51
3.5	Shapes and tool movement . . . . .	51
3.5.1	Tooling for Undercutting Toolpaths . . . . .	52
3.5.2	Generalized commands and cuts . . . . .	52
3.5.3	Movement and color . . . . .	52
3.5.4	tooldiameter . . . . .	67
3.5.5	Feeds and Speds . . . . .	69
3.6	3D Printing . . . . .	69
3.7	Difference of Stock, Rapids, and Toolpaths . . . . .	80
3.8	Output files . . . . .	80
3.8.1	Python and OpenSCAD File Handling . . . . .	80
3.8.2	DXF Overview . . . . .	84
3.8.3	G-code Overview . . . . .	91
3.9	Cutting shapes and expansion . . . . .	93
3.9.1	Building blocks . . . . .	94
3.10	(Reading) G-code Files . . . . .	113
<b>4</b>	<b>Notes</b>	<b>116</b>
4.1	Other Resources . . . . .	116
4.1.1	Coding Style . . . . .	116
4.1.2	Coding References . . . . .	116
4.1.3	Documentation Style . . . . .	117
4.2	Future . . . . .	117
4.2.1	Images . . . . .	117
4.2.2	Bézier curves in 2 dimensions . . . . .	117
4.2.3	Bézier curves in 3 dimensions . . . . .	118
4.2.4	Mathematics . . . . .	118
<b>Index</b>		<b>121</b>
Routines . . . . .		122
Variables . . . . .		123

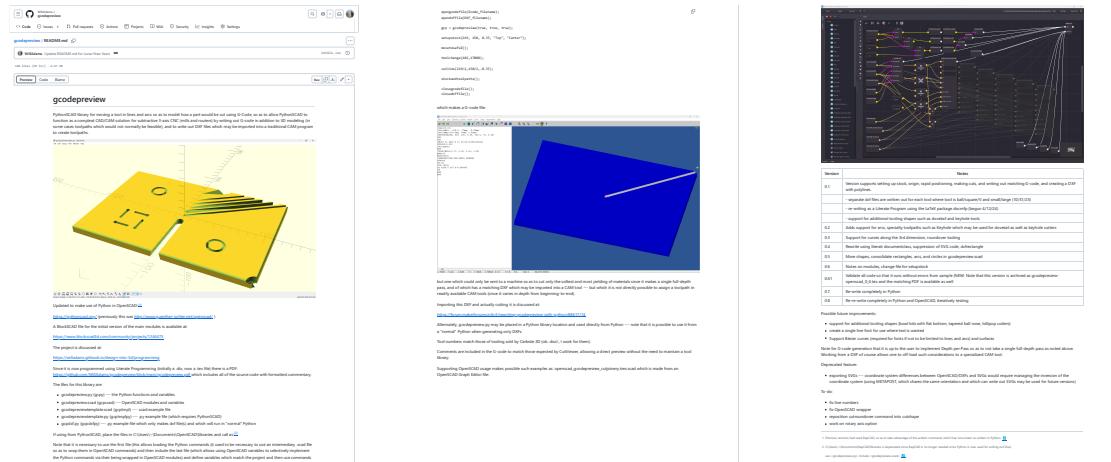
\*This file (gcodepreview) has version number v0.93, last revised 2025/11/30.<sup>1</sup>

## Contents

<b>1</b>	<b>readme.md</b>	<b>4</b>
<b>2</b>	<b>Usage and Templates</b>	<b>8</b>
2.1	gcpdxf.py . . . . .	8
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
<b>3</b>	<b>gcodepreview</b>	<b>24</b>
3.1	Cutviewer . . . . .	25
3.1.1	Stock size and placement . . . . .	25
3.1.2	Tool Shapes . . . . .	25
3.1.2.1	Tool/Mill (Square, radiused, ball-nose, and tapered-ball) . . . . .	26
3.1.2.2	Corner Rounding, (roundover) . . . . .	26
3.1.2.3	V shaped tooling (and variations) . . . . .	26
3.2	Module Naming Convention . . . . .	26
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 . . . . .	32
3.3.3	Initial Modules . . . . .	33
3.3.3.1	setupstock . . . . .	33
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 . . . . .	41
3.4.1.1.2	toolchange . . . . .	42
3.4.1.1.3	Square (including O-flute) . . . . .	42
3.4.1.1.4	Ball-nose (including tapered-ball) . . . . .	44
3.4.1.1.5	V . . . . .	45
3.4.1.1.6	Keyhole . . . . .	46
3.4.1.1.7	Bowl . . . . .	47
3.4.1.1.8	Tapered ball nose . . . . .	48
3.4.1.1.9	Roundover (cove tooling) . . . . .	48
3.4.1.1.10	Dovetails . . . . .	49
3.4.1.1.11	closing G-code . . . . .	50
3.4.2	Laser support . . . . .	51
3.5	Shapes and tool movement . . . . .	51
3.5.1	Tooling for Undercutting Toolpaths . . . . .	52
3.5.2	Generalized commands and cuts . . . . .	52
3.5.3	Movement and color . . . . .	52
3.5.3.1	toolmovement . . . . .	52
3.5.3.1.1	Square (including O-flute) . . . . .	53
3.5.3.1.2	Ball nose (including tapered ball nose) . . . . .	54
3.5.3.1.3	bowl . . . . .	54
3.5.3.1.4	V . . . . .	54
3.5.3.1.5	Keyhole . . . . .	54
3.5.3.1.6	Tapered ball nose . . . . .	55
3.5.3.1.7	Dovetails . . . . .	55
3.5.3.2	Concave toolshapes . . . . .	55
3.5.3.2.1	Roundover tooling . . . . .	56
3.5.3.3	shaftmovement . . . . .	56
3.5.3.4	tool outlines . . . . .	56
3.5.3.4.1	defineshaft . . . . .	57
3.5.3.4.2	Square (including O-flute) . . . . .	57
3.5.3.4.3	Ball-nose (including tapered-ball) . . . . .	58
3.5.3.4.4	V tool outline . . . . .	58
3.5.3.4.5	Keyhole outline . . . . .	58
3.5.3.4.6	Bowl outline . . . . .	59
3.5.3.4.7	Tapered ball nose . . . . .	59
3.5.3.4.8	Roundover (cove tooling) . . . . .	59
3.5.3.5	rapid and cut (lines) . . . . .	59
3.5.3.6	Arcs . . . . .	62
3.5.4	tooldiameter . . . . .	67
3.5.5	Feeds and Speeds . . . . .	69

3.6	3D Printing . . . . .	69
3.7	Difference of Stock, Rapids, and Toolpaths . . . . .	80
3.8	Output files . . . . .	80
3.8.1	Python and OpenSCAD File Handling . . . . .	80
3.8.2	DXF Overview . . . . .	84
3.8.2.1	Writing to DXF files . . . . .	84
3.8.2.1.1	DXF Lines and Arcs . . . . .	85
3.8.3	G-code Overview . . . . .	91
3.8.3.1	Closings . . . . .	92
3.9	Cutting shapes and expansion . . . . .	93
3.9.1	Building blocks . . . . .	94
3.9.1.1	List of shapes . . . . .	94
3.9.1.1.1	circles . . . . .	95
3.9.1.1.2	rectangles . . . . .	96
3.9.1.1.3	Keyhole toolpath and undercut tooling . . . . .	98
3.9.1.1.4	Dovetail joinery and tooling . . . . .	105
3.9.1.1.5	Full-blind box joints . . . . .	107
3.10	(Reading) G-code Files . . . . .	113
4	Notes . . . . .	116
4.1	Other Resources . . . . .	116
4.1.1	Coding Style . . . . .	116
4.1.2	Coding References . . . . .	116
4.1.3	Documentation Style . . . . .	117
4.2	Future . . . . .	117
4.2.1	Images . . . . .	117
4.2.2	Bézier curves in 2 dimensions . . . . .	117
4.2.3	Bézier curves in 3 dimensions . . . . .	118
4.2.4	Mathematics . . . . .	118
	<b>Index</b> . . . . .	<b>121</b>
	Routines . . . . .	122
	Variables . . . . .	123

## 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 all
      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 - gcpdfx.py (gcpdfxpy) --- .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
35 rdme Note that additional templates are in: https://github.com/WillAdams/gcodepreview/tree/main/templates
36 rdme
37 rdme If using from PythonSCAD, place the files in C:\Users\\~\Documents

```

```

    \OpenSCAD\libraries or, load them from Github using the command:
38 rdme
39 rdme     nimport("https://raw.githubusercontent.com/WillAdams/
                  gcodepreview/refs/heads/main/gcodepreview.py")
40 rdme
41 rdme If using gcodepreview.scad call as:
42 rdme
43 rdme     use <gcodepreview.py>
44 rdme     include <gcodepreview.scad>
45 rdme
46 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:
47 rdme
48 rdme     opengcodefile(Gcode_filename);
49 rdme     opendxfile(DXF_filename);
50 rdme
51 rdme     gcp = gcodepreview("cut", true, true);
52 rdme
53 rdme     setupstock(219, 150, 8.35, "Top", "Center");
54 rdme
55 rdme     movetosafeZ();
56 rdme
57 rdme     toolchange(102, 17000);
58 rdme
59 rdme     cutline(219/2, 150/2, -8.35);
60 rdme
61 rdme     stockandtoolpaths();
62 rdme
63 rdme     closegcodefile();
64 rdme     closedxfile();
65 rdme
66 rdme which makes a G-code file:
67 rdme
68 rdme ! [OpenSCAD template G-code file](https://raw.githubusercontent.com/
      WillAdams/gcodepreview/main/gcodepreview_template.png?raw=true)
69 rdme
70 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
      ).
71 rdme
72 rdme Importing this DXF and actually cutting it is discussed at:
73 rdme
74 rdme https://forum.makerforums.info/t/rewriting-gcodepreview-with-python
      /88617/14
75 rdme
76 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).
77 rdme
78 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.
79 rdme
80 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).
81 rdme
82 rdme Supporting OpenSCAD usage makes possible such examples as:
      openscad_gcodepreview_cutjoinery.tres.scad which is made from an
      OpenSCAD Graph Editor file:
83 rdme
84 rdme ! [OpenSCAD Graph Editor Cut Joinery File](https://raw.
      githubusercontent.com/WillAdams/gcodepreview/main/
      OSGE_cutjoinery.png?raw=true)
85 rdme
86 rdme | Version | Notes |

```

```

87 rdme | ----- | -----
88 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.
89 rdme |           | - separate dxf files are written out for each
      |           tool where tool is ball/square/V and small/large (10/31/23)
      |
90 rdme |           | - re-writing as a Literate Program using the
      |           LaTeX package docmfp (begun 4/12/24)
      |
91 rdme |           | - support for additional tooling shapes such as
      |           dovetail and keyhole tools
      |
92 rdme | 0.2           | Adds support for arcs, specialty toolpaths such
      |           as Keyhole which may be used for dovetail as well as keyhole
      |           cutters
      |
93 rdme | 0.3           | Support for curves along the 3rd dimension,
      |           roundover tooling
      |
94 rdme | 0.4           | Rewrite using literati documentclass, suppression
      |           of SVG code, dxfractangle
      |
95 rdme | 0.5           | More shapes, consolidate rectangles, arcs, and
      |           circles in gcodepreview.scad
      |
96 rdme | 0.6           | Notes on modules, change file for setupstock
      |
97 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)
98 rdme | 0.7           | Re-write completely in Python
      |
99 rdme | 0.8           | Re-re-write completely in Python and OpenSCAD,
      |           iteratively testing
      |
100 rdme | 0.801          | Add support for bowl bits with flat bottom
      |
101 rdme | 0.802          | Add support for tapered ball-nose and V tools
      |           with flat bottom
      |
102 rdme | 0.803          | Implement initial color support and joinery
      |           modules (dovetail and full blind box joint modules)
      |
103 rdme | 0.9           | Re-write to use Python lists for 3D shapes for
      |           toolpaths and rapids.
      |
104 rdme | 0.91          | Finish converting to native OpenPythonSCAD
      |           trigonometric functions.
      |
105 rdme | 0.92          | Remove multiple DXFs and unimplemented features,
      |           add hooks for 3D printing.
      |
106 rdme | 0.93          | Initial support for 3D printing.
      |
107 rdme
108 rdme Possible future improvements:
109 rdme
110 rdme - support for 4th-axis
111 rdme - support for post-processors
112 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)
113 rdme - support for additional tooling shapes (lollipop cutters)
114 rdme - create a single line font for use where text is wanted
115 rdme - Support for METAPOST and Bézier curves (latter required for
          fonts if not to be limited to lines and arcs) and surfaces
116 rdme
117 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.
118 rdme
119 rdme To-do:
120 rdme
121 rdme - implement skin()
122 rdme - determine why one quadrant of arc command doesn't work in
          OpenSCAD
123 rdme - clock-wise arcs
124 rdme - add toolpath for cutting countersinks using ball-nose tool from
          inside working out
125 rdme - verify OpenSCAD wrapper and add any missing commands for Python
126 rdme - verify support for shaft on tooling
127 rdme - create additional template and sample files
128 rdme - fully implement/verify describing/saving/loading tools using
          CutViewer comments
129 rdme
130 rdme Deprecated features:
131 rdme
132 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)
133 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.
134 rdme - multiple DXF files
135 rdme - RapCAD support
```

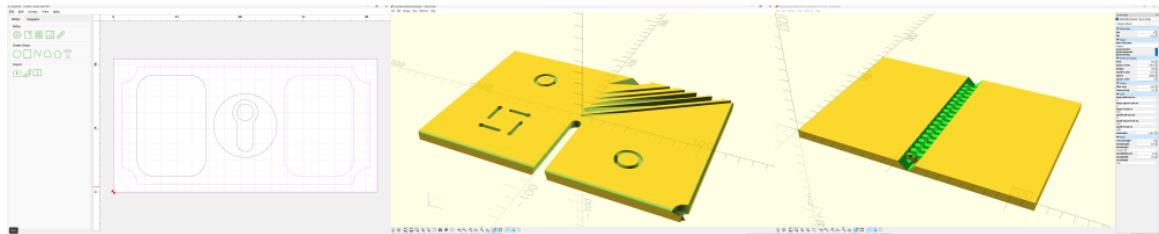
---

## 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 allow for reasonable tool (and toolpath) geometries).

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

It should be that the `readme` at the project page which serves as an overview, and this section (which serves as a collection of templates and a tutorial) are all the documentation which most users will need (and arguably is still too much). The balance of the document after this section shows all the code and implementation details, and will where appropriate show examples of usage 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.

### 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(large_square_tool_num, 0, 0, stockXwidth,
                           stockYheight)
50 gcpdxfpy
51 gcpdxfpy gcp.setdxfcolor("Red")
52 gcpdxfpy
53 gcpdxfpy gcp.dxfarc(large_square_tool_num, inset, inset, radius, 0, 90)
54 gcpdxfpy gcp.dxfarc(large_square_tool_num, stockXwidth - inset, inset,
                       radius, 90, 180)
55 gcpdxfpy gcp.dxfarc(large_square_tool_num, stockXwidth - inset, stockYheight
                       - inset, radius, 180, 270)
56 gcpdxfpy gcp.dxfarc(large_square_tool_num, inset, stockYheight - inset,
                       radius, 270, 360)
57 gcpdxfpy
58 gcpdxfpy gcp.dxfline(large_square_tool_num, inset, inset + radius, inset,
                        stockYheight - (inset + radius))
59 gcpdxfpy gcp.dxfline(large_square_tool_num, inset + radius, inset,
                        stockXwidth - (inset + radius), inset)
60 gcpdxfpy gcp.dxfline(large_square_tool_num, stockXwidth - inset, inset +
                        radius, stockXwidth - inset, stockYheight - (inset + radius))
61 gcpdxfpy gcp.dxfline(large_square_tool_num, inset + radius, stockYheight -
                        inset, stockXwidth - (inset + radius), stockYheight - inset)
62 gcpdxfpy
63 gcpdxfpy gcp.setdxfcolor("Blue")
64 gcpdxfpy
65 gcpdxfpy gcp.dxfrectangle(large_square_tool_num, radius + inset, radius,
                           stockXwidth/2 - (radius * 4), stockYheight - (radius * 2),
                           cornerstyle, radius)
66 gcpdxfpy gcp.dxfrectangle(large_square_tool_num, stockXwidth/2 + (radius *
                           2) + inset, radius, stockXwidth/2 - (radius * 4), stockYheight -
                           inset)

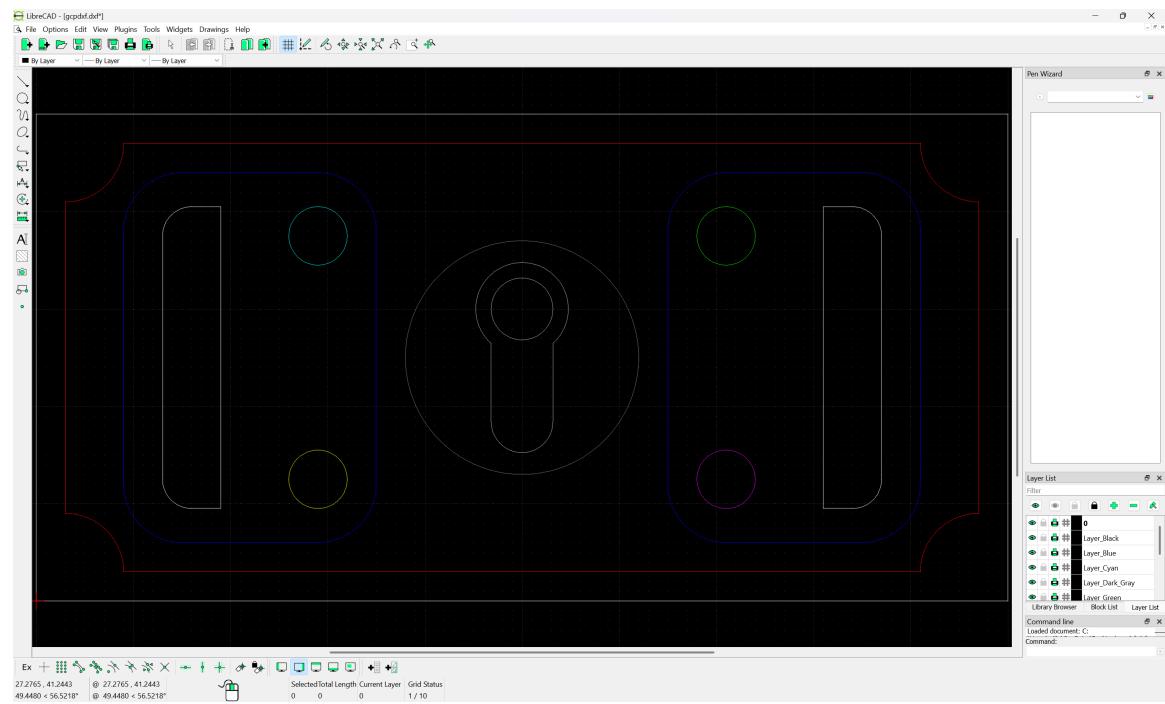
```

```

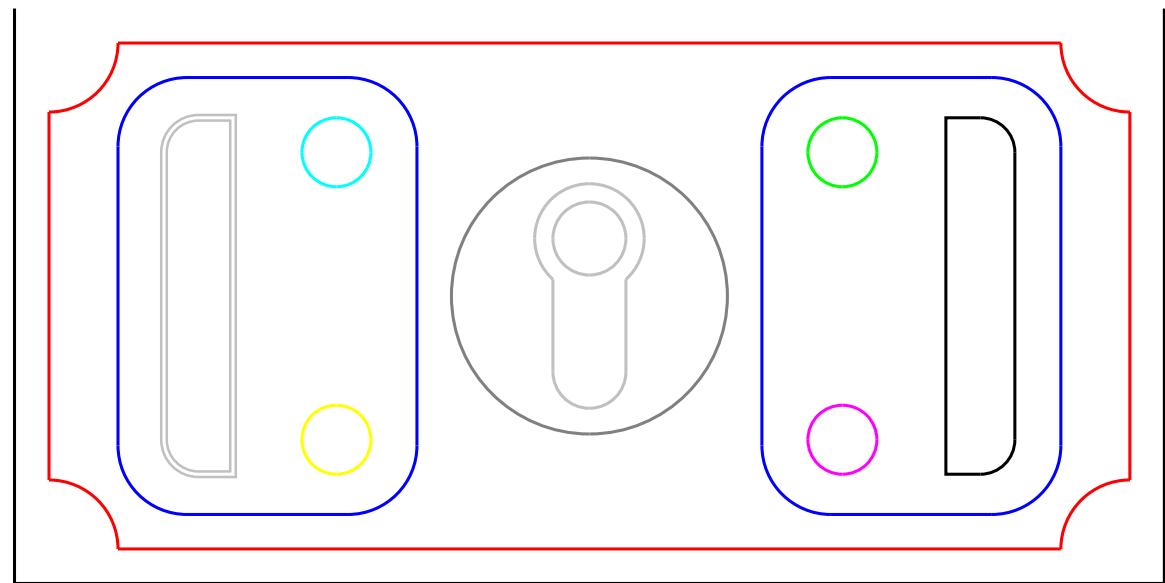
(radius * 2), cornerstyle, radius)
67 gcpdxfpy
68 gcpdxfpy gcp.setdxfcolor("Black")
69 gcpdxfpy
70 gcpdxfpy gcp.beginpolyline(large_square_tool_num)
71 gcpdxfpy gcp.addvertex(large_square_tool_num, stockXwidth*0.75+radius*1.5,
    stockYheight/4-radius/2)
72 gcpdxfpy gcp.addvertex(large_square_tool_num,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

```
11.5875)
123 gcpdxfpy
124 gcpdxfpy gcp.closedxfile()
```

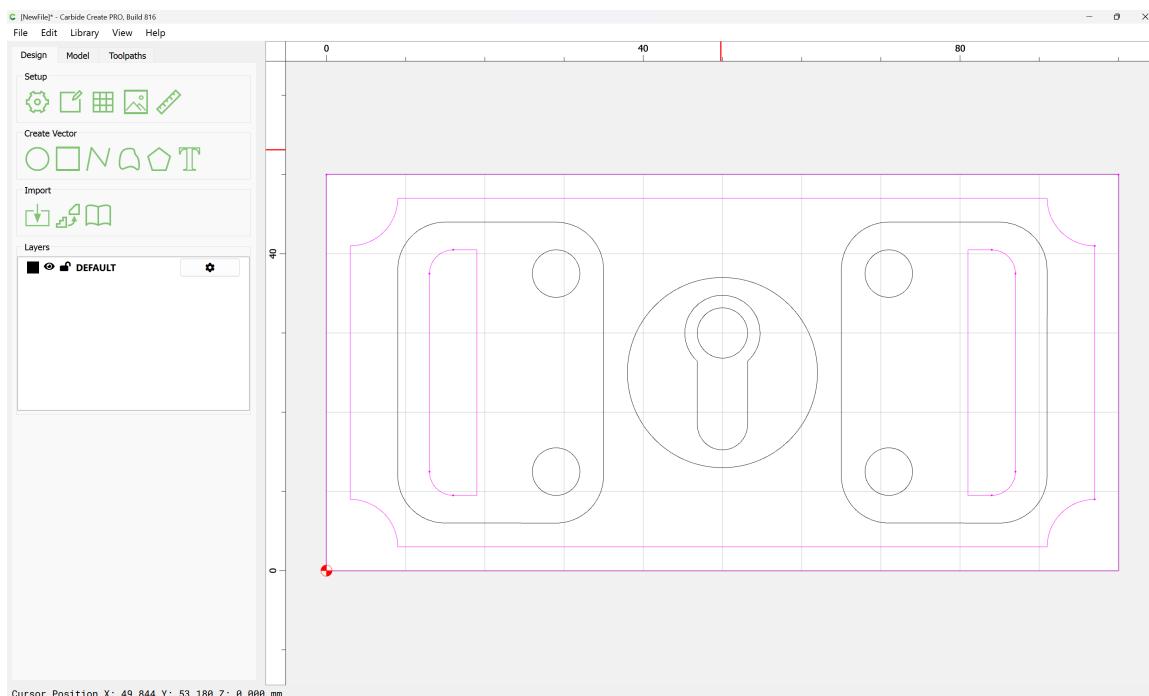
which creates a .dxf file which may be imported into any CAD program:



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



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



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

- dxfrectangle (specify lower-left 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)
- dxfarcc (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:

---

```

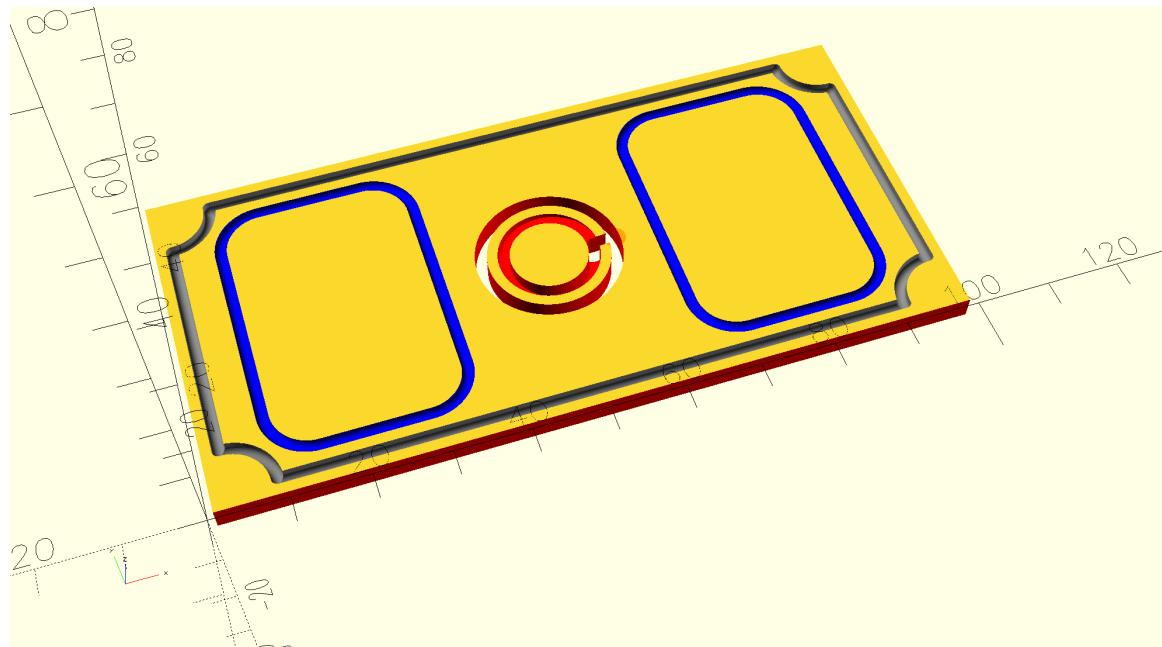
1 gpcutdxfpy from openscad import *
2 gpcutdxfpy # nimport("https://raw.githubusercontent.com/WillAdams/gcodepreview
   /refs/heads/main/gcodepreview.py")
3 gpcutdxfpy from gcodepreview import *
4 gpcutdxfpy
5 gpcutdxfpy fa = 2
6 gpcutdxfpy fs = 0.125
7 gpcutdxfpy
8 gpcutdxfpy gcp = gcodepreview("cut", # "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] */

```



```
radius, 0, stockXwidth/2 - (radius * 4), stockYheight - (radius * 2), -stockZthickness/4, radius)
96 gpcutdxfpy
97 gpcutdxfpy gcp.rapid(stockXwidth/2 + (radius * 2) + inset + radius, radius, 0,
    "laser")
98 gpcutdxfpy
99 gpcutdxfpy gcp.cutrectanglerounddxf(large_square_tool_num, stockXwidth/2 +
    radius * 2) + inset, radius, 0, stockXwidth/2 - (radius * 4),
    stockYheight - (radius * 2), -stockZthickness/4, radius)
100 gpcutdxfpy
101 gpcutdxfpy gcp.setdxfcolor("Red")
102 gpcutdxfpy
103 gpcutdxfpy gcp.rapid(stockXwidth/2 + radius, stockYheight/2, 0, "laser")
104 gpcutdxfpy
105 gpcutdxfpy gcp.toolchange(large_square_tool_num)
106 gpcutdxfpy
107 gpcutdxfpy gcp.cutcircleCC(stockXwidth/2, stockYheight/2, 0, -stockZthickness,
    radius)
108 gpcutdxfpy
109 gpcutdxfpy gcp.cutcircleCC(stockXwidth/2, stockYheight/2, -stockZthickness,
    -stockZthickness, radius*1.5)
110 gpcutdxfpy
111 gpcutdxfpy gcp.closeddffile()
112 gpcutdxfpy
113 gpcutdxfpy 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 DXF 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] */
81 gcptmplpy small_V_ratio = 0.625 # [0.25:2]
82 gcptmplpy # [Feeds and Speeds] */
83 gcptmplpy DT_ratio = 0.75 # [0.25:2]
84 gcptmplpy # [Feeds and Speeds] */
85 gcptmplpy KH_ratio = 0.75 # [0.25:2]
86 gcptmplpy # [Feeds and Speeds] */
87 gcptmplpy RO_ratio = 0.5 # [0.25:2]
88 gcptmplpy # [Feeds and Speeds] */
89 gcptmplpy MISC_ratio = 0.5 # [0.25:2]
90 gcptmplpy
91 gcptmplpy # Note that the various ratios are simply declared as a possible

```

```

        hook
92 gcptmplpy # which might be useful and how are handled is left as an exercise
93 gcptmplpy # for the reader and that they are not applied below.
94 gcptmplpy # One naive option might be to multiply by the feed rate
95 gcptmplpy # and divide by speeds.
96 gcptmplpy
97 gcptmplpy gcp = gcodepreview("cut", # "print" or "no_preview"
98 gcptmplpy                         generategcode,
99 gcptmplpy                         generatedxf,
100 gcptmplpy                        )
101 gcptmplpy
102 gcptmplpy gcp.opengcodefile(Base_filename)
103 gcptmplpy gcp.opendxfxf(file(Base_filename))
104 gcptmplpy
105 gcptmplpy gcp.setupstock(stockXwidth, stockYheight, stockZthickness,
                           zeroheight, stockzero, retractheight)
106 gcptmplpy
107 gcptmplpy gcp.movetosafeZ()
108 gcptmplpy
109 gcptmplpy gcp.toolchange(102, 10000 * small_square_ratio)
110 gcptmplpy
111 gcptmplpy gcp.rapidZ(0)
112 gcptmplpy
113 gcptmplpy gcp.cutlinedxfgc(stockXwidth/2, stockYheight/2, -stockZthickness)
114 gcptmplpy
115 gcptmplpy gcp.rapidZ(retractheight)
116 gcptmplpy gcp.toolchange(201, 10000)
117 gcptmplpy gcp.rapidXY(0, stockYheight/16)
118 gcptmplpy gcp.rapidZ(0)
119 gcptmplpy gcp.cutlinedxfgc(stockXwidth/16*7, stockYheight/2, -stockZthickness
                               )
120 gcptmplpy
121 gcptmplpy gcp.rapidZ(retractheight)
122 gcptmplpy gcp.toolchange(202, 10000)
123 gcptmplpy gcp.rapidXY(0, stockYheight/8)
124 gcptmplpy gcp.rapidZ(0)
125 gcptmplpy gcp.cutlinedxfgc(stockXwidth/16*6, stockYheight/2, -stockZthickness
                               )
126 gcptmplpy
127 gcptmplpy gcp.rapidZ(retractheight)
128 gcptmplpy gcp.toolchange(101, 10000)
129 gcptmplpy gcp.rapidXY(0, stockYheight/16*3)
130 gcptmplpy gcp.rapidZ(0)
131 gcptmplpy gcp.cutlinedxfgc(stockXwidth/16*5, stockYheight/2, -stockZthickness
                               )
132 gcptmplpy
133 gcptmplpy gcp.setzpos(retractheight)
134 gcptmplpy gcp.toolchange(390, 10000)
135 gcptmplpy gcp.rapidXY(0, stockYheight/16*4)
136 gcptmplpy gcp.rapidZ(0)
137 gcptmplpy gcp.cutlinedxfgc(stockXwidth/16*4, stockYheight/2, -stockZthickness
                               )
138 gcptmplpy gcp.rapidZ(retractheight)
139 gcptmplpy
140 gcptmplpy gcp.toolchange(301, 10000)
141 gcptmplpy gcp.rapidXY(0, stockYheight/16*6)
142 gcptmplpy gcp.rapidZ(0)
143 gcptmplpy gcp.cutlinedxfgc(stockXwidth/16*2, stockYheight/2, -stockZthickness
                               )
144 gcptmplpy
145 gcptmplpy rapids = gcp.rapid(gcp.xpos(), gcp.ypos(), retractheight)
146 gcptmplpy gcp.toolchange(102, 10000)
147 gcptmplpy
148 gcptmplpy gcp.rapid(-stockXwidth/4+stockYheight/16, +stockYheight/4, 0)
149 gcptmplpy
150 gcptmplpy #gcp.cutarcCC(0, 90, gcp.xpos()-stockYheight/16, gcp.ypos(),
                           stockYheight/16, -stockZthickness/4)
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.cutquarterCCNWdx(gcp.xpos() - stockYheight/8, gcp.ypos() -
                           stockYheight/8, -stockZthickness/2, stockYheight/8)
156 gcptmplpy gcp.cutquarterCCSWdx(gcp.xpos() + stockYheight/8, gcp.ypos() -
                           stockYheight/8, -stockZthickness/4, stockYheight/8)

```

```

    stockYheight/8, -stockZthickness * 0.75, stockYheight/8)
157 gcptmplpy gcp.cutquarterCCSEdx(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)

```

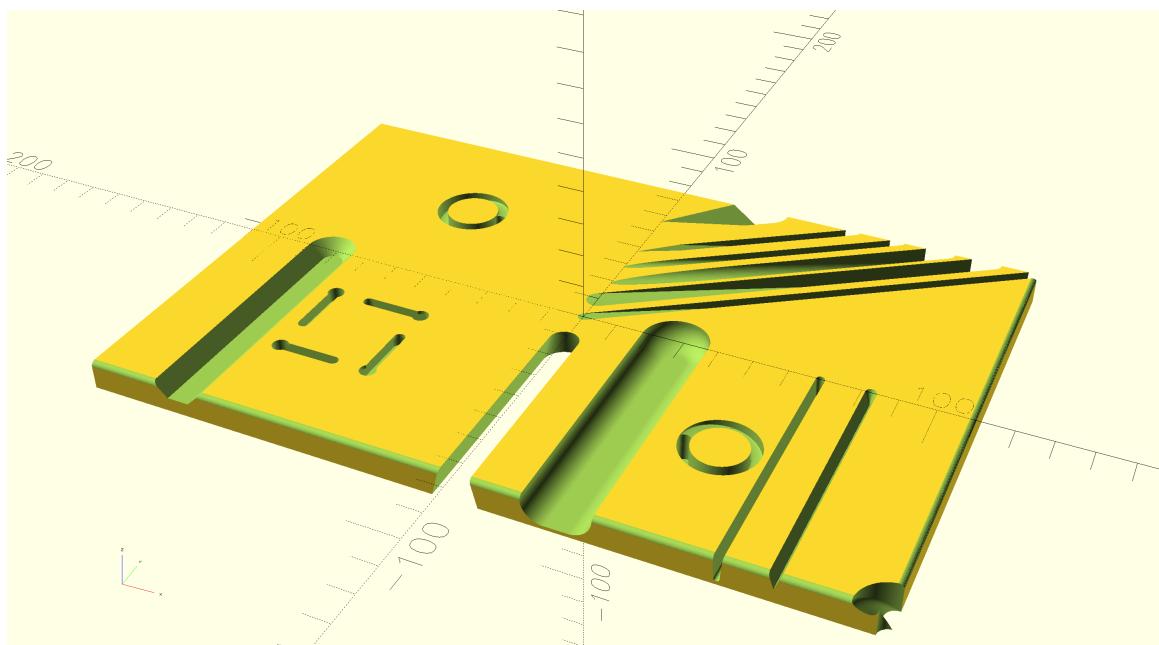
```

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.closeddfffile()

```

---

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));

```

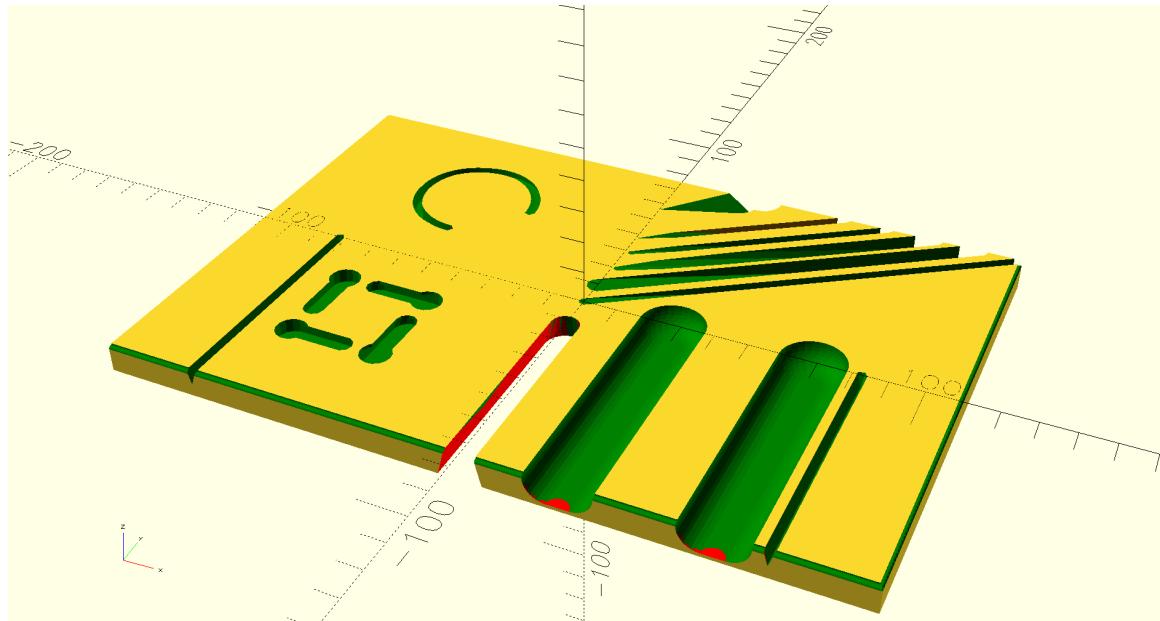
```

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 nozzlediameter = 0.4
26 gcpthreedp filamentdiameter = 1.75
27 gcpthreedp extrusionwidth = 0.6
28 gcpthreedp layerheight = 0.2
29 gcpthreedp
30 gcpthreedp gcp = gcodepreview("print", # "cut" or "no_preview"
31 gcpthreedp                         generategcode,
32 gcpthreedp                         generatedxf,
33 gcpthreedp                         )
34 gcpthreedp
35 gcpthreedp gcp.opengcodefile(Base_filename)
36 gcpthreedp
37 gcpthreedp gcp.initializeforprinting(nozzlediameter,
38 gcpthreedp                         filamentdiameter,
39 gcpthreedp                         extrusionwidth,
40 gcpthreedp                         layerheight)
41 gcpthreedp
42 gcpthreedp gcp.stockandtoolpaths("toolpaths")
43 gcpthreedp
44 gcpthreedp gcp.closegcodefile()

```

---

## 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 gcodepreview

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 4<sup>th</sup> additional axis may be worked up as a future option and supporting the work-around of two-sided (flip) machining by using an imported file as the Stock or preserving state and affording a second operation seems promising) and if desired, writing out DXF and/or G-code files (as opposed to the normal technique of rendering to a 3D model and writing out an STL or STEP or other model format and using a traditional CAM application). There are multiple modes for this, doing so may require loading up to two files:

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

```

import gcodepreview_standalone as gcp
or used in an OpenSCAD file:
use <gcodepreview.py>

```

with an additional OpenSCAD module which allows accessing it and that there is an option for loading directly from the Github repository implemented in PythonSCAD

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

Note that this architecture requires that many OpenSCAD modules are essentially “Dispatchers” (another term is “Descriptors”) which pass information from one aspect of the environment to another, but in some instances it 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 modeled as lists of `hull()` operations which must be processed as such and are differenced from the stock. The templates set up these options as noted, and ensure that `True == true`.

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

### 3.1 Cutviewer

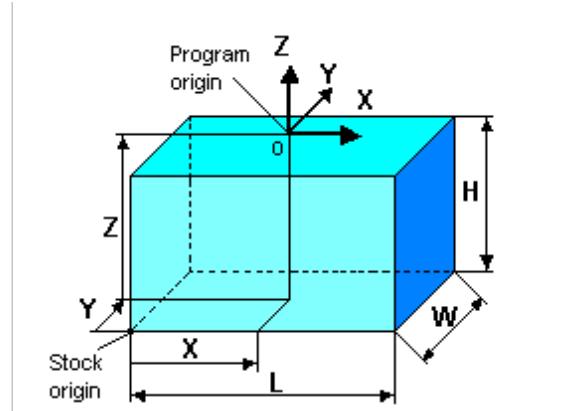
This problem space, showing the result of cutting stock using tooling in 3D has a number of tools addressing it, Camotics (formerly OpenSCAM) is an opensource option. Many tools simply create a wireframe preview such as <https://ncviewer.com/>. Cutviewer is a notable commercial program which has a unique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

#### 3.1.1 Stock size and placement

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

The CutViewer comments are in the form:

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



#### 3.1.2 Tool Shapes

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

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:



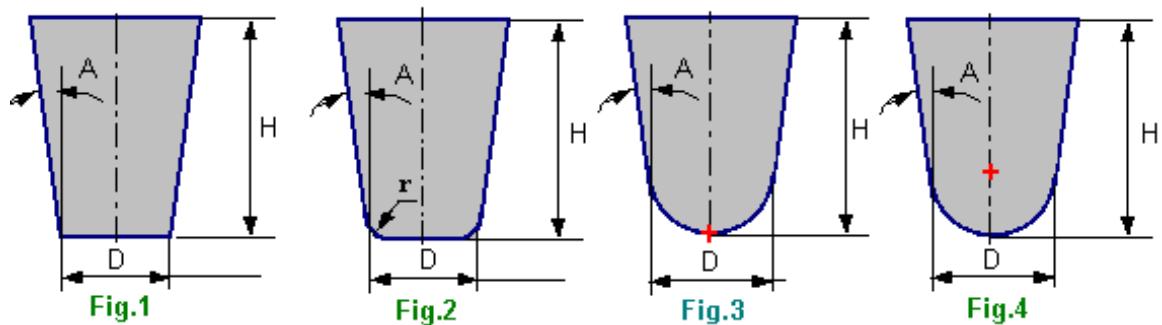
- Square (#201 and 102) — able to cut a flat bottom, perpendicular side and right angle, their simple and easily understood geometry makes them a standard choice
- Ballnose (#202 and 101) — rounded, they are the standard choice for concave and organic shapes
- V tooling (#301, 302, 311 and 312) — pointed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

### 3.1.2.1 Tool/Mill (Square, radiused, ball-nose, and tapered-ball)

The CutViewer values include:

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



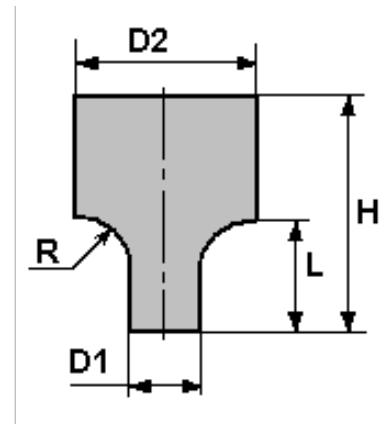
Note that it is possible to use these definitions for a wide variety of tooling, e.g., a Carbide 3D #301 V tool being represented as:

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

### 3.1.2.2 Corner Rounding, (roundover)

One notable tool option which cannot be supported using the Tool/Mill description is corner rounding/roundover tooling:

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



### 3.1.2.3 V shaped tooling (and variations)

Cutviewer has multiple V shaped tooling definitions:

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

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

## 3.2 Module Naming Convention

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

— CONFUCIUS

Naming modules 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. An exception is when there are multiple tool #s as when opening a file — collecting them all at the end is a more straight-forward approach.

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

- Both prefix and suffix
  - dxf (action (write out to DXF file), filetype)
- Prefixes
  - generate (Boolean) — used to identify which types of actions will be done (note that in the interest of brevity the check for this will be deferred until the last possible moment, see below)
  - write (action) — used to write to files, will include a check for the matching generate command, which being true will cause the write to the file to actually transpire
  - cut (action) — create tool movement removing volume from 3D object
  - 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	cutarc	cutarcgc
dxf	cutlinedxf	dxfline	cutarcdxf	dxfarcc	
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 will eventually be removed. In its place there is a mechanism where each colour is offset by the stock dimensions multiplied by the colour number, so that they are arrayed on a diagonal — when opened each such set of objects may then be easily selected and moved to the appropriate layer, then aligned against the stock.

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

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

Principles for naming modules (and variables):

- minimize use of underscores (for convenience sake, underscores are not used for index entries)
- identify which aspect of the project structure is being worked with (cut(ting), 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](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/](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 module not loaded.")
18 gcpy
19 gcpy def pygcpversion():
```

---

```

20 gcpy      thegcpversion = 0.93
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
- misc tool:
  - \* 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 gcptmpl #gcptemplate.txt --- this file will collect example usages of each
4 gcptmpl command with a brief commentary.
```

---

### 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             cutorprint = "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             cutorprint      : string
39 gcpy                         Enables creation of 3D model for cutting or
39 gcpy                         printing.
40 gcpy             generategcode : boolean
41 gcpy                         Enables writing out G-code.
42 gcpy             generatedxf   : boolean
43 gcpy                         Enables writing out DXF file(s).
44 gcpy
45 gcpy             Returns
46 gcpy             -----
47 gcpy             object
48 gcpy             The initialized gcodepreview object.
49 gcpy         """
50 gcpy         if cutorprint == "print":
51 gpy             self.generatecut = False
52 gpy             self.generateprint = True
53 gpy         elif cutorprint == "cut":
54 gpy             self.generatecut = True
55 gpy             self.generateprint = False
56 gpy         else: # no_preview
57 gpy             self.generatecut = False
58 gpy             self.generateprint = False
59 gpy         if generategcode == True:
60 gpy             self.generategcode = True
61 gpy         elif generategcode == 1:
62 gpy             self.generategcode = True
63 gpy         elif generategcode == 0:
64 gpy             self.generategcode = False
```

```

65 gcpy      else:
66 gcpy          self.generategcode = generategcode
67 gcpy      if generatedxf == True:
68 gcpy          self.generatedxf = True
69 gcpy      elif generatedxf == 1:
70 gcpy          self.generatedxf = True
71 gcpy      elif generatedxf == 0:
72 gcpy          self.generatedxf = False
73 gcpy      else:
74 gcpy          self.generatedxf = generatedxf
75 gcpy # unless multiple dxfs are enabled, the check for them is of course
76 gcpy          False
77 gcpy          self.generateddxfs = False
78 gcpy # set up 3D previewing parameters
79 gcpy          fa = gcpfa
80 gcpy          fs = gcpfs
81 gcpy          self.steps = steps
82 gcpy # initialize the machine state
83 gcpy          self.mc = "Initialized"
84 gcpy          self.mpx = float(0)
85 gcpy          self.mpy = float(0)
86 gcpy          self.mpz = 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
102 gcpy         toolchange command
103 gcpy         self.tooloutline = polygon( points
104 gcpy             = [[0,0],[3.175,0],[3.175,12.7],[0,12.7]] )
105 gcpy         self.toolprofile = polygon( points
106 gcpy             = [[0,0],[1.5875,0],[1.5875,12.7],[0,12.7]] )
107 gcpy         self.shaftoutline = polygon( points
108 gcpy             = [[0,12.7],[3.175,12.7],[3.175,25.4],[0,25.4]] )
109 gcpy         self.shaftprofile = polygon( points
110 gcpy             = [[0,12.7],[1.5875,12.7],[1.5875,25.4],[0,25.4]] )
111 gcpy         self.currenttoolshape = cylinder(h = self.flute, r = self.
112 gcpy             shaftdiameter/2)
113 gcpy         sh = cylinder(h = self.flute, r = self.shaftdiameter/2)
114 gcpy         self.currenttoolshaft = sh.translate([0,0,self.flute])
115 gcpy # debug mode requires a variable to track if it is on or off
116 gcpy #         self.debugenable = False
117 gcpy # the variables for holding 3D models must be initialized as empty
118 gcpy # lists so as to ensure that only append or extend commands are
119 gcpy # used with them
120 gcpy         self.rapids = []
121 gcpy         self.toolpaths = []
122 gcpy         print("gcodepreview\u2192class\u2192initialized")
123 gcpy #     def myfunc(self, var):
124 gcpy #         self.vv = var * var
125 gcpy #     return self.vv
126 gcpy #     def getvv(self):
127 gcpy #         return self.vv
128 gcpy #
129 gcpy #     def checkint(self):
130 gcpy #         return self.mc
131 gcpy #
132 gcpy #     def makecube(self, xdim, ydim, zdim):
133 gcpy #         self.c=cube([xdim, ydim, zdim])

```

---

### 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 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.

**toolmovements** There are two variables to record toolmovement, rapids and toolpaths. Initialized as empty lists, toolmovements will be extended to the lists, then for output, the lists will be expanded and 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             Returns
180 gcpy             -----
181 gcpy             none
182 gcpy             """
183 gcpy             self.stockXwidth = stockXwidth
184 gcpy             self.stockYheight = stockYheight
185 gcpy             self.stockZthickness = stockZthickness
186 gcpy             self.zeroheight = zeroheight
187 gcpy             self.stockzero = stockzero
188 gcpy             self.retractheight = retractheight
189 gcpy             self.stock = cube([stockXwidth, stockYheight,
190 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:  
194 gcpy                   self.writegc("(stockMin:0.00mm, 0.00mm, -", str  
195 gcpy                           (self.stockZthickness), "mm)")  
196 gcpy                   self.writegc("(stockMax:", str(self.stockXwidth  
197 gcpy                           ), "mm, ", str(stockYheight), "mm, 0.00mm)")  
198 gcpy                   self.writegc("(STOCK/BLOCK,", str(self.  
199 gcpy                           stockXwidth), ", ", str(self.stockYheight),  
200 gcpy                           ", ", str(self.stockZthickness), " 0.00
```

```

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

```

---

```

229 gcpy           if self.generategcode == True:
230 gcpy             self.writegc("(stockMin:0.00mm, " + str(self.
231 gcpy               stockYheight), " mm, " + str(0.00mm))")
232 gcpy             self.writegc("(stockMax:", str(self.stockXwidth
233 gcpy               ), " mm, " + str(0.00mm), " , " + str(self.stockZthickness)
234 gcpy               , " mm)")
235 gcpy             self.writegc("(STOCK/BLOCK," + str(self.
236 gcpy               stockXwidth), ", " + str(self.stockYheight),
237 gcpy               ", " + str(self.stockZthickness), ", " + str(0.00),
238 gcpy               ", " + str(self.stockYheight), ", " + str(0.00))
239 gcpy             if self.stockzero == "Center":
240 gcpy               self.stock = self.stock.translate([-self.
241 gcpy                 stockXwidth / 2, -self.stockYheight / 2, 0])
242 gcpy             if self.generategcode == True:
243 gcpy               self.writegc("(stockMin:" + str(-self.
244 gcpy                 stockXwidth/2), " , " + str(self.stockYheight/2),
245 gcpy                 " mm, " + str(0.00mm))")
246 gcpy               self.writegc("(stockMax:", str(self.stockXwidth
247 gcpy               /2), " mm, " + str(self.stockYheight/2), " mm,
248 gcpy               ", " + str(self.stockZthickness), " mm)")
249 gcpy               self.writegc("(STOCK/BLOCK," + str(self.
250 gcpy               stockXwidth), ", " + str(self.stockYheight),
251 gcpy               ", " + str(self.stockZthickness), ", " + str(
252 gcpy                 self.stockXwidth/2), " , " + str(self.
253 gcpy                 stockYheight/2), ", " + str(0.00)))
254 gcpy             if self.generategcode == True:
255 gcpy               self.writegc("G90");
256 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

```

---

**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_kw_args) -> None:
253 gcpy       """
254 gcpy         Print debug output if enabled.
255 gcpy         Accepts the same arguments as built-in print (except file
256 gcpy           is supported via print_kw_args).
257 gcpy
258 gcpy         if not self.debugenable:
259 gcpy           return
260 gcpy           # Build the message and print under a lock to avoid
261 gcpy             interleaving in multithreaded apps
262 gcpy           self.prefix = "DEBUG:" + str(self.prefix)
263 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:

---

```
97 gcptmpl self.debugenable = True
98 gcptmpl
99 gcptmpl testvariable = 1
100 gcptmpl
101 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:

---

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

---



---

```
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:

---

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

---

the OpenSCAD module is a descriptor as expected:

---

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

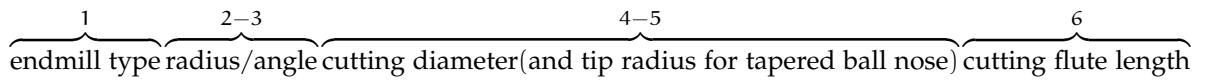
Note that it will also be necessary to write out a tool description compatible with the program CutViewer as a G-code comment so that it may be used as a 3D previewer for the G-code for tool changes in G-code. Several forms are available as described below.

### 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 ( $1/64^{\text{th}}$ )
  - 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
#251 == 100047 (also #201 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 (Amana 46170-K)
#326 == 100036 (also #102 and #274)
```

- Ball

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

- V

```
#301 == 390074
#302 == 360071
#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
```

- 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
```

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

- No way to indicate flute geometry beyond O-flute (which distinction will probably be removed)
- 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 9#####)

A further consideration is that it is not possible to represent tools unambiguously, so that given a tool definition it is possible to derive the manufacturer's tool number, *e.g.*, 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, the generated number will be used internally, then the saved manufacturer number will be exported to the G-code file, or used when generating a DXF filename for a given set of tool movements.

---

275 gcpy	<code>def currenttoolnumber(self):</code>
276 gcpy	<code>    return(self.currenttoolnum)</code>

---

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

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

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

- `radius`
- `angle`

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

`tool number` **3.4.1.1 toolchange** This command accepts a tool number and assigns its characteristics as 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 `misc`.

The original implementation created the model for the tool at the current position, and a duplicate at the end position, wrapping the twain for each end of a given movement in a `hull()` command and then applying a union. This approach will not work within Python, so it will be necessary to instead assign and select the tool as part of the `toolmovement` command.

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` and a second 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 state where tools are defined in a vendor-neutral fashion.

---

```

278 gcpy      def settoolparameters(self, tooltype, first, second, third,
279 gcpy          fourth, length = 0):
280 gcpy          if tooltype == "mill":
281 gcpy              diameter = first
282 gcpy              cornerradius = second
283 gcpy              height = third
284 gcpy              taperangle = fourth
284 gcpy              if cornerradius == 0:
285 gcpy #M6T122 (TOOL/MILL,0.80, 0.00, 1.59, 0.00)
286 gcpy #M6T112 (TOOL/MILL,1.59, 0.00, 6.35, 0.00)
287 gcpy #M6T102 (TOOL/MILL,3.17, 0.00, 12.70, 0.00)
288 gcpy #M6T201 (TOOL/MILL,6.35, 0.00, 19.05, 0.00)
289 gcpy #M6T205 (TOOL/MILL,6.35, 0.00, 25.40, 0.00)
290 gcpy #M6T251 (TOOL/MILL,6.35, 0.00, 19.05, 0.00)
291 gcpy #M6T322 (TOOL/MILL,6.35, 0.00, 19.05, 0.00)
292 gcpy #M6T324 (TOOL/MILL,6.35, 0.00, 22.22, 0.00)
293 gcpy #M6T326 (TOOL/MILL,3.17, 0.00, 12.70, 0.00)
294 gcpy #M6T602 (TOOL/MILL,25.40, 0.00, 9.91, 0.00)
295 gcpy #M6T603 (TOOL/MILL,25.40, 0.00, 9.91, 0.00)
296 gcpy #M6T274 (TOOL/MILL,3.17, 0.00, 12.70, 0.00)
297 gcpy #M6T278 (TOOL/MILL,6.35, 0.00, 19.05, 0.00)

```

```

298 gcpy #M6T282 (TOOL/MILL,2.00, 0.00, 6.35, 0.00)
299 gcpy self.endmilltype = "square"
300 gcpy self.diameter = diameter
301 gcpy self.flute = height
302 gcpy self.shaftdiameter = diameter
303 gcpy self.shaftheight = height
304 gcpy self.shaftlength = height
305 gcpy #
306 gcpy elif cornerradius > 0 and taperangle == 0:
307 gcpy #M6T121 (TOOL/MILL,0.80, 0.40, 1.59, 0.00)
308 gcpy #M6T111 (TOOL/MILL,1.59, 0.79, 6.35, 0.00)
309 gcpy #M6T101 (TOOL/MILL,3.17, 1.59, 12.70, 0.00)
310 gcpy #M6T202 (TOOL/MILL,6.35, 3.17, 19.05, 0.00)
311 gcpy #M6T325 (TOOL/MILL,6.35, 3.17, 25.40, 0.00)
312 gcpy self.endmilltype = "ball"
313 gcpy self.diameter = diameter
314 gcpy self.flute = height
315 gcpy self.shaftdiameter = diameter
316 gcpy self.shaftheight = height
317 gcpy self.shaftlength = height
318 gcpy #
319 gcpy elif taperangle > 0:
320 gcpy #M6T301 (TOOL/MILL,0.10, 0.05, 6.35, 45.00)
321 gcpy #M6T302 (TOOL/MILL,0.10, 0.05, 6.35, 30.00)
322 gcpy #M6T327 (TOOL/MILL,0.10, 0.05, 23.39, 30.00)
323 gcpy self.endmilltype = "V"
324 gcpy self.diameter = Tan(taperangle / 2) * height
325 gcpy self.flute = height
326 gcpy self.angle = taperangle
327 gcpy self.shaftdiameter = Tan(taperangle / 2) * height
328 gcpy self.shaftheight = height
329 gcpy self.shaftlength = height
330 gcpy #
331 gcpy elif tooltype == "chamfer":
332 gcpy tipdiameter = first
333 gcpy radius = second
334 gcpy height = third
335 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.

```

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

```

---

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

```

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

```

352 gcpy     self.definesquaretool(self.diameter, self.shaftheight,
353 gcpy             self.shaftlength)
354 gcpy     self.defineshaft(self.diameter, self.shaftdiameter,
355 gcpy             self.flute, 0, self.shaftlength)
356 gcpy     self.toolnumber = 10003

```

```

355 gcpy      elif (tool_number == 201) or (tool_number == 100047): #
201/251/322 (Amana 46202-K) == 100047
356 gcpy      self.writegc("(TOOL/MILL, 6.35, 0.00, 0.00, 0.00)")
357 gcpy      self.endmilltype = "square"
358 gcpy      self.diameter = 6.35
359 gcpy      self.flute = 19.05
360 gcpy      self.shaftdiameter = 6.35
361 gcpy      self.shaftheight = 19.05
362 gcpy      self.shaftlength = 20.0
363 gcpy      self.definesquaretool(self.diameter, self.shaftheight,
364 gcpy          self.shaftlength)
364 gcpy      self.defineshaft(self.diameter, self.shaftdiameter,
365 gcpy          self.flute, 0, self.shaftlength)
365 gcpy      self.toolnumber = "100047"
366 gcpy      elif (tool_number == 112) or (tool_number == 100024): #112
366 gcpy          == 100024
367 gcpy          self.writegc("(TOOL/MILL, 1.5875, 0.00, 0.00, 0.00)")
368 gcpy          self.endmilltype = "square"
369 gcpy          self.diameter = 1.5875
370 gcpy          self.flute = 6.35
371 gcpy          self.shaftdiameter = 3.175
372 gcpy          self.shaftheight = 6.35
373 gcpy          self.shaftlength = 12.0
374 gcpy          self.definesquaretool(self.diameter, self.shaftheight,
374 gcpy              self.shaftlength, (self.shaftdiameter - self.
374 gcpy                  diameter)/2)
375 gcpy          self.defineshaft(self.diameter, self.shaftdiameter,
375 gcpy              self.flute, 0, self.shaftlength)
376 gcpy          self.toolnumber = "100024"
377 gcpy      elif (tool_number == 122) or (tool_number == 100012): #122
377 gcpy          == 100012
378 gcpy          self.writegc("(TOOL/MILL, 0.79375, 0.00, 0.00, 0.00)")
379 gcpy          self.endmilltype = "square"
380 gcpy          self.diameter = 0.79375
381 gcpy          self.flute = 1.5875
382 gcpy          self.shaftdiameter = 3.175
383 gcpy          self.shaftheight = 1.5875
384 gcpy          self.shaftlength = 12.0
385 gcpy          self.definesquaretool(self.diameter, self.shaftheight,
385 gcpy              self.shaftlength, (self.shaftdiameter - self.
385 gcpy                  diameter)/2)
386 gcpy          self.defineshaft(self.diameter, self.shaftdiameter,
386 gcpy              self.flute, 0, self.shaftlength)
387 gcpy          self.toolnumber = "100012"
388 gcpy      elif (tool_number == 324): #324 (Amana 46170-K) == 100048
388 gcpy          self.writegc("(TOOL/MILL, 6.35, 0.00, 0.00, 0.00)")
389 gcpy          self.endmilltype = "square"
390 gcpy          self.diameter = 6.35
391 gcpy          self.flute = 22.225
392 gcpy          self.shaftdiameter = 6.35
393 gcpy          self.shaftheight = 22.225
394 gcpy          self.shaftlength = 20.0
395 gcpy          self.definesquaretool(self.diameter, self.shaftheight,
395 gcpy              self.shaftlength)
396 gcpy          self.defineshaft(self.diameter, self.shaftdiameter,
396 gcpy              self.flute, 0, self.shaftlength)
397 gcpy          self.toolnumber = "100048"
398 gcpy      elif (tool_number == 205) or (tool_number == 100048): #205
398 gcpy          == 100048
399 gcpy          self.writegc("(TOOL/MILL, 6.35, 0.00, 0.00, 0.00)")
400 gcpy          self.endmilltype = "square"
401 gcpy          self.diameter = 6.35
402 gcpy          self.flute = 25.4
403 gcpy          self.shaftdiameter = 6.35
404 gcpy          self.shaftheight = 25.4
405 gcpy          self.shaftlength = 20.0
406 gcpy          self.definesquaretool(self.diameter, self.shaftheight,
406 gcpy              self.shaftlength)
407 gcpy          self.defineshaft(self.diameter, self.shaftdiameter,
407 gcpy              self.flute, 0, self.shaftlength)
408 gcpy          defineKeyholetool(self.diameter, self.flute, self.
408 gcpy              shaftdiameter, self.shaftheight, self.shaftdiameter,
408 gcpy                  self.shaftlength)
409 gcpy          self.toolnumber = "100048"
410 gcpy          410 gcpy #
411 gcpy      #

```

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

---

```

412 gcpy      elif (tool_number == 282) or (tool_number == 100204): #282
413 gcpy          == 000204
414 gcpy          self.writegc("(TOOL/MILL, 2.0, 0.00, 0.00, 0.00)")
415 gcpy          self.endmilltype = "0-flute"
416 gcpy          self.diameter = 2.0
417 gcpy          self.flute = 6.35
418 gcpy          self.shaftdiameter = 6.35
419 gcpy          self.shaftheight = 6.35
420 gcpy          self.definesquaretool(self.diameter, self.shaftheight,
421 gcpy              self.shaftlength, (self.shaftdiameter - self.
422 gcpy                  diameter)/2)
423 gcpy          self.definshaft(self.diameter, self.shaftdiameter,
424 gcpy              self.flute, 0, self.shaftlength)
425 gcpy          self.toolnumber = "100204"
426 gcpy      elif (tool_number == 274) or (tool_number == 100036): #274
427 gcpy          == 000036
428 gcpy          self.writegc("(TOOL/MILL, 3.175, 0.00, 0.00, 0.00)")
429 gcpy          self.endmilltype = "0-flute"
430 gcpy          self.diameter = 3.175
431 gcpy          self.flute = 12.7
432 gcpy          self.shaftdiameter = 3.175
433 gcpy          self.shaftheight = 12.7
434 gcpy          self.shaftlength = 20.0
435 gcpy          self.definesquaretool(self.diameter, self.shaftheight,
436 gcpy              self.shaftlength)
437 gcpy          self.definshaft(self.diameter, self.shaftdiameter,
438 gcpy              self.flute, 0, self.shaftlength)
439 gcpy          self.toolnumber = "100036"
440 gcpy      elif (tool_number == 278) or (tool_number == 100047): #278
441 gcpy          == 000047
442 gcpy          self.writegc("(TOOL/MILL, 6.35, 0.00, 0.00, 0.00)")
443 gcpy          self.endmilltype = "0-flute"
444 gcpy          self.diameter = 6.35
445 gcpy          self.flute = 19.05
446 gcpy          self.shaftdiameter = 3.175
447 gcpy          self.shaftheight = 19.05
448 gcpy          self.shaftlength = 20.0
449 gcpy          self.definesquaretool(self.diameter, self.shaftheight,
450 gcpy              self.shaftlength)
451 gcpy          self.definshaft(self.diameter, self.shaftdiameter,
452 gcpy              self.flute, 0, self.shaftlength)
453 gcpy          self.toolnumber = "100047"
454 gcpy      #

```

---

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

---

```

446 gcpy      elif (tool_number == 202) or (tool_number == 204047): #202
447 gcpy          == 204047
448 gcpy          self.writegc("(TOOL/MILL, 6.35, 3.175, 0.00, 0.00)")
449 gcpy          self.endmilltype = "ball"
450 gcpy          self.diameter = 6.35
451 gcpy          self.flute = 19.05
452 gcpy          self.shaftdiameter = 6.35
453 gcpy          self.shaftheight = 19.05
454 gcpy          self.shaftlength = 20.0
455 gcpy          self.defineballnosetool(self.diameter, self.flute, self.
456 gcpy              .shaftlength)
457 gcpy          self.definshaft(self.diameter, self.shaftdiameter,
458 gcpy              self.flute, 0, self.shaftlength)
459 gcpy          self.toolnumber = "204047"
460 gcpy      elif (tool_number == 101) or (tool_number == 203036): #101
461 gcpy          == 203036
462 gcpy          self.writegc("(TOOL/MILL, 3.175, 1.5875, 0.00, 0.00)")
463 gcpy          self.endmilltype = "ball"
464 gcpy          self.diameter = 3.175
465 gcpy          self.flute = 12.7

```

```

        .shaftlength)
466 gcpy    self.defineshaft(self.diameter, self.shaftdiameter,
                           self.flute, 0, self.shaftlength)
467 gcpy    self.toolnumber = "203036"
468 gcpy    elif (tool_number == 111) or (tool_number == 202024): #111
               == 202024
469 gcpy    self.writegc("(TOOL/MILL, 1.5875, 0.79375, 0.00, 0.00)"
                           )
470 gcpy    self.endmilltype = "ball"
471 gcpy    self.diameter = 1.5875
472 gcpy    self.flute = 6.35
473 gcpy    self.shaftdiameter = 3.175
474 gcpy    self.shaftheight = 6.35
475 gcpy    self.shaftlength = 20.0
476 gcpy    self.defineballnosetool(self.diameter, self.flute, self.
                           .shaftlength, (self.shaftdiameter - self.diameter)
                           /2)
477 gcpy    self.defineshaft(self.diameter, self.shaftdiameter,
                           self.flute, 0, self.shaftlength)
478 gcpy    self.toolnumber = "202024"
479 gcpy    elif (tool_number == 121) or (tool_number == 201012): #121
               == 201012
480 gcpy    self.writegc("(TOOL/MILL, 3.175, 0.79375, 0.00, 0.00)")
481 gcpy    self.endmilltype = "ball"
482 gcpy    self.diameter = 0.79375
483 gcpy    self.flute = 1.5875
484 gcpy    self.shaftdiameter = 3.175
485 gcpy    self.shaftheight = 1.5875
486 gcpy    self.shaftlength = 20.0
487 gcpy    self.defineballnosetool(self.diameter, self.flute, self.
                           .shaftlength, (self.shaftdiameter - self.diameter)
                           /2)
488 gcpy    self.defineshaft(self.diameter, self.shaftdiameter,
                           self.flute, 0, self.shaftlength)
489 gcpy    self.toolnumber = "201012"
490 gcpy    elif (tool_number == 325) or (tool_number == 204048): #325
               (Amana 46376-K) == 204048
491 gcpy    self.writegc("(TOOL/MILL, 6.35, 3.175, 0.00, 0.00)")
492 gcpy    self.endmilltype = "ball"
493 gcpy    self.diameter = 6.35
494 gcpy    self.flute = 25.4
495 gcpy    self.shaftdiameter = 6.35
496 gcpy    self.shaftheight = 25.4
497 gcpy    self.shaftlength = 20.0
498 gcpy    self.defineballnosetool(self.diameter, self.flute, self.
                           .shaftlength, (self.shaftdiameter - self.diameter)
                           /2)
499 gcpy    self.defineshaft(self.diameter, self.shaftdiameter,
                           self.flute, 0, self.shaftlength)
500 gcpy    self.toolnumber = "204048"
501 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).

```

502 gcpy    elif (tool_number == 301) or (tool_number == 390074): #301
               == 390074
503 gcpy    self.writegc("(TOOL/MILL, 0.10, 0.05, 6.35, 45.00)")
504 gcpy    self.endmilltype = "V"
505 gcpy    self.diameter = 12.7
506 gcpy    self.flute = 6.35
507 gcpy    self.angle = 90
508 gcpy    self.shaftdiameter = 6.35
509 gcpy    self.shaftheight = 6.35
510 gcpy    self.shaftlength = 20.0
511 gcpy    self.defineVtool(self.diameter, self.flute, self.
                           shaftlength, self.shaftdiameter)
512 gcpy    self.toolnumber = "390074"
513 gcpy    elif (tool_number == 302) or (tool_number == 360071): #302
               == 360071
514 gcpy    self.writegc("(TOOL/MILL, 0.10, 0.05, 6.35, 30.00)")
515 gcpy    self.endmilltype = "V"
516 gcpy    self.diameter = 12.7
517 gcpy    self.flute = 11.067
518 gcpy    self.angle = 60

```

```

519 gcpy      self.shaftdiameter = 6.35
520 gcpy      self.shaftheight = 11.067
521 gcpy      self.shaftlength = 20.0
522 gcpy      self.defineVtool(self.diameter, self.flute, self.
                           shaftlength, self.shaftdiameter)
523 gcpy      self.toolnumber = "360071"
524 gcpy      elif (tool_number == 390) or (tool_number == 390032): #390
                  == 390032
525 gcpy      self.writegc("(TOOL/MILL,0.03,0.00,1.5875,45.00)")
526 gcpy      self.endmilltype = "V"
527 gcpy      self.diameter = 3.175
528 gcpy      self.flute = 1.5875
529 gcpy      self.angle = 90
530 gcpy      self.shaftdiameter = 3.175
531 gcpy      self.shaftheight = 1.5875
532 gcpy      self.shaftlength = 20.0
533 gcpy      self.defineVtool(self.diameter, self.flute, self.
                           shaftlength, self.shaftdiameter)
534 gcpy      self.toolnumber = "390032"
535 gcpy      elif (tool_number == 327) or (tool_number == 360098): #327
                  (Amana RC-1148) == 360098
536 gcpy      self.writegc("(TOOL/MILL,0.03,0.00,13.4874,30.00)")
537 gcpy      self.endmilltype = "V"
538 gcpy      self.diameter = 25.4
539 gcpy      self.flute = 22.134
540 gcpy      self.angle = 60
541 gcpy      self.shaftdiameter = 6.35
542 gcpy      self.shaftheight = 22.134
543 gcpy      self.shaftlength = 20.0
544 gcpy      self.defineVtool(self.diameter, self.flute, self.
                           shaftlength, self.shaftdiameter)
545 gcpy      self.toolnumber = "360098"
546 gcpy      elif (tool_number == 323) or (tool_number == 330041): #323
                  == 330041 30 degree V Amana, 45771-K
547 gcpy      self.writegc("(TOOL/MILL,0.10,0.05,11.18,15.00)")
548 gcpy      self.endmilltype = "V"
549 gcpy      self.diameter = 6.35
550 gcpy      self.flute = 11.849
551 gcpy      self.angle = 30
552 gcpy      self.shaftdiameter = 6.35
553 gcpy      self.shaftheight = 11.849
554 gcpy      self.shaftlength = 20.0
555 gcpy      self.defineVtool(self.diameter, self.flute, self.
                           shaftlength, self.shaftdiameter)
556 gcpy      self.toolnumber = "330041"
557 gcpy #

```

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

```
558 gcpy
      elif (tool_number == 374) or (tool_number == 906043): #374
          == 906043
              self.writegc("(TOOL/MILL, 9.53, 0.00, 3.17, 0.00)")
              self.endmilltype = "keyhole"
              self.diameter = 9.525
              self.flute = 3.175
              self.radius = 6.35
              self.shaftdiameter = 6.35
              self.shaftheight = 3.175
              self.shaftlength = 20.0
              self.defineKeyholetool(self.diameter, self.flute, self.
                  shaftdiameter, self.shaftheight, self.shaftdiameter,
                  self.shaftlength)
              self.toolnumber = "906043"
      elif (tool_number == 375) or (tool_number == 906053): #375
          == 906053
              self.writegc("(TOOL/MILL, 9.53, 0.00, 3.17, 0.00)")
              self.endmilltype = "keyhole"
              self.diameter = 9.525
              self.flute = 3.175
              self.radius = 8
              self.shaftdiameter = 6.35
              self.shaftheight = 3.175
              self.shaftlength = 20.0
              self.defineKeyholetool(self.diameter, self.flute, self.
                  shaftdiameter, self.shaftheight, self.shaftdiameter,
                  self.shaftlength)
              self.toolnumber = "906053"
559 gcpy
560 gcpy
561 gcpy
562 gcpy
563 gcpy
564 gcpy
565 gcpy
566 gcpy
567 gcpy
568 gcpy
569 gcpy
570 gcpy
571 gcpy
572 gcpy
573 gcpy
574 gcpy
575 gcpy
576 gcpy
577 gcpy
578 gcpy
579 gcpy
```

```

580 gcpy      elif (tool_number == 376) or (tool_number == 907040): #376
               == 907040
581 gcpy      self.writegc("(TOOL/MILL,12.7,0.00,4.77,0.00)")
582 gcpy      self.endmilltype = "keyhole"
583 gcpy      self.diameter = 12.7
584 gcpy      self.flute = 4.7625
585 gcpy      self.radius = 6.35
586 gcpy      self.shaftdiameter = 6.35
587 gcpy      self.shaftheight = 4.7625
588 gcpy      self.shaftlength = 20.0
589 gcpy      self.defineKeyholetool(self.diameter, self.flute, self.
               shaftdiameter, self.shaftheight, self.shaftdiameter,
               self.shaftlength)
590 gcpy      self.toolnumber = "907040"
591 gcpy      elif (tool_number == 378) or (tool_number == 907050): #378
               == 907050
592 gcpy      self.writegc("(TOOL/MILL,12.7,0.00,4.77,0.00)")
593 gcpy      self.endmilltype = "keyhole"
594 gcpy      self.diameter = 12.7
595 gcpy      self.flute = 4.7625
596 gcpy      self.radius = 8
597 gcpy      self.shaftdiameter = 6.35
598 gcpy      self.shaftheight = 4.7625
599 gcpy      self.shaftlength = 20.0
600 gcpy      self.defineKeyholetool(self.diameter, self.flute, self.
               shaftdiameter, self.shaftheight, self.shaftdiameter,
               self.shaftlength)
601 gcpy      self.toolnumber = "907050"
602 gcpy #

```

---

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

```

603 gcpy      elif (tool_number == 45981): #45981 == 445981
604 gcpy #Amana Carbide Tipped Bowl & Tray 1/8 Radius x 1/2 Dia x 1/2 x 1/4
               Inch Shank
605 gcpy      self.writegc("(TOOL/MILL,0.03,0.00,10.00,30.00)")
606 gcpy      self.writegc("(TOOL/MILL,15.875,6.35,19.05,0.00)")
607 gcpy      self.endmilltype = "bowl"
608 gcpy      self.diameter = 12.7
609 gcpy      self.flute = 12.7
610 gcpy      self.radius = 3.175
611 gcpy      self.shaftdiameter = 6.35
612 gcpy      self.shaftheight = 12.7
613 gcpy      self.shaftlength = 20.0
614 gcpy      self.definebowltool(self.diameter, self.flute, self.
               radius, self.shaftdiameter, self.shaftlength)
615 gcpy      self.toolnumber = "445981"
616 gcpy      elif (tool_number == 45982):#0.507/2, 4.509
617 gcpy      self.writegc("(TOOL/MILL,15.875,6.35,19.05,0.00)")
618 gcpy      self.endmilltype = "bowl"
619 gcpy      self.diameter = 19.05
620 gcpy      self.flute = 15.875
621 gcpy      self.radius = 6.35
622 gcpy      self.shaftdiameter = 6.35
623 gcpy      self.shaftheight = 15.875
624 gcpy      self.shaftlength = 20.0
625 gcpy      self.definebowltool(self.diameter, self.flute, self.
               radius, self.shaftdiameter, self.shaftlength)
626 gcpy      self.toolnumber = "445982"
627 gcpy      elif (tool_number == 1370): #1370 == 401370
628 gcpy #Whiteside Bowl & Tray Bit 1/4"SH, 1/8"R, 7/16"CD (5/16" cutting
               flute length)
629 gcpy      self.writegc("(TOOL/MILL,11.1125,8,3.175,0.00)")
630 gcpy      self.endmilltype = "bowl"
631 gcpy      self.diameter = 11.1125
632 gcpy      self.flute = 8
633 gcpy      self.radius = 3.175
634 gcpy      self.shaftdiameter = 6.35
635 gcpy      self.shaftheight = 8
636 gcpy      self.shaftlength = 20.0
637 gcpy      self.definebowltool(self.diameter, self.flute, self.
               radius, self.shaftdiameter, self.shaftlength)
638 gcpy      self.toolnumber = "401370"
639 gcpy      elif (tool_number == 1372): #1372/45982 == 401372
640 gcpy #Whiteside Bowl & Tray Bit 1/4"SH, 1/4"R, 3/4"CD (5/8" cutting
               flute length)
641 gcpy #Amana Carbide Tipped Bowl & Tray 1/4 Radius x 3/4 Dia x 5/8 x 1/4

```

```

Inch Shank
642 gcpy      self.writegc("(TOOL/MILL,19.5,15.875,6.35,0.00)")
643 gcpy      self.endmilltype = "bowl"
644 gcpy      self.diameter = 19.5
645 gcpy      self.flute = 15.875
646 gcpy      self.radius = 6.35
647 gcpy      self.shaftdiameter = 6.35
648 gcpy      self.shaftheight = 15.875
649 gcpy      self.shaftlength = 20.0
650 gcpy      self.definebowltool(self.diameter, self.flute, self.
                           radius, self.shaftdiameter, self.shaftlength)
651 gcpy      self.toolnumber = "401372"
652 gcpy #

```

---

**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.

```

653 gcpy      elif (tool_number == 501) or (tool_number == 530131): #501
               == 530131
654 gcpy      self.writegc("(TOOL/MILL,0.03,0.00,10.00,30.00)")
655 gcpy #
               self.currenttoolshape = self.toolshapes("tapered ball
               ", 3.175, 5.561, 30, 0.254)
656 gcpy      self.tooloutline = osimport("501_outline.svg")
657 gcpy      self.toolprofile = osimport("501_profile.svg")
658 gcpy      self.endmilltype = "tapered_ball"
659 gcpy      self.diameter = 3.175
660 gcpy      self.flute = 5.561
661 gcpy      self.angle = 30
662 gcpy      self.tip = 0.254
663 gcpy      self.shaftdiameter = 3.175
664 gcpy      self.shaftheight = 5.561
665 gcpy      self.shaftlength = 10.0
666 gcpy      self.toolnumber = "530131"
667 gcpy      elif (tool_number == 502) or (tool_number == 540131): #502
               == 540131
668 gcpy      self.writegc("(TOOL/MILL,0.03,0.00,10.00,20.00)")
669 gcpy #
               self.currenttoolshape = self.toolshapes("tapered ball
               ", 3.175, 4.117, 40, 0.254)
670 gcpy      self.endmilltype = "tapered_ball"
671 gcpy      self.diameter = 3.175
672 gcpy      self.flute = 4.117
673 gcpy      self.angle = 40
674 gcpy      self.tip = 0.254
675 gcpy      self.shaftdiameter = 3.175
676 gcpy      self.shaftheight = 4.117
677 gcpy      self.shaftlength = 10.0
678 gcpy      self.toolnumber = "540131"
679 gcpy #
               elif (tool_number == 204):#
680 gcpy #
               self.writegc("(")
681 gcpy #
               self.currenttoolshape = self.tapered_ball(1.5875,
               6.35, 38.1, 3.6)
682 gcpy #
               elif (tool_number == 304):#
683 gcpy #
               self.writegc("(")
684 gcpy #
               self.currenttoolshape = self.tapered_ball(3.175, 6.35,
               38.1, 2.4)
685 gcpy #

```

---

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

---

```

686 gcpy      elif (tool_number == 56125) or (tool_number == 603042):#
               0.508/2, 1.531 56125 == 603042
687 gcpy      self.writegc("(TOOL/CRMILL, 0.508, 6.35, 3.175, 7.9375,
                         3.175)")
688 gcpy      self.endmilltype = "roundover"
689 gcpy      self.tipdiameter = 0.508
690 gcpy      self.diameter = 6.35 - self.tipdiameter
691 gcpy      self.flute = 8 - self.tipdiameter
692 gcpy      self.radius = 3.175 - self.tipdiameter/2
693 gcpy      self.shaftdiameter = 6.35
694 gcpy      self.shaftheight = 8
695 gcpy      self.shaftlength = 10.0
696 gcpy      self.defineRoundoverTool(self.diameter, self.
                                         tipdiameter, self.flute, self.radius, self.
                                         shaftdiameter, self.shaftlength)
697 gcpy      self.toolnumber = "603042"
698 gcpy      elif (tool_number == 56142) or (tool_number == 602032):#
               0.508/2, 2.921 56142 == 602032
699 gcpy      self.writegc("(TOOL/CRMILL, 0.508, 3.571875, 1.5875, 5.
                           55625, 1.5875)")
700 gcpy      self.endmilltype = "roundover"
701 gcpy      self.tip = 0.508
702 gcpy      self.diameter = 3.175 - self.tip
703 gcpy      self.flute = 4.7625 - self.tip
704 gcpy      self.radius = 1.5875 - self.tip/2
705 gcpy      self.shaftdiameter = 3.175
706 gcpy      self.shaftheight = 4.7625
707 gcpy      self.shaftlength = 10.0
708 gcpy      self.toolnumber = "602032"
709 gcpy #     elif (tool_number == 312):#1.524/2, 3.175
710 gcpy #     self.writegc("(TOOL/CRMILL, Diameter1, Diameter2,
                           Radius, Height, Length)")
711 gcpy #     elif (tool_number == 1568):#0.507/2, 4.509 1568 == 603032
712 gcpy ##FIX      self.writegc("(TOOL/CRMILL, 0.17018, 9.525,
                           4.7625, 12.7, 4.7625)")
713 gcpy ##      self.currenttoolshape = self.toolshapes("roundover",
               3.175, 6.35, 3.175, 0.396875)
714 gcpy #     self.endmilltype = "roundover"
715 gcpy #     self.diameter = 3.175
716 gcpy #     self.flute = 6.35
717 gcpy #     self.radius = 3.175
718 gcpy #     self.tip = 0.396875
719 gcpy #     self.toolnumber = "603032"
720 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
721 gcpy #     elif (tool_number == 1570):#0.507/2, 4.509 1570 == 600002
               ?!?
722 gcpy #     self.writegc("(TOOL/CRMILL, 0.17018, 9.525, 4.7625,
               12.7, 4.7625)")
723 gcpy ##      self.currenttoolshape = self.toolshapes("roundover",
               4.7625, 9.525, 4.7625, 0.396875)
724 gcpy #     self.endmilltype = "roundover"
725 gcpy #     self.diameter = 4.7625
726 gcpy #     self.flute = 9.525
727 gcpy #     self.radius = 4.7625
728 gcpy #     self.tip = 0.396875
729 gcpy #     self.toolnumber = "600002"
730 gcpy #     elif (tool_number == 1572): #1572 = 604042
731 gcpy ##FIX      self.writegc("(TOOL/CRMILL, 0.17018, 9.525,
               4.7625, 12.7, 4.7625)")
732 gcpy ##      self.currenttoolshape = self.toolshapes("roundover",
               6.35, 12.7, 6.35, 0.396875)
733 gcpy #     self.endmilltype = "roundover"
734 gcpy #     self.diameter = 6.35
735 gcpy #     self.flute = 12.7
736 gcpy #     self.radius = 6.35
737 gcpy #     self.tip = 0.396875
738 gcpy #     self.toolnumber = "604042"
739 gcpy #     elif (tool_number == 1574): #1574 == 600062
740 gcpy ##FIX      self.writegc("(TOOL/CRMILL, 0.17018, 9.525,
               4.7625, 12.7, 4.7625)")
741 gcpy ##      self.currenttoolshape = self.toolshapes("roundover",
               9.525, 19.5, 9.515, 0.396875)
742 gcpy #     self.endmilltype = "roundover"
743 gcpy #     self.diameter = 9.525
744 gcpy #     self.flute = 19.5
745 gcpy #     self.radius = 9.515
746 gcpy #     self.tip = 0.396875

```

---

```
747 gcpy #           self.toolnumber = "600062"
748 gcpy #
```

---

**3.4.1.10 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).

---

```
749 gcpy      elif (tool_number == 814) or (tool_number == 814071): #814
               == 814071
750 gcpy #Item 18J1607, 1/2" 14° Dovetail Bit, 8mm shank
751 gcpy         self.writegc("(TOOL/MILL,12.7,6.367,12.7,0.00)")
752 gcpy     #    dt_bottomdiameter, dt_topdiameter, dt_height, dt_angle
               )
753 gcpy     #    https://www.leevalley.com/en-us/shop/tools/power-tool-
               accessories/router-bits/30172-dovetail-bits?item=18J1607
754 gcpy #          self.currenttoolshape = self.toolshapes("dovetail",
               12.7, 12.7, 14)
               self.endmilltype = "dovetail"
755 gcpy         self.diameter = 12.7
756 gcpy         self.flute = 12.7
757 gcpy         self.angle = 14
758 gcpy         self.toolnumber = "814071"
759 gcpy
760 gcpy     elif (tool_number == 808079) or (tool_number == 808071): #
               45828 == 808071
               self.writegc("(TOOL/MILL,12.7,6.816,20.95,0.00)")
761 gcpy
762 gcpy     #    http://www.amanatool.com/45828-carbide-tipped-dovetail
               -8-deg-x-1-2-dia-x-825-x-1-4-inch-shank.html
763 gcpy #          self.currenttoolshape = self.toolshapes("dovetail",
               12.7, 20.955, 8)
               self.endmilltype = "dovetail"
764 gcpy         self.diameter = 12.7
765 gcpy         self.flute = 20.955
766 gcpy         self.angle = 8
767 gcpy         self.toolnumber = "808071"
768 gcpy
769 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.)

**3.4.1.11 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.

---

```
770 gcpy      self.writegc("M6T", str(tool_number))
771 gcpy #    if (self.endmilltype == "square"):
772 gcpy #        speed = speed *
773 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...` Alternately, describing tools in terms of outline 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. The initial version will only create OpenSCAD commands for 3D modeling and write out matching DXF files. At a later time this will be extended with G-code support.

There are three different movements in G-code which will need to be handled. Rapid commands will be used for 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.

---

```

774 gcpy      def setcolor(self,
775 gcpy          cutcolor = "green",
776 gcpy          rapidcolor = "orange",
777 gcpy          shaftcolor = "red"):
778 gcpy          self.cutcolor = cutcolor
779 gcpy          self.rapidcolor = rapidcolor
780 gcpy          self.shaftcolor = shaftcolor

```

---

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

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.

Table 1: Colors in OpenSCAD and DXF

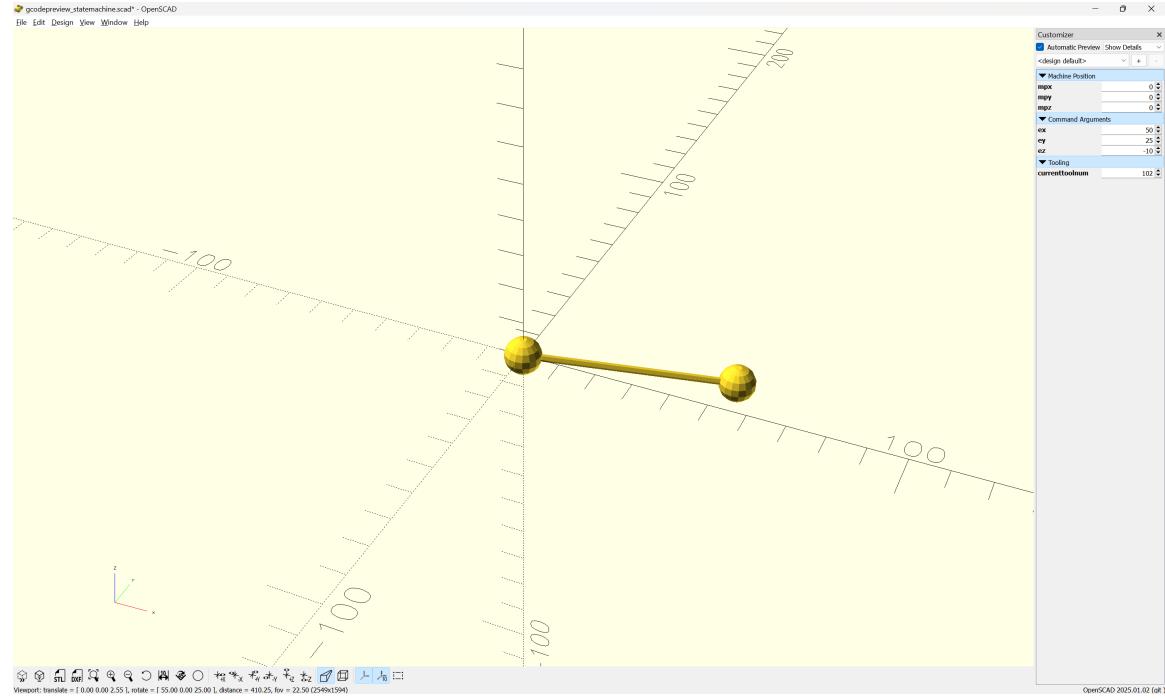
Web Colors (OpenSCAD)	DXF	Both
Black	"Black" (0)	
Red	"Red" (1)	Red
Yellow	"Yellow" (2)	Yellow
Green	"Green" (3)	Green
	"Cyan" (4)	
Blue	"Blue" (5)	Blue
	"Magenta" (6)	
White	"White" (7)	
Gray	"Dark Gray" (8)	(Dark) Gray
	"Light Gray" (9)	
Silver		
Maroon		
Olive		
Lime		
Aqua		
Teal		
Navy		
Fuchsia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

(note that the names are not case-sensitive)

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

Diagramming this is quite straight-forward — there is simply a movement made from the current position to the end. If we start at the origin,  $X_0, Y_0, Z_0$ , then it is simply a straight-line movement (rapid)/cut (possibly a partial cut in the instance of a keyhole or roundover tool), and no variables change value.

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




---

```

782 gcpy      def toolmovement(self, bx, by, bz, ex, ey, ez, step = 0):
783 gcpy          tslist = []
784 gcpy          if step > 0:
785 gcpy              steps = step
786 gcpy          else:
787 gcpy              steps = self.steps
788 gcpy #

```

---

endmill square

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

---

```

789 gcpy      if self.endmilltype == "square":

```

---

---

```

790 gcpy           ts = cylinder(r1=(self.diameter / 2), r2=(self.diameter
                     / 2), h=self.flute, center = False)
791 gcpy           tslist.append(hull(ts.translate([bx, by, bz]), ts.
                     translate([ex, ey, ez])))
792 gcpy           return tslist
793 gcpy #
794 gcpy #       if self.endmilltype == "O-flute":
795 gcpy #           ts = cylinder(r1=(self.diameter / 2), r2=(self.
                     diameter / 2), h=self.flute, center = False)
796 gcpy #           tslist.append(hull(ts.translate([bx, by, bz]), ts.
                     translate([ex, ey, ez])))
797 gcpy #           return tslist
798 gcpy #

```

---

ballnose    **3.5.3.1.2 Ball nose (including tapered ball nose)** The ballnose is modeled as a hemisphere joined with a cylinder:

---

```

799 gcpy           if self.endmilltype == "ball":
800 gcpy             b = sphere(r=(self.diameter / 2))
801 gcpy             s = cylinder(r1=(self.diameter / 2), r2=(self.diameter
                     / 2), h=self.flute, center=False)
802 gcpy             bs = union(b, s)
803 gcpy             bs = bs.translate([0, 0, (self.diameter / 2)])
804 gcpy             tslist.append(hull(bs.translate([bx, by, bz]), bs.
                     translate([ex, ey, ez])))
805 gcpy             return tslist
806 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:

---

```

807 gcpy           if self.endmilltype == "bowl":
808 gcpy             inner = cylinder(r1 = self.diameter/2 - self.radius, r2
                     = self.diameter/2 - self.radius, h = self.flute)
809 gcpy             outer = cylinder(r1 = self.diameter/2, r2 = self.
                     diameter/2, h = self.flute - self.radius)
810 gcpy             outer = outer.translate([0,0, self.radius])
811 gcpy             slices = hull(outer, inner)
812 gcpy #       slices = cylinder(r1 = 0.0001, r2 = 0.0001, h = 0.0001, center
                     =False)
813 gcpy             for i in range(1, 90 - self.steps, self.steps):
814 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)
815 gcpy               slices = hull(slices, slice.translate([0, 0, self.
                     radius - self.radius * Cos(i+self.steps)]))
816 gcpy               tslist.append(hull(slices.translate([bx, by, bz]),
                     slices.translate([ex, ey, ez])))
817 gcpy               return tslist
818 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)):

---

```

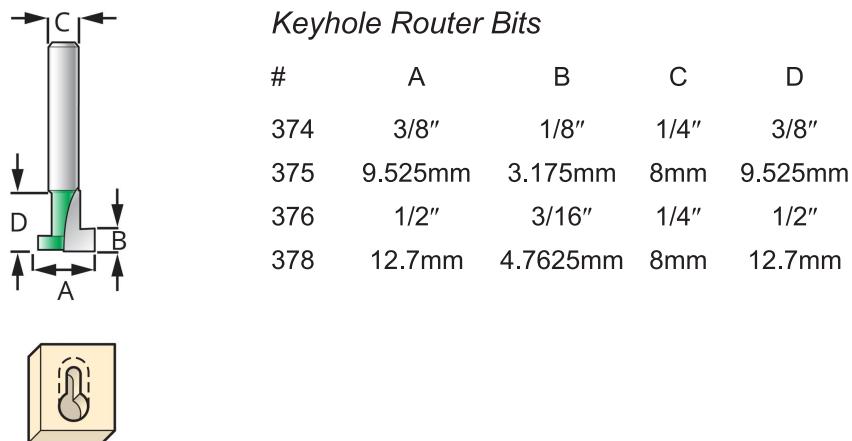
819 gcpy           if self.endmilltype == "V":
820 gcpy             v = cylinder(r1=0, r2=(self.diameter / 2), h=((self.
                     diameter / 2) / Tan((self.angle / 2))), center=False
                     )
821 gcpy #       s = cylinder(r1=(self.diameter / 2), r2=(self.
                     diameter / 2), h=self.flute, center=False)
822 gcpy #       sh = s.translate([0, 0, ((self.diameter / 2) / Tan
                     ((self.angle / 2)))])
823 gcpy             tslist.append(hull(v.translate([bx, by, bz]), v.
                     translate([ex, ey, ez])))
824 gcpy             return tslist

```

---

**3.5.3.1.5 Keyhole** Keyhole toolpaths (see: subsection [3.9.1.1.3](#)) 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.




---

```

826 gcpy      if self.endmilltype == "keyhole":
827 gcpy          kh = cylinder(r1=(self.diameter / 2), r2=(self.diameter
828 gcpy                  / 2), h=self.flute, center=False)
829 gcpy          sh = (cylinder(r1=(self.radius / 2), r2=(self.radius /
830 gcpy                  2), h=self.flute*2, center=False))
831 gcpy          tslist.append(hull(kh.translate([bx, by, bz]), kh.
832 gcpy                  translate([ex, ey, ez])))
833 gcpy          tslist.append(hull(sh.translate([bx, by, bz]), sh.
834 gcpy                  translate([ex, ey, ez])))
835 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.

---

```

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

---

```

840 gcpy      if self.endmilltype == "dovetail":
841 gcpy          dt = cylinder(r1=(self.diameter / 2), r2=(self.diameter
842 gcpy                  / 2) - self.flute * Tan(self.angle), h= self.flute,
843 gcpy                  center=False)
844 gcpy          tslist.append(hull(dt.translate([bx, by, bz]), dt.
845 gcpy                  translate([ex, ey, ez])))
846 gcpy      return tslist
847 gcpy      if self.endmilltype == "other":
848 gcpy          tslist = []
849 gcpy      def dovetail(self, dt_bottomdiameter, dt_topdiameter,
850 gcpy                  dt_height, dt_angle):
851 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-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.

---

```

849 gcpy      if self.endmilltype == "roundover":
850 gcpy          shaft = cylinder(self.steps, self.tip/2, self.tip/2)
851 gcpy          toolpath = hull(shaft.translate([bx, by, bz]), shaft.
852 gcpy              translate([ex, ey, ez]))
853 gcpy          shaft = cylinder(self.flute, self.diameter/2 + self.tip
854 gcpy              /2, self.diameter/2 + self.tip/2)
855 gcpy          toolpath = toolpath.union(hull(shaft.translate([bx, by,
856 gcpy              bz + self.radius]), shaft.translate([ex, ey, ez +
857 gcpy                  self.radius])))
858 gcpy          tslist = [toolpath]
859 gcpy          slice = cylinder(0.0001, 0.0001, 0.0001)
860 gcpy          slices = slice
861 gcpy          for i in range(1, 90 - self.steps, self.steps):
862 gcpy              dx = self.radius*cos(i)
863 gcpy              dxx = self.radius*cos(i + self.steps)
864 gcpy              dzz = self.radius*sin(i)
865 gcpy              dz = self.radius*sin(i + self.steps)
866 gcpy              dh = dz - dzz
867 gcpy              slice = cylinder(r1 = self.tip/2+self.radius-dx, r2
868 gcpy                  = self.tip/2+self.radius-dxx, h = dh)
869 gcpy          slices = slices.union(hull(slice.translate([bx, by,
870 gcpy                  bz+dz]), slice.translate([ex, ey, ez+dz])))
871 gcpy          tslist.append(slices)
872 gcpy
873 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:

---

```

868 gcpy      def shaftmovement(self, bx, by, bz, ex, ey, ez):
869 gcpy          tslist = []
870 gcpy          ts = cylinder(r1=(self.shaftdiameter / 2), r2=(self.
871 gcpy              shaftdiameter / 2), h=self.shaftlength, center = False)
872 gcpy          ts = ts.translate([0, 0, self.shaftheight])
873 gcpy          tslist.append(hull(ts.translate([bx, by, bz]), ts.translate
874 gcpy                  ([ex, ey, ez])))
875 gcpy
876 gcpy      return tslist

```

---

**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.

---

```

875 gcpy      def defineshaft(self, toolingdiameter, shaftdiameter, flute,
876 gcpy          transition, shaft):
877 gcpy          if shaftdiameter == 0:
878 gcpy              self.shaftoutline = polygon(points=[[0, flute],
879 gcpy                  diameter, flute], [diameter, shaft],[0, shaft]])
880 gcpy              self.shaftprofile = polygon(points=[[0, flute],
881 gcpy                  diameter/2 ,flute], [diameter/2, shaft], [0, shaft
882 gcpy                  ]])
883 gcpy              sh = cylinder(h = shaft, r = diameter/2)
884 gcpy              self.currenttoolshaft = sh.translate([0,0,flute])
885 gcpy          if shaftdiameter > 0:
886 gcpy              self.shaftoutline = polygon(points=[
887 gcpy                  [shaftdiameter / 2 - toolingdiameter / 2, flute],
888 gcpy                  [0, flute + transition],
889 gcpy                  [0, flute + transition + shaft],
890 gcpy                  [shaftdiameter, flute + transition + shaft],
891 gcpy                  [shaftdiameter, flute + transition],
892 gcpy                  [shaftdiameter / 2 + toolingdiameter / 2, flute],
893 gcpy                  ])
894 gcpy              self.shaftprofile = polygon( points= [
895 gcpy                  [0, flute],
896 gcpy                  [0, flute + transition + shaft],
897 gcpy                  [shaftdiameter/2, flute + transition + shaft],
898 gcpy                  [shaftdiameter/2, flute + transition],
899 gcpy                  [toolingdiameter/2, flute]
900 gcpy                  ])
901 gcpy          self.currenttoolshaft = rotate_extrude(self.
902 gcpy              shaftprofile)

```

---

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

---

```

899 gcpy      def definesquaretool(self, diameter, flute, shaft, offset = 0):
900 gcpy          self.tooloutline = polygon( points=[0 + offset,0],[
901 gcpy              diameter + offset,0],[diameter + offset,flute],[0 +
902 gcpy              offset,flute]] )

```

---

---

```

901 gcpy      self.toolprofile = polygon( points=[[0,0],[diameter/2,0],[
902 gcpy      diameter/2,flute],[0,flute]] )
903 gcpy      self.currenttoolshape = cylinder(h = flute, r = diameter/2)
904 gcpy      sh = cylinder(h = flute, r = diameter/2)

```

---

**3.5.3.4.3 Ball-nose (including tapered-ball)** 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.

---

```

905 gcpy      def defineballnosetool(self, diameter, flute, shaft, offset =
906 gcpy      0):
907 gcpy      s = square([diameter,flute - diameter/2])
908 gcpy      sh = s.translate([0 + offset, diameter/2])
909 gcpy      c = circle(d=diameter)
910 gcpy      b = c.translate([diameter/2 + offset, diameter/2])
911 gcpy      self.tooloutline = union(sh, b)
912 gcpy      #
913 gcpy      s = square([diameter/2,flute - diameter/2])
914 gcpy      sh = s.translate([0, diameter/2])
915 gcpy      c = circle(d=diameter)
916 gcpy      b = c.translate([0, diameter/2])
917 gcpy      bn = union(sh, b)
918 gcpy      bns = bn.translate([0, diameter/2])
919 gcpy      thein = square([diameter/2,flute])
920 gcpy      theins = thein.translate([diameter/2, 0])
921 gcpy      self.toolprofile = intersection(thein, bn)
922 gcpy      self.shaftprofile = polygon( points=[[0,flute],[diameter/2,
923 gcpy      flute],[diameter/2,shaft],[0,shaft]] )
924 gcpy      #
925 gcpy      b = self.toolprofile
926 gcpy      bn = b.translate([-diameter/2, 0])
927 gcpy      self.currenttoolshape = rotate_extrude(self.toolprofile)
928 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.

---

```

930 gcpy      def defineVtool(self, diameter, flute, shaft, shaftdiameter =
931 gcpy      0):
932 gcpy      self.tooloutline = polygon([[diameter/2, 0], [diameter,
933 gcpy      flute], [0, flute]])
934 gcpy      self.toolprofile = polygon([[0, 0], [diameter/2, flute],
935 gcpy      [0, flute]])
936 gcpy      #
937 gcpy      if shaftdiameter == 0:
938 gcpy      shaftdiameter = diameter
939 gcpy      self.shaftprofile = polygon([[0, flute], [shaftdiameter/2,
940 gcpy      flute], [shaftdiameter/2, flute + shaft], [0, flute +
941 gcpy      shaft]])
942 gcpy      self.currenttoolshape = rotate_extrude(self.toolprofile)
943 gcpy      self.currenttoolshaft = rotate_extrude(self.shaftprofile)
944 gcpy

```

---

**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.

---

```

946 gcpy      def defineKeyholetool(self, diameter, flute, narrowdiameter,
947 gcpy      narrowflute, shaftdiameter, shaftlength):
948 gcpy      self.tooloutline = polygon([[0, 0], [diameter, 0], [
949 gcpy      diameter, flute], [diameter/2 + narrowdiameter/2, flute
950 gcpy      ], [diameter/2 + narrowdiameter/2, flute + narrowflute],
951 gcpy      [diameter/2 - narrowdiameter/2, flute + narrowflute], [
952 gcpy      diameter/2 - narrowdiameter/2, flute], [0, flute]])

```

---

---

```

951 gcpy      self.toolprofile = polygon([[0, 0], [diameter/2, 0], [
952 gcpy #      diameter/2, flute], [narrowdiameter/2, flute], [
953 gcpy #      narrowdiameter/2, flute + narrowflute], [0, flute +
954 gcpy #      narrowflute]])]
955 gcpy #
956 gcpy      self.currenttoolshape = rotate_extrude(self.toolprofile)
957 gcpy #
958 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.

---

```

960 gcpy      def definebowltool(self, diameter, flute, radius, shaftdiameter
961 gcpy #      , shaftlength):
962 gcpy #
963 gcpy      self.tooloutline =
964 gcpy #
965 gcpy      self.toolprofile = polygon([[0,0], [diameter/2, 0, radius],
966 gcpy #      [diameter/2, radius], [diameter/2, flute], [0, flute]])
967 gcpy #
968 gcpy #
969 gcpy      self.currenttoolshape = rotate_extrude(self.toolprofile)

```

---

**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.

---

```

971 gcpy      def defineRoundovertool(self, diameter, tipdiameter, flute,
972 gcpy #      radius, shaftdiameter, shaftlength):
973 gcpy #      self.tip = 0.508
974 gcpy #      self.diameter = 6.35 - self.tip
975 gcpy #      self.flute = 8 - self.tip
976 gcpy #      self.radius = 3.175 - self.tip/2
977 gcpy #      self.shaftdiameter = 6.35
978 gcpy #      self.shaftheight = 8
979 gcpy #      self.shaftlength = 10.0
980 gcpy #      print(diameter)
981 gcpy #      print(tipdiameter)
982 gcpy #      print(flute)
983 gcpy #      print(radius)
984 gcpy #      print(shaftdiameter)
985 gcpy #      self.tooloutline =
986 gcpy #
987 gcpy      self.toolprofile = polygon([[0,0], [tipdiameter/2, 0], [
988 gcpy #      diameter/2, flute], [0, flute]])
989 gcpy      self.shaftprofile = polygon([[0,flute], [shaftdiameter/2,
990 gcpy #      flute], [shaftdiameter/2, flute + shaftlength], [0,
991 gcpy #      flute + shaftlength]])]
992 gcpy #
993 gcpy      self.currenttoolshape = rotate_extrude(self.toolprofile)

```

---

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.

---

```

995 gcpy      def rapid(self, ex, ey, ez, laser = 0):
996 gcpy #       print(self.rapidcolor)
997 gcpy #       if laser == 0:
998 gcpy #           tm = self.toolmovement(self.xpos(), self.ypos(), self.
999 gcpy #               zpos(), ex, ey, ez)
1000 gcpy #           tm = color(tm, self.shaftcolor)
1001 gcpy #           ts = self.shaftmovement(self.xpos(), self.ypos(), self.
1002 gcpy #               zpos(), ex, ey, ez)
1003 gcpy #           ts = color(ts, self.rapidcolor)
1004 gcpy #           self.toolpaths.extend([tm, ts])
1005 gcpy #       self.setxpos(ex)
1006 gcpy #       self.setypos(ey)
1007 gcpy #       self.setzpos(ez)
1008 gcpy #       def cutline(self, ex, ey, ez):
1009 gcpy #           print(self.cutcolor)
1010 gcpy #           print(ex, ey, ez)
1011 gcpy #           tm = self.toolmovement(self.xpos(), self.ypos(), self.zpos
1012 gcpy #               (), ex, ey, ez)
1013 gcpy #           tm = color(tm, self.cutcolor)
1014 gcpy #           ts = self.shaftmovement(self.xpos(), self.ypos(), self.zpos
1015 gcpy #               (), ex, ey, ez)
1016 gcpy #           ts = color(ts, self.rapidcolor)
1017 gcpy #           self.setxpos(ex)
1018 gcpy #           self.setypos(ey)
1019 gcpy #           self.setzpos(ez)
1020 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:

---

```

1019 gcpy      def movetosafeZ(self):
1020 gcpy #       rapid = self.rapid(self.xpos(), self.ypos(), self.
1021 gcpy #           retractheight)
1022 gcpy #       if self.generatepaths == True:
1023 gcpy #           rapid = self.rapid(self.xpos(), self.ypos(), self.
1024 gcpy #               retractheight)
1025 gcpy #           self.rapids = self.rapids.union(rapid)
1026 gcpy #       else:
1027 gcpy #           if (generategcode == true) {
1028 gcpy #               // writecomment("PREPOSITION FOR RAPID PLUNGE");Z25.650
1029 gcpy #               //G1Z24.663F381.0, "F", str(plunge)
1030 gcpy #               if self.generatepaths == False:
1031 gcpy #                   return rapid
1032 gcpy #               else:
1033 gcpy #                   return cube([0.001, 0.001, 0.001])
1034 gcpy #       return rapid
1035 gcpy #       def rapidXYZ(self, ex, ey, ez):
1036 gcpy #           rapid = self.rapid(ex, ey, ez)
1037 gcpy #           if self.generatepaths == False:
1038 gcpy #               return rapid
1039 gcpy #       def rapidXY(self, ey):
1040 gcpy #           rapid = self.rapid(ex, ey, self.zpos())
1041 gcpy #           if self.generatepaths == True:
1042 gcpy #               self.rapids = self.rapids.union(rapid)
1043 gcpy #           else:
1044 gcpy #               if self.generatepaths == False:
1045 gcpy #                   return rapid
1046 gcpy #       def rapidXZ(self, ex, ez):
1047 gcpy #           rapid = self.rapid(ex, self.ypos(), ez)
1048 gcpy #           if self.generatepaths == False:
1049 gcpy #               return rapid
1050 gcpy #       def rapidYZ(self, ey, ez):
1051 gcpy #           rapid = self.rapid(self.xpos(), ey, ez)
1052 gcpy #           if self.generatepaths == False:
1053 gcpy #               return rapid
1054 gcpy #

```

---

---

```

1055 gcpy           return rapid
1056 gcpy
1057 gcpy     def rapidX(self, ex):
1058 gcpy         rapid = self.rapid(ex, self.ypos(), self.zpos())
1059 gcpy #       if self.generatepaths == False:
1060 gcpy         return rapid
1061 gcpy
1062 gcpy     def rapidY(self, ey):
1063 gcpy         rapid = self.rapid(self.xpos(), ey, self.zpos())
1064 gcpy #       if self.generatepaths == False:
1065 gcpy         return rapid
1066 gcpy
1067 gcpy     def rapidZ(self, ez):
1068 gcpy         rapid = [self.rapid(self.xpos(), self.ypos(), ez)]
1069 gcpy #       if self.generatepaths == True:
1070 gcpy #           self.rapids = self.rapids.union(rapid)
1071 gcpy #       else:
1072 gcpy #           if self.generatepaths == False:
1073 gcpy         return rapid

```

---

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

---

```

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

---

```

1075 gcpy     def cutlinedxf(self, ex, ey, ez):
1076 gcpy         self.dxfline(self.currenttoolnumber(), self.xpos(), self.
1077 gcpy             ypos(), ex, ey)
1078 gcpy         self.cutline(ex, ey, ez)
1079 gcpy     def cutlinedxfgc(self, ex, ey, ez):
1080 gcpy         self.dxfline(self.currenttoolnumber(), self.xpos(), self.
1081 gcpy             ypos(), ex, ey)
1082 gcpy         self.writegc("G01\u21D3X", str(ex), "\u21D3Y", str(ey), "\u21D3Z", str(ez)
1083 gcpy             )
1084 gcpy         self.cutline(ex, ey, ez)
1085 gcpy     def cutvertexdxf(self, ex, ey, ez):
1086 gcpy         self.addvertex(self.currenttoolnumber(), ex, ey)
1087 gcpy         self.writegc("G01\u21D3X", str(ex), "\u21D3Y", str(ey), "\u21D3Z", str(ez)
1088 gcpy             )
1089 gcpy         self.cutline(ex, ey, ez)
1090 gcpy     def cutlineXYZwithfeed(self, ex, ey, ez, feed):
1091 gcpy         return self.cutline(ex, ey, ez)
1092 gcpy     def cutlineXYZ(self, ex, ey, ez):
1093 gcpy         return self.cutline(ex, ey, ez)
1094 gcpy
1095 gcpy     def cutlineXYwithfeed(self, ex, ey, feed):
1096 gcpy         return self.cutline(ex, ey, self.zpos())
1097 gcpy
1098 gcpy     def cutlineXY(self, ex, ey):

```

```

1099 gcpy           return self.cutline(ex, ey, self.zpos())
1100 gcpy
1101 gcpy     def cutlineXZwithfeed(self, ex, ez, feed):
1102 gcpy         return self.cutline(ex, self.ypos(), ez)
1103 gcpy
1104 gcpy     def cutlineXZ(self, ex, ez):
1105 gcpy         return self.cutline(ex, self.ypos(), ez)
1106 gcpy
1107 gcpy     def cutlineXwithfeed(self, ex, feed):
1108 gcpy         return self.cutline(ex, self.ypos(), self.zpos())
1109 gcpy
1110 gcpy     def cutlineX(self, ex):
1111 gcpy         return self.cutline(ex, self.ypos(), self.zpos())
1112 gcpy
1113 gcpy     def cutlineYZ(self, ey, ez):
1114 gcpy         return self.cutline(self.xpos(), ey, ez)
1115 gcpy
1116 gcpy     def cutlineYwithfeed(self, ey, feed):
1117 gcpy         return self.cutline(self.xpos(), ey, self.zpos())
1118 gcpy
1119 gcpy     def cutlineY(self, ey):
1120 gcpy         return self.cutline(self.xpos(), ey, self.zpos())
1121 gcpy
1122 gcpy     def cutlineZgcfeed(self, ez, feed):
1123 gcpy         self.writegc("G01 Z", str(ez), "F", str(feed))
1124 gcpy         return self.cutline(self.xpos(), self.ypos(), ez)
1125 gcpy
1126 gcpy     def cutlineZwithfeed(self, ez, feed):
1127 gcpy         return self.cutline(self.xpos(), self.ypos(), ez)
1128 gcpy
1129 gcpy     def cutlineZ(self, ez):
1130 gcpy         return self.cutline(self.xpos(), self.ypos(), ez)

```

---

The matching OpenSCAD command is a descriptor:

```

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

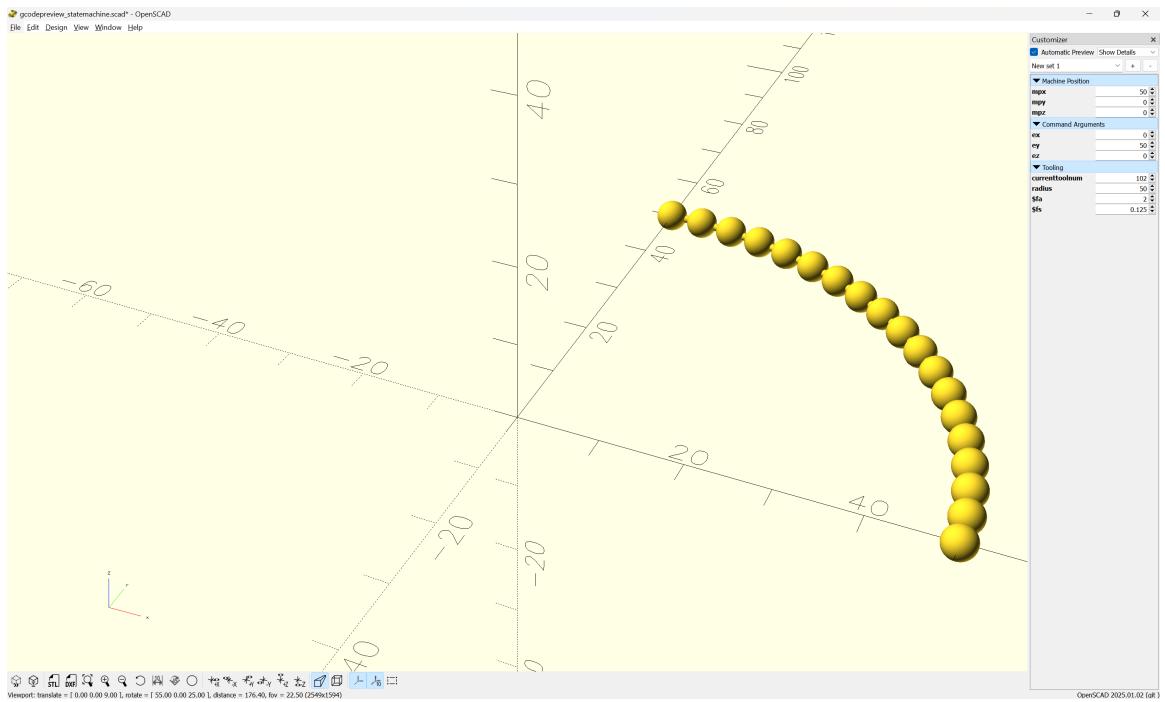
- `ex` — note that the matching origins (`bx, by, bz`) as well as the (current) toolnumber are accessed using the appropriate commands for machine position
- `ey`
- `ez` — allowing a different Z position will make possible threading and similar helical tool-paths
- `xcenter` — the center position will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

---

```

1132 gcpy      def cutarcCC(self, barc, earc, xcenter, ycenter, radius,
1133 gcpy          tpzreldim, stepsizearc=1):
1134 gcpy          tpzinc = tpzreldim / (earc - barc)
1135 gcpy          i = barc
1136 gcpy          while i < earc:
1137 gcpy              self.cutline(xcenter + radius * Cos(i), ycenter +
1138 gcpy                  radius * Sin(i), self.zpos() + tpzinc)
1139 gcpy          i += stepsizearc
1140 gcpy          self.setxpos(xcenter + radius * Cos(earc))
1141 gcpy          self.setypos(ycenter + radius * Sin(earc))
1142 gcpy      def cutarcCW(self, barc, earc, xcenter, ycenter, radius,
1143 gcpy          tpzreldim, stepsizearc=1):
1144 gcpy          print(str(self.zpos()))
1145 gcpy          print(str(ez))
1146 gcpy          print(str(barc - earc))
1147 gcpy          tpzinc = ez - self.zpos() / (barc - earc)
1148 gcpy          print(str(tpzinc))
1149 gcpy          global toolpath
1150 gcpy          print("Entering n toolpath")
1151 gcpy          tpzinc = tpzreldim / (barc - earc)
1152 gcpy          cts = self.currenttoolshape
1153 gcpy          toolpath = cts
1154 gcpy          toolpath = toolpath.translate([self.xpos(), self.ypos(),
1155 gcpy              self.zpos()])
1156 gcpy          toolpath = []
1157 gcpy          i = barc
1158 gcpy          while i > earc:
1159 gcpy              self.cutline(xcenter + radius * Cos(i), ycenter +
1160 gcpy                  radius * Sin(i), self.zpos() + tpzinc)
1161 gcpy              self.setxpos(xcenter + radius * Cos(i))
1162 gcpy              self.setypos(ycenter + radius * Sin(i))
1163 gcpy              print(str(self.xpos()), str(self.ypos()), str(self.zpos(
1164 gcpy                  ()))))
1165 gcpy              self.setzpos(self.zpos() + tpzinc)
1166 gcpy              i += abs(stepsizearc) * -1
1167 gcpy              self.dxfarctoolnumber(), xcenter, ycenter,
1168 gcpy                  radius, barc, earc)
1169 gcpy              if self.generatepaths == True:
1170 gcpy                  print("Unioning n toolpath")
1171 gcpy                  self.toolpaths = self.toolpaths.union(toolpath)
1172 gcpy              else:
1173 gcpy                  return toolpath
1174 gcpy          else:
1175 gcpy              return cube([0.01, 0.01, 0.01])

```

---

Alternately, the command for using rotate\_extrude is quite straight-forward:

---

```

1175 gcpy      def extrudearcCC(self, barc, earc, xcenter, ycenter, radius,
1176 gcpy          tpzreldim, stepsizearc=1):
1177 gcpy          tm = self.toolmovement(self.xpos(), self.ypos(), self.zpos(
1178 gcpy              (), ex, ey, ez)
1179 gcpy          tm = union(self.toolshape.translate(self.xpos(), self.ypos(
1180 gcpy              (), self.zpos())),
1181 gcpy                  self.toolshape.translate(),
1182 gcpy                  tooloutline.translate([r-3.175, 0, 0]),
1183 gcpy                  rotate_extrude(angle=ang2-ang1).rotz(ang1) + G3_center

```

---

```

1180 gcpy
1181 gcpy      tm = color(tm, self.cutcolor)
1182 gcpy      ts = self.shaftmovement(self.xpos(), self.ypos(), self.zpos
1183 gcpy          (), ex, ey, ez)
1184 gcpy      ts = color(ts, self.rapidcolor)
1185 gcpy      self.setxpos(ex)
1186 gcpy      self.setypos(ey)
1187 gcpy      self.setzpos(ez)
1187 gcpy      self.toolpaths.extend([tm, ts])

```

---

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

```

1189 gcpy      def cutarcCWdxf(self, barc, earc, xcenter, ycenter, radius,
1190 gcpy          tpzreldim, stepsizearc=1):
1191 gcpy          self.cutarcCW(barc, earc, xcenter, ycenter, radius,
1192 gcpy          tpzreldim, stepsizearc=1)
1193 gcpy          self.dxfarc(self.currenttoolnumber(), xcenter, ycenter,
1194 gcpy          radius, earc, barc)
1195 gcpy      if self.generatepaths == False:
1196 gcpy      return toolpath
1197 gcpy      else:
1198 gcpy      return cube([0.01, 0.01, 0.01])
1199 gcpy      def cutarcCCdxf(self, barc, earc, xcenter, ycenter, radius,
1200 gcpy          tpzreldim, stepsizearc=1):
1201 gcpy          self.cutarcCC(barc, earc, xcenter, ycenter, radius,
1202 gcpy          tpzreldim, stepsizearc=1)
1203 gcpy          self.dxfarc(self.currenttoolnumber(), xcenter, ycenter,
1204 gcpy          radius, barc, earc)

```

---

Matching OpenSCAD modules are easily made:

```

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

```

---

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

```

1201 gcpy      def cutquarterCCNE(self, ex, ey, ez, radius):
1202 gcpy          if self.zpos() == ez:
1203 gcpy              tpzinc = 0
1204 gcpy          else:
1205 gcpy              tpzinc = (ez - self.zpos()) / 90
1206 gcpy      print("tpzinc ", tpzinc)
1207 gcpy      i = 1
1208 gcpy      while i < 91:
1209 gcpy          self.cutline(ex + radius * Cos(i), ey - radius + radius
1210 gcpy          * Sin(i), self.zpos()+tpzinc)
1211 gcpy          i += 1
1212 gcpy      def cutquarterCCNW(self, ex, ey, ez, radius):
1213 gcpy          if self.zpos() == ez:
1214 gcpy              tpzinc = 0
1215 gcpy          else:
1216 gcpy              tpzinc = (ez - self.zpos()) / 90
1217 gcpy      tpzinc = (self.zpos() + ez) / 90
1218 gcpy      self.debug("tpzinc", tpzinc)
1219 gcpy      i = 91
1220 gcpy      while i < 181:
1221 gcpy          self.cutline(ex + radius + radius * Cos(i), ey + radius
1222 gcpy          * Sin(i), self.zpos()+tpzinc)
1223 gcpy          i += 1
1224 gcpy      def cutquarterCCSW(self, ex, ey, ez, radius):
1225 gcpy          if self.zpos() == ez:
1226 gcpy              tpzinc = 0
1227 gcpy          else:
1228 gcpy              tpzinc = (ez - self.zpos()) / 90
1229 gcpy      tpzinc = (self.zpos() + ez) / 90
1230 gcpy      print("tpzinc ", tpzinc)
1231 gcpy      i = 181
1232 gcpy      while i < 271:

```

```

1233 gcpy           self.cutline(ex + radius * Cos(i), ey + radius + radius
1234 gcpy           * Sin(i), self.zpos() + tpzinc)
1235 gcpy           i += 1
1236 gcpy       def cutquarterCCSE(self, ex, ey, ez, radius):
1237 gcpy           if self.zpos() == ez:
1238 gcpy               tpzinc = 0
1239 gcpy           else:
1240 gcpy               tpzinc = (ez - self.zpos()) / 90
1241 gcpy #               tpzinc = (self.zpos() + ez) / 90
1242 gcpy #               print("tpzinc ", tpzinc)
1243 gcpy           i = 271
1244 gcpy           while i < 361:
1245 gcpy               self.cutline(ex - radius + radius * Cos(i), ey + radius
1246 gcpy               * Sin(i), self.zpos() + tpzinc)
1247 gcpy           i += 1
1248 gcpy       def cutquarterCCNEdxf(self, ex, ey, ez, radius):
1249 gcpy           self.cutquarterCCNE(ex, ey, ez, radius)
1250 gcpy           self.dxfarc(self.currenttoolnumber(), ex, ey - radius,
1251 gcpy           radius, 0, 90)
1252 gcpy       def cutquarterCCNWdxr(self, ex, ey, ez, radius):
1253 gcpy           self.cutquarterCCNW(ex, ey, ez, radius)
1254 gcpy           self.dxfarc(self.currenttoolnumber(), ex + radius, ey,
1255 gcpy           radius, 90, 180)
1256 gcpy       def cutquarterCCSWdxr(self, ex, ey, ez, radius):
1257 gcpy           self.cutquarterCCSW(ex, ey, ez, radius)
1258 gcpy           self.dxfarc(self.currenttoolnumber(), ex, ey + radius,
1259 gcpy           radius, 180, 270)
1260 gcpy       def cutquarterCCSEdxr(self, ex, ey, ez, radius):
1261 gcpy           self.cutquarterCCSE(ex, ey, ez, radius)
1262 gcpy           self.dxfarc(self.currenttoolnumber(), ex - radius, ey,
1263 gcpy           radius, 270, 360)

```

---

```

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

---

```
124 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:

---

```
1264 gcpy      def tool_diameter(self, ptd_tool, ptd_depth):
1265 gcpy # Square 122, 112, 102, 201
1266 gcpy      if ptd_tool == 122:
1267 gcpy          return 0.79375
1268 gcpy      if ptd_tool == 112:
1269 gcpy          return 1.5875
1270 gcpy      if ptd_tool == 102:
1271 gcpy          return 3.175
1272 gcpy      if ptd_tool == 201:
1273 gcpy          return 6.35
1274 gcpy # Ball 121, 111, 101, 202
1275 gcpy      if ptd_tool == 122:
1276 gcpy          if ptd_depth > 0.396875:
1277 gcpy              return 0.79375
1278 gcpy          else:
1279 gcpy              return ptd_tool
1280 gcpy      if ptd_tool == 112:
1281 gcpy          if ptd_depth > 0.79375:
1282 gcpy              return 1.5875
1283 gcpy          else:
1284 gcpy              return ptd_tool
1285 gcpy      if ptd_tool == 101:
1286 gcpy          if ptd_depth > 1.5875:
1287 gcpy              return 3.175
1288 gcpy          else:
1289 gcpy              return ptd_tool
1290 gcpy      if ptd_tool == 202:
1291 gcpy          if ptd_depth > 3.175:
1292 gcpy              return 6.35
1293 gcpy          else:
1294 gcpy              return ptd_tool
1295 gcpy # V 301, 302, 390
1296 gcpy      if ptd_tool == 301:
1297 gcpy          return ptd_tool
1298 gcpy      if ptd_tool == 302:
1299 gcpy          return ptd_tool
1300 gcpy      if ptd_tool == 390:
1301 gcpy          return ptd_tool
1302 gcpy # Keyhole
1303 gcpy      if ptd_tool == 374:
1304 gcpy          if ptd_depth < 3.175:
1305 gcpy              return 9.525
1306 gcpy          else:
1307 gcpy              return 6.35
1308 gcpy      if ptd_tool == 375:
1309 gcpy          if ptd_depth < 3.175:
1310 gcpy              return 9.525
1311 gcpy          else:
1312 gcpy              return 8
1313 gcpy      if ptd_tool == 376:
1314 gcpy          if ptd_depth < 4.7625:
1315 gcpy              return 12.7
1316 gcpy          else:
1317 gcpy              return 6.35
1318 gcpy      if ptd_tool == 378:
1319 gcpy          if ptd_depth < 4.7625:
1320 gcpy              return 12.7
1321 gcpy          else:
1322 gcpy              return 8
1323 gcpy # Dovetail
1324 gcpy      if ptd_tool == 814:
1325 gcpy          if ptd_depth > 12.7:
1326 gcpy              return 6.35
1327 gcpy          else:
1328 gcpy              return ptd_tool
1329 gcpy      if ptd_tool == 808079:
```

---

```

1330 gcpy           if ptd_depth > 20.95:
1331 gcpy           return 6.816
1332 gcpy           else:
1333 gcpy           return ptd_tool
1334 gcpy # Bowl Bit
1335 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
1336 gcpy           if ptd_tool == 45982:
1337 gcpy           if ptd_depth > 6.35:
1338 gcpy           return 15.875
1339 gcpy           else:
1340 gcpy           return ptd_tool
1341 gcpy # Tapered Ball Nose
1342 gcpy           if ptd_tool == 204:
1343 gcpy           if ptd_depth > 6.35:
1344 gcpy           return ptd_tool
1345 gcpy           if ptd_tool == 304:
1346 gcpy           if ptd_depth > 6.35:
1347 gcpy           return ptd_tool
1348 gcpy           else:
1349 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:

```

1351 gcpy     def tool_radius(self, ptd_tool, ptd_depth):
1352 gcpy         tr = self.tool_diameter(ptd_tool, ptd_depth)/2
1353 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

## 3.6 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.

```
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

```

The various commands for machine functionality are quite straight-forward, with each added as a descriptive module.

---

```

1355 gcpy     def setfansoff(self):
1356 gcpy         writegc("M106\u00d7S0")
1357 gcpy
1358 gcpy     def setfanspeed(self, fan, speed):
1359 gcpy         writegc("M106\u00d7P", fan, "\u00d7S", speed)
1360 gcpy
1361 gcpy     def pauseforclearbuffer(self):
1362 gcpy         writegc("M400\u00d7;uwait\u00d7for\u00d7buffer\u00d7to\u00d7clear")

```

---

Note that certain commands will require setting values which will need to be tracked and used for calculations.

---

```

1364 gcpy     def setfeedratio(self, feedratio):
1365 gcpy         writegc("M220\u00d7S", feedratio)
1366 gcpy         self.feedratio = feedratio
1367 gcpy
1368 gcpy     def setspeedratio(self, speedratio):
1369 gcpy         writegc("M221\u00d7S", speedratio)
1370 gcpy         self.speedratio = speedratio

```

---

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.

---

```

1372 gcpy #Set extruder temperature: M104 [T<index>] [S<temperature>]
1373 gcpy     def setextrudertemperature(self, temperature):
1374 gcpy         writegc("M104\u00d7S", temperature)
1375 gcpy         self.extrudertemperature = temperature
1376 gcpy
1377 gcpy #Set bed temperature: M140 [S<temperature>]
1378 gcpy     def setbedtemperature(self, temperature):
1379 gcpy         writegc("M140\u00d7S", temperature)
1380 gcpy         self.bedtemperature = temperature
1381 gcpy
1382 gcpy #Set bed temperature and wait: M190 S<temperature>
1383 gcpy #Note: M190 always waits for temperature to settle at requested
           value
1384 gcpy     def setandwaitforbedtemperature(self, temperature):
1385 gcpy         writegc("M190\u00d7S", temperature)
1386 gcpy         self.bedtemperature = temperature
1387 gcpy
1388 gcpy #Set extruder temperature and wait: M109 [T<index>] S<temperature>
1389 gcpy #Note: M109 always waits for temperature to settle at requested
           value
1390 gcpy     def setandwaitforextrudertemperature(self, temperature):
1391 gcpy         writegc("M109\u00d7S", temperature)

```

---

```
1392 gcpy      self.bedtemperature = temperature
```

---

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

---

```
1394 gcpy      def initializeforprinting(self, nozzlediameter = 0.4,
                                         filamentdiameter = 1.75, extrusionwidth = 0.6, layerheight =
                                         0.2):
1395 gcpy          self.writegc("G90")
1396 gcpy          self.writegc("G28; home")
1397 gcpy          self.writegc("M729; Clean_Nozzle")
1398 gcpy          self.nozzlediameter = nozzlediameter
1399 gcpy          self.extrusionwidth = extrusionwidth
1400 gcpy          self.layerheight = layerheight
1401 gcpy          self.toolpaths = []
1402 gcpy          self.feedrate = 0
1403 gcpy          fr = filamentdiameter/2
1404 gcpy          self.extrusion_normal_length = 1 / 3.14159 * (fr * fr)
```

---



---

```
1406 gcpy #Set acceleration: M204 S<value> OR M204 P<value> T<value>
1407 gcpy #Note: If S is not specified and both P and T are specified, then
             the acceleration is set to the minimum of P and T. If only one
             of P or T is specified, the command has no effect.
1408 gcpy      def setacceleration(self, acceleration):
1409 gcpy          writegc("M204S", acceleration)
1410 gcpy          self.acceleration = acceleration
1411 gcpy
```

---



---

```
1412 gcpy #Use absolute/relative distances for extrusion: M82, M83
1413 gcpy      def setextrusionabsolute(self, acceleration):
1414 gcpy          writegc("M83")
1415 gcpy          self.extrusionabsolute = true
```

---



---

```
1417 gcpy #Set build percentage: M73 P<percent>
1418 gcpy      def setbuildpercentage(self, percent):
1419 gcpy          writegc("M73P", percent)
1420 gcpy          self.percent = percent
```

---

The program [https://github.com/FullControlXYZ/fullcontrol/blob/master/models/hex\\_adapter.ipynb](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 / toolcha
```

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) / ($\pi * (0.875)^2) 0.033 mm$/$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 ---

```

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.

---

```
1422 gcpy #Move (G0 or G1): G1 [X<pos>] [Y<pos>] [Z<pos>] [E<pos>] [F<speed>]
1423 gcpy     def extrude(self, ex, ey, ez, extrusionwidth = 0, layerheight =
0, feedrate = 0):
1424 gcpy         if extrusionwidth > 0:
1425 gcpy             self.extrusionwidth = extrusionwidth
1426 gcpy         if layerheight > 0:
1427 gcpy             self.layerheight = layerheight
1428 gcpy         if feedrate > 0:
1429 gcpy             self.feedrate = feedrate
1430 gcpy         if self.extrusionwidth == self.layerheight:
1431 gcpy             c = sphere(self.layerheight/2)
1432 gcpy         else:
1433 gcpy             ew = self.extrusionwidth
1434 gcpy             lh = self.layerheight
1435 gcpy             i = circle(lh/2)
1436 gcpy             j = i.translate([0,lh/2,0])
1437 gcpy             k = intersection(j,square([lh,lh]))
1438 gcpy             l = k.translate([ew/2-lh/2,0,0])
1439 gcpy             m = union(l, square([ew/2-lh/2, lh]))
1440 gcpy             c = rotate_extrude(m)
1441 gcpy             c = c.translate([0,0,-self.layerheight])
1442 gcpy             tslist = hull(c.translate([self.xpos(), self.ypos(),self.
zpos()]), c.translate([ex, ey, ez]))
1443 gcpy             self.toolpaths.append(tslist)
1444 gcpy             #volume = r^2 * length
1445 gcpy             # + extrusionwidth-layerheight * layerheight * length
1446 gcpy             distance = math.dist([self.xpos(), self.ypos(), self.zpos()]
[ex, ey, ez])
1447 gcpy             print("Distance=", distance)
1448 gcpy             v = self.extrusionwidth-self.layerheight * self.layerheight
* distance + 3.14159 * self.layerheight/2 * self.
layerheight/2 * distance
1449 gcpy             print("Volume=",v)
1450 gcpy             el = self.extrusion_normal_length * v
1451 gcpy             print("Extrusion_length=",el)
1452 gcpy             self.writegc("X"+str(ex)+"Y"+str(ey)+ "Z"+str(ez)+ "E"+
str(el)+"F"+str(self.feedrate))
```

---

The system Fullcontrolgcode <https://fullcontrolgcode.com/> 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/](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

```

### 3.7 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`:

---

```

1454 gcpy      def stockandtoolpaths(self, option = "stockandtoolpaths"):
1455 gcpy          if option == "stock":
1456 gcpy              show(self.stock)
1457 gcpy          elif option == "toolpaths":
1458 gcpy              show(self.toolpaths)
1459 gcpy          elif option == "rapids":
1460 gcpy              show(self.rapids)
1461 gcpy      else:
1462 gcpy          part = self.stock.difference(self.rapids)
1463 gcpy          part = self.stock.difference(self.toolpaths)
1464 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.

---

```

1466 gcpy      def showtooloutline(self):
1467 gcpy          to = union(self.tooloutline, self.shaftoutline)
1468 gcpy          show(to)
1469 gcpy
1470 gcpy      def showtoolprofile(self):
1471 gcpy          to = union(self.toolprofile, self.shaftprofile)
1472 gcpy          show(to)
1473 gcpy
1474 gcpy      def showtoolshape(self):
1475 gcpy          to = union(self.currenttoolshape, self.currenttoolshaft)
1476 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:

---

```

1478 gcpy      def returnstockandtoolpaths(self):
1479 gcpy          part = self.stock.difference(self.toolpaths)
1480 gcpy          return part

```

---

and then make use of that specific command for OpenSCAD:

---

```

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

```

---

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

### 3.8 Output files

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

#### 3.8.1 Python and OpenSCAD File Handling

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

---

```

1482 gcpy      def writegc(self, *arguments):
1483 gcpy          if self.generategcode == True:
1484 gcpy              line_to_write = ""
1485 gcpy              for element in arguments:
1486 gcpy                  line_to_write += element
1487 gcpy              self.gc.write(line_to_write)

```

---

```

1488 gcpy           self.gc.write("\n")
1489 gcpy
1490 gcpy     def writedxf(self, toolnumber, *arguments):
1491 gcpy #         global dxfclosed
1492 gcpy         line_to_write = ""
1493 gcpy         for element in arguments:
1494 gcpy             line_to_write += element
1495 gcpy         if self.generatedxf == True:
1496 gcpy             if self.dxfclosed == False:
1497 gcpy                 self.dxf.write(line_to_write)
1498 gcpy                 self.dxf.write("\n")
1499 gcpy         if self.generateddxfs == True:
1500 gcpy             self.writedxfs(toolnumber, line_to_write)
1501 gcpy
1502 gcpy     def writedxfs(self, toolnumber, line_to_write):
1503 gcpy #         print("Processing writing toolnumber", toolnumber)
1504 gcpy #
1505 gcpy #         line_to_write = ""
1506 gcpy #
1507 gcpy         for element in arguments:
1508 gcpy             line_to_write += element
1509 gcpy         if (toolnumber == 0):
1510 gcpy             return
1511 gcpy         elif self.generateddxfs == True:
1512 gcpy             if (self.large_square_tool_num == toolnumber):
1513 gcpy                 self.dxflglsq.write(line_to_write)
1514 gcpy                 self.dxflglsq.write("\n")
1515 gcpy             if (self.small_square_tool_num == toolnumber):
1516 gcpy                 self.dfxsmsq.write(line_to_write)
1517 gcpy                 self.dfxsmsq.write("\n")
1518 gcpy             if (self.large_ball_tool_num == toolnumber):
1519 gcpy                 self.dxflgbl.write(line_to_write)
1520 gcpy                 self.dxflgbl.write("\n")
1521 gcpy             if (self.small_ball_tool_num == toolnumber):
1522 gcpy                 self.dfxsmbl.write(line_to_write)
1523 gcpy                 self.dfxsmbl.write("\n")
1524 gcpy             if (self.large_V_tool_num == toolnumber):
1525 gcpy                 self.dxflgV.write(line_to_write)
1526 gcpy                 self.dxflgV.write("\n")
1527 gcpy             if (self.small_V_tool_num == toolnumber):
1528 gcpy                 self.dfxsmV.write(line_to_write)
1529 gcpy                 self.dfxsmV.write("\n")
1530 gcpy             if (self.DT_tool_num == toolnumber):
1531 gcpy                 self.dxfDT.write(line_to_write)
1532 gcpy                 self.dxfDT.write("\n")
1533 gcpy             if (self.KH_tool_num == toolnumber):
1534 gcpy                 self.dxfKH.write(line_to_write)
1535 gcpy                 self.dxfKH.write("\n")
1536 gcpy             if (self.Roundover_tool_num == toolnumber):
1537 gcpy                 self.dxfRt.write(line_to_write)
1538 gcpy                 self.dxfRt.write("\n")
1539 gcpy             if (self.MISC_tool_num == toolnumber):
1540 gcpy                 self.dxfMt.write(line_to_write)
1541 gcpy                 self.dxfMt.write("\n")

```

---

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

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

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

```

130 gpcscad module opendxfile(basefilename){
131 gpcscad     gcp.opendxfile(basefilename);
132 gpcscad }
133 gpcscad
134 gpcscad module opendxfiles(Base_filename, large_square_tool_num,
135 gpcscad     small_square_tool_num, large_ball_tool_num, small_ball_tool_num,
136 gpcscad     large_V_tool_num, small_V_tool_num, DT_tool_num, KH_tool_num,
137 gpcscad     Roundover_tool_num, MISC_tool_num) {
138 gpcscad     gcp.opendxfiles(Base_filename, large_square_tool_num,
139 gpcscad     small_square_tool_num, large_ball_tool_num,
140 gpcscad     small_ball_tool_num, large_V_tool_num, small_V_tool_num,
141 gpcscad     DT_tool_num, KH_tool_num, Roundover_tool_num, MISC_tool_num)

```

```
136 gpcscad }
```

---

opengcodefile With matching OpenSCAD commands: opengcodefile for OpenSCAD:

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

---

and Python:

```
1541 gcpy     def opengcodefile(self, basefilename = "export",
1542 gcpy           currenttoolnum = 102,
1543 gcpy           toolradius = 3.175,
1544 gcpy           plunge = 400,
1545 gcpy           feed = 1600,
1546 gcpy           speed = 10000
1547 gcpy           ):
1548 gcpy           self.basefilename = basefilename
1549 gcpy           self.currenttoolnum = currenttoolnum
1550 gcpy           self.toolradius = toolradius
1551 gcpy           self.plunge = plunge
1552 gcpy           self.feed = feed
1553 gcpy           self.speed = speed
1554 gcpy           if self.generategcode == True:
1555 gcpy               self.gcodefilename = basefilename + ".nc"
1556 gcpy               self.gc = open(self.gcodefilename, "w")
1557 gcpy               self.writegc("(DesignFile:" + self.basefilename + ")"
1558 gcpy               )
1559 gcpy     def opendxfile(self, basefilename = "export"):
1560 gcpy         self.basefilename = basefilename
1561 gcpy #         global generateddxfs
1562 gcpy #         global dxfclosed
1563 gcpy         self.dxfclosed = False
1564 gcpy         self.dxfcolor = "Black"
1565 gcpy         if self.generatedxf == True:
1566 gcpy             self.generateddxfs = False
1567 gcpy             self.dxffilename = basefilename + ".dxf"
1568 gcpy             self.dxf = open(self.dxffilename, "w")
1569 gcpy             self.dxfpreamble(-1)
1570 gcpy     def opendxfiles(self, basefilename = "export",
1571 gcpy           large_square_tool_num = 0,
1572 gcpy           small_square_tool_num = 0,
1573 gcpy           large_ball_tool_num = 0,
1574 gcpy           small_ball_tool_num = 0,
1575 gcpy           large_V_tool_num = 0,
1576 gcpy           small_V_tool_num = 0,
1577 gcpy           DT_tool_num = 0,
1578 gcpy           KH_tool_num = 0,
1579 gcpy           Roundover_tool_num = 0,
1580 gcpy           MISC_tool_num = 0):
1581 gcpy           global generateddxfs
1582 gcpy #           self.basefilename = basefilename
1583 gcpy           self.generateddxfs = True
1584 gcpy           self.large_square_tool_num = large_square_tool_num
1585 gcpy           self.small_square_tool_num = small_square_tool_num
1586 gcpy           self.large_ball_tool_num = large_ball_tool_num
1587 gcpy           self.small_ball_tool_num = small_ball_tool_num
1588 gcpy           self.large_V_tool_num = large_V_tool_num
1589 gcpy           self.small_V_tool_num = small_V_tool_num
1590 gcpy           self.DT_tool_num = DT_tool_num
1591 gcpy           self.KH_tool_num = KH_tool_num
1592 gcpy           self.Roundover_tool_num = Roundover_tool_num
1593 gcpy           self.MISC_tool_num = MISC_tool_num
1594 gcpy           if self.generatedxf == True:
1595 gcpy               if (large_square_tool_num > 0):
1596 gcpy                   self.dxflgsqfilename = basefilename + str(
1597 gcpy                     large_square_tool_num) + ".dxf"
1598 gcpy #                   print("Opening ", str(self.dxflgsqfilename))
1599 gcpy                   self.dxflgsq = open(self.dxflgsqfilename, "w")
1600 gcpy               if (small_square_tool_num > 0):
1601 gcpy                   print("Opening small square")
1602 gcpy                   self.dfxsmsqfilename = basefilename + str(
```

```

1603 gcpy
1604 gcpy
1605 gcpy #
1606 gcpy
1607 gcpy
1608 gcpy
1609 gcpy #
1610 gcpy
1611 gcpy
1612 gcpy
1613 gcpy #
1614 gcpy
1615 gcpy
1616 gcpy
1617 gcpy #
1618 gcpy
1619 gcpy
1620 gcpy
1621 gcpy #
1622 gcpy
1623 gcpy
1624 gcpy
1625 gcpy #
1626 gcpy
1627 gcpy
1628 gcpy
1629 gcpy #
1630 gcpy
1631 gcpy
1632 gcpy
1633 gcpy #
1634 gcpy
1635 gcpy
    small_square_tool_num) + ".dxf"
    self.dfxsmsg = open(self.dfxsmsgfilename, "w")
if (large_ball_tool_num > 0):
    print("Opening large ball")
    self.dxflgblfilename = basefilename + str(
        large_ball_tool_num) + ".dxf"
    self.dxflgbl = open(self.dxflgblfilename, "w")
if (small_ball_tool_num > 0):
    print("Opening small ball")
    self.dfxsmbllfilename = basefilename + str(
        small_ball_tool_num) + ".dxf"
    self.dfxsmbll = open(self.dfxsmbllfilename, "w")
if (large_V_tool_num > 0):
    print("Opening large V")
    self.dxflgVfilename = basefilename + str(
        large_V_tool_num) + ".dxf"
    self.dxflgV = open(self.dxflgVfilename, "w")
if (small_V_tool_num > 0):
    print("Opening small V")
    self.dfxsmVfilename = basefilename + str(
        small_V_tool_num) + ".dxf"
    self.dfxsmV = open(self.dfxsmVfilename, "w")
if (DT_tool_num > 0):
    print("Opening DT")
    self.dfxDTfilename = basefilename + str(DT_tool_num)
        ) + ".dxf"
    self.dfxDT = open(self.dfxDTfilename, "w")
if (KH_tool_num > 0):
    print("Opening KH")
    self.dfxKHfilename = basefilename + str(KH_tool_num)
        ) + ".dxf"
    self.dfxKH = open(self.dfxKHfilename, "w")
if (Roundover_tool_num > 0):
    print("Opening Rt")
    self.dfxRtfilename = basefilename + str(
        Roundover_tool_num) + ".dxf"
    self.dfxRt = open(self.dfxRtfilename, "w")
if (MISC_tool_num > 0):
    print("Opening Mt")
    self.dfxMtfilename = basefilename + str(
        MISC_tool_num) + ".dxf"
    self.dfxMt = open(self.dfxMtfilename, "w")

```

---

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

```

1636 gcpy
1637 gcpy
1638 gcpy
1639 gcpy
1640 gcpy
1641 gcpy
1642 gcpy
1643 gcpy
1644 gcpy
1645 gcpy
1646 gcpy
1647 gcpy
1648 gcpy
1649 gcpy
1650 gcpy
1651 gcpy
1652 gcpy
1653 gcpy
1654 gcpy
1655 gcpy
    if (large_square_tool_num > 0):
        self.dfxpreamble(large_square_tool_num)
    if (small_square_tool_num > 0):
        self.dfxpreamble(small_square_tool_num)
    if (large_ball_tool_num > 0):
        self.dfxpreamble(large_ball_tool_num)
    if (small_ball_tool_num > 0):
        self.dfxpreamble(small_ball_tool_num)
    if (large_V_tool_num > 0):
        self.dfxpreamble(large_V_tool_num)
    if (small_V_tool_num > 0):
        self.dfxpreamble(small_V_tool_num)
    if (DT_tool_num > 0):
        self.dfxpreamble(DT_tool_num)
    if (KH_tool_num > 0):
        self.dfxpreamble(KH_tool_num)
    if (Roundover_tool_num > 0):
        self.dfxpreamble(Roundover_tool_num)
    if (MISC_tool_num > 0):
        self.dfxpreamble(MISC_tool_num)

```

---

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

Future considerations:

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

to it,

### 3.8.2 DXF Overview

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

```

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

```

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

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

- |              |                                        |
|--------------|----------------------------------------|
| writedxflbl  | • Ball nose, large (lbl) writedxflbl   |
| writedxfsmbl | • Ball nose, small (smbl) writedxfsmbl |
| writedxflgsq | • Square, large (lgsq) writedxflgsq    |
| writedxfsmfq | • Square, small (smsq) writedxfsmfq    |
| writedxflgV  | • V, large (lgV) writedxflgV           |
| writedxfsmV  | • V, small (smV) writedxfsmV           |
| writedxfKH   | • Keyhole (KH) writedxfKH              |
| writedxfDT   | • Dovetail (DT) writedxfDT             |

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

---

```

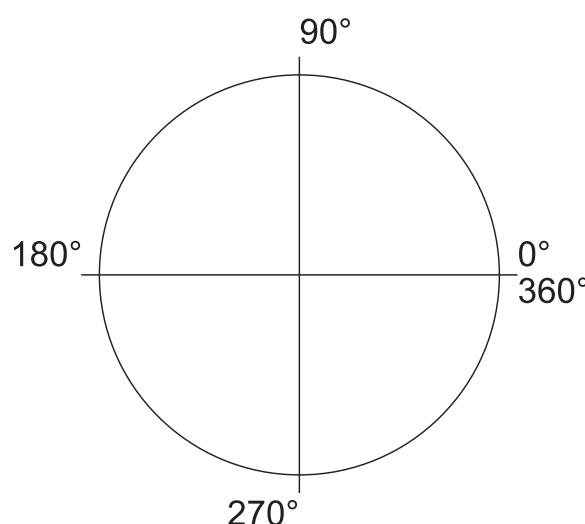
1657 gcpy      def dxfpreamble(self, tn):
1658 gcpy      #          self.writedxf(tn, str(tn))
1659 gcpy      self.writedxf(tn, "0")
1660 gcpy      self.writedxf(tn, "SECTION")
1661 gcpy      self.writedxf(tn, "2")
1662 gcpy      self.writedxf(tn, "ENTITIES")

```

---

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

- dxfline
  - beginpolyline  
  addvertex  
closepolyline
  - dxfarc
  - circle — a notable option would be for the arc to close on itself, creating a circle 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:

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

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

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

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

---

```
1664 gcpy      def setdxfcolor(self, color):
1665 gcpy          self.dxfcolor = color
1666 gcpy          self.cutcolor = color
1667 gcpy
1668 gcpy      def writedxfcolor(self, tn):
1669 gcpy          self.writedxf(tn, "8")
1670 gcpy          if (self.dxfcolor == "Black"):
1671 gcpy              self.writedxf(tn, "Layer_Black")
1672 gcpy          if (self.dxfcolor == "Red"):
1673 gcpy              self.writedxf(tn, "Layer_Red")
1674 gcpy          if (self.dxfcolor == "Yellow"):
1675 gcpy              self.writedxf(tn, "Layer_Yellow")
1676 gcpy          if (self.dxfcolor == "Green"):
1677 gcpy              self.writedxf(tn, "Layer_Green")
1678 gcpy          if (self.dxfcolor == "Cyan"):
1679 gcpy              self.writedxf(tn, "Layer_Cyan")
1680 gcpy          if (self.dxfcolor == "Blue"):
1681 gcpy              self.writedxf(tn, "Layer_Blue")
1682 gcpy          if (self.dxfcolor == "Magenta"):
1683 gcpy              self.writedxf(tn, "Layer_Magenta")
1684 gcpy          if (self.dxfcolor == "White"):
```

```

1685 gcpy           self.writedxf(tn, "Layer_White")
1686 gcpy           if (self.dxfcolor == "DarkGray"):
1687 gcpy             self.writedxf(tn, "Layer_Dark_Gray")
1688 gcpy           if (self.dxfcolor == "LightGray"):
1689 gcpy             self.writedxf(tn, "Layer_Light_Gray")
1690 gcpy
1691 gcpy           self.writedxf(tn, "62")
1692 gcpy           if (self.dxfcolor == "Black"):
1693 gcpy             self.writedxf(tn, "0")
1694 gcpy           if (self.dxfcolor == "Red"):
1695 gcpy             self.writedxf(tn, "1")
1696 gcpy           if (self.dxfcolor == "Yellow"):
1697 gcpy             self.writedxf(tn, "2")
1698 gcpy           if (self.dxfcolor == "Green"):
1699 gcpy             self.writedxf(tn, "3")
1700 gcpy           if (self.dxfcolor == "Cyan"):
1701 gcpy             self.writedxf(tn, "4")
1702 gcpy           if (self.dxfcolor == "Blue"):
1703 gcpy             self.writedxf(tn, "5")
1704 gcpy           if (self.dxfcolor == "Magenta"):
1705 gcpy             self.writedxf(tn, "6")
1706 gcpy           if (self.dxfcolor == "White"):
1707 gcpy             self.writedxf(tn, "7")
1708 gcpy           if (self.dxfcolor == "DarkGray"):
1709 gcpy             self.writedxf(tn, "8")
1710 gcpy           if (self.dxfcolor == "LightGray"):
1711 gcpy             self.writedxf(tn, "9")

```

---

```

142 gpcscad module setdxfcolor(color){
143 gpcscad     gcp.setdxfcolor(color);
144 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, 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.

```

1713 gcpy     def dxfline(self, tn, xbegin, ybegin, xend, yend):
1714 gcpy         self.writedxf(tn, "0")
1715 gcpy         self.writedxf(tn, "LINE")
1716 gcpy #
1717 gcpy         self.writedxfcolor(tn)
1718 gcpy #
1719 gcpy         self.writedxf(tn, "10")
1720 gcpy         self.writedxf(tn, str(xbegin))
1721 gcpy         self.writedxf(tn, "20")
1722 gcpy         self.writedxf(tn, str(ybegin))
1723 gcpy         self.writedxf(tn, "30")
1724 gcpy         self.writedxf(tn, "0.0")
1725 gcpy         self.writedxf(tn, "11")
1726 gcpy         self.writedxf(tn, str(xend))
1727 gcpy         self.writedxf(tn, "21")
1728 gcpy         self.writedxf(tn, str(yend))
1729 gcpy         self.writedxf(tn, "31")
1730 gcpy         self.writedxf(tn, "0.0")

```

---

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

First, begin the polyline:

```

1732 gcpy     def beginpolyline(self, tn):#, xbegin, ybegin
1733 gcpy         self.writedxf(tn, "0")
1734 gcpy         self.writedxf(tn, "POLYLINE")
1735 gcpy         self.writedxf(tn, "8")

```

---

---

```

1736 gcpy      self.writedxf(tn, "default")
1737 gcpy      self.writedxf(tn, "66")
1738 gcpy      self.writedxf(tn, "1")
1739 gcpy #
1740 gcpy      self.writedxfcolor(tn)
1741 gcpy #
1742 gcpy      self.writedxf(tn, "10")
1743 gcpy      self.writedxf(tn, str(xbegin))
1744 gcpy      self.writedxf(tn, "20")
1745 gcpy      self.writedxf(tn, str(ybegin))
1746 gcpy      self.writedxf(tn, "30")
1747 gcpy      self.writedxf(tn, "0.0")
1748 gcpy      self.writedxf(tn, "70")
1749 gcpy      self.writedxf(tn, "0")

```

---

then add as many vertexes as are wanted:

---

```

1751 gcpy      def addvertex(self, tn, xend, yend):
1752 gcpy          self.writedxf(tn, "0")
1753 gcpy          self.writedxf(tn, "VERTEX")
1754 gcpy          self.writedxf(tn, "8")
1755 gcpy          self.writedxf(tn, "default")
1756 gcpy          self.writedxf(tn, "70")
1757 gcpy          self.writedxf(tn, "32")
1758 gcpy          self.writedxf(tn, "10")
1759 gcpy          self.writedxf(tn, str(xend))
1760 gcpy          self.writedxf(tn, "20")
1761 gcpy          self.writedxf(tn, str(yend))
1762 gcpy          self.writedxf(tn, "30")
1763 gcpy          self.writedxf(tn, "0.0")

```

---

then end the sequence:

---

```

1765 gcpy      def closepolyline(self, tn):
1766 gcpy          self.writedxf(tn, "0")
1767 gcpy          self.writedxf(tn, "SEQEND")

```

---

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

---

```

1769 gcpy      def dxfarl(self, tn, xcenter, ycenter, radius, anglebegin,
1770 gcpy          endangle):
1771 gcpy          if (self.generatedxf == True):
1772 gcpy              self.writedxf(tn, "0")
1773 gcpy              self.writedxf(tn, "ARC")
1774 gcpy              self.writedxfcolor(tn)
1775 gcpy              self.writedxf(tn, "10")
1776 gcpy              self.writedxf(tn, str(xcenter))
1777 gcpy              self.writedxf(tn, "20")
1778 gcpy              self.writedxf(tn, str(ycenter))
1779 gcpy              self.writedxf(tn, "40")
1780 gcpy              self.writedxf(tn, str(radius))
1781 gcpy              self.writedxf(tn, "50")
1782 gcpy              self.writedxf(tn, str(anglebegin))
1783 gcpy              self.writedxf(tn, "51")
1784 gcpy              self.writedxf(tn, str(endangle))
1785 gcpy
1786 gcpy
1787 gcpy      def gcodearc(self, tn, xcenter, ycenter, radius, anglebegin,
1788 gcpy          endangle):
1789 gcpy          if (self.generategcode == True):
1790 gcpy              self.writegc(tn, "(0)")

```

---

The various textual versions are quite obvious, and due to the requirements of G-code, it is straight-forward to include the G-code in them if it is wanted.

---

```

1791 gcpy      def cutarcNECCdxlf(self, ex, ey, ez, xcenter, ycenter, radius):
1792 gcpy          global toolpath
1793 gcpy          toolpath = self.currenttool()
1794 gcpy          toolpath = toolpath.translate([self.xpos(), self.ypos(),
1795 gcpy              self.zpos()])
1796 gcpy          self.dxfarc(self.currenttoolnumber(), xcenter, ycenter,
1797 gcpy              radius, 0, 90)
1798 gcpy          if (self.zpos == ez):

```

```

1797 gcpy           self.settzpos(0)
1798 gcpy
1799 gcpy
1800 gcpy #
1801 gcpy #
1802 gcpy #
1803 gcpy #
1804 gcpy #
1805 gcpy
1806 gcpy #
1807 gcpy #
1808 gcpy #
1809 gcpy #
1810 gcpy
1811 gcpy
1812 gcpy
1813 gcpy #
1814 gcpy #
1815 gcpy #
1816 gcpy
1817 gcpy
1818 gcpy
1819 gcpy
1820 gcpy
1821 gcpy #
1822 gcpy #
1823 gcpy #
1824 gcpy #
1825 gcpy #
1826 gcpy #
1827 gcpy #
1828 gcpy
1829 gcpy
1830 gcpy
1831 gcpy
1832 gcpy #
1833 gcpy #
1834 gcpy #
1835 gcpy
1836 gcpy
1837 gcpy
1838 gcpy
1839 gcpy
1840 gcpy #
1841 gcpy #
1842 gcpy #
1843 gcpy
1844 gcpy
1845 gcpy #
1846 gcpy
1847 gcpy
1848 gcpy
1849 gcpy
1850 gcpy
1851 gcpy #
1852 gcpy #
1853 gcpy #
1854 gcpy
1855 gcpy
1856 gcpy
1857 gcpy
1858 gcpy
1859 gcpy #
1860 gcpy #
1861 gcpy #
1862 gcpy
1863 gcpy
1864 gcpy #
1865 gcpy
1866 gcpy
           self.setxpos(ex)
           self.setypos(ey)
           self.setzpos(ez)
           if self.generatepaths == True:
               print("Unioning cutarcNECCdxfs toolpath")
           self.arcloop(1, 90, xcenter, ycenter, radius)
           self.toolpaths = self.toolpaths.union(toolpath)
       else:
           toolpath = self.arcloop(1, 90, xcenter, ycenter,
           radius)
           print("Returning cutarcNECCdxfs toolpath")
       return toolpath

   def cutarcNWCCdxfs(self, ex, ey, ez, xcenter, ycenter, radius):
       global toolpath
       toolpath = self.currenttool()
       toolpath = toolpath.translate([self.xpos(), self.ypos(),
       self.zpos()])
       self.dxfarc(self.currenttoolnumber(), xcenter, ycenter,
       radius, 90, 180)
       if (self.zpos == ez):
           self.settzpos(0)
       else:
           self.settzpos((self.zpos()-ez)/90)
           self.setxpos(ex)
           self.setypos(ey)
           self.setzpos(ez)
           if self.generatepaths == True:
               self.arcloop(91, 180, xcenter, ycenter, radius)
               self.toolpaths = self.toolpaths.union(toolpath)
           else:
               toolpath = self.arcloop(91, 180, xcenter, ycenter, radius)
       return toolpath

   def cutarcSWCCdxfs(self, ex, ey, ez, xcenter, ycenter, radius):
       global toolpath
       toolpath = self.currenttool()
       toolpath = toolpath.translate([self.xpos(), self.ypos(),
       self.zpos()])
       self.dxfarc(self.currenttoolnumber(), xcenter, ycenter,
       radius, 180, 270)
       if (self.zpos == ez):
           self.settzpos(0)
       else:
           self.settzpos((self.zpos()-ez)/90)
           self.setxpos(ex)
           self.setypos(ey)
           self.setzpos(ez)
           if self.generatepaths == True:
               self.arcloop(181, 270, xcenter, ycenter, radius)
               self.toolpaths = self.toolpaths.union(toolpath)
           else:
               toolpath = self.arcloop(181, 270, xcenter, ycenter,
               radius)
       return toolpath

   def cutarcSECCdxfs(self, ex, ey, ez, xcenter, ycenter, radius):
       global toolpath
       toolpath = self.currenttool()
       toolpath = toolpath.translate([self.xpos(), self.ypos(),
       self.zpos()])
       self.dxfarc(self.currenttoolnumber(), xcenter, ycenter,
       radius, 270, 360)
       if (self.zpos == ez):
           self.settzpos(0)
       else:
           self.settzpos((self.zpos()-ez)/90)
           self.setxpos(ex)
           self.setypos(ey)
           self.setzpos(ez)
           if self.generatepaths == True:
               self.arcloop(271, 360, xcenter, ycenter, radius)
               self.toolpaths = self.toolpaths.union(toolpath)
           else:
               toolpath = self.arcloop(271, 360, xcenter, ycenter,
               radius)
       return toolpath

```

```

                radius)
1867 gcpy         return toolpath
1868 gcpy
1869 gcpy     def cutarcNECWdx (self, ex, ey, ez, xcenter, ycenter, radius):
1870 gcpy #         global toolpath
1871 gcpy #         toolpath = self.currenttool()
1872 gcpy #         toolpath = toolpath.translate([self.xpos(), self.ypos(),
1873 gcpy             self.zpos()])
1874 gcpy         self.dxfarc(self.currenttoolnumber(), xcenter, ycenter,
1875 gcpy             radius, 0, 90)
1876 gcpy     if (self.zpos == ez):
1877 gcpy         self.settzpos(0)
1878 gcpy     else:
1879 gcpy         self.settzpos((self.zpos()-ez)/90)
1880 gcpy         self.setxpos(ex)
1881 gcpy         self.setypos(ey)
1882 gcpy         self.setzpos(ez)
1883 gcpy     if self.generatepaths == True:
1884 gcpy         self.narcloop(89, 0, xcenter, ycenter, radius)
1885 gcpy         self.toolpaths = self.toolpaths.union(toolpath)
1886 gcpy     else:
1887 gcpy         toolpath = self.narcloop(89, 0, xcenter, ycenter,
1888 gcpy             radius)
1889 gcpy         return toolpath
1890 gcpy     def cutarcSECWdx (self, ex, ey, ez, xcenter, ycenter, radius):
1891 gcpy #         global toolpath
1892 gcpy #         toolpath = self.currenttool()
1893 gcpy #         toolpath = toolpath.translate([self.xpos(), self.ypos(),
1894 gcpy             self.zpos()])
1895 gcpy         self.dxfarc(self.currenttoolnumber(), xcenter, ycenter,
1896 gcpy             radius, 270, 360)
1897 gcpy     if (self.zpos == ez):
1898 gcpy         self.settzpos(0)
1899 gcpy     else:
1900 gcpy         self.settzpos((self.zpos()-ez)/90)
1901 gcpy         self.setxpos(ex)
1902 gcpy         self.setypos(ey)
1903 gcpy         self.setzpos(ez)
1904 gcpy     if self.generatepaths == True:
1905 gcpy         self.narcloop(359, 270, xcenter, ycenter, radius)
1906 gcpy         self.toolpaths = self.toolpaths.union(toolpath)
1907 gcpy     else:
1908 gcpy         toolpath = self.narcloop(359, 270, xcenter, ycenter,
1909 gcpy             radius)
1910 gcpy         return toolpath
1911 gcpy     def cutarcSWCWdx (self, ex, ey, ez, xcenter, ycenter, radius):
1912 gcpy #         global toolpath
1913 gcpy #         toolpath = self.currenttool()
1914 gcpy #         toolpath = toolpath.translate([self.xpos(), self.ypos(),
1915 gcpy             self.zpos()])
1916 gcpy         self.dxfarc(self.currenttoolnumber(), xcenter, ycenter,
1917 gcpy             radius, 180, 270)
1918 gcpy     if (self.zpos == ez):
1919 gcpy         self.settzpos(0)
1920 gcpy     else:
1921 gcpy         self.settzpos((self.zpos()-ez)/90)
1922 gcpy         self.setxpos(ex)
1923 gcpy         self.setypos(ey)
1924 gcpy         self.setzpos(ez)
1925 gcpy     if self.generatepaths == True:
1926 gcpy         self.narcloop(269, 180, xcenter, ycenter, radius)
1927 gcpy         self.toolpaths = self.toolpaths.union(toolpath)
1928 gcpy     else:
1929 gcpy         toolpath = self.narcloop(269, 180, xcenter, ycenter,
1930 gcpy             radius)
1931 gcpy         return toolpath
1932 gcpy     def cutarcNWCWdx (self, ex, ey, ez, xcenter, ycenter, radius):
1933 gcpy #         global toolpath
1934 gcpy #         toolpath = self.currenttool()
1935 gcpy #         toolpath = toolpath.translate([self.xpos(), self.ypos(),
1936 gcpy             self.zpos()])
1937 gcpy         self.dxfarc(self.currenttoolnumber(), xcenter, ycenter,
1938 gcpy             radius, 90, 180)
1939 gcpy     if (self.zpos == ez):
1940 gcpy         self.settzpos(0)

```

```

1933 gcpy      else:
1934 gcpy          self.settzpos((self.zpos()-ez)/90)
1935 gcpy #      self.setxpos(ex)
1936 gcpy #      self.setypos(ey)
1937 gcpy #      self.setzpos(ez)
1938 gcpy      if self.generatepaths == True:
1939 gcpy          self.narcloop(179, 90, xcenter, ycenter, radius)
1940 gcpy #          self.toolpaths = self.toolpaths.union(toolpath)
1941 gcpy      else:
1942 gcpy          toolpath = self.narcloop(179, 90, xcenter, ycenter,
1943 gcpy                  radius)
1943 gcpy      return toolpath

```

---

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

```

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

```

```

1945 gcpy      def arcCCgc(self, ex, ey, ez, xcenter, ycenter, radius):
1946 gcpy          self.writegc("G03\u00D7X", str(ex), "\u00D7Y", str(ey), "\u00D7Z", str(ez),
1947 gcpy                  , "\u00D7R", str(radius))
1948 gcpy      def arcCWgc(self, ex, ey, ez, xcenter, ycenter, radius):
1949 gcpy          self.writegc("G02\u00D7X", str(ex), "\u00D7Y", str(ey), "\u00D7Z", str(ez),
1950 gcpy                  , "\u00D7R", str(radius))

```

---

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

```

1951 gcpy      def cutarcNECCdxfgc(self, ex, ey, ez, xcenter, ycenter, radius):
1952 gcpy          :
1953 gcpy          self.arcCCgc(ex, ey, ez, xcenter, ycenter, radius)
1954 gcpy      if self.generatepaths == True:
1955 gcpy          self.cutarcNECCdxf(ex, ey, ez, xcenter, ycenter, radius)
1956 gcpy      else:
1957 gcpy          return self.cutarcNECCdxf(ex, ey, ez, xcenter, ycenter,
1958 gcpy                  radius)
1959 gcpy      def cutarcNWCCdxfgc(self, ex, ey, ez, xcenter, ycenter, radius):
1960 gcpy          :
1961 gcpy          self.arcCCgc(ex, ey, ez, xcenter, ycenter, radius)
1962 gcpy      if self.generatepaths == False:
1963 gcpy          return self.cutarcNWCCdxf(ex, ey, ez, xcenter, ycenter,
1964 gcpy                  radius)
1965 gcpy      def cutarcSWCCdxfgc(self, ex, ey, ez, xcenter, ycenter, radius):
1966 gcpy          :
1967 gcpy          self.arcCCgc(ex, ey, ez, xcenter, ycenter, radius)
1968 gcpy      if self.generatepaths == False:
1969 gcpy          return self.cutarcSWCCdxf(ex, ey, ez, xcenter, ycenter,
1970 gcpy                  radius)
1971 gcpy      def cutarcSECCdxfgc(self, ex, ey, ez, xcenter, ycenter, radius):
1972 gcpy          :
1973 gcpy          self.arcCCgc(ex, ey, ez, xcenter, ycenter, radius)
1974 gcpy      if self.generatepaths == False:
1975 gcpy          return self.cutarcSECCdxf(ex, ey, ez, xcenter, ycenter,
1976 gcpy                  radius)
1977 gcpy      def cutarcNECWdxfgc(self, ex, ey, ez, xcenter, ycenter, radius):
1978 gcpy          :
1979 gcpy          self.arcCWgc(ex, ey, ez, xcenter, ycenter, radius)
1980 gcpy      if self.generatepaths == False:
1981 gcpy          return self.cutarcNECWdxf(ex, ey, ez, xcenter, ycenter,
1982 gcpy                  radius)
1983 gcpy      def cutarcSECWdxfgc(self, ex, ey, ez, xcenter, ycenter, radius):
1984 gcpy          :

```

```

1984 gcpy          self.arcCWgc(ex, ey, ez, xcenter, ycenter, radius)
1985 gcpy          if self.generatepaths == False:
1986 gcpy              return self.cutarcSWCWdx(x, ey, ez, xcenter, ycenter,
1987 gcpy                  radius)
1988 gcpy          def cutarcNWCWdxfgc(self, ex, ey, ez, xcenter, ycenter, radius):
1989 gcpy              :
1990 gcpy          self.arcCWgc(ex, ey, ez, xcenter, ycenter, radius)
1991 gcpy          if self.generatepaths == False:
1992 gcpy              return self.cutarcNWCWdx(x, ey, ez, xcenter, ycenter,
1993 gcpy                  radius)

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

```

### 3.8.3 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) GOXYO 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: )	opengcodefile(s)(...); setupstock(...)
(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	
(Move to safe Z to avoid workholding)	movetosafez()
G53G0Z-5.000	
(Toolpath: Contour Toolpath 1)	toolchange(...);
M05	
(TOOL/MILL, 3.17, 0.00, 0.00, 0.00)	
M6T102	
M03S10000	
(PREPOSITION FOR RAPID PLUNGE)	writecomment(...)
G0X0.000Y-152.400	rapid(...)
Z0.250	rapid(...)
G1Z-1.000F203.2	cutwithfeed(...);
X109.500Y-77.400F508.0	cutwithfeed(...);
X57.918Y16.302Z-0.726	
Y22.023Z-1.023	
X61.190Z-0.681	
Y21.643	
X57.681	
Z12.700	
M05	closegcodefile();
M02	

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)

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

---

```

1993 gcpy      def dxfpostamble(self, tn):
1994 gcpy #          self.writedxf(tn, str(tn))
1995 gcpy          self.writedxf(tn, "0")
1996 gcpy          self.writedxf(tn, "ENDSEC")
1997 gcpy          self.writedxf(tn, "0")
1998 gcpy          self.writedxf(tn, "EOF")

```

---



---

```

2000 gcpy      def gcodepostamble(self):
2001 gcpy          self.writegc("Z12.700")
2002 gcpy          self.writegc("M05")
2003 gcpy          self.writegc("M02")

```

---

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

---

```

2005 gcpy      def closegcodefile(self):
2006 gcpy          if self.generategcode == True:
2007 gcpy              self.gcodepostamble()
2008 gcpy              self.gc.close()
2009 gcpy

```

```

2010 gcpy      def closedxfffile(self):
2011 gcpy          if self.generateddxf == True:
2012 gcpy #              global dxfclosed
2013 gcpy          self.dxfpostamble(-1)
2014 gcpy #              self.dxfclosed = True
2015 gcpy          self.dxf.close()
2016 gcpy
2017 gcpy      def closedxfffiles(self):
2018 gcpy          if self.generateddxfs == True:
2019 gcpy              if (self.large_square_tool_num > 0):
2020 gcpy                  self.dxfpostamble(self.large_square_tool_num)
2021 gcpy              if (self.small_square_tool_num > 0):
2022 gcpy                  self.dxfpostamble(self.small_square_tool_num)
2023 gcpy              if (self.large_ball_tool_num > 0):
2024 gcpy                  self.dxfpostamble(self.large_ball_tool_num)
2025 gcpy              if (self.small_ball_tool_num > 0):
2026 gcpy                  self.dxfpostamble(self.small_ball_tool_num)
2027 gcpy              if (self.large_V_tool_num > 0):
2028 gcpy                  self.dxfpostamble(self.large_V_tool_num)
2029 gcpy              if (self.small_V_tool_num > 0):
2030 gcpy                  self.dxfpostamble(self.small_V_tool_num)
2031 gcpy              if (self.DT_tool_num > 0):
2032 gcpy                  self.dxfpostamble(self.DT_tool_num)
2033 gcpy              if (self.KH_tool_num > 0):
2034 gcpy                  self.dxfpostamble(self.KH_tool_num)
2035 gcpy              if (self.Roundover_tool_num > 0):
2036 gcpy                  self.dxfpostamble(self.Roundover_tool_num)
2037 gcpy              if (self.MISC_tool_num > 0):
2038 gcpy                  self.dxfpostamble(self.MISC_tool_num)
2039 gcpy
2040 gcpy          if (self.large_square_tool_num > 0):
2041 gcpy              self.dxflgsq.close()
2042 gcpy          if (self.small_square_tool_num > 0):
2043 gcpy              self.dfxmsq.close()
2044 gcpy          if (self.large_ball_tool_num > 0):
2045 gcpy              self.dxflgbl.close()
2046 gcpy          if (self.small_ball_tool_num > 0):
2047 gcpy              self.dfxsmbl.close()
2048 gcpy          if (self.large_V_tool_num > 0):
2049 gcpy              self.dxflgV.close()
2050 gcpy          if (self.small_V_tool_num > 0):
2051 gcpy              self.dfxsmV.close()
2052 gcpy          if (self.DT_tool_num > 0):
2053 gcpy              self.dxfDT.close()
2054 gcpy          if (self.KH_tool_num > 0):
2055 gcpy              self.dxfKH.close()
2056 gcpy          if (self.Roundover_tool_num > 0):
2057 gcpy              self.dxfRt.close()
2058 gcpy          if (self.MISC_tool_num > 0):
2059 gcpy              self.dxfMt.close()

```

---

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

```

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

```

---

### 3.9 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 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.

### 3.9.1 Building blocks

The outlines of shapes will be defined using:

- lines — dxfline
- arcs — dxfarc

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

**3.9.1.1 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
  - **semicircle/circular/half-circle segment** (arc and a straight line); see also sector below
  - arch—curve possibly smoothly joining a pair of straight lines with a flat bottom
  - lens/vesica piscis (two convex curves)
  - lune/crescent (one convex, one concave curve)
  - heart (two curves)
  - tomoe (comma shape)—non-arc curves
- 3
  - **triangle**
    - \* equilateral
    - \* isosceles
    - \* right triangle
    - \* scalene
  - **(circular) sector** (two straight edges, one convex arc)
    - \* quadrant ( $90^\circ$ )
    - \* sextants ( $60^\circ$ )
    - \* octants ( $45^\circ$ )
  - deltoid curve (three concave arcs)
  - Reuleaux triangle (three convex arcs)
  - arbelos (one convex, two concave arcs)
  - two straight edges, one concave arc—an example is the hyperbolic sector<sup>1</sup>
  - two convex, one concave arc
- 4
  - **rectangle (including square)** — dxfrectangle, dxfrectangleround
  - **parallelogram**
  - **rhombus**
  - **trapezoid/trapezium**
  - **kite**

---

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

- ring/annulus segment (straight line, concave arc, straight line, convex arc)
- astroid (four concave arcs)
- **salinon** (four semicircles)
- three straight lines and one concave arc

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

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

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

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

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

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

Some further considerations:

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

### 3.9.1.1.1 circles Circles are made up of a series of arcs:

---

```

2061 gcpy      def dxfcircle(self, tool_num, xcenter, ycenter, radius):
2062 gcpy          self.dxfarc(tool_num, xcenter, ycenter, radius, 0, 90)
2063 gcpy          self.dxfarc(tool_num, xcenter, ycenter, radius, 90, 180)
2064 gcpy          self.dxfarc(tool_num, xcenter, ycenter, radius, 180, 270)
2065 gcpy          self.dxfarc(tool_num, xcenter, ycenter, radius, 270, 360)

```

---

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

Center is specified, but the actual entry point is the right-most edge.

---

```

2067 gcpy     def cutcircleCC(self, xcenter, ycenter, bz, ez, radius):
2068 gcpy         self.setzpos(bz)
2069 gcpy         self.cutquarterCCNE(xcenter, ycenter + radius, self.zpos()
2070 gcpy             + ez/4, radius)
2071 gcpy         self.cutquarterCCNW(xcenter - radius, ycenter, self.zpos()
2072 gcpy             + ez/4, radius)
2073 gcpy         self.cutquarterCCSW(xcenter, ycenter - radius, self.zpos()
2074 gcpy             + ez/4, radius)
2075 gcpy     def cutcircleCCdxfs(self, xcenter, ycenter, bz, ez, radius):
2076 gcpy         self.cutcircleCC(self, xcenter, ycenter, bz, ez, radius)
2077 gcpy         self.dxfcircle(self, tool_num, xcenter, ycenter, radius)

```

---

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

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

---

```

2078 gcpy     def dxfrectangle(self, tool_num, xorigin, yorigin, xwidth,
2079 gcpy         yheight, corners = "Square", radius = 6):
2080 gcpy         if corners == "Square":
2081 gcpy             self.dxfline(tool_num, xorigin, yorigin, xorigin +
2082 gcpy                 xwidth, yorigin)
2083 gcpy             self.dxfline(tool_num, xorigin + xwidth, yorigin,
2084 gcpy                 xorigin + xwidth, yorigin + yheight)
2085 gcpy             self.dxfline(tool_num, xorigin + xwidth, yorigin +
2086 gcpy                 yheight, xorigin, yorigin + yheight)
2087 gcpy             self.dxfline(tool_num, xorigin, yorigin + yheight,
2088 gcpy                 xorigin, yorigin)
2089 gcpy         elif corners == "Fillet":
2090 gcpy             self.dxfrectangleround(tool_num, xorigin, yorigin,
2091 gcpy                 xwidth, yheight, radius)
2092 gcpy         elif corners == "Chamfer":
2093 gcpy             self.dxfrectanglechamfer(tool_num, xorigin, yorigin,
2094 gcpy                 xwidth, yheight, radius)
2095 gcpy         elif corners == "Flipped_Fillet":
2096 gcpy             self.dxfrectangleflippedfillet(tool_num, xorigin,
2097 gcpy                 yorigin, xwidth, yheight, radius)

```

---

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

---

```

2098 gcpy     def dxfrectangleround(self, tool_num, xorigin, yorigin, xwidth,
2099 gcpy         yheight, radius):
2100 gcpy # begin section
2101 gcpy         self.writedxf(tool_num, "0")
2102 gcpy         self.writedxf(tool_num, "SECTION")
2103 gcpy         self.writedxf(tool_num, "2")
2104 gcpy         self.writedxf(tool_num, "ENTITIES")
2105 gcpy         self.writedxf(tool_num, "0")
2106 gcpy         self.writedxf(tool_num, "LWPOLYLINE")
2107 gcpy         self.writedxf(tool_num, "5")
2108 gcpy         self.writedxf(tool_num, "4E")
2109 gcpy         self.writedxf(tool_num, "100")
2110 gcpy         self.writedxf(tool_num, "AcDbEntity")
2111 gcpy         self.writedxf(tool_num, "8")
2112 gcpy         self.writedxf(tool_num, "0")
2113 gcpy         self.writedxf(tool_num, "6")
2114 gcpy         self.writedxf(tool_num, "ByLayer")
2115 gcpy #
2116 gcpy         self.writedxf(tool_num, "370")
2117 gcpy         self.writedxf(tool_num, "-1")
2118 gcpy         self.writedxf(tool_num, "100")
2119 gcpy         self.writedxf(tool_num, "AcDbPolyline")
2120 gcpy         self.writedxf(tool_num, "90")
2121 gcpy         self.writedxf(tool_num, "8")
2122 gcpy         self.writedxf(tool_num, "70")
2123 gcpy         self.writedxf(tool_num, "1")
2124 gcpy         self.writedxf(tool_num, "43")
2125 gcpy         self.writedxf(tool_num, "0")
2126 gcpy #1 upper right corner before arc (counter-clockwise)
2127 gcpy         self.writedxf(tool_num, "10")

```

---

```

2122 gcpy           self.writedxf(tool_num, str(xorigin + xwidth))
2123 gcpy           self.writedxf(tool_num, "20")
2124 gcpy           self.writedxf(tool_num, str(yorigin + yheight - radius))
2125 gcpy           self.writedxf(tool_num, "42")
2126 gcpy           self.writedxf(tool_num, "0.414213562373095")
2127 gcpy #2 upper right corner after arc
2128 gcpy           self.writedxf(tool_num, "10")
2129 gcpy           self.writedxf(tool_num, str(xorigin + xwidth - radius))
2130 gcpy           self.writedxf(tool_num, "20")
2131 gcpy           self.writedxf(tool_num, str(yorigin + yheight))
2132 gcpy #3 upper left corner before arc (counter-clockwise)
2133 gcpy           self.writedxf(tool_num, "10")
2134 gcpy           self.writedxf(tool_num, str(xorigin + radius))
2135 gcpy           self.writedxf(tool_num, "20")
2136 gcpy           self.writedxf(tool_num, str(yorigin + yheight))
2137 gcpy           self.writedxf(tool_num, "42")
2138 gcpy           self.writedxf(tool_num, "0.414213562373095")
2139 gcpy #4 upper left corner after arc
2140 gcpy           self.writedxf(tool_num, "10")
2141 gcpy           self.writedxf(tool_num, str(xorigin))
2142 gcpy           self.writedxf(tool_num, "20")
2143 gcpy           self.writedxf(tool_num, str(yorigin + yheight - radius))
2144 gcpy #5 lower left corner before arc (counter-clockwise)
2145 gcpy           self.writedxf(tool_num, "10")
2146 gcpy           self.writedxf(tool_num, str(xorigin))
2147 gcpy           self.writedxf(tool_num, "20")
2148 gcpy           self.writedxf(tool_num, str(yorigin + radius))
2149 gcpy           self.writedxf(tool_num, "42")
2150 gcpy           self.writedxf(tool_num, "0.414213562373095")
2151 gcpy #6 lower left corner after arc
2152 gcpy           self.writedxf(tool_num, "10")
2153 gcpy           self.writedxf(tool_num, str(xorigin + radius))
2154 gcpy           self.writedxf(tool_num, "20")
2155 gcpy           self.writedxf(tool_num, str(yorigin))
2156 gcpy #7 lower right corner before arc (counter-clockwise)
2157 gcpy           self.writedxf(tool_num, "10")
2158 gcpy           self.writedxf(tool_num, str(xorigin + xwidth - radius))
2159 gcpy           self.writedxf(tool_num, "20")
2160 gcpy           self.writedxf(tool_num, str(yorigin))
2161 gcpy           self.writedxf(tool_num, "42")
2162 gcpy           self.writedxf(tool_num, "0.414213562373095")
2163 gcpy #8 lower right corner after arc
2164 gcpy           self.writedxf(tool_num, "10")
2165 gcpy           self.writedxf(tool_num, str(xorigin + xwidth))
2166 gcpy           self.writedxf(tool_num, "20")
2167 gcpy           self.writedxf(tool_num, str(yorigin + radius))
2168 gcpy # end current section
2169 gcpy           self.writedxf(tool_num, "0")
2170 gcpy           self.writedxf(tool_num, "SEQEND")

```

---

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

```

2172 gcpy   def dxfrectanglechamfer(self, tool_num, xorigin, yorigin,
2173 gcpy       xwidth, yheight, radius):
2174 gcpy       self.dxfline(tool_num, xorigin + radius, yorigin, xorigin,
2175 gcpy           yorigin + radius)
2176 gcpy       self.dxfline(tool_num, xorigin, yorigin + yheight - radius,
2177 gcpy           xorigin + radius, yorigin + yheight)
2178 gcpy       self.dxfline(tool_num, xorigin + xwidth - radius, yorigin +
2179 gcpy           yheight, xorigin + xwidth, yorigin + yheight - radius)
2180 gcpy       self.dxfline(tool_num, xorigin + xwidth - radius, yorigin +
2181 gcpy           yheight, xorigin + radius, yorigin + yheight)
2182 gcpy       self.dxfline(tool_num, xorigin, yorigin + yheight - radius,
2183 gcpy           xorigin, yorigin + radius)

```

---

Flipped Fillet:

```

2183 gcpy   def dxfrectangleflippedfillet(self, tool_num, xorigin, yorigin,
2184 gcpy       xwidth, yheight, radius):

```

---

---

```

2184 gcpy      self.dxfarc(tool_num, xorigin, yorigin, radius, 0, 90)
2185 gcpy      self.dxfarc(tool_num, xorigin + xwidth, yorigin, radius,
                           90, 180)
2186 gcpy      self.dxfarc(tool_num, xorigin + xwidth, yorigin + yheight,
                           radius, 180, 270)
2187 gcpy      self.dxfarc(tool_num, xorigin, yorigin + yheight, radius,
                           270, 360)
2188 gcpy      self.dxfline(tool_num, xorigin + radius, yorigin, xorigin +
                           xwidth - radius, yorigin)
2189 gcpy      self.dxfline(tool_num, xorigin + xwidth, yorigin + radius,
                           xorigin + xwidth, yorigin + yheight - radius)
2190 gcpy      self.dxfline(tool_num, xorigin + xwidth - radius, yorigin +
                           yheight, xorigin + radius, yorigin + yheight)
2191 gcpy      self.dxfline(tool_num, xorigin, yorigin + yheight - radius,
                           xorigin, yorigin + radius)

```

---

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

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

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

---

```

2194 gcpy      def cutrectangle(self, tool_num, bx, by, bz, xwidth, yheight,
                           zdepth):
2195 gcpy          self.cutline(bx, by, bz)
2196 gcpy          self.cutline(bx, by, bz - zdepth)
2197 gcpy          self.cutline(bx + xwidth, by, bz - zdepth)
2198 gcpy          self.cutline(bx + xwidth, by + yheight, bz - zdepth)
2199 gcpy          self.cutline(bx, by + yheight, bz - zdepth)
2200 gcpy          self.cutline(bx, by, bz - zdepth)
2201 gcpy
2202 gcpy      def cutrectangledxf(self, tool_num, bx, by, bz, xwidth, yheight,
                           zdepth):
2203 gcpy          self.cutrectangle(tool_num, bx, by, bz, xwidth, yheight,
                           zdepth)
2204 gcpy          self.dxfrectangle(tool_num, bx, by, xwidth, yheight, "Square")

```

---

The rounded forms instantiate a radius:

---

```

2206 gcpy      def cutrectangleround(self, tool_num, bx, by, bz, xwidth,
                           yheight, zdepth, radius):
2207 gcpy      #           self.rapid(bx + radius, by, bz)
2208 gcpy          self.cutline(bx + radius, by, bz + zdepth)
2209 gcpy          self.cutline(bx + xwidth - radius, by, bz + zdepth)
2210 gcpy          self.cutquarterCCSE(bx + xwidth, by + radius, bz + zdepth,
                           radius)
2211 gcpy          self.cutline(bx + xwidth, by + yheight - radius, bz +
                           zdepth)
2212 gcpy          self.cutquarterCCNE(bx + xwidth - radius, by + yheight, bz
                           + zdepth, radius)
2213 gcpy          self.cutline(bx + radius, by + yheight, bz + zdepth)
2214 gcpy          self.cutquarterCCNW(bx, by + yheight - radius, bz + zdepth,
                           radius)
2215 gcpy          self.cutline(bx, by + radius, bz + zdepth)
2216 gcpy          self.cutquarterCCSW(bx + radius, by, bz + zdepth, radius)
2217 gcpy
2218 gcpy      def cutrectangleroundddxf(self, tool_num, bx, by, bz, xwidth,
                           yheight, zdepth, radius):
2219 gcpy          self.cutrectangleround(tool_num, bx, by, bz, xwidth,
                           yheight, zdepth, radius)
2220 gcpy          self.dxfrectangleround(tool_num, bx, by, xwidth, yheight,
                           radius)

```

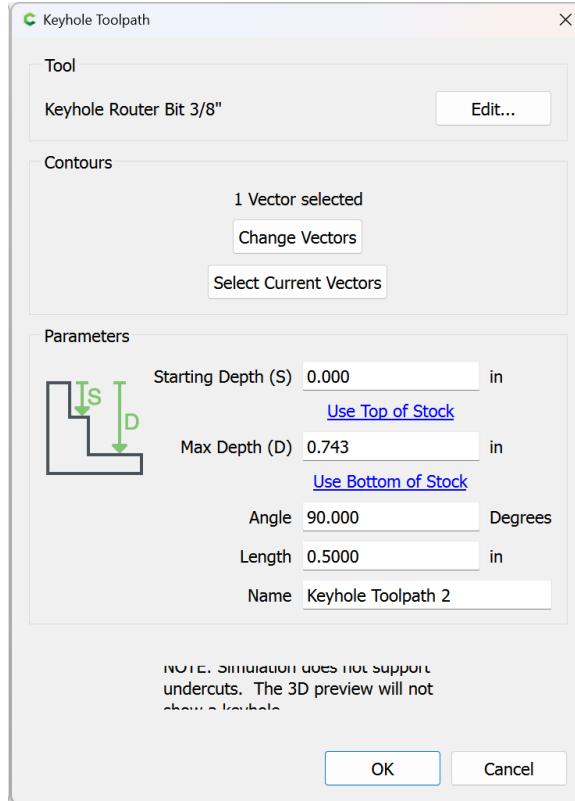
---

**3.9.1.1.3 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:

---

```

2222 gcpy      def cutkeyholegcdxf(self, kh_tool_num, kh_start_depth,
2223 gcpy          kh_max_depth, kht_direction, kh_distance):
2224 gcpy          if (kht_direction == "N"):
2225 gcpy              toolpath = self.cutKHgcdxf(kh_tool_num, kh_start_depth,
2226 gcpy                  kh_max_depth, 90, kh_distance)
2227 gcpy          elif (kht_direction == "S"):
2228 gcpy              toolpath = self.cutKHgcdxf(kh_tool_num, kh_start_depth,
2229 gcpy                  kh_max_depth, 270, kh_distance)
2230 gcpy          elif (kht_direction == "E"):
2231 gcpy              toolpath = self.cutKHgcdxf(kh_tool_num, kh_start_depth,
2232 gcpy                  kh_max_depth, 0, kh_distance)
2233 gcpy          elif (kht_direction == "W"):
2234 gcpy              toolpath = self.cutKHgcdxf(kh_tool_num, kh_start_depth,
2235 gcpy                  kh_max_depth, 180, kh_distance)
2236 gcpy      if self.generatepaths == True:

```

---

```

2232 gcpy #           self.toolpaths = union([self.toolpaths, toolpath])
2233 gcpy     return toolpath
2234 gcpy #
2235 gcpy #           else:
2235 gcpy #               return cube([0.01, 0.01, 0.01])

```

---



---

```

174 gpcscad module cutkeyholegcdxf(kh_tool_num, kh_start_depth, kh_max_depth,
174 gpcscad     kht_direction, kh_distance){
175 gpcscad     gcp.cutkeyholegcdxf(kh_tool_num, kh_start_depth, kh_max_depth,
175 gpcscad         kht_direction, kh_distance);
176 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). The first task is to place a circle at the origin which is invariant of angle:

---

```

2237 gcpy     def cutKHgcdxf(self, kh_tool_num, kh_start_depth, kh_max_depth,
2237 gcpy         kh_angle, kh_distance):
2238 gcpy         oXpos = self.xpos()
2239 gcpy         oYpos = self.ypos()
2240 gcpy         self.dxfKH(kh_tool_num, self.xpos(), self.ypos(),
2240 gcpy             kh_start_depth, kh_max_depth, kh_angle, kh_distance)
2241 gcpy         toolpath = self.cutline(self.xpos(), self.ypos(), -
2241 gcpy             kh_max_depth)
2242 gcpy         self.setxpos(oXpos)
2243 gcpy         self.setypos(oYpos)
2244 gcpy #
2244 gcpy     if self.generatepaths == False:
2245 gcpy     return toolpath
2246 gcpy #
2246 gcpy     else:
2247 gcpy #         return cube([0.001, 0.001, 0.001])

```

---



---

```

2249 gcpy     def dxfKH(self, kh_tool_num, oXpos, oYpos, kh_start_depth,
2249 gcpy         kh_max_depth, kh_angle, kh_distance):
2250 gcpy #         oXpos = self.xpos()
2251 gcpy #         oYpos = self.ypos()
2252 gcpy #Circle at entry hole
2253 gcpy         self.dxfarc(kh_tool_num, oXpos, oYpos, self.tool_radius(
2253 gcpy             kh_tool_num, 7), 0, 90)
2254 gcpy         self.dxfarc(kh_tool_num, oXpos, oYpos, self.tool_radius(
2254 gcpy             kh_tool_num, 7), 90, 180)
2255 gcpy         self.dxfarc(kh_tool_num, oXpos, oYpos, self.tool_radius(
2255 gcpy             kh_tool_num, 7), 180, 270)
2256 gcpy         self.dxfarc(kh_tool_num, oXpos, oYpos, self.tool_radius(
2256 gcpy             kh_tool_num, 7), 270, 360)

```

---

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

---

```

2257 gcpy #pre-calculate needed values
2258 gcpy     r = self.tool_radius(kh_tool_num, 7)
2259 gcpy #
2259 gcpy     print(r)
2260 gcpy     rt = self.tool_radius(kh_tool_num, 1)
2261 gcpy #
2261 gcpy     print(rt)
2262 gcpy     ro = math.sqrt((self.tool_radius(kh_tool_num, 1))**2-(self.
2262 gcpy         tool_radius(kh_tool_num, 7))**2)
2263 gcpy #
2263 gcpy     print(ro)
2264 gcpy     angle = math.degrees(math.acos(ro/rt))
2265 gcpy #Outlines of entry hole and slot
2266 gcpy     if (kh_angle == 0):
2267 gcpy #Lower left of entry hole
2268 gcpy         self.dxfarc(kh_tool_num, self.xpos(), self.ypos(), self
2268 gcpy             .tool_radius(kh_tool_num, 1), 180, 270)
2269 gcpy #Upper left of entry hole
2270 gcpy         self.dxfarc(kh_tool_num, self.xpos(), self.ypos(), self
2270 gcpy             .tool_radius(kh_tool_num, 1), 90, 180)
2271 gcpy #Upper right of entry hole
2272 gcpy     self.dxfarc(kh_tool_num, self.xpos(), self.ypos(), rt,
2272 gcpy         41.810, 90)
2273 gcpy     self.dxfarc(kh_tool_num, self.xpos(), self.ypos(), rt,
2273 gcpy         angle, 90)
2274 gcpy #Lower right of entry hole
2275 gcpy     self.dxfarc(kh_tool_num, self.xpos(), self.ypos(), rt,
2275 gcpy         270, 360-angle)
2276 gcpy #
2276 gcpy     self.dxfarc(kh_tool_num, self.xpos(), self.ypos(),

```

```

        self.tool_radius(kh_tool_num, 1), 270, 270+math.acos(self.
        tool_diameter(kh_tool_num, 5)/self.tool_diameter(kh_tool_num, 1)
        ))
2277 gcpy #Actual line of cut
2278 gcpy #           self.dxfline(kh_tool_num, self.xpos(), self.ypos(),
        self.xpos()+kh_distance, self.ypos())
2279 gcpy #upper right of end of slot (kh_max_depth+4.36))/2
2280 gcpy           self.dxfarc(kh_tool_num, self.xpos()+kh_distance, self.
        ypos(), self.tool_diameter(kh_tool_num, (
        kh_max_depth+4.36))/2, 0, 90)
2281 gcpy #lower right of end of slot
2282 gcpy           self.dxfarc(kh_tool_num, self.xpos()+kh_distance, self.
        ypos(), self.tool_diameter(kh_tool_num, (
        kh_max_depth+4.36))/2, 270, 360)
2283 gcpy #upper right slot
2284 gcpy           self.dxfline(kh_tool_num, self.xpos()+ro, self.ypos()-
        self.tool_diameter(kh_tool_num, 7)/2), self.xpos()+
        kh_distance, self.ypos()-(self.tool_diameter(
        kh_tool_num, 7)/2))
2285 gcpy #
           self.dxfline(kh_tool_num, self.xpos()+(math.sqrt((self.
        .tool_diameter(kh_tool_num, 1)^2)-(self.tool_diameter(
        kh_tool_num, 5)^2))/2), self.ypos()+(self.tool_diameter(
        kh_tool_num, (kh_max_depth))/2, ((kh_max_depth-6.34))/2)^2-
        (self.tool_diameter(kh_tool_num, (kh_max_depth-6.34))/2)^2, self.
        xpos()+(self.tool_diameter(kh_tool_num, (kh_max_depth))/2, kh_tool_num)
2286 gcpy #end position at top of slot
2287 gcpy #lower right slot
2288 gcpy           self.dxfline(kh_tool_num, self.xpos()+ro, self.ypos()+
        self.tool_diameter(kh_tool_num, 7)/2), self.xpos()+
        kh_distance, self.ypos()+(self.tool_diameter(
        kh_tool_num, 7)/2))
2289 gcpy #
           dxflne(kh_tool_num, self.xpos()+(math.sqrt((self.
        tool_diameter(kh_tool_num, 1)^2)-(self.tool_diameter(kh_tool_num,
        5)^2))/2), self.ypos()-(self.tool_diameter(kh_tool_num, (
        kh_max_depth))/2, ((kh_max_depth-6.34))/2)^2-
        (self.
        tool_diameter(kh_tool_num, (kh_max_depth-6.34))/2)^2, self.xpos
        ()+kh_distance, self.ypos()-(self.tool_diameter(kh_tool_num, (
        kh_max_depth))/2, KH_tool_num)
2290 gcpy #end position at top of slot
2291 gcpy #
           hull(){
2292 gcpy #
           translate([xpos(), ypos(), zpos()]){
2293 gcpy #
           keyhole_shaft(6.35, 9.525);
2294 gcpy #
           }
2295 gcpy #
           translate([xpos(), ypos(), zpos()-kh_max_depth]){
2296 gcpy #
           keyhole_shaft(6.35, 9.525);
2297 gcpy #
           }
2298 gcpy #
           }
2299 gcpy #
           hull(){
2300 gcpy #
           translate([xpos(), ypos(), zpos()-kh_max_depth]){
2301 gcpy #
           keyhole_shaft(6.35, 9.525);
2302 gcpy #
           }
2303 gcpy #
           translate([xpos()+kh_distance, ypos(), zpos()-kh_max_depth])
           {
2304 gcpy #
           keyhole_shaft(6.35, 9.525);
2305 gcpy #
           }
2306 gcpy #
           }
2307 gcpy #
           cutwithfeed(getxpos(), getypos(), -kh_max_depth, feed);
2308 gcpy #
           cutwithfeed(getxpos()+kh_distance, getypos(), -kh_max_depth,
           feed);
2309 gcpy #
           setxpos(getxpos()-kh_distance);
2310 gcpy #
           } else if (kh_angle > 0 && kh_angle < 90) {
2311 gcpy #//echo(kh_angle);
2312 gcpy #
           dxfarc(getxpos(), getypos(), tool_diameter(KH_tool_num, (
           kh_max_depth))/2, 90+kh_angle, 180+kh_angle, KH_tool_num);
2313 gcpy #
           dxfarc(getxpos(), getypos(), tool_diameter(KH_tool_num, (
           kh_max_depth))/2, 180+kh_angle, 270+kh_angle, KH_tool_num);
2314 gcpy #dxfarc(getxpos(), getypos(), tool_diameter(KH_tool_num, (
           kh_max_depth))/2, kh_angle+asin((tool_diameter(KH_tool_num, (
           kh_max_depth+4.36))/2)/(tool_diameter(KH_tool_num, (kh_max_depth
           ))/2)), 90+kh_angle, KH_tool_num);
2315 gcpy #dxfarc(getxpos(), getypos(), tool_diameter(KH_tool_num, (
           kh_max_depth))/2, 270+kh_angle, 360+kh_angle-asin((tool_diameter
           (KH_tool_num, (kh_max_depth+4.36))/2)/(tool_diameter(KH_tool_num
           , (kh_max_depth))/2)), KH_tool_num);
2316 gcpy #dxfarc(getxpos()+(kh_distance*cos(kh_angle)),
2317 gcpy #
           getypos()+(kh_distance*sin(kh_angle)), tool_diameter(KH_tool_num
           , (kh_max_depth+4.36))/2, 0+kh_angle, 90+kh_angle, KH_tool_num);

```

```

2318 gcpy #dxfarc(getxpos()+(kh_distance*cos(kh_angle)), getypos()+(  

    kh_distance*sin(kh_angle)), tool_diameter(KH_tool_num, (  

        kh_max_depth+4.36))/2, 270+kh_angle, 360+kh_angle, KH_tool_num);  

2319 gcpy #dxfline( getxpos() +tool_diameter(KH_tool_num, (kh_max_depth))/2*  

    cos(kh_angle+asin((tool_diameter(KH_tool_num, (kh_max_depth)  

        +4.36))/2)/(tool_diameter(KH_tool_num, (kh_max_depth))/2))),  

2320 gcpy # getypos() +tool_diameter(KH_tool_num, (kh_max_depth))/2*sin(  

    kh_angle+asin((tool_diameter(KH_tool_num, (kh_max_depth+4.36))  

        /2)/(tool_diameter(KH_tool_num, (kh_max_depth))/2))),  

2321 gcpy # getxpos() +(kh_distance*cos(kh_angle))-((tool_diameter(KH_tool_num  

        , (kh_max_depth+4.36))/2)*sin(kh_angle)),  

2322 gcpy # getypos() +(kh_distance*sin(kh_angle))+((tool_diameter(KH_tool_num  

        , (kh_max_depth+4.36))/2)*cos(kh_angle)), KH_tool_num);  

2323 gcpy #//echo("a", tool_diameter(KH_tool_num, (kh_max_depth+4.36))/2);  

2324 gcpy #//echo("c", tool_diameter(KH_tool_num, (kh_max_depth))/2);  

2325 gcpy #echo("Aangle", asin((tool_diameter(KH_tool_num, (kh_max_depth  

        +4.36))/2)/(tool_diameter(KH_tool_num, (kh_max_depth))/2)));  

2326 gcpy #//echo(kh_angle);  

2327 gcpy # cutwithfeed(getxpos()+(kh_distance*cos(kh_angle)), getypos()+(  

    kh_distance*sin(kh_angle)), -kh_max_depth, feed);  

2328 gcpy #           toolpath = toolpath.union(self.cutline(self.xpos() +  

    kh_distance, self.ypos(), -kh_max_depth))  

2329 gcpy     elif (kh_angle == 90):  

2330 gcpy #Lower left of entry hole  

2331 gcpy         self.dxfarc(kh_tool_num, oXpos, oYpos, self.tool_radius  

            (kh_tool_num, 1), 180, 270)  

2332 gcpy #Lower right of entry hole  

2333 gcpy         self.dxfarc(kh_tool_num, oXpos, oYpos, self.tool_radius  

            (kh_tool_num, 1), 270, 360)  

2334 gcpy #left slot  

2335 gcpy         self.dxfline(kh_tool_num, oXpos-r, oYpos+ro, oXpos-r,  

            oYpos+kh_distance)  

2336 gcpy #right slot  

2337 gcpy         self.dxfline(kh_tool_num, oXpos+r, oYpos+ro, oXpos+r,  

            oYpos+kh_distance)  

2338 gcpy #upper left of end of slot  

2339 gcpy         self.dxfarc(kh_tool_num, oXpos, oYpos+kh_distance, r,  

            90, 180)  

2340 gcpy #upper right of end of slot  

2341 gcpy         self.dxfarc(kh_tool_num, oXpos, oYpos+kh_distance, r,  

            0, 90)  

2342 gcpy #Upper right of entry hole  

2343 gcpy         self.dxfarc(kh_tool_num, oXpos, oYpos, rt, 0, 90-angle)  

2344 gcpy #Upper left of entry hole  

2345 gcpy         self.dxfarc(kh_tool_num, oXpos, oYpos, rt, 90+angle,  

            180)  

2346 gcpy #           toolpath = toolpath.union(self.cutline(oXpos, oYpos +  

    kh_distance, -kh_max_depth))  

2347 gcpy     elif (kh_angle == 180):  

2348 gcpy #Lower right of entry hole  

2349 gcpy         self.dxfarc(kh_tool_num, oXpos, oYpos, self.tool_radius  

            (kh_tool_num, 1), 270, 360)  

2350 gcpy #Upper right of entry hole  

2351 gcpy         self.dxfarc(kh_tool_num, oXpos, oYpos, self.tool_radius  

            (kh_tool_num, 1), 0, 90)  

2352 gcpy #Upper left of entry hole  

2353 gcpy         self.dxfarc(kh_tool_num, oXpos, oYpos, rt, 90, 180-  

            angle)  

2354 gcpy #Lower left of entry hole  

2355 gcpy         self.dxfarc(kh_tool_num, oXpos, oYpos, rt, 180+angle,  

            270)  

2356 gcpy #upper slot  

2357 gcpy         self.dxfline(kh_tool_num, oXpos-ro, oYpos-r, oXpos -  

            kh_distance, oYpos-r)  

2358 gcpy #lower slot  

2359 gcpy         self.dxfline(kh_tool_num, oXpos-ro, oYpos+r, oXpos -  

            kh_distance, oYpos+r)  

2360 gcpy #upper left of end of slot  

2361 gcpy         self.dxfarc(kh_tool_num, oXpos-kh_distance, oYpos, r,  

            90, 180)  

2362 gcpy #lower left of end of slot  

2363 gcpy         self.dxfarc(kh_tool_num, oXpos-kh_distance, oYpos, r,  

            180, 270)  

2364 gcpy #           toolpath = toolpath.union(self.cutline(oXpos -  

    kh_distance, oYpos, -kh_max_depth))  

2365 gcpy     elif (kh_angle == 270):  

2366 gcpy #Upper left of entry hole  

2367 gcpy         self.dxfarc(kh_tool_num, oXpos, oYpos, self.tool_radius

```

```

            (kh_tool_num, 1), 90, 180)
2368 gcpy #Upper right of entry hole
2369 gcpy           self.dxfarc(kh_tool_num, oXpos, oYpos, self.tool_radius
                           (kh_tool_num, 1), 0, 90)
2370 gcpy #left slot
2371 gcpy           self.dxfline(kh_tool_num, oXpos-r, oYpos-ro, oXpos-r,
                           oYpos-kh_distance)
2372 gcpy #right slot
2373 gcpy           self.dxfline(kh_tool_num, oXpos+r, oYpos-ro, oXpos+r,
                           oYpos-kh_distance)
2374 gcpy #lower left of end of slot
2375 gcpy           self.dxfarc(kh_tool_num, oXpos, oYpos-kh_distance, r,
                           180, 270)
2376 gcpy #lower right of end of slot
2377 gcpy           self.dxfarc(kh_tool_num, oXpos, oYpos-kh_distance, r,
                           270, 360)
2378 gcpy #lower right of entry hole
2379 gcpy           self.dxfarc(kh_tool_num, oXpos, oYpos, rt, 180, 270-
                           angle)
2380 gcpy #lower left of entry hole
2381 gcpy           self.dxfarc(kh_tool_num, oXpos, oYpos, rt, 270+angle,
                           360)
2382 gcpy #
           toolpath = toolpath.union(self.cutline(oXpos, oYpos-
                           kh_distance, -kh_max_depth))
2383 gcpy #
           print(self.zpos())
2384 gcpy #
           self.setxpos(oXpos)
2385 gcpy #
           self.setypos(oYpos)
2386 gcpy #
           if self.generatepaths == False:
2387 gcpy #
           return toolpath
2388 gcpy
2389 gcpy # } else if (kh_angle == 90) {
2390 gcpy #
           //Lower left of entry hole
2391 gcpy #
           dxfarc(getxpos(), getypos(), 9.525/2, 180, 270, KH_tool_num);
2392 gcpy #
           //Lower right of entry hole
2393 gcpy #
           dxfarc(getxpos(), getypos(), 9.525/2, 270, 360, KH_tool_num);
2394 gcpy #
           //Upper right of entry hole
2395 gcpy #
           dxfarc(getxpos(), getypos(), 9.525/2, 0, acos(tool_diameter(
                           KH_tool_num, 5)/tool_diameter(KH_tool_num, 1)), KH_tool_num);
2396 gcpy #
           //Upper left of entry hole
2397 gcpy #
           dxfarc(getxpos(), getypos(), 9.525/2, 180-acos(tool_diameter(
                           KH_tool_num, 5)/tool_diameter(KH_tool_num, 1)), 180, KH_tool_num
                           );
2398 gcpy #
           //Actual line of cut
2399 gcpy #
           dxfline(getxpos(), getypos(), getxpos(), getypos()+kh_distance
                           );
2400 gcpy #
           //upper right of slot
2401 gcpy #
           dxfarc(getxpos(), getypos()+kh_distance, tool_diameter(
                           KH_tool_num, (kh_max_depth+4.36))/2, 0, 90, KH_tool_num);
2402 gcpy #
           //upper left of slot
2403 gcpy #
           dxfarc(getxpos(), getypos()+kh_distance, tool_diameter(
                           KH_tool_num, (kh_max_depth+6.35))/2, 90, 180, KH_tool_num);
2404 gcpy #
           //right of slot
2405 gcpy #
           dxfline(
2406 gcpy #
               getxpos() + tool_diameter(KH_tool_num, (kh_max_depth))/2,
2407 gcpy #
               getypos() + (math.sqrt((tool_diameter(KH_tool_num, 1)^2)-
                           tool_diameter(KH_tool_num, 5)^2))/2, //((kh_max_depth-6.34))/
                           2^2-(tool_diameter(KH_tool_num, (kh_max_depth-6.34))/2)^2,
2408 gcpy #
               getxpos() + tool_diameter(KH_tool_num, (kh_max_depth))/2,
2409 gcpy #
               //end position at top of slot
2410 gcpy #
               getypos() + kh_distance,
2411 gcpy #
               KH_tool_num);
2412 gcpy #
           dxfline(getxpos()-tool_diameter(KH_tool_num, (kh_max_depth))/2,
                           getypos() + (math.sqrt((tool_diameter(KH_tool_num, 1)^2)-
                           tool_diameter(KH_tool_num, 5)^2))/2, getxpos()-tool_diameter(
                           KH_tool_num, (kh_max_depth+6.35))/2, getypos() + kh_distance,
                           KH_tool_num);
2413 gcpy #
           hull() {
2414 gcpy #
               translate([xpos(), ypos(), zpos()]) {
2415 gcpy #
                   keyhole_shaft(6.35, 9.525);
2416 gcpy #
               }
2417 gcpy #
               translate([xpos(), ypos(), zpos()-kh_max_depth]) {
2418 gcpy #
                   keyhole_shaft(6.35, 9.525);
2419 gcpy #
               }
2420 gcpy #
           }
2421 gcpy #
           hull() {
2422 gcpy #
               translate([xpos(), ypos(), zpos()-kh_max_depth]) {
2423 gcpy #
                   keyhole_shaft(6.35, 9.525);
2424 gcpy #
               }

```

```

2425 gcpy #      translate([xpos(), ypos()+kh_distance, zpos()-kh_max_depth])
2426 gcpy #      {
2427 gcpy #      keyhole_shaft(6.35, 9.525);
2428 gcpy #
2429 gcpy #      }
2430 gcpy #      cutwithfeed(getxpos(), getypos(), -kh_max_depth, feed);
2431 gcpy #      cutwithfeed(getxpos(), getypos()+kh_distance, -kh_max_depth,
2432 gcpy #      feed);
2433 gcpy #      setypos(getypos()-kh_distance);
2434 gcpy # } else if (kh_angle == 180) {
2435 gcpy # //Lower right of entry hole
2436 gcpy # dxfarc(getxpos(), getypos(), 9.525/2, 270, 360, KH_tool_num);
2437 gcpy # //Upper right of entry hole
2438 gcpy # dxfarc(getxpos(), getypos(), 9.525/2, 0, 90, KH_tool_num);
2439 gcpy # //Upper left of entry hole
2440 gcpy # dxfarc(getxpos(), getypos(), 9.525/2, 90, 90+acos(
2441 gcpy #      tool_diameter(KH_tool_num, 5)/tool_diameter(KH_tool_num, 1)),
2442 gcpy #      KH_tool_num);
2443 gcpy # //Lower left of entry hole
2444 gcpy # dxfarc(getxpos(), getypos(), 9.525/2, 270-acos(tool_diameter(
2445 gcpy #      KH_tool_num, 5)/tool_diameter(KH_tool_num, 1)), 270, KH_tool_num)
2446 gcpy # );
2447 gcpy # //upper left slot
2448 gcpy # dxfline(
2449 gcpy #      getxpos()-(math.sqrt((tool_diameter(KH_tool_num, 1)^2)-
2450 gcpy #      tool_diameter(KH_tool_num, 5)^2))/2,
2451 gcpy #      getypos()+tool_diameter(KH_tool_num, (kh_max_depth))/2,
2452 gcpy # //((kh_max_depth-6.34))/2)^2-(tool_diameter(KH_tool_num, (
2453 gcpy #      kh_max_depth-6.34))/2)^2,
2454 gcpy #      getxpos()-kh_distance,
2455 gcpy # //end position at top of slot
2456 gcpy # dxfline(
2457 gcpy #      getxpos()-(math.sqrt((tool_diameter(KH_tool_num, 1)^2)-
2458 gcpy #      tool_diameter(KH_tool_num, 5)^2))/2,
2459 gcpy #      getypos()-tool_diameter(KH_tool_num, (kh_max_depth))/2,
2460 gcpy # //((kh_max_depth-6.34))/2)^2-(tool_diameter(KH_tool_num, (
2461 gcpy #      kh_max_depth-6.34))/2)^2,
2462 gcpy #      getxpos()-kh_distance,
2463 gcpy # //end position at top of slot
2464 gcpy # hull(){
2465 gcpy #      translate([xpos(), ypos(), zpos()]){
2466 gcpy #      keyhole_shaft(6.35, 9.525);
2467 gcpy #      }
2468 gcpy #      translate([xpos(), ypos(), zpos()-kh_max_depth]){
2469 gcpy #      keyhole_shaft(6.35, 9.525);
2470 gcpy #      }
2471 gcpy #      hull(){
2472 gcpy #      translate([xpos(), ypos(), zpos()-kh_max_depth]){
2473 gcpy #      keyhole_shaft(6.35, 9.525);
2474 gcpy #      }
2475 gcpy #      translate([xpos()-kh_distance, ypos(), zpos()-kh_max_depth])
2476 gcpy #      {
2477 gcpy #      keyhole_shaft(6.35, 9.525);
2478 gcpy #      }
2479 gcpy #      cutwithfeed(getxpos(), getypos(), -kh_max_depth, feed);
2480 gcpy #      cutwithfeed(getxpos()-kh_distance, getypos(), -kh_max_depth,
2481 gcpy #      feed);
2482 gcpy #      setxpos(getxpos()+kh_distance);
2483 gcpy # } else if (kh_angle == 270) {
2484 gcpy # //Upper right of entry hole
2485 gcpy # dxfarc(getxpos(), getypos(), 9.525/2, 0, 90, KH_tool_num);

```

```

2486 gcpy #      dxfarc(getxpos(), getypos(), 9.525/2, 90, 180, KH_tool_num);
2487 gcpy #      //lower right of slot
2488 gcpy #      dxfarc(getxpos(), getypos()-kh_distance, tool_diameter(
2489 gcpy #          KH_tool_num, (kh_max_depth+4.36))/2, 270, 360, KH_tool_num);
2490 gcpy #      //lower left of slot
2491 gcpy #      dxfarc(getxpos(), getypos()-kh_distance, tool_diameter(
2492 gcpy #          KH_tool_num, (kh_max_depth+4.36))/2, 180, 270, KH_tool_num);
2493 gcpy #      //Actual line of cut
2492 gcpy #      dxfline(getxpos(), getypos(), getxpos(), getypos()-kh_distance
2493 gcpy #      );
2493 gcpy #      //right of slot
2494 gcpy #      dxfline(
2495 gcpy #          getxpos() + tool_diameter(KH_tool_num, (kh_max_depth))/2,
2496 gcpy #          getypos() - (math.sqrt((tool_diameter(KH_tool_num, 1)^2) -
2497 gcpy #              tool_diameter(KH_tool_num, 5)^2))/2, //((kh_max_depth-6.34))
2498 gcpy #              /2)^2 - (tool_diameter(KH_tool_num, (kh_max_depth-6.34))/2)^2,
2499 gcpy #              getxpos() + tool_diameter(KH_tool_num, (kh_max_depth))/2,
2500 gcpy #      //end position at top of slot
2501 gcpy #      getypos() - kh_distance,
2502 gcpy #      KH_tool_num);
2501 gcpy #      //left of slot
2502 gcpy #      dxfline(
2503 gcpy #          getxpos() - tool_diameter(KH_tool_num, (kh_max_depth))/2,
2504 gcpy #          getypos() - (math.sqrt((tool_diameter(KH_tool_num, 1)^2) -
2505 gcpy #              tool_diameter(KH_tool_num, 5)^2))/2, //((kh_max_depth-6.34))
2506 gcpy #              /2)^2 - (tool_diameter(KH_tool_num, (kh_max_depth-6.34))/2)^2,
2507 gcpy #              getxpos() - tool_diameter(KH_tool_num, (kh_max_depth))/2,
2508 gcpy #      //end position at top of slot
2509 gcpy #      getypos() - kh_distance,
2509 gcpy #      KH_tool_num);
2510 gcpy #      //Lower right of entry hole
2510 gcpy #      dxfarc(getxpos(), getypos(), 9.525/2, 360-acos(tool_diameter(
2511 gcpy #          KH_tool_num, 5)/tool_diameter(KH_tool_num, 1)), 360, KH_tool_num)
2511 gcpy #      );
2512 gcpy #      //Lower left of entry hole
2512 gcpy #      dxfarc(getxpos(), getypos(), 9.525/2, 180, 180+acos(
2513 gcpy #          tool_diameter(KH_tool_num, 5)/tool_diameter(KH_tool_num, 1)),
2513 gcpy #          KH_tool_num);
2513 gcpy #      hull(){
2514 gcpy #          translate([xpos(), ypos(), zpos()]){
2515 gcpy #              keyhole_shaft(6.35, 9.525);
2516 gcpy #          }
2517 gcpy #          translate([xpos(), ypos(), zpos()-kh_max_depth]){
2518 gcpy #              keyhole_shaft(6.35, 9.525);
2519 gcpy #          }
2520 gcpy #      }
2521 gcpy #      hull(){
2522 gcpy #          translate([xpos(), ypos(), zpos()-kh_max_depth]){
2523 gcpy #              keyhole_shaft(6.35, 9.525);
2524 gcpy #          }
2525 gcpy #          translate([xpos(), ypos()-kh_distance, zpos()-kh_max_depth])
2525 gcpy #      {
2526 gcpy #          keyhole_shaft(6.35, 9.525);
2527 gcpy #      }
2528 gcpy #  }
2529 gcpy #  cutwithfeed(getxpos(), getypos(), -kh_max_depth, feed);
2530 gcpy #  cutwithfeed(getxpos(), getypos()-kh_distance, -kh_max_depth,
2530 gcpy #      feed);
2531 gcpy #  setypos(getypos() + kh_distance);
2532 gcpy # }
2533 gcpy # }
```

---

**3.9.1.1.4 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:

---

```

2535 gcpy     def cut_pins(self, Joint_Width, stockZthickness,
2536 gcpy         Number_of_Dovetails, Spacing, Proportion, DTT_diameter,
2537 gcpy         DTT_angle):
2538 gcpy         DTO = Tan(DTT_angle) * (stockZthickness * Proportion)
2539 gcpy         DTR = DTT_diameter/2 - DTO
2540 gcpy         cpr = self.rapidXY(0, stockZthickness + Spacing/2)
2541 gcpy         ctp = self.cutlinedxfgc(self.xpos(), self.ypos(), -
2542 gcpy             stockZthickness * Proportion)
2543 gcpy         #           ctp = ctp.union(self.cutlinedxfgc(Joint_Width / (
2544 gcpy             Number_of_Dovetails * 2), self.ypos(), -stockZthickness *
2545 gcpy             Proportion))
2546 gcpy         i = 1
2547 gcpy         while i < Number_of_Dovetails * 2:
2548 gcpy             print(i)
2549 gcpy             ctp = ctp.union(self.cutlinedxfgc(i * (Joint_Width / (
2550 gcpy                 Number_of_Dovetails * 2)), self.ypos(), -
2551 gcpy                 stockZthickness * Proportion))
2552 gcpy             ctp = ctp.union(self.cutlinedxfgc(i * (Joint_Width / (
2553 gcpy                 Number_of_Dovetails * 2)), (stockZthickness +
2554 gcpy                 Spacing) + (stockZthickness * Proportion) - (
2555 gcpy                 DTT_diameter/2), -(stockZthickness * Proportion)))
2556 gcpy             ctp = ctp.union(self.cutlinedxfgc(i * (Joint_Width / (
2557 gcpy                 Number_of_Dovetails * 2)), stockZthickness + Spacing
2558 gcpy                 / 2, -(stockZthickness * Proportion)))
2559 gcpy             ctp = ctp.union(self.cutlinedxfgc((i + 1) * (
2560 gcpy                 Joint_Width / (Number_of_Dovetails * 2)),
2561 gcpy                 stockZthickness + Spacing/2, -(stockZthickness *
2562 gcpy                 Proportion)))
2563 gcpy             self.dxrectangleround(self.currenttoolnumber(),
2564 gcpy                 i * (Joint_Width / (Number_of_Dovetails * 2))-DTR,
2565 gcpy                 stockZthickness + (Spacing/2) - DTR,
2566 gcpy                 DTR * 2,
2567 gcpy                 (stockZthickness * Proportion) + Spacing/2 + DTR *
2568 gcpy                 2 - (DTT_diameter/2),
2569 gcpy                 DTR)
2570 gcpy             i += 2
2571 gcpy             self.rapidZ(0)
2572 gcpy             return ctp

```

---

and

---

```

2573 gcpy     def cut_tails(self, Joint_Width, stockZthickness,
2574 gcpy         Number_of_Dovetails, Spacing, Proportion, DTT_diameter,
2575 gcpy         DTT_angle):
2576 gcpy         DTO = Tan(DTT_angle) * (stockZthickness * Proportion)
2577 gcpy         DTR = DTT_diameter/2 - DTO
2578 gcpy         cpr = self.rapidXY(0, 0)
2579 gcpy         ctp = self.cutlinedxfgc(self.xpos(), self.ypos(), -
2580 gcpy             stockZthickness * Proportion)
2581 gcpy         ctp = ctp.union(self.cutlinedxfgc(
2582 gcpy             Joint_Width / (Number_of_Dovetails * 2) - (DTT_diameter
2583 gcpy             - DTO),
2584 gcpy             self.ypos(),
2585 gcpy             -stockZthickness * Proportion))
2586 gcpy         i = 1
2587 gcpy         while i < Number_of_Dovetails * 2:
2588 gcpy             ctp = ctp.union(self.cutlinedxfgc(
2589 gcpy                 i * (Joint_Width / (Number_of_Dovetails * 2)) - (
2590 gcpy                     DTT_diameter - DTO),
2591 gcpy                     stockZthickness * Proportion - DTT_diameter / 2,
2592 gcpy                     -(stockZthickness * Proportion)))
2593 gcpy             ctp = ctp.union(self.cutarcCWdx(180, 90,
2594 gcpy                 i * (Joint_Width / (Number_of_Dovetails * 2)),
2595 gcpy                 stockZthickness * Proportion - DTT_diameter / 2,
2596 gcpy                 self.ypos(),
2597 gcpy                 DTT_diameter - DTO, 0, 1))
2598 gcpy             ctp = ctp.union(self.cutarcCWdx(90, 0,
2599 gcpy                 i * (Joint_Width / (Number_of_Dovetails * 2)),
2600 gcpy                 stockZthickness * Proportion - DTT_diameter / 2,
2601 gcpy                 DTT_diameter - DTO, 0, 1))
2602 gcpy             ctp = ctp.union(self.cutlinedxfgc(
2603 gcpy                 i * (Joint_Width / (Number_of_Dovetails * 2)) + (
2604 gcpy                     DTT_diameter - DTO),
2605 gcpy                     0,
2606 gcpy                     -(stockZthickness * Proportion)))
2607 gcpy             ctp = ctp.union(self.cutlinedxfgc(
2608 gcpy                 (i + 2) * (Joint_Width / (Number_of_Dovetails * 2)))

```

---

```

2588 gcpy           - (DTT_diameter - DTO),
2589 gcpy           0,
2590 gcpy           -(stockZthickness * Proportion)))
2591 gcpy           i += 2
2592 gcpy           self.rapidZ(0)
2593 gcpy           self.rapidXY(0, 0)
2594 gcpy           ctp = ctp.union(self.cutlinedxfgc(self.xpos(), self.ypos(),
2595 gcpy           -stockZthickness * Proportion))
2596 gcpy           self.dxfarc(self.currenttoolnumber(), 0, 0, DTR, 180, 270)
2597 gcpy           self.dxfline(self.currenttoolnumber(), -DTR, 0, -DTR,
2598 gcpy           stockZthickness + DTR)
2599 gcpy           self.dxfarc(self.currenttoolnumber(), 0, stockZthickness +
2600 gcpy           DTR, DTR, 90, 180)
2601 gcpy           self.dxfline(self.currenttoolnumber(), 0, stockZthickness +
2602 gcpy           DTR * 2, Joint_Width, stockZthickness + DTR * 2)
i = 0
while i < Number_of_Dovetails * 2:
    ctp = ctp.union(self.cutline(i * (Joint_Width /
2603 gcpy           Number_of_Dovetails * 2)), stockZthickness + DTO, -( stockZthickness * Proportion)))
2604 gcpy           ctp = ctp.union(self.cutline((i+2) * (Joint_Width /
2605 gcpy           Number_of_Dovetails * 2)), stockZthickness + DTO, -( stockZthickness * Proportion)))
2606 gcpy           ctp = ctp.union(self.cutline((i+2) * (Joint_Width /
2607 gcpy           Number_of_Dovetails * 2)), 0, -(stockZthickness *
2608 gcpy           Proportion))
self.dxfarc(self.currenttoolnumber(), i * (Joint_Width
2609 gcpy           / (Number_of_Dovetails * 2)), 0, DTR, 270, 360)
self.dxfline(self.currenttoolnumber(),
2610 gcpy           i * (Joint_Width / (Number_of_Dovetails * 2)) + DTR
2611 gcpy           ,
2612 gcpy           0,
2613 gcpy           i * (Joint_Width / (Number_of_Dovetails * 2)) + DTR
2614 gcpy           , stockZthickness * Proportion - DTT_diameter /
2615 gcpy           2)
2616 gcpy           self.dxfarc(self.currenttoolnumber(), Joint_Width,
2617 gcpy           stockZthickness + DTR, DTR, 0, 90)
2618 gcpy           self.dxfline(self.currenttoolnumber(), Joint_Width + DTR,
2619 gcpy           stockZthickness + DTR, Joint_Width + DTR, 0)
self.dxfarc(self.currenttoolnumber(), Joint_Width, 0, DTR,
270, 360)
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.

**3.9.1.1.5 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:

---

```

2621 gcpy      def Full_Blind_Finger_Joint_square(self, bx, by, orientation,
2622 gcpy          side, width, thickness, Number_of_Pins, largeVdiameter,
2623 gcpy          smallDiameter, normalormirror = "Default"):
2624 gcpy      # Joint_Orientation = "Horizontal" "Even" == "Lower", "Odd" ==
2625 gcpy          "Upper"
2626 gcpy      # Joint_Orientation = "Vertical" "Even" == "Left", "Odd" == "
2627 gcpy          "Right"
2628 gcpy      if (orientation == "Vertical"):
2629 gcpy          if (normalormirror == "Default" and side != "Both"):
2630 gcpy              if (side == "Left"):
2631 gcpy                  normalormirror = "Even"
2632 gcpy              if (side == "Right"):
2633 gcpy                  normalormirror = "Odd"
2634 gcpy      if (orientation == "Horizontal"):
2635 gcpy          if (normalormirror == "Default" and side != "Both"):
2636 gcpy              if (side == "Lower"):
2637 gcpy                  normalormirror = "Even"
2638 gcpy              if (side == "Upper"):
2639 gcpy                  normalormirror = "Odd"
2640 gcpy      Finger_Width = ((Number_of_Pins * 2) - 1) * smallDiameter *
2641 gcpy          1.1
2642 gcpy      Finger_Origin = width/2 - Finger_Width/2
2643 gcpy      rapid = self.rapidZ(0)
2644 gcpy      self.setdxfcolor("Cyan")
2645 gcpy      rapid = rapid.union(self.rapidXY(bx, by))
2646 gcpy      toolpath = (self.Finger_Joint_square(bx, by, orientation,
2647 gcpy          side, width, thickness, Number_of_Pins, Finger_Origin,
2648 gcpy          smallDiameter))
2649 gcpy      if (orientation == "Vertical"):
2650 gcpy          if (side == "Both"):
2651 gcpy              toolpath = self.cutrectangleroundddxf(self.
2652 gcpy                  currenttoolnum, bx - (thickness - smallDiameter
2653 gcpy                  /2), by-smallDiameter/2, 0, (thickness * 2) -
2654 gcpy                  smallDiameter, width+smallDiameter,
2655 gcpy                  (smallDiameter / 2) / Tan(45), smallDiameter/2)
2656 gcpy          if (side == "Left"):
2657 gcpy              toolpath = self.cutrectangleroundddxf(self.
2658 gcpy                  currenttoolnum, bx - (smallDiameter/2), by-
2659 gcpy                  smallDiameter/2, 0, thickness, width+
2660 gcpy                  smallDiameter, ((smallDiameter / 2) / Tan(45)),
2661 gcpy                  smallDiameter/2)
2662 gcpy          if (side == "Right"):
2663 gcpy              toolpath = self.cutrectangleroundddxf(
2664 gcpy                  self.currenttoolnum,
2665 gcpy                  bx-smallDiameter/2,
2666 gcpy                  by - (thickness - smallDiameter/2),
2667 gcpy                  0,
2668 gcpy                  width+smallDiameter,
2669 gcpy                  (thickness * 2) - smallDiameter,
2670 gcpy                  (smallDiameter / 2) / Tan(45),
2671 gcpy                  smallDiameter/2)
2672 gcpy      toolpath = toolpath.union(self.Finger_Joint_square(bx, by,
2673 gcpy          orientation, side, width, thickness, Number_of_Pins,
2674 gcpy          Finger_Origin, smallDiameter))
2675 gcpy      if (orientation == "Horizontal"):
2676 gcpy          if (side == "Both"):
2677 gcpy              toolpath = self.cutrectangleroundddxf(
2678 gcpy                  self.currenttoolnum,
2679 gcpy                  bx-smallDiameter/2,
2680 gcpy                  by - (thickness - smallDiameter/2),
2681 gcpy                  0,
2682 gcpy                  width+smallDiameter,
2683 gcpy                  (thickness * 2) - smallDiameter,
2684 gcpy                  (smallDiameter / 2) / Tan(45),
2685 gcpy                  smallDiameter/2)
2686 gcpy          if (side == "Lower"):
2687 gcpy              toolpath = self.cutrectangleroundddxf(
2688 gcpy                  self.currenttoolnum,
```

```

2664 gcpy                                bx - (smallDiameter/2),
2665 gcpy                                by - smallDiameter/2,
2666 gcpy                                0,
2667 gcpy                                width+smallDiameter,
2668 gcpy                                thickness,
2669 gcpy                                ((smallDiameter / 2) / Tan(45)),
2670 gcpy                                smallDiameter/2)
2671 gcpy                                if (side == "Upper"):
2672 gcpy                                toolpath = self.cutrectanglerounddxf(
2673 gcpy                                  self.currenttoolnum,
2674 gcpy                                  bx - smallDiameter/2,
2675 gcpy                                  by - (thickness - smallDiameter/2),
2676 gcpy                                  0,
2677 gcpy                                  width+smallDiameter,
2678 gcpy                                  thickness,
2679 gcpy                                  ((smallDiameter / 2) / Tan(45)),
2680 gcpy                                  smallDiameter/2)
2681 gcpy                                toolpath = toolpath.union(self.Finger_Joint_square(bx, by,
2682 gcpy                                  orientation, side, width, thickness, Number_of_Pins,
2683 gcpy                                  Finger_Origin, smallDiameter))
2684 gcpy                                return toolpath

def Finger_Joint_square(self, bx, by, orientation, side, width,
2685 gcpy                                  thickness, Number_of_Pins, Finger_Origin, smallDiameter,
2686 gcpy                                  normalormirror = "Default"):
2687 gcpy                                  jointdepth = -(thickness - (smallDiameter / 2) / Tan(45))
# Joint_Orientation = "Horizontal" "Even" == "Lower", "Odd" ==
2688 gcpy                                  "Upper"
# Joint_Orientation = "Vertical" "Even" == "Left", "Odd" == "
2689 gcpy                                  Right"
2690 gcpy                                  if (orientation == "Vertical"):
2691 gcpy                                      if (normalormirror == "Default" and side != "Both"):
2692 gcpy                                          if (side == "Left"):
2693 gcpy                                              normalormirror = "Even"
2694 gcpy                                          if (side == "Right"):
2695 gcpy                                              normalormirror = "Odd"
2696 gcpy                                  if (orientation == "Horizontal"):
2697 gcpy                                      if (normalormirror == "Default" and side != "Both"):
2698 gcpy                                          if (side == "Lower"):
2699 gcpy                                              normalormirror = "Even"
2700 gcpy                                          if (side == "Upper"):
2701 gcpy                                              normalormirror = "Odd"
2702 gcpy                                  radius = smallDiameter/2
2703 gcpy                                  jointwidth = thickness - smallDiameter
2704 gcpy                                  toolpath = self.currenttool()
2705 gcpy                                  rapid = self.rapidZ(0)
2706 gcpy                                  self.setdxfcolor("Blue")
2707 gcpy                                  toolpath = toolpath.union(self.cutlineZgcfeed(jointdepth
2708 gcpy                                  ,1000))
2709 gcpy                                  self.beginpolyline(self.currenttool())
2710 gcpy                                  if (orientation == "Vertical"):
2711 gcpy                                      rapid = rapid.union(self.rapidXY(bx, by + Finger_Origin
2712 gcpy                                      ))
2713 gcpy                                      self.addvertex(self.currenttoolnumber(), self.xpos(),
2714 gcpy                                      self.ypos())
2715 gcpy                                      toolpath = toolpath.union(self.cutlineZgcfeed(
2716 gcpy                                      jointdepth,1000))
2717 gcpy                                      i = 0
2718 gcpy                                      while i <= Number_of_Pins - 1:
2719 gcpy                                          if (side == "Right"):
2720 gcpy                                              toolpath = toolpath.union(self.cutvertexdxf(

```

```

                jointdepth))
2721 gcpy   if (side == "Left"):
2722 gcpy       toolpath = toolpath.union(self.cutvertexdxf(
2723 gcpy           self.xpos(), self.ypos() + smallDiameter +
2724 gcpy               radius/5, jointdepth))
2725 gcpy   #   if (side == "Right" or side == "Both"):
2726 gcpy       if (i < (Number_of_Pins - 1)):
2727 gcpy           print(i)
2728 gcpy           toolpath = toolpath.union(self.cutvertexdxf(
2729 gcpy               (self.xpos(), self.ypos() + radius,
2730 gcpy                   jointdepth))
2731 gcpy           toolpath = toolpath.union(self.cutvertexdxf(
2732 gcpy               (self.xpos() - jointwidth, self.ypos(),
2733 gcpy                   jointdepth))
2734 gcpy           toolpath = toolpath.union(self.cutvertexdxf(
2735 gcpy               (self.xpos() + jointwidth, self.ypos(),
2736 gcpy                   jointdepth))
2737 gcpy           i += 1
2738 gcpy   # Joint_Orientation = "Horizontal" "Even" == "Lower", "Odd" ==
2739 gcpy       "Upper"
2740 gcpy   if (orientation == "Horizontal"):
2741 gcpy       rapid = rapid.union(self.rapidXY(bx + Finger_Origin, by
2742 gcpy           ))
2743 gcpy   self.addvertex(self.currenttoolnumber(), self.xpos(),
2744 gcpy       self.ypos())
2745 gcpy   toolpath = toolpath.union(self.cutlineZgcfeed(
2746 gcpy       jointdepth,1000))
2747 gcpy   i = 0
2748 gcpy   while i <= Number_of_Pins - 1:
2749 gcpy   if (side == "Upper"):
2750 gcpy       toolpath = toolpath.union(self.cutvertexdxf(
2751 gcpy           self.xpos() + smallDiameter + radius/5, self
2752 gcpy               .ypos(), jointdepth))
2753 gcpy   if (side == "Lower" or side == "Both"):
2754 gcpy       toolpath = toolpath.union(self.cutvertexdxf(
2755 gcpy           self.xpos() + radius, self.ypos(),
2756 gcpy               jointdepth))
2757 gcpy       toolpath = toolpath.union(self.cutvertexdxf(
2758 gcpy           (self.xpos(), self.ypos() - jointwidth,
2759 gcpy               jointdepth))
2760 gcpy   if (side == "Lower"):
2761 gcpy       toolpath = toolpath.union(self.cutvertexdxf(
2762 gcpy           self.xpos() + smallDiameter + radius/5, self
2763 gcpy               .ypos(), jointdepth))
2764 gcpy   if (side == "Upper" or side == "Both"):
2765 gcpy   if (i < (Number_of_Pins - 1)):
2766 gcpy       print(i)
2767 gcpy       toolpath = toolpath.union(self.cutvertexdxf(
2768 gcpy           (self.xpos() + radius, self.ypos(),
2769 gcpy               jointdepth))
2770 gcpy   toolpath = toolpath.union(self.cutvertexdxf(
2771 gcpy           (self.xpos(), self.ypos() - jointwidth,
2772 gcpy               jointdepth))
2773 gcpy   toolpath = toolpath.union(self.cutvertexdxf(
2774 gcpy           (self.xpos() + radius, self.ypos(),
2775 gcpy               jointdepth)))
2776 gcpy   i += 1

```

```

2758 gcpy         self.closepolyline(self.currenttoolnumber())
2759 gcpy         return toolpath
2760 gcpy
2761 gcpy     def Full_Blind_Finger_Joint_smallV(self, bx, by, orientation,
2762 gcpy             side, width, thickness, Number_of_Pins, largeVdiameter,
2763 gcpy             smallDiameter):
2764 gcpy             rapid = self.rapidZ(0)
2765 gcpy             #   rapid = rapid.union(self.rapidXY(bx, by))
2766 gcpy             self.setdxfcolor("Red")
2767 gcpy             if (orientation == "Vertical"):
2768 gcpy                 rapid = rapid.union(self.rapidXY(bx, by - smallDiameter
2769 gcpy                     /6))
2770 gcpy                 toolpath = self.cutlineZgcfeed(-thickness,1000)
2771 gcpy                 toolpath = self.cutlinedxfgc(bx, by + width +
2772 gcpy                     smallDiameter/6, - thickness)
2773 gcpy             if (orientation == "Horizontal"):
2774 gcpy                 rapid = rapid.union(self.rapidXY(bx - smallDiameter/6,
2775 gcpy                     by))
2776 gcpy                 toolpath = self.cutlineZgcfeed(-thickness,1000)
2777 gcpy                 toolpath = self.cutlinedxfgc(bx + width + smallDiameter
2778 gcpy                     /6, by, -thickness)
2779 gcpy             #   rapid = self.rapidZ(0)
2780 gcpy
2781 gcpy             return toolpath
2782 gcpy
2783 gcpy     def Full_Blind_Finger_Joint_largeV(self, bx, by, orientation,
2784 gcpy             side, width, thickness, Number_of_Pins, largeVdiameter,
2785 gcpy             smallDiameter):
2786 gcpy             radius = smallDiameter/2
2787 gcpy             rapid = self.rapidZ(0)
2788 gcpy             Finger_Width = ((Number_of_Pins * 2) - 1) * smallDiameter *
2789 gcpy                 1.1
2790 gcpy             Finger_Origin = width/2 - Finger_Width/2
2791 gcpy             #   rapid = rapid.union(self.rapidXY(bx, by))
2792 gcpy             # Joint_Orientation = "Horizontal" "Even" == "Lower", "Odd" ==
2793 gcpy                 "Upper"
2794 gcpy             # Joint_Orientation = "Vertical" "Even" == "Left", "Odd" == "
2795 gcpy                 "Right"
2796 gcpy                 if (orientation == "Vertical"):
2797 gcpy                     rapid = rapid.union(self.rapidXY(bx, by))
2798 gcpy                     toolpath = self.cutlineZgcfeed(-thickness,1000)
2799 gcpy                     toolpath = toolpath.union(self.cutlinedxfgc(bx, by +
2800 gcpy                         Finger_Origin, -thickness))
2801 gcpy                     rapid = self.rapidZ(0)
2802 gcpy                     rapid = rapid.union(self.rapidXY(bx, by + width -
2803 gcpy                         Finger_Origin))
2804 gcpy                     self.setdxfcolor("Blue")
2805 gcpy                     toolpath = toolpath.union(self.cutlineZgcfeed(-
2806 gcpy                         thickness,1000))
2807 gcpy                     toolpath = toolpath.union(self.cutlinedxfgc(bx +
2808 gcpy                         width, -thickness))
2809 gcpy                     if (side == "Left" or side == "Both"):
2810 gcpy                         rapid = self.rapidZ(0)
2811 gcpy                         self.setdxfcolor("DarkGray")
2812 gcpy                         rapid = rapid.union(self.rapidXY(bx+thickness -(
2813 gcpy                             smallDiameter / 2) / Tan(45), by - radius/2))
2814 gcpy                         toolpath = toolpath.union(self.cutlineZgcfeed(-(
2815 gcpy                             smallDiameter / 2) / Tan(45),10000))
2816 gcpy                         toolpath = toolpath.union(self.cutlinedxfgc(bx+
2817 gcpy                             thickness-(smallDiameter / 2) / Tan(45), by +
2818 gcpy                             width + radius/2, -(smallDiameter / 2) / Tan(45)
2819 gcpy                             ))
2820 gcpy                         rapid = self.rapidZ(0)
2821 gcpy                         self.setdxfcolor("Green")
2822 gcpy                         rapid = rapid.union(self.rapidXY(bx+thickness/2, by
2823 gcpy                             +width))
2824 gcpy                         toolpath = toolpath.union(self.cutlineZgcfeed(-
2825 gcpy                             thickness/2,1000))
2826 gcpy                         toolpath = toolpath.union(self.cutlinedxfgc(bx+
2827 gcpy                             thickness/2, by + width -thickness, -thickness
2828 gcpy                             /2))
2829 gcpy                         rapid = self.rapidZ(0)
2830 gcpy                         rapid = rapid.union(self.rapidXY(bx+thickness/2, by
2831 gcpy                             ))
2832 gcpy                         toolpath = toolpath.union(self.cutlineZgcfeed(-
2833 gcpy                             thickness/2,1000))
2834 gcpy                         toolpath = toolpath.union(self.cutlinedxfgc(bx+
2835 gcpy                             thickness/2, by +thickness, -thickness/2))

```

```

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
# Joint_Orientation = "Horizontal" "Even" == "Lower", "Odd" ==
"Upper"
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
2854 gcpy
2855 gcpy

    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)
        rapid = rapid.union(self.rapidXY(bx, by+thickness
            /2))
        toolpath = toolpath.union(self.cutlineZgcfeed(-
            thickness/2,1000))
        toolpath = toolpath.union(self.cutlinedxfgc(bx +
            thickness, by+thickness/2, -thickness/2))
    if (side == "Upper" 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)

```

```

2856 gcpy           self.setdxfcolor("Green")
2857 gcpy           rapid = rapid.union(self.rapidXY(bx+width, by-
2858 gcpy           thickness/2))
2859 gcpy           toolpath = toolpath.union(self.cutlineZgcfeed(-
2860 gcpy           thickness/2,1000))
2861 gcpy           toolpath = toolpath.union(self.cutlinedxfgc(bx +
2862 gcpy           width -thickness, by-thickness/2, -thickness/2))
2863 gcpy           rapid = self.rapidZ(0)
2864 gcpy           rapid = rapid.union(self.rapidXY(bx, by-thickness
2865 gcpy           /2))
2866 gcpy           toolpath = toolpath.union(self.cutlineZgcfeed(-
2867 gcpy           thickness/2,1000))
2868 gcpy           toolpath = toolpath.union(self.cutlinedxfgc(bx +
2869 gcpy           thickness, by-thickness/2, -thickness/2))
2870 gcpy           rapid = self.rapidZ(0)
2871 gcpy           return toolpath
2872 gcpy       def Full_Blind_Finger_Joint(self, bx, by, orientation, side,
2873 gcpy           width, thickness, largeVdiameter, smallDiameter,
2874 gcpy           normalormirror = "Default", squaretool = 102, smallV = 390,
2875 gcpy           largeV = 301):
2876 gcpy           Number_of_Pins = int((width - thickness * 2) / (
2877 gcpy           smallDiameter * 2.2) / 2) + 0.0) * 2 + 1
2878 gcpy           print("Number of Pins: ", Number_of_Pins)
2879 gcpy           self.movetosafeZ()
2880 gcpy           self.toolchange(squaretool, 17000)
2881 gcpy           toolpath = self.Full_Blind_Finger_Joint_square(bx, by,
2882 gcpy           orientation, side, width, thickness, Number_of_Pins,
2883 gcpy           largeVdiameter, smallDiameter)
2884 gcpy           self.movetosafeZ()
2885 gcpy           self.toolchange(smallV, 17000)
2886 gcpy           toolpath = toolpath.union(self.
2887 gcpy           Full_Blind_Finger_Joint_smallV(bx, by, orientation, side
2888 gcpy           , width, thickness, Number_of_Pins, largeVdiameter,
2889 gcpy           smallDiameter))
2890 gcpy           self.toolchange(largeV, 17000)
2891 gcpy           toolpath = toolpath.union(self.
2892 gcpy           Full_Blind_Finger_Joint_largeV(bx, by, orientation, side
2893 gcpy           , width, thickness, Number_of_Pins, largeVdiameter,
2894 gcpy           smallDiameter))
2895 gcpy           return toolpath

```

---

### 3.10 (Reading) G-code Files

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

First, a template file will be necessary:

```

1 gcpncpy #Requires OpenPythonSCAD, so load support for 3D modeling in that
          tool:
2 gcpncpy from openscad import *
3 gcpncpy
4 gcpncpy #The gcodepreview library must be loaded, either from github (first
          line below) or from a local library (second line below),
          uncomment one and comment out the other, depending on where one
          wishes to load from
5 gcpncpy #nimport("https://raw.githubusercontent.com/WillAdams/gcodepreview/
          refs/heads/main/gcodepreview.py")
6 gcpncpy from gcodepreview import *
7 gcpncpy
8 gcpncpy #The file to be loaded must be specified:
9 gcpncpy #gc_file = "filename_of_G-code_file_to_process.nc"
10 gcpncpy #
11 gcpncpy #if using windows the full filepath should be provided with
          backslashes replaced with double 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 gcodepreview object:
15 gcpncpy gcp = gcodepreview("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:

---

```

2880 gcpy      def previewgcodefile(self, gc_file):
2881 gcpy          gc_file = open(gc_file, 'r')
2882 gcpy          gcfilecontents = []
2883 gcpy          with gc_file as file:
2884 gcpy              for line in file:
2885 gcpy                  command = line
2886 gcpy                  gcfilecontents.append(line)
2887 gcpy
2888 gcpy          numlinesfound = 0
2889 gcpy          for line in gcfilecontents:
2890 gcpy              print(line)
2891 gcpy              if line[:10] == "(stockMin:":
2892 gcpy                  subdivisions = line.split()
2893 gcpy                  extentleft = float(subdivisions[0][10:-3])
2894 gcpy                  extentfb = float(subdivisions[1][-3:])
2895 gcpy                  extentd = float(subdivisions[2][-3:])
2896 gcpy                  numlinesfound = numlinesfound + 1
2897 gcpy              if line[:13] == "(STOCK/BLOCK,":
2898 gcpy                  subdivisions = line.split()
2899 gcpy                  sizeX = float(subdivisions[0][13:-1])
2900 gcpy                  sizeY = float(subdivisions[1][-1])
2901 gcpy                  sizeZ = float(subdivisions[4][-1])
2902 gcpy                  numlinesfound = numlinesfound + 1
2903 gcpy              if line[:3] == "G21":
2904 gcpy                  units = "mm"
2905 gcpy                  numlinesfound = numlinesfound + 1
2906 gcpy              if numlinesfound >=3:
2907 gcpy                  break
2908 gcpy      print(numlinesfound)

```

---

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

```

2910 gcpy          self.setupcuttingarea(sizeX, sizeY, sizeZ, extentleft,
2911 gcpy                      extentfb, extentd)
2912 gcpy          commands = []
2913 gcpy          for line in gcfilecontents:
2914 gcpy              Xc = 0
2915 gcpy              Yc = 0
2916 gcpy              Zc = 0
2917 gcpy              Fc = 0
2918 gcpy              Xp = 0.0
2919 gcpy              Yp = 0.0
2920 gcpy              Zp = 0.0
2921 gcpy              if line == "G53G0Z-5.000\n":
2922 gcpy                  self.movetosafeZ()
2923 gcpy              if line[:3] == "M6T":
2924 gcpy                  tool = int(line[3:])
2925 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)

```

---

```

2926 gcpy           if line[:2] == "G0":
2927 gcpy             machinestate = "rapid"
2928 gcpy           if line[:2] == "G1":
2929 gcpy             machinestate = "cutline"
2930 gcpy           if line[:2] == "G0" or line[:2] == "G1" or line[:1] ==
2931 gcpy               "X" or line[:1] == "Y" or line[:1] == "Z":
2932 gcpy               if "F" in line:
2933 gcpy                 Fplus = line.split("F")
2934 gcpy                 Fc = 1
2935 gcpy                 fr = float(Fplus[1])
2936 gcpy                 line = Fplus[0]
2937 gcpy           if "Z" in line:
2938 gcpy                 Zplus = line.split("Z")
2939 gcpy                 Zc = 1
2940 gcpy                 Zp = float(Zplus[1])
2941 gcpy                 line = Zplus[0]
2942 gcpy           if "Y" in line:
2943 gcpy                 Yplus = line.split("Y")
2944 gcpy                 Yc = 1
2945 gcpy                 Yp = float(Yplus[1])
2946 gcpy                 line = Yplus[0]
2947 gcpy           if "X" in line:
2948 gcpy                 Xplus = line.split("X")
2949 gcpy                 Xc = 1
2950 gcpy                 Xp = float(Xplus[1])
2951 gcpy           if Zc == 1:
2952 gcpy               if Yc == 1:
2953 gcpy                   if machinestate == "rapid":
2954 gcpy                     command = "rapidXYZ(" + str(Xp) + "
2955 gcpy                         ", " + str(Yp) + ", " + str(Zp) +
2956 gcpy                         ")"
2957 gcpy                     self.rapidXYZ(Xp, Yp, Zp)
2958 gcpy               else:
2959 gcpy                   command = "cutlineXYZ(" + str(Xp) +
2960 gcpy                         ", " + str(Yp) + ", " + str(Zp) +
2961 gcpy                         ")"
2962 gcpy                     self.cutlineXYZ(Xp, Yp, Zp)
2963 gcpy               else:
2964 gcpy                   if machinestate == "rapid":
2965 gcpy                     command = "rapidYZ(" + str(Yp) + ",
2966 gcpy                         ", " + str(Zp) + ")"
2967 gcpy                     self.rapidYZ(Yp, Zp)
2968 gcpy               else:
2969 gcpy                   command = "cutlineYZ(" + str(Yp) +
2970 gcpy                         ", " + str(Zp) + ")"
2971 gcpy                     self.cutlineYZ(Yp, Zp)
2972 gcpy               else:
2973 gcpy                   if Xc == 1:
2974 gcpy                       if machinestate == "rapid":
2975 gcpy                         command = "rapidXZ(" + str(Xp) + ",
2976 gcpy                           ", " + str(Zp) + ")"
2977 gcpy                         self.rapidXZ(Xp, Zp)
2978 gcpy               else:
2979 gcpy                   command = "cutlineXZ(" + str(Xp) +
2980 gcpy                         ", " + str(Zp) + ")"
2981 gcpy                     self.cutlineXZ(Xp, Zp)
2982 gcpy               else:
2983 gcpy                   if Yc == 1:
2984 gcpy                       if machinestate == "rapid":
2985 gcpy                         command = "rapidXY(" + str(Xp) + ",
2986 gcpy                           ", " + str(Yp) + ")"
2987 gcpy                         self.rapidXY(Xp, Yp)
2988 gcpy               else:
2989 gcpy                   command = "cutlineXY(" + str(Xp) +
2990 gcpy                         ", " + str(Yp) + ")"
2991 gcpy                     self.cutlineXY(Xp, Yp)
2992 gcpy               else:
2993 gcpy                   if machinestate == "rapid":
```

```

2992 gcpy
2993 gcpy
2994 gcpy
2995 gcpy
2996 gcpy
2997 gcpy
2998 gcpy
2999 gcpy
3000 gcpy
3001 gcpy
3002 gcpy
3003 gcpy
3004 gcpy
3005 gcpy #
3006 gcpy #
3007 gcpy #
3008 gcpy #
3009 gcpy #
3010 gcpy #
3011 gcpy
3012 gcpy #
3013 gcpy #
3014 gcpy
3015 gcpy #
3016 gcpy

    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("/n")
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

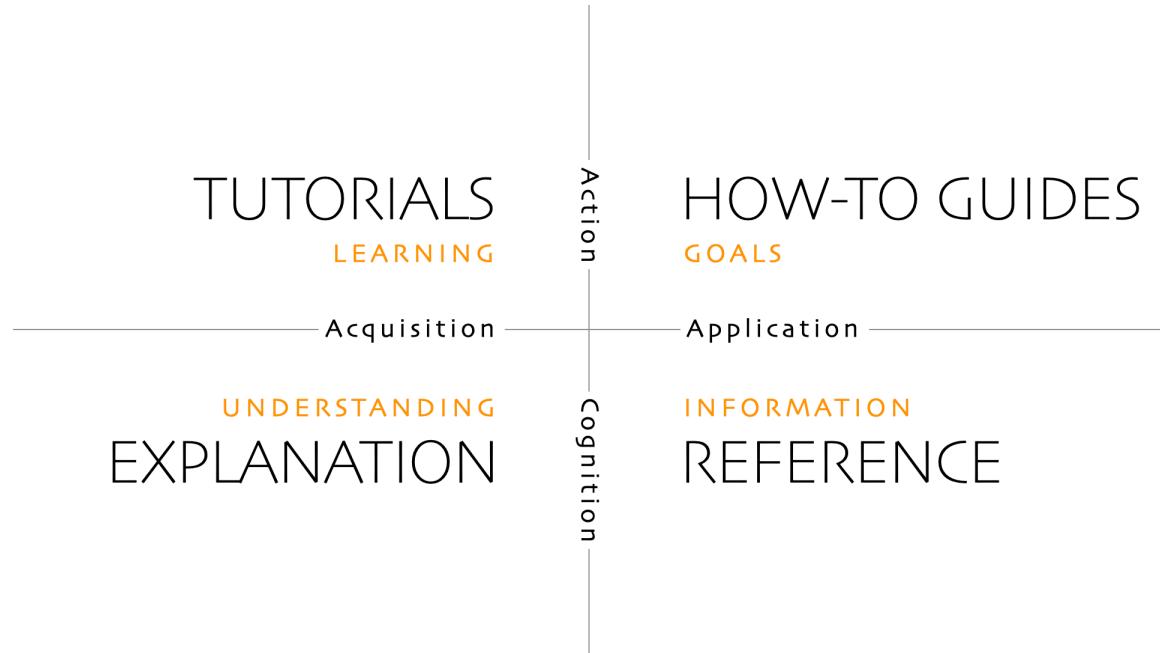
<https://thewhitetulip.gitbook.io/py/06-file-handling>

### 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:

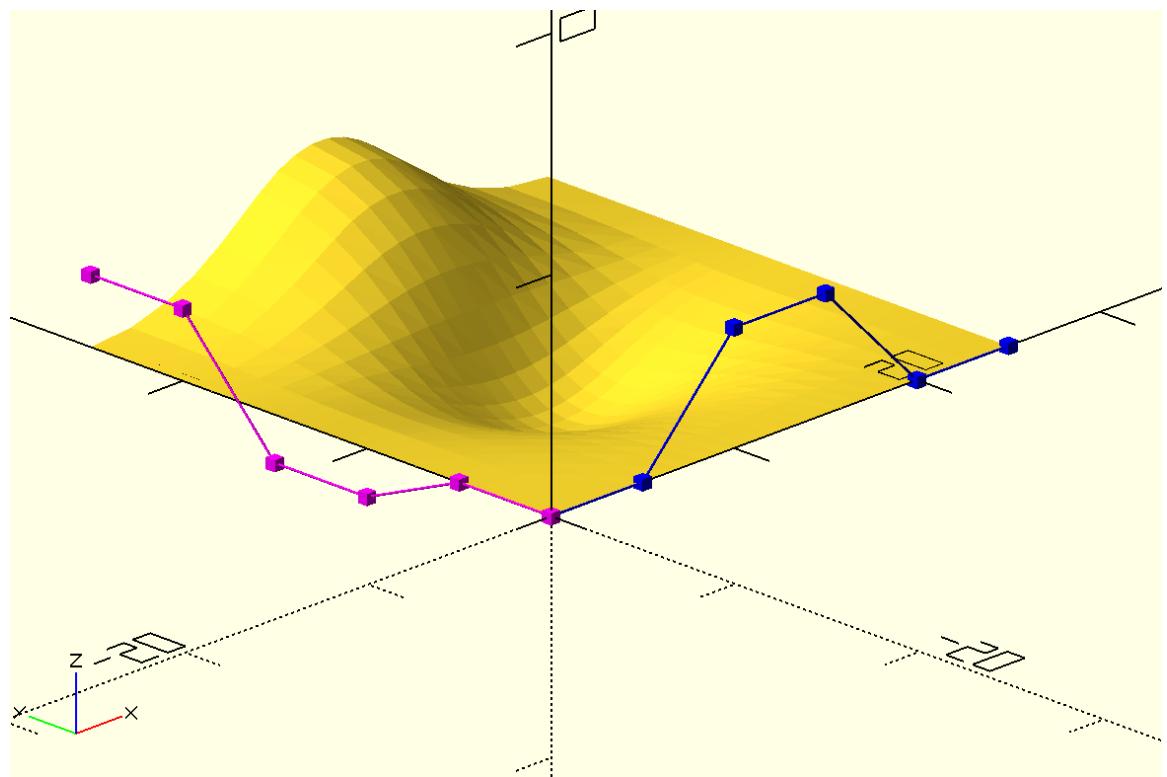
3 planes \* 3 Béziers \* (2 on-curve + 2 off-curve points) == 36 coordinate pairs

which is a marked contrast to representations such as:

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

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

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



<https://pages.mtu.edu/~shene/COURSES/cs3621/NOTES/notes.html>

c.f., <https://github.com/BelfrySCAD/BOSL2/wiki/nurbs.scad> and [https://old.reddit.com/r/OpenPythonSCAD/comments/1gjcz4z/pythonscad\\_will\\_get\\_a\\_new\\_spline\\_function/](https://old.reddit.com/r/OpenPythonSCAD/comments/1gjcz4z/pythonscad_will_get_a_new_spline_function/)

### 4.2.4 Mathematics

<https://elementsofprogramming.com/>

## References

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

## Command Glossary

. [25](#)

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

# Index

addvertex, 85  
ballnose, 54  
beginpolyline, 85  
closedxfile, 92, 93  
closegcodefile, 92, 93  
close polyline, 85  
currenttool, 33  
currenttoolnum, 33  
cut..., 51, 61  
cutarcCC, 63  
cutarcCW, 63  
cutkeyhole toolpath, 98  
cutKHgcdxf, 100  
cutline, 61  
cutrectangle, 98  
diameter, 34  
dovetail, 55  
dxfar, 85  
dxfcircle, 85  
dxflne, 85  
dxffpostamble, 92, 93  
dxfpreamble, 84  
dxfwrite, 84  
endmill square, 53  
endmill v, 54  
endmilltype, 34  
feed, 69  
flute, 34  
gcodepreview, 31  
    writeln, 80  
gcp.setupstock, 34  
generatecut, 28  
generatedxf, 28  
generategcode, 28  
init, 31  
mpx, 33  
mpy, 33  
mpz, 33  
opendxfile, 81  
opengcodefile, 81, 82  
plunge, 69  
previugcodefile, 113  
ra, 34  
rapid, 59  
rapid..., 51  
rapids, 34, 37  
roundover, 56  
settoolparameters, 41  
setupstock, 34  
    gcodepreview, 34  
setxpos, 33  
setypos, 33  
setzpos, 33  
shaftmovement, 52, 56  
speed, 69  
stockzero, 34  
subroutine  
    gcodepreview, 34  
    writeln, 80  
tip, 34  
tool diameter, 68  
tool number, 41  
tool radius, 69  
toolchange, 41, 42  
toolmovement, 33, 34, 37, 42, 52  
toolpaths, 34, 37  
tpzinc, 33  
writedxfDT, 84  
writedxfKH, 84  
writedxflbl, 84  
writedxfagsq, 84  
writedxfglV, 84  
writedxfsmbl, 84  
writedxfsmgsq, 84  
writedxfsmV, 84  
xpos, 33  
ypos, 33  
zeroheight, 34  
zpos, 33

## Routines

addvertex, 85  
ballnose, 54  
beginpolyline, 85  
closedxffile, 92, 93  
closegcodefile, 92, 93  
close polyline, 85  
cut..., 51, 61  
cutarcCC, 63  
cutarcCW, 63  
cutkeyhole toolpath, 98  
cutKHgwdx, 100  
cutline, 61  
cutrectangle, 98  
dovetail, 55  
dxfar, 85  
dxfcircle, 85  
dxflne, 85  
dxffostamble, 92, 93  
dxfpreamble, 84  
dxfwrite, 84  
endmill square, 53  
endmill v, 54  
gcodepreview, 31, 34  
gcp.setupstock, 34  
init, 31  
opendxffile, 81  
opengcodefile, 81, 82  
previewgcodefile, 113  
rapid, 59  
rapid..., 51  
roundover, 56  
setupstock, 34  
setxpos, 33  
setypos, 33  
setzpos, 33  
shaftmovement, 52, 56  
tool diameter, 68  
tool radius, 69  
toolchange, 41, 42  
toolmovement, 33, 34, 37, 42, 52  
writedxfDT, 84  
writedxfKH, 84  
writedxflbl, 84  
writedxfgsq, 84  
writedxfglV, 84  
writedxfsmbl, 84  
writedxfsmgsq, 84  
writedxfsmV, 84  
writeln, 80  
xpos, 33  
ypos, 33  
zpos, 33

## Variables

currenttool, 33  
currenttoolnum, 33  
diameter, 34  
endmilltype, 34  
feed, 69  
flute, 34  
generatecut, 28  
generatedxf, 28  
generategcode, 28  
mpx, 33  
mpy, 33  
mpz, 33  
plunge, 69  
ra, 34  
rapids, 34, 37  
settoolparameters, 41  
speed, 69  
stockzero, 34  
tip, 34  
tool number, 41  
toolchange, 41  
toolpaths, 34, 37  
tpzinc, 33  
zeroheight, 34