

# 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 . . . . .	27
3.2.1	Parameters and Default Values . . . . .	29
3.3	Implementation files and gcodepreview class . . . . .	29
3.3.1	init . . . . .	31
3.3.2	Position and Variables . . . . .	33
3.3.3	Initial Modules . . . . .	33
3.3.4	Adjustments and Additions . . . . .	37
3.4	Tools and Shapes and Changes . . . . .	37
3.4.1	Numbering for Tools . . . . .	38
3.4.2	Laser support . . . . .	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
3.5.6	3D Printing . . . . .	69
3.6	Difference of Stock, Rapids, and Toolpaths . . . . .	80
3.7	Output files . . . . .	80
3.7.1	Python and OpenSCAD File Handling . . . . .	80
3.7.2	DXF Overview . . . . .	84
3.7.3	G-code Overview . . . . .	91
3.8	Cutting shapes and expansion . . . . .	93
3.8.1	Building blocks . . . . .	94
3.9	(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

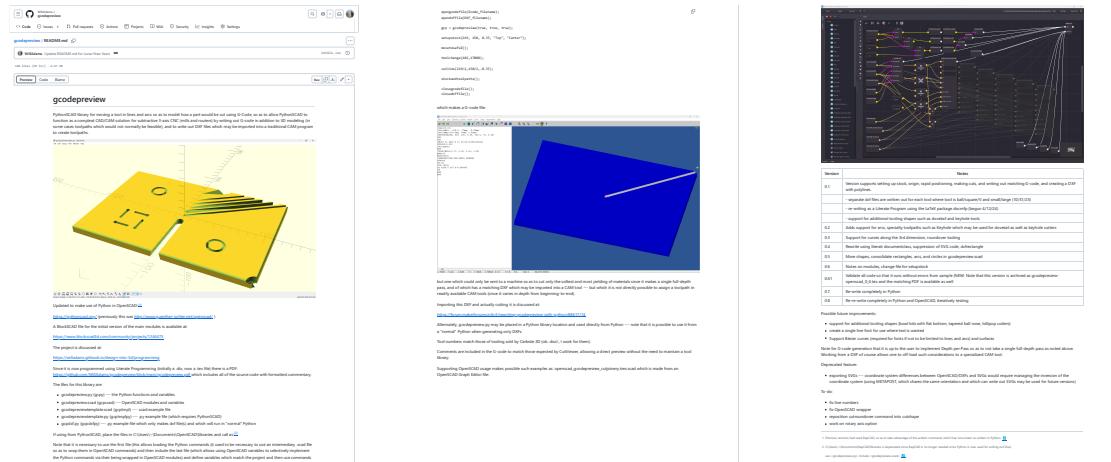
\*This file (gcodepreview) has version number v0.93, last revised 2025/11/30.<sup>1</sup>

## 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 . . . . .	27
3.2.1	Parameters and Default Values . . . . .	29
3.3	Implementation files and gcodepreview class . . . . .	29
3.3.1	init . . . . .	31
3.3.2	Position and Variables . . . . .	33
3.3.3	Initial Modules . . . . .	33
3.3.3.1	setupstock . . . . .	34
3.3.3.2	setupcuttingarea . . . . .	36
3.3.3.3	debug . . . . .	36
3.3.4	Adjustments and Additions . . . . .	37
3.4	Tools and Shapes and Changes . . . . .	37
3.4.1	Numbering for Tools . . . . .	38
3.4.1.1	toolchange . . . . .	41
3.4.1.1.1	settoolparameters . . . . .	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) . . . . .	49
3.4.1.1.10	Dovetails . . . . .	50
3.4.1.1.11	closing G-code . . . . .	51
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 . . . . .	53
3.5.3.1.1	Square (including O-flute) . . . . .	54
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 . . . . .	55
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) . . . . .	58
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) . . . . .	60
3.5.3.6	Arcs . . . . .	62
3.5.4	tooldiameter . . . . .	67
3.5.5	Feeds and Speeds . . . . .	69

3.5.6	3D Printing . . . . .	69
3.5.6.1	Sample 3D printing file . . . . .	70
3.5.6.2	Feed and Speed ratio . . . . .	73
3.5.6.3	Time and Firmware for 3D printers . . . . .	73
3.6	Difference of Stock, Rapids, and Toolpaths . . . . .	80
3.7	Output files . . . . .	80
3.7.1	Python and OpenSCAD File Handling . . . . .	80
3.7.2	DXF Overview . . . . .	84
3.7.2.1	Writing to DXF files . . . . .	84
3.7.2.1.1	DXF Lines and Arcs . . . . .	85
3.7.3	G-code Overview . . . . .	91
3.7.3.1	Closings . . . . .	92
3.8	Cutting shapes and expansion . . . . .	93
3.8.1	Building blocks . . . . .	94
3.8.1.1	List of shapes . . . . .	94
3.8.1.1.1	circles . . . . .	95
3.8.1.1.2	rectangles . . . . .	96
3.8.1.1.3	Keyhole toolpath and undercut tooling . . . . .	98
3.8.1.1.4	Dovetail joinery and tooling . . . . .	105
3.8.1.1.5	Full-blind box joints . . . . .	107
3.9	(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
	Index . . . . .	121
	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
```

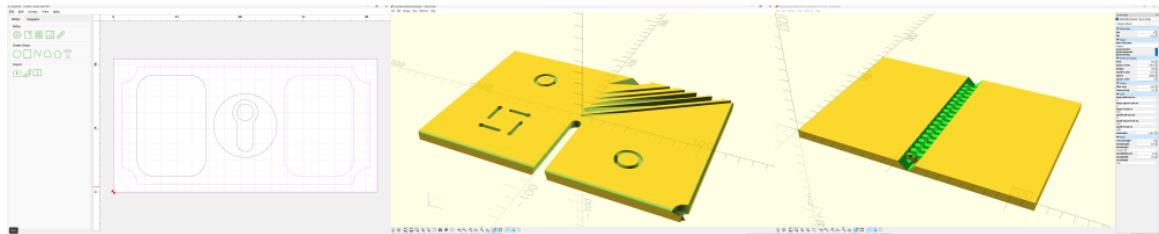
---

## 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.dxfrctangle(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.dxfrctangle(large_square_tool_num, radius + inset, radius,
                           stockXwidth/2 - (radius * 4), stockYheight - (radius * 2),
                           cornerstyle, radius)
66 gcpdxfpy gcp.dxfrctangle(large_square_tool_num, stockXwidth/2 + (radius *
                           2) + inset, radius, stockXwidth/2 - (radius * 4), stockYheight -
                           inset)

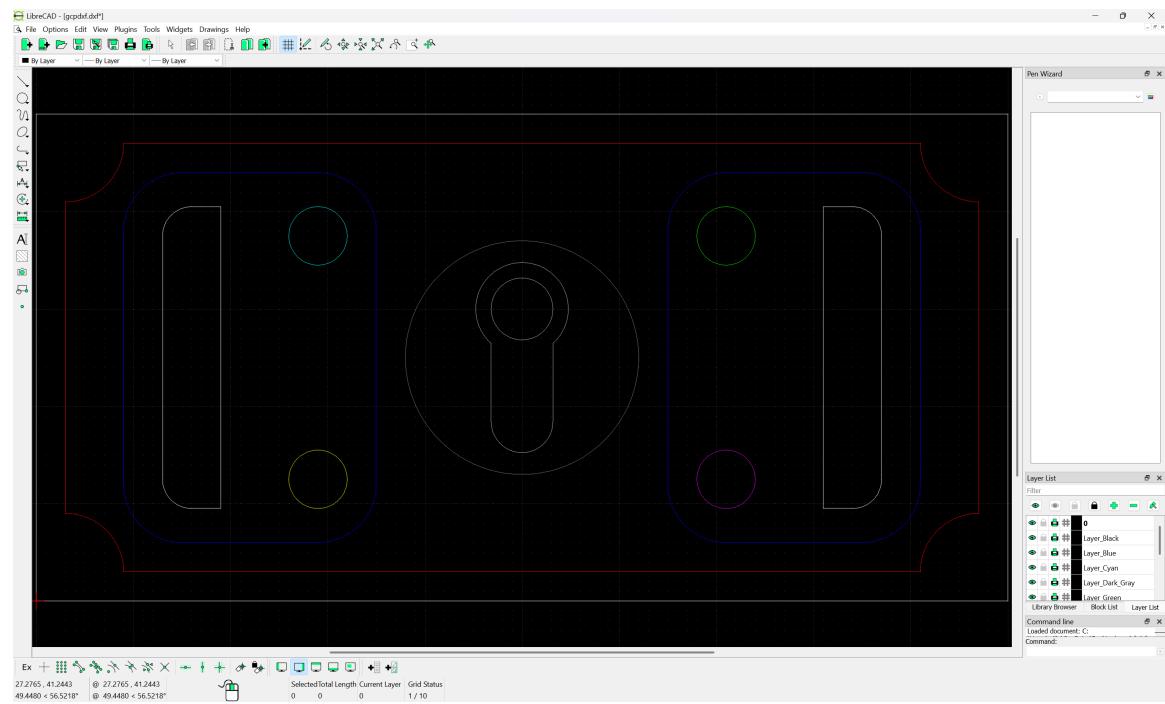
```

```

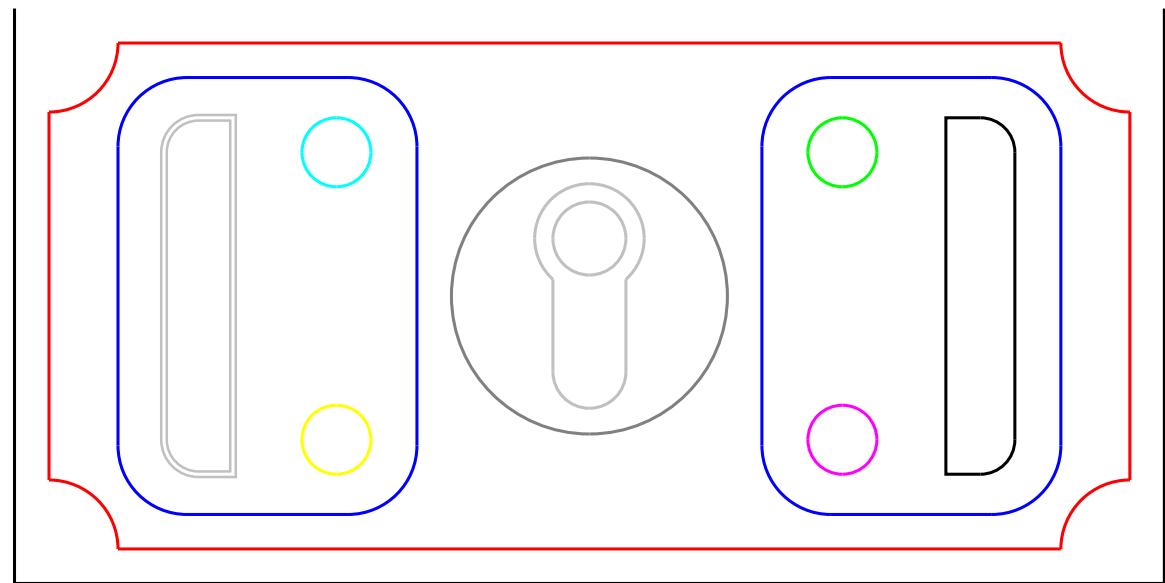
(radius * 2), cornerstyle, radius)
67 gcpdxfpy
68 gcpdxfpy gcp.setdxfcolor("Black")
69 gcpdxfpy
70 gcpdxfpy gcp.beginpolyline(large_square_tool_num)
71 gcpdxfpy gcp.addvertex(large_square_tool_num, stockXwidth*0.75+radius*1.5,
    stockYheight/4-radius/2)
72 gcpdxfpy gcp.addvertex(large_square_tool_num, 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()
```

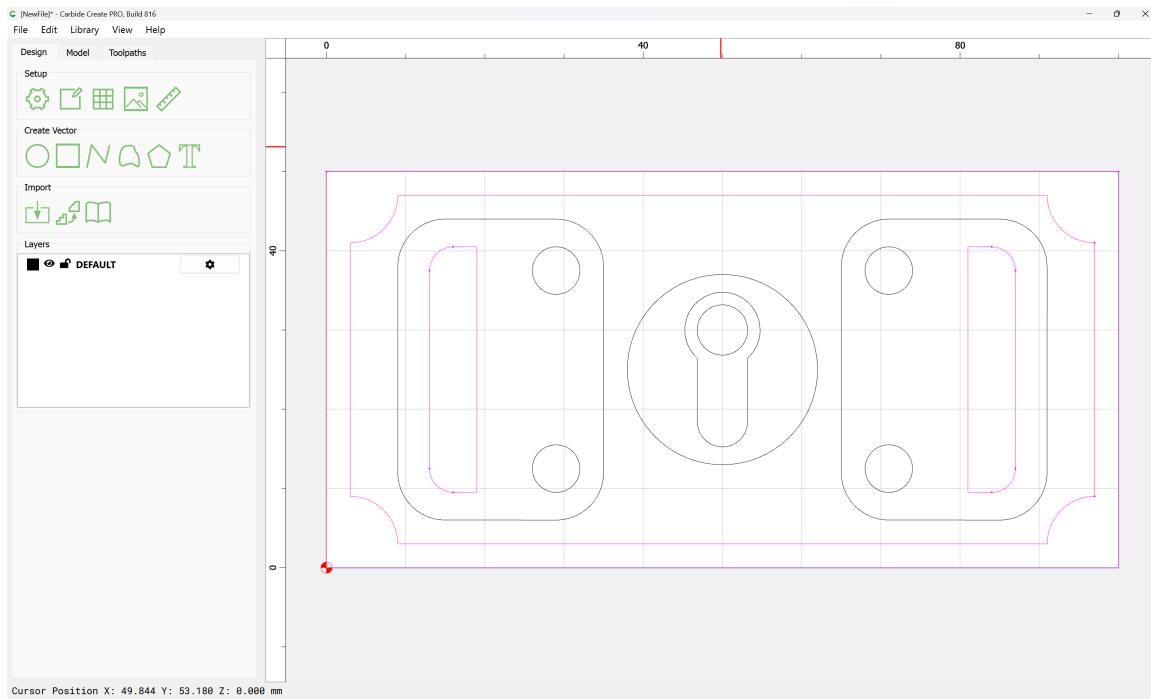
which creates a .dxf file which may be imported into any CAD program:



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] */

```



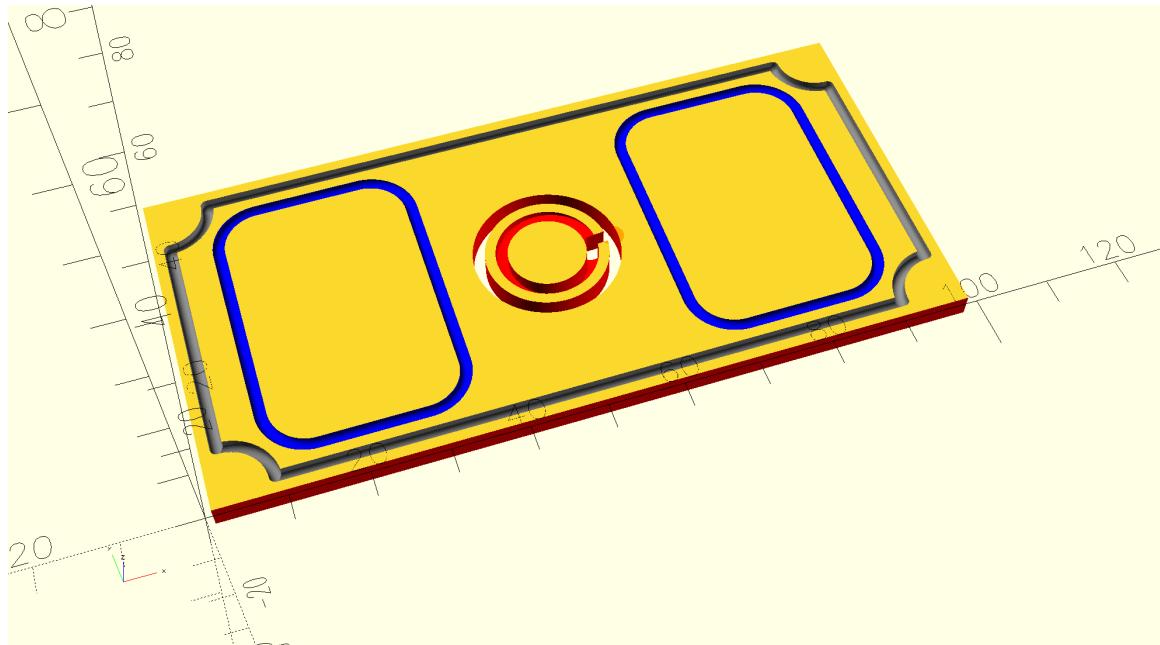
```

        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.cutrectanglerounddx( 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.closedxfile()
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 DXFs or G-code will cut requires the use of PythonSCAD.

```

1 gcptmplpy #!/usr/bin/env python
2 gcptmplpy
3 gcptmplpy import sys
4 gcptmplpy
5 gcptmplpy try:
6 gcptmplpy     if 'gcodepreview' in sys.modules:
7 gcptmplpy         del sys.modules['gcodepreview']
8 gcptmplpy except AttributeError:
9 gcptmplpy     pass
10 gcptmplpy
11 gcptmplpy from gcodepreview import *
12 gcptmplpy
13 gcptmplpy fa = 2
14 gcptmplpy fs = 0.125
15 gcptmplpy
16 gcptmplpy # [Export] */
17 gcptmplpy Base_filename = "aexport"

```

```

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

```

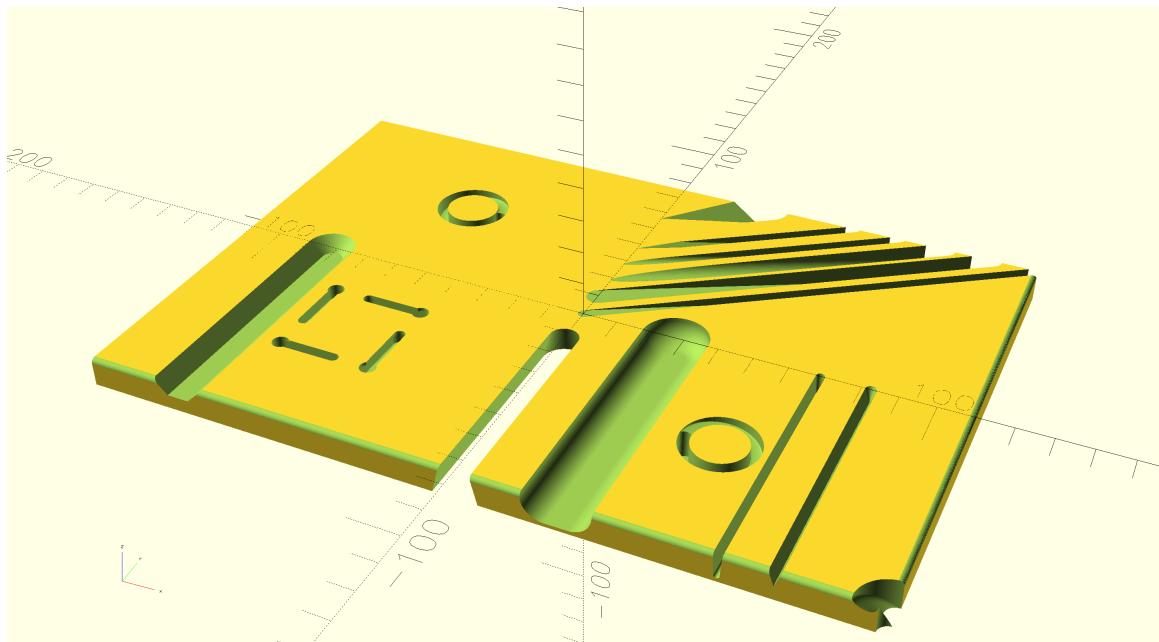
```

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));

```

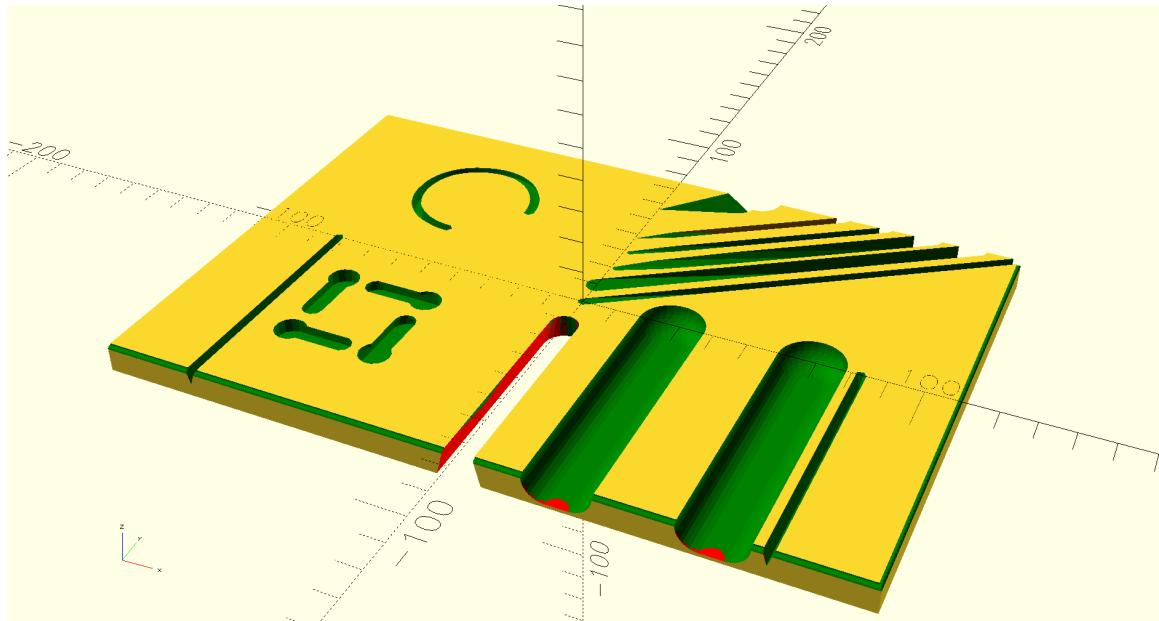
```

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.moveatfeedrate(0,0,layerheight,20000)
43 gcpthreedp gcp.extrude(1, 2, layerheight)
44 gcpthreedp
45 gcpthreedp gcp.stockandtoolpaths("toolpaths")
46 gcpthreedp
47 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 (gcp) — 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 output(<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

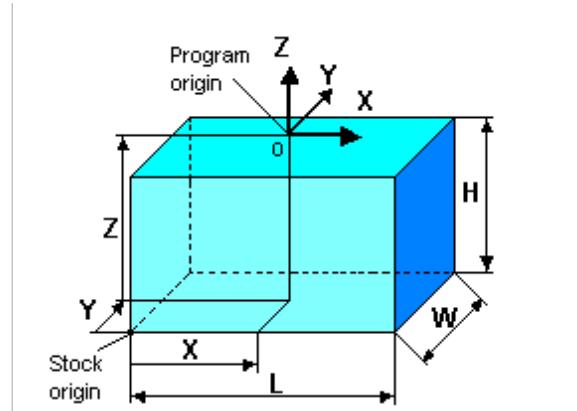
This problem space, showing the result of cutting stock using tooling in 3D has a number of tools addressing it, Camotics (formerly OpenSCAM) is an opensource option. Many tools simply create a wireframe preview such as <https://ncviewer.com/>. Cutviewer is a notable commercial program which has a unique 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:

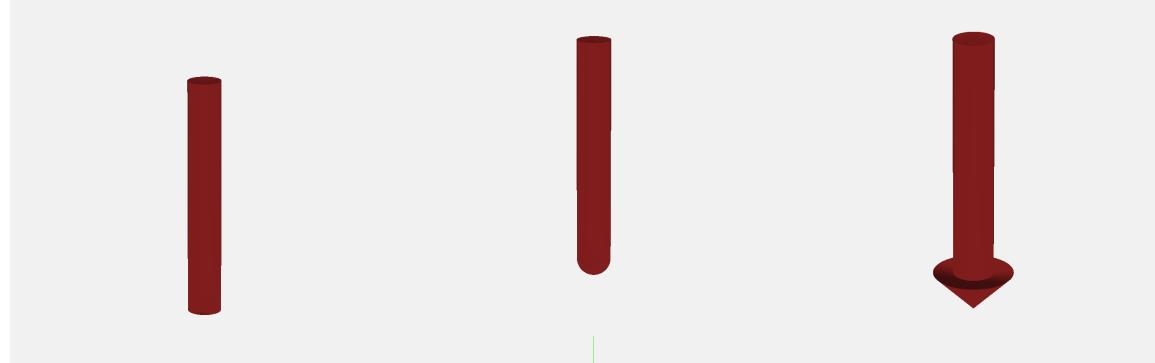
```
(STOCK/BLOCK, Length, Width, Height, Origin X, Origin Y, Origin Z)
```



#### 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:



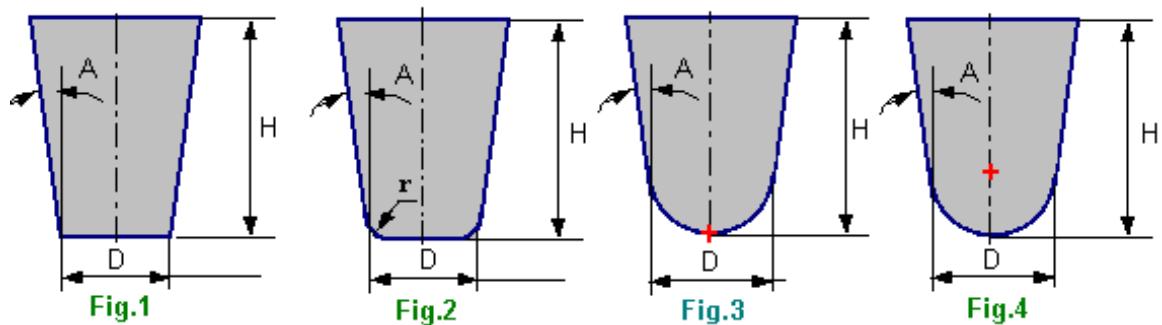
- Square (#201 and 102) — able to cut a flat bottom, perpendicular side and right angle, their simple and easily understood geometry makes them a standard choice
- Ballnose (#202 and 101) — rounded, they are the standard choice for concave and organic shapes
- V tooling (#301, 302, 311 and 312) — pointed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



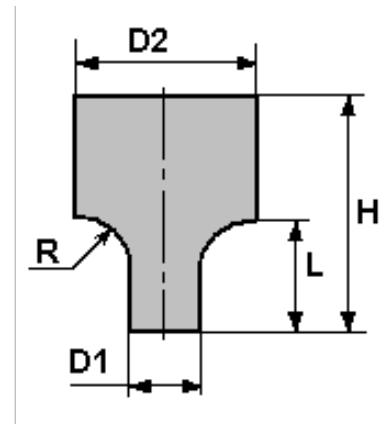
Note that it is possible to use these definitions for a wide variety of tooling, e.g., a Carbide 3D #301 V tool being represented as:

(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	
dxf	cutlinedxf	dxfline		cutarcdxf	dxfarc
gcode	cutlinegc		linegc	cutarcgc	arcgc
			cutlinedxfgc		cutarcdxfgc

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

The gcode and dxf columns and the matter of having specific commands which encompass generategcode those file types is tied up in having the internal variables generategcode, generatedxf and im- generatedxf plementations, and a strong argument could be made that this should simply be handled by generatecut if...then structures using those variables. The addition of a generatecut variable adds the necessary symmetry. Note that an early option to output a separate file for each tool used has since been deprecated and 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/.gcode file, it will be necessary to have a matching command for the combinatorial possibilities, hence:

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

Principles for naming modules (and variables):

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

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

- rectangle
  - cutrectangle
  - cutrectangleround

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

There are multiple modes for programming PythonSCAD:

- Python — in gcodepreview this allows writing out dxf files and using mutable variables (this is done in current versions of this project)
- OpenSCAD — see: <https://openscad.org/documentation.html>
- Programming in Python, calling Python from OpenSCAD using dispatchers/descriptors (this is done in current versions of this project)
- Programming in OpenSCAD with variables and calling Python — this requires 3 files and was originally used in the project as written up at: [https://github.com/WillAdams/gcodepreview/blob/main/gcodepreview-openscad\\_0\\_6.pdf](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 gcpy                 self.generatecut = False
52 gpy                 self.generateprint = True
53 gpy                 self.gcodefilext = ".gcode"
54 gpy             elif cutorprint == "cut":
55 gpy                 self.generatecut = True
56 gpy                 self.generateprint = False
57 gpy                 self.gcodefilext = ".nc"
58 gpy             else: # no_preview
59 gpy                 self.generatecut = False
60 gpy                 self.generateprint = False
61 gpy             if generategcode == True:
62 gpy                 self.generategcode = True
63 gpy             elif generategcode == 1:
64 gpy                 self.generategcode = True
```

```

65 gcpy      elif generategcode == 0:
66 gcpy          self.generategcode = False
67 gcpy      else:
68 gcpy          self.generategcode = generategcode
69 gcpy      if generatedxf == True:
70 gcpy          self.generatedxf = True
71 gcpy      elif generatedxf == 1:
72 gcpy          self.generatedxf = True
73 gcpy      elif generatedxf == 0:
74 gcpy          self.generatedxf = False
75 gcpy      else:
76 gcpy          self.generatedxf = generatedxf
77 gcpy # unless multiple dxfs are enabled, the check for them is of course
78 gcpy          False
79 gcpy      self.generateddxfs = False
80 gcpy      fa = gcpfa
81 gcpy      fs = gcpfs
82 gcpy      self.steps = steps
83 gcpy # initialize the machine state
84 gcpy      self.mc = "Initialized"
85 gcpy      self.mpx = float(0)
86 gcpy      self.mpy = float(0)
87 gcpy      self.mpz = float(0)
88 gcpy      self.tpz = float(0)
89 gcpy # initialize the toolpath state
90 gcpy      self.retractheight = 5
91 gcpy # initialize the DEFAULT tool
92 gcpy      self.currenttoolnum = 102
93 gcpy      self.endmilltype = "square"
94 gcpy      self.diameter = 3.175
95 gcpy      self.flute = 12.7
96 gcpy      self.shaftdiameter = 3.175
97 gcpy      self.shaftheight = 12.7
98 gcpy      self.shaftlength = 19.5
99 gcpy      self.toolnumber = "100036"
100 gcpy     self.cutcolor = "green"
101 gcpy     self.rapidcolor = "orange"
102 gcpy     self.shaftcolor = "red"
103 gcpy # the command definesquaretool(3.175, 12.7, 20) is used in the
104 gcpy      toolchange command
104 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
105 gcpy          =[[0,0],[1.5875,0],[1.5875,12.7],[0,12.7]] )
106 gcpy      self.shaftoutline = polygon( points
106 gcpy          =[[0,12.7],[3.175,12.7],[3.175,25.4],[0,25.4]] )
107 gcpy      self.shaftprofile = polygon( points
107 gcpy          =[[0,12.7],[1.5875,12.7],[1.5875,25.4],[0,25.4]] )
108 gcpy      self.currenttoolshape = cylinder(h = self.flute, r = self.
108 gcpy          shaftdiameter/2)
109 gcpy      sh = cylinder(h = self.flute, r = self.shaftdiameter/2)
110 gcpy      self.currenttoolshaft = sh.translate([0,0,self.flute])
111 gcpy # debug mode requires a variable to track if it is on or off
112 gcpy      self.debugenable = False
113 gcpy # the variables for holding 3D models must be initialized as empty
113 gcpy      lists so as to ensure that only append or extend commands are
113 gcpy      used with them
114 gcpy      self.rapids = []
115 gcpy      self.toolpaths = []
116 gcpy      print("gcodepreview\u2225class\u2225initialized")
117 gcpy
118 gcpy #     def myfunc(self, var):
119 gcpy #         self.vv = var * var
120 gcpy #     return self.vv
121 gcpy #
122 gcpy #     def getvv(self):
123 gcpy #         return self.vv
124 gcpy #
125 gcpy #     def checkint(self):
126 gcpy #         return self.mc
127 gcpy #
128 gcpy #     def makecube(self, xdim, ydim, zdim):
129 gcpy #         self.c=cube([xdim, ydim, zdim])
130 gcpy #
131 gcpy #     def placecube(self):
132 gcpy #         show(self.c)
133 gcpy #

```

---

```
134 gcpy #     def instantiatecube(self):
135 gcpy #         return self.c
```

---

### 3.3.2 Position and Variables

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

The first such variables are for xyz position:

mpx	• mpx
mpy	• mpy
mpz	• mpz

Similarly, for some toolpaths it will be necessary to track the depth along the Z-axis as the toolpath `tpzinc` is cut out, or the increment which a cut advances — this is done using an internal variable, `tpzinc`.

It will further be necessary to have a variable for the current tool:

currenttoolnum	• currenttoolnum
----------------	------------------

Note that the `currenttoolnum` variable should always be accessed and used for any specification of a tool, being read in whenever a tool is to be made use of, or a parameter or aspect of the tool needs to be used in a calculation.

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

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

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

---

```
137 gcpy     def xpos(self):
138 gcpy         return self.mpx
139 gcpy
140 gcpy     def ypos(self):
141 gcpy         return self.mpy
142 gcpy
143 gcpy     def zpos(self):
144 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	• setypos
setzpos	• setzpos

```
146 gcpy     def setxpos(self, newxpos):
147 gcpy         self.mpx = newxpos
148 gcpy
149 gcpy     def setypos(self, newypos):
150 gcpy         self.mpy = newypos
151 gcpy
152 gcpy     def setzpos(self, newzpos):
153 gcpy         self.mpz = newzpos
```

---

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

### 3.3.3 Initial Modules

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

- mpx
- mpy
- mpz

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

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

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

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

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

---

```

155 gcpy      def setupstock(self, stockXwidth,
156 gcpy          stockYheight,
157 gcpy          stockZthickness,
158 gcpy          zeroheight,
159 gcpy          stockzero,
160 gcpy          retractheight):
161 gcpy          """
162 gcpy      Set up blank/stock for material and position/zero.
163 gcpy
164 gcpy      Parameters
165 gcpy      -----
166 gcpy      stockXwidth : float
167 gcpy          X extent/dimension
168 gcpy      stockYheight : float
169 gcpy          Y extent/dimension
170 gcpy      stockZthickness : boolean
171 gcpy          Z extent/dimension
172 gcpy      zeroheight : string
173 gcpy          Top or Bottom, determines if Z extent will
174 gcpy          be positive or negative
175 gcpy      stockzero : string
176 gcpy          Lower-Left, Center-Left, Top-Left, Center,
177 gcpy          determines XY position of stock
178 gcpy      retractheight : float
179 gcpy          Distance which tool retracts above surface
180 gcpy          of stock.
181 gcpy
182 gcpy      Returns
183 gcpy      -----
184 gcpy      none
185 gcpy
186 gcpy      self.stockXwidth = stockXwidth
187 gcpy      self.stockYheight = stockYheight
188 gcpy      self.stockZthickness = stockZthickness
189 gcpy      self.zeroheight = zeroheight
190 gcpy      self.stockzero = stockzero
191 gcpy      self.retractheight = retractheight
192 gcpy      self.stock = cube([stockXwidth, stockYheight,
193 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.

---

```

191 gcpy      if self.zeroheight == "Top":
192 gcpy          if self.stockzero == "Lower-Left":
193 gcpy              self.stock = self.stock.translate([0, 0, -self.
194 gcpy                  stockZthickness])
195 gcpy          if self.generategcode == True:
196 gcpy              self.writegc("(stockMin:0.00mm,△0.00mm,△-", str
197 gcpy                  (self.stockZthickness), "mm)")
198 gcpy              self.writegc("(stockMax:", str(self.stockXwidth
199 gcpy

```

---

```

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

```

---

```

229 gcpy           str(self.stockYheight/2), ",_0.00mm");
230 gcpy           if self.stockzero == "Top-Left":
231 gcpy               self.stock = self.stock.translate([0, -self.
232 gcpy                   stockYheight, 0])
233 gcpy           if self.generategcode == True:
234 gcpy               self.writegc("(stockMin:0.00mm,_-", str(self.
235 gcpy                   stockYheight), "mm,_0.00mm)")
236 gcpy               self.writegc("(stockMax:", str(self.stockXwidth
237 gcpy                   ), "mm,_0.00mm,_", str(self.stockZthickness)
238 gcpy                   , "mm)")
239 gcpy               self.writegc("(STOCK/BLOCK,_", str(self.
240 gcpy                   stockXwidth), ",_", str(self.stockYheight),
241 gcpy                   ",_", str(self.stockZthickness), ",_0.00,_",
242 gcpy                   str(self.stockYheight), ",_0.00)")
243 gcpy           if self.generategcode == True:
244 gcpy               self.writegc("G90");
245 gcpy               self.writegc("G21");

```

---

Note that while the #102 is declared as a default tool, while it was originally necessary to call a tool change after invoking `setupstock`, in the 2024.09.03 version of PythonSCAD this requirement went away when an update which interfered with persistently setting a variable directly was fixed. The `setupstock` command is required if working with a 3D project, creating the block of stock which the following toolpath commands will cut away. Note that since Python in OpenPython-SCAD defers output of the 3D model, it is possible to define it once, then set up all the specifics for each possible positioning of the stock in terms of origin.

The OpenSCAD version is simply a descriptor:

---

```

36 gpcscad module setupstock(stockXwidth, stockYheight, stockZthickness,
37 gpcscad     zeroheight, stockzero, retractheight) {
38 gpcscad     gcp.setupstock(stockXwidth, stockYheight, stockZthickness,
39 gpcscad         zeroheight, stockzero, retractheight);
40 gpcscad }

```

---

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

---

```

245 gcpy     def setupcuttingarea(self, sizeX, sizeY, sizeZ, extentleft,
246 gcpy         extentfb, extentd):
247 gcpy         self.initializemachinestate()
248 gcpy         c=cube([sizeX,sizeY,sizeZ])
249 gcpy         c = c.translate([extentleft,extentfb,extentd])
250 gcpy         self.stock = c
251 gcpy         self.toolpaths = []
252 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:

---

```

253 gcpy     def debug(self, *args: any, sep: str = " ", end: str = "\n", **kwargs) -> None:
254 gcpy         """
255 gcpy             Print debug output if enabled.
256 gcpy
257 gcpy             Accepts the same arguments as built-in print (except file
258 gcpy                 is supported via print_kwarg).
259 gcpy
260 gcpy             if not self.debugenable:
261 gcpy                 return

```

---

```

261 gcpy      # Build the message and print under a lock to avoid
               interleaving in multithreaded apps
262 gcpy      self.prefix = "DEBUG: "
263 gcpy      msg = self.prefix + sep.join(map(str, args))
264 gcpy      with self._lock:
265 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:

---

```

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

```

---

```

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

```

---

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

---

```

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

```

---

the OpenSCAD module is a descriptor as expected:

---

```

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

```

---

## 3.4 Tools and Shapes and Changes

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

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

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

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

```

tooltype = "mill"
diameter = first
cornerradius = second
height = third
taperangle
length

```

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

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

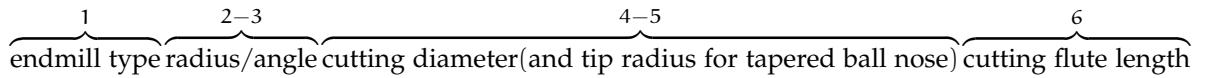
Note that it will also be necessary to write out a tool description compatible with the program CutViewer as a G-code comment so that it may be used as a 3D previewer for the G-code for tool changes in G-code. Several forms are available as described below.

### 3.4.1 Numbering for Tools

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

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

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



- 1st digit: endmill type:
  - 0 - manufacturer number
  - 1 - square (incl. "O"-flute)
  - 2 - ball
  - 3 - V
  - 4 - bowl
  - 5 - tapered ball
  - 6 - roundover
  - 7 - thread-cutting
  - 8 - dovetail
  - 9 - other (e.g., user-defined, or unsupported tools, keyhole, lollipop, &c.)
- 2nd and 3rd digits shape radius (ball/roundover) or angle (V), 2nd and 3rd digit together 10–99 indicate measurement in tenth of a millimeter. 2nd digit:
  - 0 - Imperial (00 indicates n/a or square)
  - any other value for both the 2nd and 3rd digits together indicate a metric measurement or an angle in degrees
- 3rd digit (if 2nd is 0 indicating Imperial)
  - 1 -  $1/32^{\text{nd}}$
  - 2 -  $1/16$
  - 3 -  $1/8$
  - 4 -  $1/4$
  - 5 -  $5/16$
  - 6 -  $3/8$
  - 7 -  $1/2$
  - 8 -  $3/4$
  - 9 -  $>1''$  or other
- 4th and 5th digits cutting diameter as 2nd and 3rd above except 4th digit indicates tip radius for tapered ball nose and such tooling is only represented in Imperial measure:
- 4th digit (tapered ball nose)
  - 1 - 0.01 in (this is the 0.254mm of the #501 and 502)

- 2 - 0.015625 in (1/64th)
- 3 - 0.0295
- 4 - 0.03125 in (1/32nd)
- 5 - 0.0335
- 6 - 0.0354
- 7 - 0.0625 in (1/16th)
- 8 - 0.125 in (1/8th)
- 9 - 0.25 in (1/4)
- 6th digit cutting flute length:
  - 0 - other
  - 1 - calculate based on V angle
  - 2 - 1/16
  - 3 - 1/8
  - 4 - 1/4
  - 5 - 5/16
  - 6 - 1/2
  - 7 - 3/4
  - 8 - "long reach" or greater than 3/4"
  - 9 - calculate based on radius
- or 6th digit tip diameter for roundover tooling (added to cutting diameter to arrive at actual cutting diameter — note that these values are the same as for the tip radius of the #501 and 502)
  - 1 - 0.01 in
  - 2 - 0.015625 in (1/64th)
  - 3 - 0.0295
  - 4 - 0.03125 in (1/32nd)
  - 5 - 0.0335
  - 6 - 0.0354
  - 7 - 0.0625 in (1/16th)
  - 8 - 0.125 in (1/8th)
  - 9 - 0.25 in (1/4)

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

- Square

```
#122 == 100012
#112 == 100024
#102 == 100036 (also #274 and #326 (Amana 46200-K))
#201 == 100047 (also #251 and #322 (Amana 46202-K))
#205 == 100048
#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.

---

```
277 gcpy     def currenttoolnumber(self):
278 gcpy         return(self.currenttoolnum)
```

---

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

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

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

- radius
- angle

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

tool number **3.4.1.1 toolchange** This command accepts a tool number and assigns its characteristics as 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 settoolparameters** Not currently used, this command is intended for a state where tools are defined in a vendor-neutral fashion.

---

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

---

```

295 gcpy #M6T326 (TOOL/MILL,3.17, 0.00, 12.70, 0.00)
296 gcpy #M6T602 (TOOL/MILL,25.40, 0.00, 9.91, 0.00)
297 gcpy #M6T603 (TOOL/MILL,25.40, 0.00, 9.91, 0.00)
298 gcpy #M6T274 (TOOL/MILL,3.17, 0.00, 12.70, 0.00)
299 gcpy #M6T278 (TOOL/MILL,6.35, 0.00, 19.05, 0.00)
300 gcpy #M6T282 (TOOL/MILL,2.00, 0.00, 6.35, 0.00)
301 gcpy           self.endmilltype = "square"
302 gcpy           self.diameter = diameter
303 gcpy           self.flute = height
304 gcpy           self.shaftdiameter = diameter
305 gcpy           self.shaftheight = height
306 gcpy           self.shaftlength = height
307 gcpy #
308 gcpy           elif cornerradius > 0 and taperangle == 0:
309 gcpy #M6T121 (TOOL/MILL,0.80, 0.40, 1.59, 0.00)
310 gcpy #M6T111 (TOOL/MILL,1.59, 0.79, 6.35, 0.00)
311 gcpy #M6T101 (TOOL/MILL,3.17, 1.59, 12.70, 0.00)
312 gcpy #M6T202 (TOOL/MILL,6.35, 3.17, 19.05, 0.00)
313 gcpy #M6T325 (TOOL/MILL,6.35, 3.17, 25.40, 0.00)
314 gcpy           self.endmilltype = "ball"
315 gcpy           self.diameter = diameter
316 gcpy           self.flute = height
317 gcpy           self.shaftdiameter = diameter
318 gcpy           self.shaftheight = height
319 gcpy           self.shaftlength = height
320 gcpy #
321 gcpy           elif taperangle > 0:
322 gcpy #M6T301 (TOOL/MILL,0.10, 0.05, 6.35, 45.00)
323 gcpy #M6T302 (TOOL/MILL,0.10, 0.05, 6.35, 30.00)
324 gcpy #M6T327 (TOOL/MILL,0.10, 0.05, 23.39, 30.00)
325 gcpy           self.endmilltype = "V"
326 gcpy           self.diameter = Tan(taperangle / 2) * height
327 gcpy           self.flute = height
328 gcpy           self.angle = taperangle
329 gcpy           self.shaftdiameter = Tan(taperangle / 2) * height
330 gcpy           self.shaftheight = height
331 gcpy           self.shaftlength = height
332 gcpy #
333 gcpy           elif tooltype == "chamfer":
334 gcpy             tipdiameter = first
335 gcpy             radius = second
336 gcpy             height = third
337 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.

```

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

```

---

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

```

346 gcpy     if (tool_number == 102) or (tool_number == 100036): #
347 gcpy       102/326 == 100036
348 gcpy       self.writegc("(TOOL/MILL, 3.175, 0.00, 0.00, 0.00)")
349 gcpy       self.endmilltype = "square"
350 gcpy       self.diameter = 3.175
351 gcpy       self.flute = 12.7
352 gcpy       self.shaftdiameter = 3.175
353 gcpy       self.shaftheight = 12.7
354 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:

```

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

```

---

```
412 gcpy           self.toolnumber = "100048"
413 gcpy #
```

---

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

---

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

---

**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.

---

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

```

463 gcpy           self.flute = 12.7
464 gcpy           self.shaftdiameter = 3.175
465 gcpy           self.shaftheight = 12.7
466 gcpy           self.shaftlength = 20.0
467 gcpy           self.defineballnose(tool.diameter, self.flute, self
                     .shaftlength)
468 gcpy           self.defineshaft(tool.diameter, self.shaftdiameter,
                     self.flute, 0, self.shaftlength)
469 gcpy           self.toolnumber = "203036"
470 gcpy           elif (tool_number == 111) or (tool_number == 202024): #111
                     == 202024
471 gcpy           self.writegc("(TOOL/MILL, 1.5875, 0.79375, 0.00, 0.00)"
                     )
472 gcpy           self.endmilltype = "ball"
473 gcpy           self.diameter = 1.5875
474 gcpy           self.flute = 6.35
475 gcpy           self.shaftdiameter = 3.175
476 gcpy           self.shaftheight = 6.35
477 gcpy           self.shaftlength = 20.0
478 gcpy           self.defineballnose(tool.diameter, self.flute, self
                     .shaftlength, (self.shaftdiameter - self.diameter)
                     /2)
479 gcpy           self.defineshaft(tool.diameter, self.shaftdiameter,
                     self.flute, 0, self.shaftlength)
480 gcpy           self.toolnumber = "202024"
481 gcpy           elif (tool_number == 121) or (tool_number == 201012): #121
                     == 201012
482 gcpy           self.writegc("(TOOL/MILL, 3.175, 0.79375, 0.00, 0.00)")
483 gcpy           self.endmilltype = "ball"
484 gcpy           self.diameter = 0.79375
485 gcpy           self.flute = 1.5875
486 gcpy           self.shaftdiameter = 3.175
487 gcpy           self.shaftheight = 1.5875
488 gcpy           self.shaftlength = 20.0
489 gcpy           self.defineballnose(tool.diameter, self.flute, self
                     .shaftlength, (self.shaftdiameter - self.diameter)
                     /2)
490 gcpy           self.defineshaft(tool.diameter, self.shaftdiameter,
                     self.flute, 0, self.shaftlength)
491 gcpy           self.toolnumber = "201012"
492 gcpy           elif (tool_number == 325) or (tool_number == 204048): #325
                     (Amana 46376-K) == 204048
493 gcpy           self.writegc("(TOOL/MILL, 6.35, 3.175, 0.00, 0.00)")
494 gcpy           self.endmilltype = "ball"
495 gcpy           self.diameter = 6.35
496 gcpy           self.flute = 25.4
497 gcpy           self.shaftdiameter = 6.35
498 gcpy           self.shaftheight = 25.4
499 gcpy           self.shaftlength = 20.0
500 gcpy           self.defineballnose(tool.diameter, self.flute, self
                     .shaftlength, (self.shaftdiameter - self.diameter)
                     /2)
501 gcpy           self.defineshaft(tool.diameter, self.shaftdiameter,
                     self.flute, 0, self.shaftlength)
502 gcpy           self.toolnumber = "204048"
503 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).

```

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

```

```

516 gcpy          self.writegc("(TOOL/MILL,0.10,0.05,6.35,30.00)")
517 gcpy          self.endmilltype = "V"
518 gcpy          self.diameter = 12.7
519 gcpy          self.flute = 11.067
520 gcpy          self.angle = 60
521 gcpy          self.shaftdiameter = 6.35
522 gcpy          self.shaftheight = 11.067
523 gcpy          self.shaftlength = 20.0
524 gcpy          self.defineVtool(self.diameter, self.flute, self.
525 gcpy                  shaftlength, self.shaftdiameter)
526 gcpy          self.toolnumber = "360071"
527 gcpy      elif (tool_number == 390) or (tool_number == 390032): #390
528 gcpy          == 390032
529 gcpy          self.writegc("(TOOL/MILL,0.03,0.00,1.5875,45.00)")
530 gcpy          self.endmilltype = "V"
531 gcpy          self.diameter = 3.175
532 gcpy          self.flute = 1.5875
533 gcpy          self.angle = 90
534 gcpy          self.shaftdiameter = 3.175
535 gcpy          self.shaftheight = 1.5875
536 gcpy          self.shaftlength = 20.0
537 gcpy          self.defineVtool(self.diameter, self.flute, self.
538 gcpy                  shaftlength, self.shaftdiameter)
539 gcpy          self.toolnumber = "390032"
540 gcpy      elif (tool_number == 327) or (tool_number == 360098): #327
541 gcpy          (Amana RC-1148) == 360098
542 gcpy          self.writegc("(TOOL/MILL,0.03,0.00,13.4874,30.00)")
543 gcpy          self.endmilltype = "V"
544 gcpy          self.diameter = 25.4
545 gcpy          self.flute = 22.134
546 gcpy          self.angle = 60
547 gcpy          self.shaftdiameter = 6.35
548 gcpy          self.shaftheight = 22.134
549 gcpy          self.shaftlength = 20.0
550 gcpy          self.defineVtool(self.diameter, self.flute, self.
551 gcpy                  shaftlength, self.shaftdiameter)
552 gcpy          self.toolnumber = "360098"
553 gcpy      elif (tool_number == 323) or (tool_number == 330041): #323
554 gcpy          == 330041 30 degree V Amana, 45771-K
555 gcpy          self.writegc("(TOOL/MILL,0.10,0.05,11.18,15.00)")
556 gcpy          self.endmilltype = "V"
557 gcpy          self.diameter = 6.35
558 gcpy          self.flute = 11.849
559 gcpy          self.angle = 30
560 gcpy          self.shaftdiameter = 6.35
561 gcpy          self.shaftheight = 11.849
562 gcpy          self.shaftlength = 20.0
563 gcpy          self.defineVtool(self.diameter, self.flute, self.
564 gcpy                  shaftlength, self.shaftdiameter)
565 gcpy          self.toolnumber = "330041"
566 gcpy      #

```

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

```
560 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
```

```

579 gcpy           self.shaftlength = 20.0
580 gcpy           self.defineKeyholetool(self.diameter, self.flute, self.
581 gcpy             shaftdiameter, self.shaftheight, self.shaftdiameter,
582 gcpy               self.shaftlength)
583 gcpy           self.toolnumber = "906053"
584 gcpy           elif (tool_number == 376) or (tool_number == 907040): #376
585 gcpy             == 907040
586 gcpy             self.writegc("(TOOL/MILL,12.7,0.00,4.77,0.00)")
587 gcpy             self.endmilltype = "keyhole"
588 gcpy             self.diameter = 12.7
589 gcpy             self.flute = 4.7625
590 gcpy             self.radius = 6.35
591 gcpy             self.shaftdiameter = 6.35
592 gcpy             self.shaftheight = 4.7625
593 gcpy             self.shaftlength = 20.0
594 gcpy             self.defineKeyholetool(self.diameter, self.flute, self.
595 gcpy               shaftdiameter, self.shaftheight, self.shaftdiameter,
596 gcpy                 self.shaftlength)
597 gcpy             self.toolnumber = "907040"
598 gcpy             elif (tool_number == 378) or (tool_number == 907050): #378
599 gcpy               == 907050
600 gcpy             self.writegc("(TOOL/MILL,12.7,0.00,4.77,0.00)")
601 gcpy             self.endmilltype = "keyhole"
602 gcpy             self.diameter = 12.7
603 gcpy             self.flute = 4.7625
604 gcpy             self.radius = 8
605 gcpy             self.shaftdiameter = 6.35
606 gcpy             self.shaftheight = 4.7625
607 gcpy             self.shaftlength = 20.0
608 gcpy             self.defineKeyholetool(self.diameter, self.flute, self.
609 gcpy               shaftdiameter, self.shaftheight, self.shaftdiameter,
610 gcpy                 self.shaftlength)
611 gcpy             self.toolnumber = "907050"
612 gcpy #

```

---

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

```

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

```

---

```

640 gcpy           self.toolnumber = "401370"
641 gcpy         elif (tool_number == 1372): #1372/45982 == 401372
642 gcpy #Whiteside Bowl & Tray Bit 1/4"SH, 1/4"R, 3/4"CD (5/8" cutting
               flute length)
643 gcpy #Amana Carbide Tipped Bowl & Tray 1/4 Radius x 3/4 Dia x 5/8 x 1/4
               Inch Shank
644 gcpy           self.writegc("(TOOL/MILL,19.5,15.875,6.35,0.00)")
645 gcpy           self.endmilltype = "bowl"
646 gcpy           self.diameter = 19.5
647 gcpy           self.flute = 15.875
648 gcpy           self.radius = 6.35
649 gcpy           self.shaftdiameter = 6.35
650 gcpy           self.shaftheight = 15.875
651 gcpy           self.shaftlength = 20.0
652 gcpy           self.definebowltool(self.diameter, self.flute, self.
               radius, self.shaftdiameter, self.shaftlength)
653 gcpy           self.toolnumber = "401372"
654 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.

---

```

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

```

---

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

---

```

688 gcpy      elif (tool_number == 56125) or (tool_number == 603042):#
               0.508/2, 1.531 56125 == 603042
689 gcpy      self.writegc("(TOOL/CRMILL, 0.508, 6.35, 3.175, 7.9375,
                           3.175)")
690 gcpy      self.endmilltype = "roundover"
691 gcpy      self.tipdiameter = 0.508
692 gcpy      self.diameter = 6.35 - self.tipdiameter
693 gcpy      self.flute = 8 - self.tipdiameter
694 gcpy      self.radius = 3.175 - self.tipdiameter/2
695 gcpy      self.shaftdiameter = 6.35
696 gcpy      self.shaftheight = 8
697 gcpy      self.shaftlength = 10.0
698 gcpy      self.defineRoundoverTool(self.diameter, self.
                                         tipdiameter, self.flute, self.radius, self.
                                         shaftdiameter, self.shaftlength)
699 gcpy      self.toolnumber = "603042"
700 gcpy      elif (tool_number == 56142) or (tool_number == 602032):#
               0.508/2, 2.921 56142 == 602032
701 gcpy      self.writegc("(TOOL/CRMILL, 0.508, 3.571875, 1.5875,
                           5.55625, 1.5875)")
702 gcpy      self.endmilltype = "roundover"
703 gcpy      self.tip = 0.508
704 gcpy      self.diameter = 3.175 - self.tip
705 gcpy      self.flute = 4.7625 - self.tip
706 gcpy      self.radius = 1.5875 - self.tip/2
707 gcpy      self.shaftdiameter = 3.175
708 gcpy      self.shaftheight = 4.7625
709 gcpy      self.shaftlength = 10.0
710 gcpy      self.toolnumber = "602032"
711 gcpy #      elif (tool_number == 312):#1.524/2, 3.175
712 gcpy #      self.writegc("(TOOL/CRMILL, Diameter1, Diameter2,
                           Radius, Height, Length)")
713 gcpy #      elif (tool_number == 1568):#0.507/2, 4.509 1568 == 603032
714 gcpy ##FIX      self.writegc("(TOOL/CRMILL, 0.17018, 9.525,
                           4.7625, 12.7, 4.7625)")
715 gcpy ##      self.currenttoolshape = self.toolshapes("roundover",
                           3.175, 6.35, 3.175, 0.396875)
716 gcpy #      self.endmilltype = "roundover"
717 gcpy #      self.diameter = 3.175
718 gcpy #      self.flute = 6.35
719 gcpy #      self.radius = 3.175
720 gcpy #      self.tip = 0.396875
721 gcpy #      self.toolnumber = "603032"
722 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
723 gcpy #      elif (tool_number == 1570):#0.507/2, 4.509 1570 == 600002
               ?!?
724 gcpy #      self.writegc("(TOOL/CRMILL, 0.17018, 9.525, 4.7625,
                           12.7, 4.7625)")
725 gcpy ##      self.currenttoolshape = self.toolshapes("roundover",
                           4.7625, 9.525, 4.7625, 0.396875)
726 gcpy #      self.endmilltype = "roundover"
727 gcpy #      self.diameter = 4.7625
728 gcpy #      self.flute = 9.525
729 gcpy #      self.radius = 4.7625
730 gcpy #      self.tip = 0.396875
731 gcpy #      self.toolnumber = "600002"
732 gcpy #      elif (tool_number == 1572): #1572 = 604042
733 gcpy ##FIX      self.writegc("(TOOL/CRMILL, 0.17018, 9.525,
                           4.7625, 12.7, 4.7625)")
734 gcpy ##      self.currenttoolshape = self.toolshapes("roundover",
                           6.35, 12.7, 6.35, 0.396875)
735 gcpy #      self.endmilltype = "roundover"
736 gcpy #      self.diameter = 6.35
737 gcpy #      self.flute = 12.7
738 gcpy #      self.radius = 6.35
739 gcpy #      self.tip = 0.396875
740 gcpy #      self.toolnumber = "604042"
741 gcpy #      elif (tool_number == 1574): #1574 == 600062
742 gcpy ##FIX      self.writegc("(TOOL/CRMILL, 0.17018, 9.525,
                           4.7625, 12.7, 4.7625)")
743 gcpy ##      self.currenttoolshape = self.toolshapes("roundover",
                           9.525, 19.5, 9.515, 0.396875)
744 gcpy #      self.endmilltype = "roundover"
745 gcpy #      self.diameter = 9.525

```

---

```

746 gcpy #           self.flute = 19.5
747 gcpy #           self.radius = 9.515
748 gcpy #           self.tip = 0.396875
749 gcpy #           self.toolnumber = "600062"
750 gcpy #

```

---

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

---

```

751 gcpy      elif (tool_number == 814) or (tool_number == 814071): #814
                == 814071
752 gcpy #Item 18J1607, 1/2" 14° Dovetail Bit, 8mm shank
753 gcpy      self.writegc("(TOOL/MILL,12.7,6.367,12.7,0.00)")
754 gcpy      #    dt_bottomdiameter, dt_topdiameter, dt_height, dt_angle
                  )
755 gcpy      #    https://www.leevalley.com/en-us/shop/tools/power-tool-
                  accessories/router-bits/30172-dovetail-bits?item=18J1607
756 gcpy #      self.currenttoolshape = self.toolshapes("dovetail",
                  12.7, 12.7, 14)
757 gcpy      self.endmilltype = "dovetail"
758 gcpy      self.diameter = 12.7
759 gcpy      self.flute = 12.7
760 gcpy      self.angle = 14
761 gcpy      self.toolnumber = "814071"
762 gcpy      elif (tool_number == 808079) or (tool_number == 808071): #
                  45828 == 808071
                  self.writegc("(TOOL/MILL,12.7,6.816,20.95,0.00)")
763 gcpy      #    http://www.amanatool.com/45828-carbide-tipped-dovetail
                  -8-deg-x-1-2-dia-x-825-x-1-4-inch-shank.html
764 gcpy      #      self.currenttoolshape = self.toolshapes("dovetail",
                  12.7, 20.955, 8)
765 gcpy #      self.endmilltype = "dovetail"
766 gcpy      self.diameter = 12.7
767 gcpy      self.flute = 20.955
768 gcpy      self.angle = 8
769 gcpy      self.toolnumber = "808071"
770 gcpy #
771 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.

---

```
772 gcpy      self.writegc("M6T", str(tool_number))
773 gcpy #    if (self.endmilltype == "square"):
774 gcpy #        speed = speed *
775 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.

---

```

776 gcpy    def setcolor(self,
777 gcpy        cutcolor = "green",
778 gcpy        rapidcolor = "orange",
779 gcpy        shaftcolor = "red"):
780 gcpy            self.cutcolor = cutcolor
781 gcpy            self.rapidcolor = rapidcolor
782 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.

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		

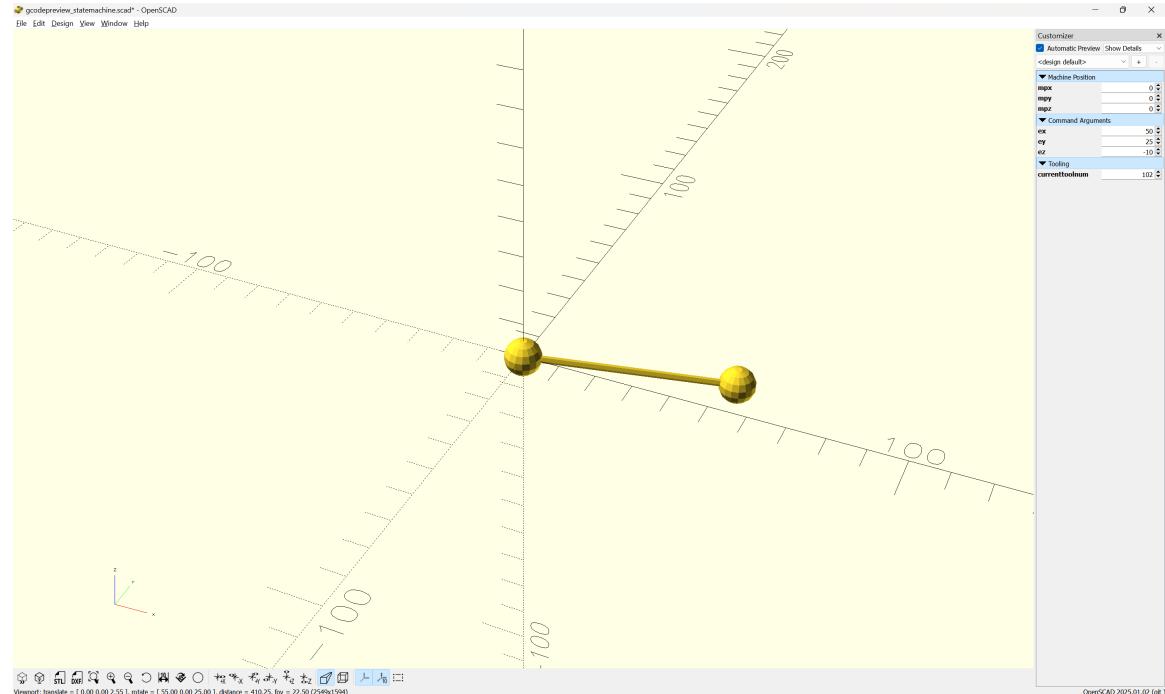
(note that the names are not case-sensitive)

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

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

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):



```

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

```

endmill square

**3.5.3.1.1 Square (including O-flute)**

The endmill square is a simple cylinder:

---

```

791 gcpy      if self.endmilltype == "square":
792 gcpy          ts = cylinder(r1=(self.diameter / 2), r2=(self.diameter
793 gcpy              / 2), h=self.flute, center = False)
794 gcpy          tslist.append(hull(ts.translate([bx, by, bz]), ts.
795 gcpy          translate([ex, ey, ez])))
796 gcpy #
797 gcpy #      if self.endmilltype == "O-flute":
798 gcpy #          ts = cylinder(r1=(self.diameter / 2), r2=(self.
799 gcpy #              diameter / 2), h=self.flute, center = False)
800 gcpy #          tslist.append(hull(ts.translate([bx, by, bz]), ts.
801 gcpy #              translate([ex, ey, ez])))
802 gcpy #
803 gcpy #
804 gcpy #
805 gcpy #
806 gcpy #
807 gcpy #
808 gcpy #

```

---

ballnose

**3.5.3.1.2 Ball nose (including tapered ball nose)**

The ballnose is modeled as a hemisphere

joined with a cylinder:

---

```

801 gcpy      if self.endmilltype == "ball":
802 gcpy          b = sphere(r=(self.diameter / 2))
803 gcpy          s = cylinder(r1=(self.diameter / 2), r2=(self.diameter
804 gcpy              / 2), h=self.flute, center=False)
805 gcpy          bs = union(b, s)
806 gcpy          bs = bs.translate([0, 0, (self.diameter / 2)])
807 gcpy          tslist.append(hull(bs.translate([bx, by, bz]), bs.
808 gcpy          translate([ex, ey, ez])))
809 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:

---

```

809 gcpy      if self.endmilltype == "bowl":
810 gcpy          inner = cylinder(r1 = self.diameter/2 - self.radius, r2
811 gcpy              = self.diameter/2 - self.radius, h = self.flute)
812 gcpy          outer = cylinder(r1 = self.diameter/2, r2 = self.
813 gcpy              diameter/2, h = self.flute - self.radius)
814 gcpy #      slices = hull(outer, inner)
815 gcpy          slices = cylinder(r1 = 0.0001, r2 = 0.0001, h = 0.0001, center
816 gcpy          =False)
817 gcpy          for i in range(1, 90 - self.steps, self.steps):
818 gcpy              slice = cylinder(r1 = self.diameter / 2 - self.
819 gcpy                  radius + self.radius * Sin(i), r2 = self.
820 gcpy                  diameter / 2 - self.radius + self.radius * Sin(i
821 gcpy                  +self.steps), h = self.radius/90, center=False)
822 gcpy          slices = hull(slices, slice.translate([0, 0, self.
823 gcpy                  radius - self.radius * Cos(i+self.steps)]))
824 gcpy          tslist.append(hull(slices.translate([bx, by, bz]),
825 gcpy                  slices.translate([ex, ey, ez])))
826 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)):

---

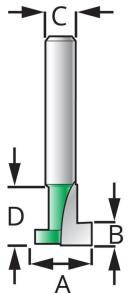
```

821 gcpy      if self.endmilltype == "V":
822 gcpy          v = cylinder(r1=0, r2=(self.diameter / 2), h=((self.
823 gcpy              diameter / 2) / Tan((self.angle / 2))), center=False
824 gcpy #
825 gcpy          s = cylinder(r1=(self.diameter / 2), r2=(self.
826 gcpy              diameter / 2), h=self.flute, center=False)

```

---

**3.5.3.1.5 Keyhole** Keyhole toolpaths (see: subsection 3.8.1.1.3) are intended for use with tooling which projects beyond the narrower shaft and so will cut usefully underneath the visible surface. Also described as “undercut” tooling, but see below.



**Keyhole Router Bits**

#	A	B	C	D
374	3/8"	1/8"	1/4"	3/8"
375	9.525mm	3.175mm	8mm	9.525mm
376	1/2"	3/16"	1/4"	1/2"
378	12.7mm	4.7625mm	8mm	12.7mm




---

```

828 gcpy      if self.endmilltype == "keyhole":
829 gcpy          kh = cylinder(r1=(self.diameter / 2), r2=(self.diameter
830 gcpy                  / 2), h=self.flute, center=False)
831 gcpy          sh = (cylinder(r1=(self.radius / 2), r2=(self.radius /
832 gcpy                  2), h=self.flute*2, center=False))
833 gcpy          tslist.append(hull(kh.translate([bx, by, bz]), kh.
834 gcpy                  translate([ex, ey, ez])))
835 gcpy          tslist.append(hull(sh.translate([bx, by, bz]), sh.
836 gcpy                  translate([ex, ey, ez])))
837 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.

---

```

835 gcpy      if self.endmilltype == "tapered_ball":
836 gcpy          b = sphere(r=(self.tip / 2))
837 gcpy          s = cylinder(r1=(self.tip / 2), r2=(self.diameter / 2),
838 gcpy                  h=self.flute, center=False)
839 gcpy          bshape = union(b, s)
840 gcpy          tslist.append(hull(bshape.translate([bx, by, bz]),
841 gcpy                  bshape.translate([ex, ey, ez])))
842 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)

---

```

842 gcpy      if self.endmilltype == "dovetail":
843 gcpy          dt = cylinder(r1=(self.diameter / 2), r2=(self.diameter
844 gcpy                  / 2) - self.flute * Tan(self.angle), h= self.flute,
845 gcpy                  center=False)
846 gcpy          tslist.append(hull(dt.translate([bx, by, bz]), dt.
847 gcpy                  translate([ex, ey, ez])))
848 gcpy      return tslist
849 gcpy      if self.endmilltype == "other":
850 gcpy          tslist = []
851 gcpy      def dovetail(self, dt_bottomdiameter, dt_topdiameter,
852 gcpy          dt_height, dt_angle):
853 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.

---

```

851 gcpy      if self.endmilltype == "roundover":
852 gcpy          shaft = cylinder(self.steps, self.tip/2, self.tip/2)
853 gcpy          toolpath = hull(shaft.translate([bx, by, bz]), shaft.
854 gcpy              translate([ex, ey, ez]))
855 gcpy          shaft = cylinder(self.flute, self.diameter/2 + self.tip
856 gcpy              /2, self.diameter/2 + self.tip/2)
857 gcpy          toolpath = toolpath.union(hull(shaft.translate([bx, by,
858 gcpy              bz + self.radius]), shaft.translate([ex, ey, ez +
859 gcpy                  self.radius])))
860 gcpy          tslist = [toolpath]
861 gcpy          slice = cylinder(0.0001, 0.0001, 0.0001)
862 gcpy          slices = slice
863 gcpy          for i in range(1, 90 - self.steps, self.steps):
864 gcpy              dx = self.radius*cos(i)
865 gcpy              dxx = self.radius*cos(i + self.steps)
866 gcpy              dzz = self.radius*sin(i)
867 gcpy              dz = self.radius*sin(i + self.steps)
868 gcpy              dh = dz - dzz
869 gcpy              slice = cylinder(r1 = self.tip/2+self.radius-dx, r2
870 gcpy                  = self.tip/2+self.radius-dxx, h = dh)
871 gcpy              slices = slices.union(hull(slice.translate([bx, by,
872 gcpy                  bz+dz]), slice.translate([ex, ey, ez+dz])))
873 gcpy          tslist.append(slices)
874 gcpy
875 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:

---

```

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

---

```

877 gcpy      def defineshaft(self, toolingdiameter, shaftdiameter, flute,
878 gcpy          transition, shaft):
879 gcpy          if shaftdiameter == 0:
880 gcpy              self.shaftoutline = polygon(points=[[0, flute], [
881 gcpy                  diameter, flute], [diameter, shaft],[0, shaft]])
882 gcpy              self.shaftprofile = polygon(points=[[0, flute], [
883 gcpy                  diameter/2 ,flute], [diameter/2, shaft], [0, shaft
884 gcpy                  ]])
885 gcpy              sh = cylinder(h = shaft, r = diameter/2)
886 gcpy              self.currenttoolshaft = sh.translate([0,0,flute])
887 gcpy          if shaftdiameter > 0:
888 gcpy              self.shaftoutline = polygon(points=[[
889 gcpy                  shaftdiameter / 2 - toolingdiameter / 2, flute],
890 gcpy                  [0, flute + transition],
891 gcpy                  [0, flute + transition + shaft],
892 gcpy                  [shaftdiameter, flute + transition + shaft],
893 gcpy                  [shaftdiameter, flute + transition],
894 gcpy                  [shaftdiameter / 2 + toolingdiameter / 2, flute],
895 gcpy                  ])
896 gcpy              self.shaftprofile = polygon( points= [
897 gcpy                  [0, flute],
898 gcpy                  [0, flute + transition + shaft],
899 gcpy                  [shaftdiameter/2, flute + transition + shaft],
900 gcpy                  [shaftdiameter/2, flute + transition],
901 gcpy                  [toolingdiameter/2, flute]
902 gcpy                  ])
903 gcpy          self.currenttoolshaft = rotate_extrude(self.
904 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.

---

```

901 gcpy      def definesquaretool(self, diameter, flute, shaft, offset = 0):
902 gcpy          self.tooloutline = polygon( points=[[0 + offset,0],[diameter + offset,0],[0 +
903 gcpy              diameter + offset,flute],[0 + offset,flute]] )
904 gcpy          self.toolprofile = polygon( points=[[0,0],[diameter/2,0],[diameter/2,flute],[0,flute]] )
905 gcpy          self.currenttoolshape = cylinder(h = flute, r = diameter/2)
906 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.

---

```

907 gcpy      def defineballnosetool(self, diameter, flute, shaft, offset = 0):
908 gcpy          s = square([diameter,flute - diameter/2])
909 gcpy          sh = s.translate([0 + offset, diameter/2])
910 gcpy          c = circle(d=diameter)
911 gcpy          b = c.translate([diameter/2 + offset, diameter/2])
912 gcpy          self.tooloutline = union(sh, b)
913 gcpy #
914 gcpy          s = square([diameter/2,flute - diameter/2])
915 gcpy          sh = s.translate([0, diameter/2])
916 gcpy          c = circle(d=diameter)
917 gcpy          b = c.translate([0, diameter/2])
918 gcpy          bn = union(sh, b)
919 gcpy #
920 gcpy          bns = bn.translate([0, diameter/2])
921 gcpy #
922 gcpy          thein = square([diameter/2,flute])
923 gcpy #
924 gcpy          theins = thein.translate([diameter/2, 0])
925 gcpy          self.toolprofile = intersection(thein, bn)
926 gcpy #
927 gcpy          self.shaftprofile = polygon( points=[[0,flute],[diameter/2,flute],[diameter/2,shaft],[0,shaft]] )
928 gcpy #
929 gcpy          b = self.toolprofile
930 gcpy          bn = b.translate([-diameter/2, 0])
931 gcpy          self.currenttoolshape = rotate_extrude(self.toolprofile)
932 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.

---

```

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

```

---

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

---

```

948 gcpy      def defineKeyholetool(self, diameter, flute, narrowdiameter,
949 gcpy          narrowflute, shaftdiameter, shaftlength):

```

---

```

950 gcpy      self.tooloutline = polygon([[0, 0], [diameter, 0], [
951 gcpy #      diameter, flute], [diameter/2 + narrowdiameter/2, flute
952 gcpy #      ], [diameter/2 + narrowdiameter/2, flute + narrowflute],
953 gcpy #      [diameter/2 - narrowdiameter/2, flute + narrowflute], [
954 gcpy #      diameter/2 - narrowdiameter/2, flute], [0, flute]])
955 gcpy      self.toolprofile = polygon([[0, 0], [diameter/2, 0], [
956 gcpy #      diameter/2, flute], [narrowdiameter/2, flute], [
957 gcpy #      narrowdiameter/2, flute + narrowflute], [0, flute +
958 gcpy #      narrowflute]])
959 gcpy #      self.currenttoolshape = rotate_extrude(self.toolprofile)
960 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.

```

962 gcpy      def definebowltool(self, diameter, flute, radius, shaftdiameter
963 gcpy #      , shaftlength):
964 gcpy #
965 gcpy      self.tooloutline =
966 gcpy #
967 gcpy      self.toolprofile = polygon([[0,0], [diameter/2, 0, radius],
968 gcpy #      [diameter/2, radius], [diameter/2, flute], [0, flute]])
969 gcpy      self.shaftprofile = polygon([[0,flute], [shaftdiameter/2,
970 gcpy #      flute], [shaftdiameter/2, flute + shaftlength], [0,
971 gcpy #      flute + shaftlength]])
972 gcpy      self.currenttoolshape = rotate_extrude(self.toolprofile)
973 gcpy      self.currenttoolshaft = rotate_extrude(self.shaftprofile)

```

---

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

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

```

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

```

---

995 gcpy	self.currenttoolshaft = rotate_extrude(self.shaftprofile)
----------	---

---

**rapid 3.5.3.5 rapid and cut (lines)** A matching pair of commands is made for these, and rapid is used as the basis for a series of commands which match typical usages of G0.

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

---

997 gcpy	<b>def</b> rapid(self, ex, ey, ez, laser = 0):
998 gcpy #	print(self.rapidcolor)
999 gcpy	<b>if</b> self.generateprint == True:
1000 gcpy	laser = 1
1001 gcpy	<b>if</b> laser == 0:
1002 gcpy	tm = self.toolmovement(self.xpos(), self.ypos(), self.zpos(), ex, ey, ez)
1003 gcpy	tm = color(tm, self.shaftcolor)
1004 gcpy	ts = self.shaftmovement(self.xpos(), self.ypos(), self.zpos(), ex, ey, ez)
1005 gcpy	ts = color(ts, self.rapidcolor)
1006 gcpy	self.toolpaths.extend([tm, ts])
1007 gcpy	self.setxpos(ex)
1008 gcpy	self.setypos(ey)
1009 gcpy	self.setzpos(ez)
1010 gcpy	
1011 gcpy	<b>def</b> cutline(self, ex, ey, ez):
1012 gcpy #	print(self.cutcolor)
1013 gcpy #	print(ex, ey, ez)
1014 gcpy	tm = self.toolmovement(self.xpos(), self.ypos(), self.zpos(), ex, ey, ez)
1015 gcpy	tm = color(tm, self.cutcolor)
1016 gcpy	ts = self.shaftmovement(self.xpos(), self.ypos(), self.zpos(), ex, ey, ez)
1017 gcpy	ts = color(ts, self.rapidcolor)
1018 gcpy	self.setxpos(ex)
1019 gcpy	self.setypos(ey)
1020 gcpy	self.setzpos(ez)
1021 gcpy	<b>if</b> self.generatecut == True:
1022 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:

---

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

---

```

1051 gcpy
1052 gcpy     def rapidXZ(self, ex, ez):
1053 gcpy         rapid = self.rapid(ex, self.ypos(), ez)
1054 gcpy #             if self.generatepaths == False:
1055 gcpy             return rapid
1056 gcpy
1057 gcpy     def rapidYZ(self, ey, ez):
1058 gcpy         rapid = self.rapid(self.xpos(), ey, ez)
1059 gcpy #             if self.generatepaths == False:
1060 gcpy             return rapid
1061 gcpy
1062 gcpy     def rapidX(self, ex):
1063 gcpy         rapid = self.rapid(ex, self.ypos(), self.zpos())
1064 gcpy #             if self.generatepaths == False:
1065 gcpy             return rapid
1066 gcpy
1067 gcpy     def rapidY(self, ey):
1068 gcpy         rapid = self.rapid(self.xpos(), ey, self.zpos())
1069 gcpy #             if self.generatepaths == False:
1070 gcpy             return rapid
1071 gcpy
1072 gcpy     def rapidZ(self, ez):
1073 gcpy         rapid = [self.rapid(self.xpos(), self.ypos(), ez)]
1074 gcpy #             if self.generatepaths == True:
1075 gcpy #                 self.rapids = self.rapids.union(rapid)
1076 gcpy #
1077 gcpy #             if self.generatepaths == False:
1078 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:

```

1080 gcpy     def moveatfeedrate(self, ex, ey, ez, f):
1081 gcpy         self.writegc("G01\u00D7X", str(ex), "\u00D7Y", str(ey), "\u00D7Z", str(ez)
1082 gcpy             , "\u00D7F", str(f))
1083 gcpy         return self.cutline(ex, ey, ez)
1084 gcpy
1085 gcpy     def cutlinedxf(self, ex, ey, ez):
1086 gcpy         self.dxfline(self.currenttoolnumber(), self.xpos(), self.
1087 gcpy             ypos(), ex, ey)
1088 gcpy         self.cutline(ex, ey, ez)
1089 gcpy
1090 gcpy     def cutlinedxfgc(self, ex, ey, ez):
1091 gcpy         self.dxfline(self.currenttoolnumber(), self.xpos(), self.
1092 gcpy             ypos(), ex, ey)
1093 gcpy         self.writegc("G01\u00D7X", str(ex), "\u00D7Y", str(ey), "\u00D7Z", str(ez)
1094 gcpy             )

```

---

```

1095 gcpy           self.writegc("G01\u21d3X", str(ex), "\u21d3Y", str(ey), "\u21d3Z", str(ez)
                           )
1096 gcpy           self.cutline(ex, ey, ez)
1097 gcpy
1098 gcpy   def cutlineXYZwithfeed(self, ex, ey, ez, feed):
1099 gcpy       return self.cutline(ex, ey, ez)
1100 gcpy
1101 gcpy   def cutlineXYZ(self, ex, ey, ez):
1102 gcpy       return self.cutline(ex, ey, ez)
1103 gcpy
1104 gcpy   def cutlineXYwithfeed(self, ex, ey, feed):
1105 gcpy       return self.cutline(ex, ey, self.zpos())
1106 gcpy
1107 gcpy   def cutlineXY(self, ex, ey):
1108 gcpy       return self.cutline(ex, ey, self.zpos())
1109 gcpy
1110 gcpy   def cutlineXZwithfeed(self, ex, ez, feed):
1111 gcpy       return self.cutline(ex, self.ypos(), ez)
1112 gcpy
1113 gcpy   def cutlineXZ(self, ex, ez):
1114 gcpy       return self.cutline(ex, self.ypos(), ez)
1115 gcpy
1116 gcpy   def cutlineXwithfeed(self, ex, feed):
1117 gcpy       return self.cutline(ex, self.ypos(), self.zpos())
1118 gcpy
1119 gcpy   def cutlineX(self, ex):
1120 gcpy       return self.cutline(ex, self.ypos(), self.zpos())
1121 gcpy
1122 gcpy   def cutlineYZ(self, ey, ez):
1123 gcpy       return self.cutline(self.xpos(), ey, ez)
1124 gcpy
1125 gcpy   def cutlineYwithfeed(self, ey, feed):
1126 gcpy       return self.cutline(self.xpos(), ey, self.zpos())
1127 gcpy
1128 gcpy   def cutlineY(self, ey):
1129 gcpy       return self.cutline(self.xpos(), ey, self.zpos())
1130 gcpy
1131 gcpy   def cutlineZgcfed(self, ez, feed):
1132 gcpy       self.writegc("G01\u21d3Z", str(ez), "F", str(feed))
1133 gcpy       return self.cutline(self.xpos(), self.ypos(), ez)
1134 gcpy
1135 gcpy   def cutlineZwithfeed(self, ez, feed):
1136 gcpy       return self.cutline(self.xpos(), self.ypos(), ez)
1137 gcpy
1138 gcpy   def cutlineZ(self, ez):
1139 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 cutlineZgcfed(ez, feed){
81 gpcscad     gcp.cutlineZgcfed(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:

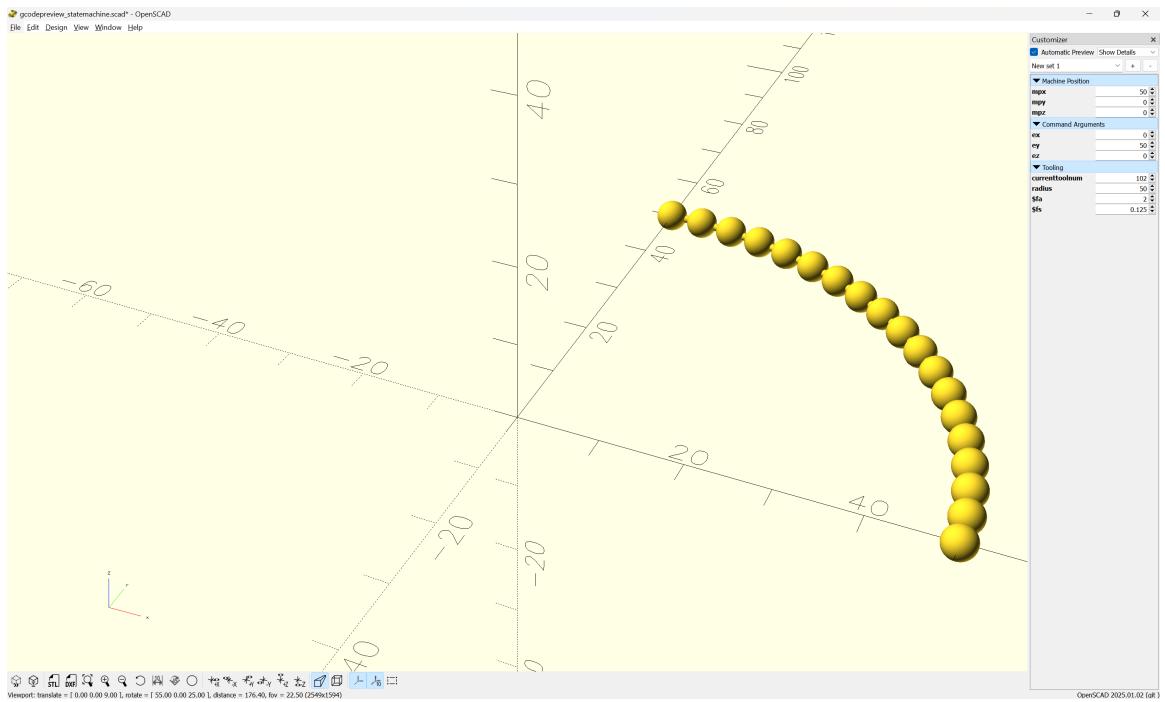
- `ex` — note that the matching origins (`bx, by, bz`) as well as the (current) toolnumber are accessed using the appropriate commands for machine position
- `ey`
- `ez` — allowing a different Z position will make possible threading and similar helical tool-paths
- `xcenter` — the center position will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

---

```

1141 gcpy      def cutarcCC(self, barc, earc, xcenter, ycenter, radius,
1142 gcpy          tpzreldim, stepsizearc=1):
1143 gcpy          tpzinc = tpzreldim / (earc - barc)
1144 gcpy          i = barc
1145 gcpy          while i < earc:
1146 gcpy              self.cutline(xcenter + radius * Cos(i), ycenter +
1147 gcpy                  radius * Sin(i), self.zpos() + tpzinc)
1148 gcpy          i += stepsizearc
1149 gcpy          self.setxpos(xcenter + radius * Cos(earc))
1150 gcpy          self.setypos(ycenter + radius * Sin(earc))
1151 gcpy      def cutarcCW(self, barc, earc, xcenter, ycenter, radius,
1152 gcpy          tpzreldim, stepsizearc=1):
1153 gcpy          print(str(self.zpos()))
1154 gcpy          print(str(ez))
1155 gcpy          print(str(barc - earc))
1156 gcpy          tpzinc = ez - self.zpos() / (barc - earc)
1157 gcpy          print(str(tpzinc))
1158 gcpy          global toolpath
1159 gcpy          print("Entering n toolpath")
1160 gcpy          tpzinc = tpzreldim / (barc - earc)
1161 gcpy          cts = self.currenttoolshape
1162 gcpy          toolpath = cts
1163 gcpy          toolpath = toolpath.translate([self.xpos(), self.ypos(),
1164 gcpy              self.zpos()])
1165 gcpy          toolpath = []
1166 gcpy          i = barc
1167 gcpy          while i > earc:
1168 gcpy              self.cutline(xcenter + radius * Cos(i), ycenter +
1169 gcpy                  radius * Sin(i), self.zpos() + tpzinc)
1170 gcpy              self.setxpos(xcenter + radius * Cos(i))
1171 gcpy              self.setypos(ycenter + radius * Sin(i))
1172 gcpy              print(str(self.xpos()), str(self.ypos()), str(self.zpos(
1173 gcpy                  )))
1174 gcpy              self.setzpos(self.zpos() + tpzinc)
1175 gcpy              i += abs(stepsizearc) * -1
1176 gcpy              self.dxfarcc(self.currenttoolnumber(), xcenter, ycenter,
1177 gcpy                  radius, barc, earc)
1178 gcpy              if self.generatepaths == True:
1179 gcpy                  print("Unioning n toolpath")
1180 gcpy                  self.toolpaths = self.toolpaths.union(toolpath)
1181 gcpy              else:
1182 gcpy                  return toolpath
1183 gcpy          else:
1184 gcpy              return cube([0.01, 0.01, 0.01])

```

---

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

---

```

1184 gcpy      def extrudearcCC(self, barc, earc, xcenter, ycenter, radius,
1185 gcpy          tpzreldim, stepsizearc=1):
1186 gcpy          tm = self.toolmovement(self.xpos(), self.ypos(), self.zpos(
1187 gcpy              ), ex, ey, ez)
1188 gcpy          tm = union(self.toolshape.translate(self.xpos(), self.ypos(
1189 gcpy              ), self.zpos()),
1190 gcpy                  self.toolshape.translate(),
1191 gcpy                  tooloutline.translate([r-3.175, 0, 0]),
1192 gcpy                  rotate_extrude(angle=ang2-ang1).rotz(ang1) + G3_center

```

---

```

1189 gcpy
1190 gcpy      tm = color(tm, self.cutcolor)
1191 gcpy      ts = self.shaftmovement(self.xpos(), self.ypos(), self.zpos
1192 gcpy          (), ex, ey, ez)
1193 gcpy      ts = color(ts, self.rapidcolor)
1194 gcpy      self.setxpos(ex)
1195 gcpy      self.setypos(ey)
1196 gcpy      self.setzpos(ez)
1196 gcpy      self.toolpaths.extend([tm, ts])

```

---

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

```

1198 gcpy      def cutarcCWdxf(self, barc, earc, xcenter, ycenter, radius,
1199 gcpy          tpzreldim, stepsizearc=1):
1200 gcpy          self.cutarcCW(barc, earc, xcenter, ycenter, radius,
1201 gcpy          tpzreldim, stepsizearc=1)
1202 gcpy          self.dxfarc(self.currenttoolnumber(), xcenter, ycenter,
1203 gcpy          radius, earc, barc)
1204 gcpy      if self.generatepaths == False:
1205 gcpy      return toolpath
1206 gcpy      else:
1207 gcpy          return cube([0.01, 0.01, 0.01])
1208 gcpy      def cutarcCCdxf(self, barc, earc, xcenter, ycenter, radius,
1209 gcpy          tpzreldim, stepsizearc=1):
1210 gcpy          self.cutarcCC(barc, earc, xcenter, ycenter, radius,
1211 gcpy          tpzreldim, stepsizearc=1)
1212 gcpy          self.dxfarc(self.currenttoolnumber(), xcenter, ycenter,
1213 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:

```

1210 gcpy      def cutquarterCCNE(self, ex, ey, ez, radius):
1211 gcpy          if self.zpos() == ez:
1212 gcpy              tpzinc = 0
1213 gcpy          else:
1214 gcpy              tpzinc = (ez - self.zpos()) / 90
1215 gcpy      print("tpzinc ", tpzinc)
1216 gcpy      i = 1
1217 gcpy      while i < 91:
1218 gcpy          self.cutline(ex + radius * Cos(i), ey - radius + radius
1219 gcpy          * Sin(i), self.zpos()+tpzinc)
1220 gcpy          i += 1
1221 gcpy      def cutquarterCCNW(self, ex, ey, ez, radius):
1222 gcpy          if self.zpos() == ez:
1223 gcpy              tpzinc = 0
1224 gcpy          else:
1225 gcpy              tpzinc = (ez - self.zpos()) / 90
1226 gcpy      tpzinc = (self.zpos() + ez) / 90
1227 gcpy      self.debug("tpzinc", tpzinc)
1228 gcpy      i = 91
1229 gcpy      while i < 181:
1230 gcpy          self.cutline(ex + radius + radius * Cos(i), ey + radius
1231 gcpy          * Sin(i), self.zpos()+tpzinc)
1232 gcpy          i += 1
1233 gcpy      def cutquarterCCSW(self, ex, ey, ez, radius):
1234 gcpy          if self.zpos() == ez:
1235 gcpy              tpzinc = 0
1236 gcpy          else:
1237 gcpy              tpzinc = (ez - self.zpos()) / 90
1238 gcpy      tpzinc = (self.zpos() + ez) / 90
1239 gcpy      print("tpzinc ", tpzinc)
1240 gcpy      i = 181
1241 gcpy      while i < 271:

```

```

1242 gcpy           self.cutline(ex + radius * Cos(i), ey + radius + radius
1243 gcpy           * Sin(i), self.zpos() + tpzinc)
1244 gcpy           i += 1
1245 gcpy       def cutquarterCCSE(self, ex, ey, ez, radius):
1246 gcpy           if self.zpos() == ez:
1247 gcpy               tpzinc = 0
1248 gcpy           else:
1249 gcpy               tpzinc = (ez - self.zpos()) / 90
1250 gcpy           tpzinc = (self.zpos() + ez) / 90
1251 gcpy           print("tpzinc ", tpzinc)
1252 gcpy           i = 271
1253 gcpy           while i < 361:
1254 gcpy               self.cutline(ex - radius + radius * Cos(i), ey + radius
1255 gcpy               * Sin(i), self.zpos() + tpzinc)
1256 gcpy               i += 1
1257 gcpy       def cutquarterCCNEdxf(self, ex, ey, ez, radius):
1258 gcpy           self.cutquarterCCNE(ex, ey, ez, radius)
1259 gcpy           self.dxfarc(self.currenttoolnumber(), ex, ey - radius,
1260 gcpy           radius, 0, 90)
1261 gcpy       def cutquarterCCNWdxr(self, ex, ey, ez, radius):
1262 gcpy           self.cutquarterCCNW(ex, ey, ez, radius)
1263 gcpy           self.dxfarc(self.currenttoolnumber(), ex + radius, ey,
1264 gcpy           radius, 90, 180)
1265 gcpy       def cutquarterCCSWdxr(self, ex, ey, ez, radius):
1266 gcpy           self.cutquarterCCSW(ex, ey, ez, radius)
1267 gcpy           self.dxfarc(self.currenttoolnumber(), ex, ey + radius,
1268 gcpy           radius, 180, 270)
1269 gcpy       def cutquarterCCSEdxr(self, ex, ey, ez, radius):
1270 gcpy           self.cutquarterCCSE(ex, ey, ez, radius)
1271 gcpy           self.dxfarc(self.currenttoolnumber(), ex - radius, ey,
1272 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:

```
1273 gcpy      def tool_diameter(self, ptd_tool, ptd_depth):
1274 gcpy # Square 122, 112, 102, 201
1275 gcpy      if ptd_tool == 122:
1276 gcpy          return 0.79375
1277 gcpy      if ptd_tool == 112:
1278 gcpy          return 1.5875
1279 gcpy      if ptd_tool == 102:
1280 gcpy          return 3.175
1281 gcpy      if ptd_tool == 201:
1282 gcpy          return 6.35
1283 gcpy # Ball 121, 111, 101, 202
1284 gcpy      if ptd_tool == 122:
1285 gcpy          if ptd_depth > 0.396875:
1286 gcpy              return 0.79375
1287 gcpy          else:
1288 gcpy              return ptd_tool
1289 gcpy      if ptd_tool == 112:
1290 gcpy          if ptd_depth > 0.79375:
1291 gcpy              return 1.5875
1292 gcpy          else:
1293 gcpy              return ptd_tool
1294 gcpy      if ptd_tool == 101:
1295 gcpy          if ptd_depth > 1.5875:
1296 gcpy              return 3.175
1297 gcpy          else:
1298 gcpy              return ptd_tool
1299 gcpy      if ptd_tool == 202:
1300 gcpy          if ptd_depth > 3.175:
1301 gcpy              return 6.35
1302 gcpy          else:
1303 gcpy              return ptd_tool
1304 gcpy # V 301, 302, 390
1305 gcpy      if ptd_tool == 301:
1306 gcpy          return ptd_tool
1307 gcpy      if ptd_tool == 302:
1308 gcpy          return ptd_tool
1309 gcpy      if ptd_tool == 390:
1310 gcpy          return ptd_tool
1311 gcpy # Keyhole
1312 gcpy      if ptd_tool == 374:
1313 gcpy          if ptd_depth < 3.175:
1314 gcpy              return 9.525
1315 gcpy          else:
1316 gcpy              return 6.35
1317 gcpy      if ptd_tool == 375:
1318 gcpy          if ptd_depth < 3.175:
1319 gcpy              return 9.525
1320 gcpy          else:
1321 gcpy              return 8
1322 gcpy      if ptd_tool == 376:
1323 gcpy          if ptd_depth < 4.7625:
1324 gcpy              return 12.7
1325 gcpy          else:
1326 gcpy              return 6.35
1327 gcpy      if ptd_tool == 378:
1328 gcpy          if ptd_depth < 4.7625:
1329 gcpy              return 12.7
1330 gcpy          else:
1331 gcpy              return 8
1332 gcpy # Dovetail
1333 gcpy      if ptd_tool == 814:
1334 gcpy          if ptd_depth > 12.7:
1335 gcpy              return 6.35
1336 gcpy          else:
1337 gcpy              return ptd_tool
1338 gcpy      if ptd_tool == 808079:
```

```

1339 gcpy           if ptd_depth > 20.95:
1340 gcpy           return 6.816
1341 gcpy           else:
1342 gcpy           return ptd_tool
1343 gcpy # Bowl Bit
1344 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
1345 gcpy           if ptd_tool == 45982:
1346 gcpy           if ptd_depth > 6.35:
1347 gcpy           return 15.875
1348 gcpy           else:
1349 gcpy           return ptd_tool
1350 gcpy # Tapered Ball Nose
1351 gcpy           if ptd_tool == 204:
1352 gcpy           if ptd_depth > 6.35:
1353 gcpy           return ptd_tool
1354 gcpy           if ptd_tool == 304:
1355 gcpy           if ptd_depth > 6.35:
1356 gcpy           return ptd_tool
1357 gcpy           else:
1358 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:

```

1360 gcpy     def tool_radius(self, ptd_tool, ptd_depth):
1361 gcpy         tr = self.tool_diameter(ptd_tool, ptd_depth)/2
1362 gcpy         return tr

```

---

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

### 3.5.5 Feeds and Speeds

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

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

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

### 3.5.6 3D Printing

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

- fan(s) on/off
- extruder(s)
- Heater(s)
- temperature(s)
- accelerometers
- load cells
- Filament Sensor(s)
- Filament Cutter(s)
- Display Status

Message

Build Percentage

(Clear) Message

- any additional commands such as “Clean Nozzle”

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

### 3.5.6.1 Sample 3D printing file

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

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

M106 P3 S255
;Prevent PLA from jamming

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

M204 S5000 ;Call exterior wall print acceleration

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

G3 I-1 J0 Z0.6 F1200.0 ;Move to side a little
M73 P27 R0
G1 F20000
G92 E0 ;Reset Extruder
;LAYER_COUNT:1
;LAYER:0
```

```
G90
G21
M83 ; use relative distances for extrusion
; filament start gcode
M106 P3 S200

;LAYER_CHANGE
;Z:0.2
;HEIGHT:0.2
;BEFORE_LAYER_CHANGE
;0.2
G92 EO

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.

---

```

1364 gcpy     def setfansoff(self):
1365 gcpy         writegc("M106\u00d7S0")
1366 gcpy
1367 gcpy     def setfanspeed(self, fan, speed):
1368 gcpy         writegc("M106\u00d7P", fan, "\u00d7S", speed)
1369 gcpy
1370 gcpy     def pauseforclearbuffer(self):
1371 gcpy         writegc("M400\u00d7;uwait\u00d7for\u00d7buffer\u00d7to\u00d7clear")

```

---

**3.5.6.2 Feed and Speed ratio** Note that certain commands will require setting values which will need to be tracked and used for calculations.

---

```

1373 gcpy     def setfeedratio(self, feedratio):
1374 gcpy         writegc("M220\u00d7S", feedratio)
1375 gcpy         self.feedratio = feedratio
1376 gcpy
1377 gcpy     def setspeedratio(self, speedratio):
1378 gcpy         writegc("M221\u00d7S", speedratio)
1379 gcpy         self.speedratio = speedratio

```

---

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

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

---

```

1381 gcpy #Set extruder temperature: M104 [T<index>] [S<temperature>]
1382 gcpy     def setextrudertemperature(self, temperature):
1383 gcpy         writegc("M104\u00d7S", temperature)
1384 gcpy         self.extrudertemperature = temperature
1385 gcpy
1386 gcpy #Set bed temperature: M140 [S<temperature>]
1387 gcpy     def setbedtemperature(self, temperature):
1388 gcpy         writegc("M140\u00d7S", temperature)
1389 gcpy         self.bedtemperature = temperature
1390 gcpy
1391 gcpy #Set bed temperature and wait: M190 S<temperature>
1392 gcpy #Note: M190 always waits for temperature to settle at requested
           value
1393 gcpy     def setandwaitforbedtemperature(self, temperature):
1394 gcpy         writegc("M190\u00d7S", temperature)
1395 gcpy         self.bedtemperature = temperature

```

---

```

1396 gcpy
1397 gcpy #Set extruder temperature and wait: M109 [T<index>] S<temperature>
1398 gcpy #Note: M109 always waits for temperature to settle at requested
           value
1399 gcpy     def setandwaitforbedtemperature(self, temperature):
1400 gcpy         writegc("M190\u2022S", temperature)
1401 gcpy         self.bedtemperature = temperature

```

---

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

---

```

1403 gcpy     def initializeforprinting(self, nozzlediameter = 0.4,
           filamentdiameter = 1.75, extrusionwidth = 0.6, layerheight =
           0.2):
1404 gcpy         self.writegc("G90")
1405 gcpy         self.writegc("G28\u2022;home")
1406 gcpy         self.writegc("M729\u2022;Clean\u2022Nozzle")
1407 gcpy         self.nozzlediameter = nozzlediameter
1408 gcpy         self.extrusionwidth = extrusionwidth
1409 gcpy         self.layerheight = layerheight
1410 gcpy         self.toolpaths = []
1411 gcpy         self.feedrate = 0
1412 gcpy         fr = filamentdiameter/2
1413 gcpy         self.extrusion_normal_length = 1 / 3.14159 * (fr * fr)

```

---



---

```

1415 gcpy #Set acceleration: M204 S<value> OR M204 P<value> T<value>
1416 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.
1417 gcpy     def setacceleration(self, acceleration):
1418 gcpy         writegc("M204\u2022S", acceleration)
1419 gcpy         self.acceleration = acceleration
1420 gcpy
1421 gcpy #Use absolute/relative distances for extrusion: M82, M83
1422 gcpy     def setextrusionabsolute(self, acceleration):
1423 gcpy         writegc("M83")
1424 gcpy         self.extrusionabsolute = true

```

---



---

```

1426 gcpy #Set build percentage: M73 P<percent>
1427 gcpy     def setbuildpercentage(self, percent):
1428 gcpy         writegc("M73\u2022P", percent)
1429 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 / toolchan

```

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

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

```

; --- Start of G-code: Demonstration of Layer and Extrusion Concepts ---
G21 ; Set units to millimeters
G90 ; Use absolute positioning
M82 ; Set extruder to absolute mode
M104 S200 ; Set extruder temperature to 200\u00b0C

```

```

M140 S60 ; Set bed temperature to 60°C
M190 S60 ; Wait for bed to reach target temp
M109 S200 ; Wait for extruder to reach target temp
G28 ; Home all axes

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

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

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

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

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

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

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

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

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

```

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.

---

```

1431 gcpy #Move (G0 or G1): G1 [X<pos>] [Y<pos>] [Z<pos>] [E<pos>] [F<speed>]
1432 gcpy     def extrude(self, ex, ey, ez, extrusionwidth = 0, layerheight =
               0, feedrate = 0):
1433 gcpy         if extrusionwidth > 0:
1434 gcpy             self.extrusionwidth = extrusionwidth
1435 gcpy         if layerheight > 0:
1436 gcpy             self.layerheight = layerheight
1437 gcpy         if feedrate > 0:
1438 gcpy             self.feedrate = feedrate
1439 gcpy         if self.extrusionwidth == self.layerheight:
1440 gcpy             c = sphere(self.layerheight/2)
1441 gcpy         else:
1442 gcpy             ew = self.extrusionwidth
1443 gcpy             lh = self.layerheight
1444 gcpy             i = circle(lh/2)
1445 gcpy             j = i.translate([0, lh/2, 0])
1446 gcpy             k = intersection(j, square([lh, lh]))
1447 gcpy             l = k.translate([ew/2-lh/2, 0, 0])
1448 gcpy             m = union(l, square([ew/2-lh/2, lh]))
1449 gcpy             c = rotate_extrude(m)
1450 gcpy             c = c.translate([0, 0, -self.layerheight])
1451 gcpy             tslist = hull(c.translate([self.xpos(), self.ypos(), self.
                           zpos()]), c.translate([ex, ey, ez]))
1452 gcpy             self.toolpaths.append(tslist)
1453 gcpy             #volume = π r^2 * length
1454 gcpy             #           + extrusionwidth-layerheight * layerheight * length
1455 gcpy             distance = math.dist([self.xpos(), self.ypos(), self.zpos()]
                           , [ex, ey, ez])
1456 gcpy             print("Distance = ", distance)
1457 gcpy             v = self.extrusionwidth-self.layerheight * self.layerheight
                           * distance + 3.14159 * self.layerheight/2 * self.
                           layerheight/2 * distance
1458 gcpy             print("Volume = ", v)
1459 gcpy             el = self.extrusion_normal_length * v
1460 gcpy             print("Extrusion length = ", el)
1461 gcpy             self.writegc("G01 X"+str(ex)+" Y"+str(ey)+" Z"+str(ez)+ " F"+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

```

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

At the end of cutting it will be necessary to subtract the accumulated toolpaths and rapids from the stock.

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

---

```

1463 gcpy      def stockandtoolpaths(self, option = "stockandtoolpaths"):
1464 gcpy          if option == "stock":
1465 gcpy              show(self.stock)
1466 gcpy          elif option == "toolpaths":
1467 gcpy              show(self.toolpaths)
1468 gcpy          elif option == "rapids":
1469 gcpy              show(self.rapids)
1470 gcpy      else:
1471 gcpy          part = self.stock.difference(self.rapids)
1472 gcpy          part = self.stock.difference(self.toolpaths)
1473 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.

---

```

1475 gcpy      def showtooloutline(self):
1476 gcpy          to = union(self.tooloutline, self.shaftoutline)
1477 gcpy          show(to)
1478 gcpy
1479 gcpy      def showtoolprofile(self):
1480 gcpy          to = union(self.toolprofile, self.shaftprofile)
1481 gcpy          show(to)
1482 gcpy
1483 gcpy      def showtoolshape(self):
1484 gcpy          to = union(self.currenttoolshape, self.currenttoolshaft)
1485 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:

---

```

1487 gcpy      def returnstockandtoolpaths(self):
1488 gcpy          part = self.stock.difference(self.toolpaths)
1489 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.

### 3.7 Output files

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

#### 3.7.1 Python and OpenSCAD File Handling

The class `gcodepreview` will need additional commands for opening files. The original implementation in RapSCAD used a command `writeln` — fortunately, this command is easily re-created in Python, though it is made as a separate file for each sort of file which may be opened. Note that

the dxfs commands will be wrapped up with if/elif blocks which will write to additional file(s) based on tool number as set up above.

---

```

1491 gcpy     def writegc(self, *arguments):
1492 gcpy         if self.generategcode == True:
1493 gcpy             line_to_write = ""
1494 gcpy             for element in arguments:
1495 gcpy                 line_to_write += element
1496 gcpy                 self.gc.write(line_to_write)
1497 gcpy                 self.gc.write("\n")
1498 gcpy
1499 gcpy     def writedxf(self, toolnumber, *arguments):
1500 gcpy #         global dxfclosed
1501 gcpy         line_to_write = ""
1502 gcpy         for element in arguments:
1503 gcpy             line_to_write += element
1504 gcpy         if self.generateddxf == True:
1505 gcpy             if self.dxfclosed == False:
1506 gcpy                 self.dxf.write(line_to_write)
1507 gcpy                 self.dxf.write("\n")
1508 gcpy         if self.generateddxfs == True:
1509 gcpy             self.writedxfs(toolnumber, line_to_write)
1510 gcpy
1511 gcpy     def writedxfs(self, toolnumber, line_to_write):
1512 gcpy #         print("Processing writing toolnumber", toolnumber)
1513 gcpy #         line_to_write = ""
1514 gcpy #
1515 gcpy #
1516 gcpy
1517 gcpy
1518 gcpy
1519 gcpy     elif self.generateddxfs == True:
1520 gcpy         if (self.large_square_tool_num == toolnumber):
1521 gcpy             self.dxflgsq.write(line_to_write)
1522 gcpy             self.dxflgsq.write("\n")
1523 gcpy         if (self.small_square_tool_num == toolnumber):
1524 gcpy             self.dxfsmssq.write(line_to_write)
1525 gcpy             self.dxfsmssq.write("\n")
1526 gcpy         if (self.large_ball_tool_num == toolnumber):
1527 gcpy             self.dxflgbl.write(line_to_write)
1528 gcpy             self.dxflgbl.write("\n")
1529 gcpy         if (self.small_ball_tool_num == toolnumber):
1530 gcpy             self.dxfsmbl.write(line_to_write)
1531 gcpy             self.dxfsmbl.write("\n")
1532 gcpy         if (self.large_V_tool_num == toolnumber):
1533 gcpy             self.dxflgV.write(line_to_write)
1534 gcpy             self.dxflgV.write("\n")
1535 gcpy         if (self.small_V_tool_num == toolnumber):
1536 gcpy             self.dxfsmV.write(line_to_write)
1537 gcpy             self.dxfsmV.write("\n")
1538 gcpy         if (self.DT_tool_num == toolnumber):
1539 gcpy             self.dxfDT.write(line_to_write)
1540 gcpy             self.dxfDT.write("\n")
1541 gcpy         if (self.KH_tool_num == toolnumber):
1542 gcpy             self.dxfKH.write(line_to_write)
1543 gcpy             self.dxfKH.write("\n")
1544 gcpy         if (self.Roundover_tool_num == toolnumber):
1545 gcpy             self.dxfRt.write(line_to_write)
1546 gcpy             self.dxfRt.write("\n")
1547 gcpy         if (self.MISC_tool_num == toolnumber):
1548 gcpy             self.dxfMt.write(line_to_write)

```

---

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

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

---

```

130 gpcscad module opendxffile(basefilename){
131 gpcscad     gcp.opendxffile(basefilename);
132 gpcscad }

```

---

```

133 gpcscad
134 gpcscad module opendxffiles(Base_filename, large_square_tool_num,
        small_square_tool_num, large_ball_tool_num, small_ball_tool_num,
        large_V_tool_num, small_V_tool_num, DT_tool_num, KH_tool_num,
        Roundover_tool_num, MISC_tool_num) {
135 gpcscad     gcp.opendxffiles(Base_filename, large_square_tool_num,
        small_square_tool_num, large_ball_tool_num,
        small_ball_tool_num, large_V_tool_num, small_V_tool_num,
        DT_tool_num, KH_tool_num, Roundover_tool_num, MISC_tool_num)
        ;
136 gpcscad }
```

---

opengcodefile    With matching OpenSCAD commands: opengcodefile for OpenSCAD:

```

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

---

and Python:

```

1550 gcpy     def opengcodefile(self, basefilename = "export",
1551 gcpy         currenttoolnum = 102,
1552 gcpy         toolradius = 3.175,
1553 gcpy         plunge = 400,
1554 gcpy         feed = 1600,
1555 gcpy         speed = 10000
1556 gcpy         ):
1557 gcpy             self.basefilename = basefilename
1558 gcpy             self.currenttoolnum = currenttoolnum
1559 gcpy             self.toolradius = toolradius
1560 gcpy             self.plunge = plunge
1561 gcpy             self.feed = feed
1562 gcpy             self.speed = speed
1563 gcpy             if self.generategcode == True:
1564 gcpy                 self.gcodename = basefilename + self.gcodefilext
1565 gcpy                 self.gc = open(self.gcodename, "w")
1566 gcpy                 self.writegc("(DesignFile:" + self.basefilename + ")"
1567 gcpy
1568 gcpy         def opendxffile(self, basefilename = "export"):
1569 gcpy             self.basefilename = basefilename
1570 gcpy #             global generateddxfs
1571 gcpy #             global dxfclosed
1572 gcpy             self.dxfclosed = False
1573 gcpy             self.dxfcolor = "Black"
1574 gcpy             if self.generatedxf == True:
1575 gcpy                 self.generatedxfs = False
1576 gcpy                 self.dxffilename = basefilename + ".dxf"
1577 gcpy                 self.dxf = open(self.dxffilename, "w")
1578 gcpy                 self.dxfpreamble(-1)
1579 gcpy
1580 gcpy         def opendxffiles(self, basefilename = "export",
1581 gcpy             large_square_tool_num = 0,
1582 gcpy             small_square_tool_num = 0,
1583 gcpy             large_ball_tool_num = 0,
1584 gcpy             small_ball_tool_num = 0,
1585 gcpy             large_V_tool_num = 0,
1586 gcpy             small_V_tool_num = 0,
1587 gcpy             DT_tool_num = 0,
1588 gcpy             KH_tool_num = 0,
1589 gcpy             Roundover_tool_num = 0,
1590 gcpy             MISC_tool_num = 0):
1591 gcpy #
1592 gcpy             global generateddxfs
1593 gcpy             self.basefilename = basefilename
1594 gcpy             self.generatedxfs = True
1595 gcpy             self.large_square_tool_num = large_square_tool_num
1596 gcpy             self.small_square_tool_num = small_square_tool_num
1597 gcpy             self.large_ball_tool_num = large_ball_tool_num
1598 gcpy             self.small_ball_tool_num = small_ball_tool_num
1599 gcpy             self.large_V_tool_num = large_V_tool_num
1600 gcpy             self.small_V_tool_num = small_V_tool_num
1601 gcpy             self.DT_tool_num = DT_tool_num
1602 gcpy             self.KH_tool_num = KH_tool_num
1603 gcpy             self.Roundover_tool_num = Roundover_tool_num
1604 gcpy             self.MISC_tool_num = MISC_tool_num
```

```

1604 gcpy           if self.generateddxf == True:
1605 gcpy           if (large_square_tool_num > 0):
1606 gcpy             self.dxfLgSqfilename = basefilename + str(
1607 gcpy               large_square_tool_num) + ".dxf"
1608 gcpy               print("Opening ", str(self.dxfLgSqfilename))
1609 gcpy               self.dxfLgSq = open(self.dxfLgSqfilename, "w")
1610 gcpy               if (small_square_tool_num > 0):
1611 gcpy                 print("Opening small square")
1612 gcpy                 self.dxfSmSqfilename = basefilename + str(
1613 gcpy                   small_square_tool_num) + ".dxf"
1614 gcpy                   self.dxfSmSq = open(self.dxfSmSqfilename, "w")
1615 gcpy               if (large_ball_tool_num > 0):
1616 gcpy                 print("Opening large ball")
1617 gcpy                 self.dxfLgBlfilename = basefilename + str(
1618 gcpy                   large_ball_tool_num) + ".dxf"
1619 gcpy                   self.dxfLgBl = open(self.dxfLgBlfilename, "w")
1620 gcpy               if (small_ball_tool_num > 0):
1621 gcpy                 print("Opening small ball")
1622 gcpy                 self.dxfSmBlfilename = basefilename + str(
1623 gcpy                   small_ball_tool_num) + ".dxf"
1624 gcpy                   self.dxfSmBl = open(self.dxfSmBlfilename, "w")
1625 gcpy               if (large_V_tool_num > 0):
1626 gcpy                 print("Opening large V")
1627 gcpy                 self.dxfLgVfilename = basefilename + str(
1628 gcpy                   large_V_tool_num) + ".dxf"
1629 gcpy                   self.dxfLgV = open(self.dxfLgVfilename, "w")
1630 gcpy               if (small_V_tool_num > 0):
1631 gcpy                 print("Opening small V")
1632 gcpy                 self.dxfSmVfilename = basefilename + str(
1633 gcpy                   small_V_tool_num) + ".dxf"
1634 gcpy                   self.dxfSmV = open(self.dxfSmVfilename, "w")
1635 gcpy               if (DT_tool_num > 0):
1636 gcpy                 print("Opening DT")
1637 gcpy                 self.dxfDTfilename = basefilename + str(DT_tool_num)
1638 gcpy                   ) + ".dxf"
1639 gcpy                   self.dxfDT = open(self.dxfDTfilename, "w")
1640 gcpy               if (KH_tool_num > 0):
1641 gcpy                 print("Opening KH")
1642 gcpy                 self.dxfKHfilename = basefilename + str(KH_tool_num)
1643 gcpy                   ) + ".dxf"
1644 gcpy                   self.dxfKH = open(self.dxfKHfilename, "w")

```

---

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

```

1645 gcpy           if (large_square_tool_num > 0):
1646 gcpy             self.dxfPreamble(large_square_tool_num)
1647 gcpy             if (small_square_tool_num > 0):
1648 gcpy               self.dxfPreamble(small_square_tool_num)
1649 gcpy             if (large_ball_tool_num > 0):
1650 gcpy               self.dxfPreamble(large_ball_tool_num)
1651 gcpy               if (small_ball_tool_num > 0):
1652 gcpy                 self.dxfPreamble(small_ball_tool_num)
1653 gcpy               if (large_V_tool_num > 0):
1654 gcpy                 self.dxfPreamble(large_V_tool_num)
1655 gcpy                 if (small_V_tool_num > 0):
1656 gcpy                   self.dxfPreamble(small_V_tool_num)
1657 gcpy                 if (DT_tool_num > 0):
1658 gcpy                   self.dxfPreamble(DT_tool_num)
1659 gcpy                   if (KH_tool_num > 0):
1660 gcpy                     self.dxfPreamble(KH_tool_num)
1661 gcpy                     if (Roundover_tool_num > 0):
1662 gcpy                       self.dxfPreamble(Roundover_tool_num)
1663 gcpy                     if (MISC_tool_num > 0):
1664 gcpy                       self.dxfPreamble(MISC_tool_num)

```

---

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

generated.

Future considerations:

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

### 3.7.2 DXF Overview

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

```

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

```

**3.7.2.1 Writing to DXF files** When the command to open .dxf files is called it is passed all of the variables for the various tool types/sizes, and based on a value being greater than zero, the matching file is opened, and in addition, the main DXF which is always written to is opened as well. On the gripping hand, each element which may be written to a DXF file will have a user module as well as an internal module which will be called by it so as to write to the file for the `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:

- |                            |  |
|----------------------------|--|
| <code>writedxflbl</code>   | • Ball nose, large (lbl) <code>writedxflbl</code>  |
| <code>writedxfsmbl</code>  | • Ball nose, small (smb) <code>writedxfsmbl</code> |
| <code>writedxfllgsq</code> | • Square, large (lgsq) <code>writedxfllgsq</code>  |
| <code>writedxfsmssq</code> | • Square, small (smsq) <code>writedxfsmssq</code>  |
| <code>writedxflgV</code>   | • V, large (lgV) <code>writedxflgV</code>          |
| <code>writedxfsmV</code>   | • V, small (smV) <code>writedxfsmV</code>          |
| <code>writedxfKH</code>    | • Keyhole (KH) <code>writedxfKH</code>             |

- 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.

---

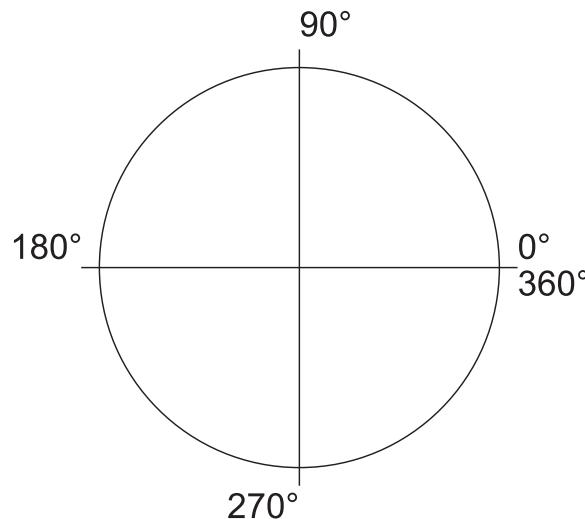
```
1666 gcpy     def writedxf(self, tn):
1667 gcpy #         self.writedxf(tn, str(tn))
1668 gcpy         self.writedxf(tn, "0")
1669 gcpy         self.writedxf(tn, "SECTION")
1670 gcpy         self.writedxf(tn, "2")
1671 gcpy         self.writedxf(tn, "ENTITIES")
```

---

### 3.7.2.1.1 DXF Lines and Arcs

There are several elements which may be written to a DXF:

- dxfline • a line dxfline
- beginpolyline • connected lines beginpolyline/addvertex/closepolyline
- closepolyline • arc dxfarcs
- dxfarcs • 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:

```
dfarc(10, 10, 5, 0, 90, small_square_tool_num);
dfarc(10, 10, 5, 90, 180, small_square_tool_num);
dfarc(10, 10, 5, 180, 270, small_square_tool_num);
dfarc(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:

---

```
1673 gcpy     def setdxfcolor(self, color):
1674 gcpy         self.dxfcolor = color
1675 gcpy         self.cutcolor = color
1676 gcpy
1677 gcpy     def writedxfcolor(self, tn):
1678 gcpy         self.writedxf(tn, "8")
1679 gcpy         if (self.dxfcolor == "Black"):
```

```

1680 gcpy           self.writedxf(tn, "Layer_Black")
1681 gcpy           if (self.dxfcolor == "Red"):
1682 gcpy             self.writedxf(tn, "Layer_Red")
1683 gcpy           if (self.dxfcolor == "Yellow"):
1684 gcpy             self.writedxf(tn, "Layer_Yellow")
1685 gcpy           if (self.dxfcolor == "Green"):
1686 gcpy             self.writedxf(tn, "Layer_Green")
1687 gcpy           if (self.dxfcolor == "Cyan"):
1688 gcpy             self.writedxf(tn, "Layer_Cyan")
1689 gcpy           if (self.dxfcolor == "Blue"):
1690 gcpy             self.writedxf(tn, "Layer_Blue")
1691 gcpy           if (self.dxfcolor == "Magenta"):
1692 gcpy             self.writedxf(tn, "Layer_Magenta")
1693 gcpy           if (self.dxfcolor == "White"):
1694 gcpy             self.writedxf(tn, "Layer_White")
1695 gcpy           if (self.dxfcolor == "DarkGray"):
1696 gcpy             self.writedxf(tn, "Layer_Dark_Gray")
1697 gcpy           if (self.dxfcolor == "LightGray"):
1698 gcpy             self.writedxf(tn, "Layer_Light_Gray")
1699 gcpy
1700 gcpy           self.writedxf(tn, "62")
1701 gcpy           if (self.dxfcolor == "Black"):
1702 gcpy             self.writedxf(tn, "0")
1703 gcpy           if (self.dxfcolor == "Red"):
1704 gcpy             self.writedxf(tn, "1")
1705 gcpy           if (self.dxfcolor == "Yellow"):
1706 gcpy             self.writedxf(tn, "2")
1707 gcpy           if (self.dxfcolor == "Green"):
1708 gcpy             self.writedxf(tn, "3")
1709 gcpy           if (self.dxfcolor == "Cyan"):
1710 gcpy             self.writedxf(tn, "4")
1711 gcpy           if (self.dxfcolor == "Blue"):
1712 gcpy             self.writedxf(tn, "5")
1713 gcpy           if (self.dxfcolor == "Magenta"):
1714 gcpy             self.writedxf(tn, "6")
1715 gcpy           if (self.dxfcolor == "White"):
1716 gcpy             self.writedxf(tn, "7")
1717 gcpy           if (self.dxfcolor == "DarkGray"):
1718 gcpy             self.writedxf(tn, "8")
1719 gcpy           if (self.dxfcolor == "LightGray"):
1720 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.

```

1722 gcpy     def dxfline(self, tn, xbegin, ybegin, xend, yend):
1723 gcpy       self.writedxf(tn, "0")
1724 gcpy       self.writedxf(tn, "LINE")
1725 gcpy #
1726 gcpy       self.writedxfcolor(tn)
1727 gcpy #
1728 gcpy       self.writedxf(tn, "10")
1729 gcpy       self.writedxf(tn, str(xbegin))
1730 gcpy       self.writedxf(tn, "20")
1731 gcpy       self.writedxf(tn, str(ybegin))
1732 gcpy       self.writedxf(tn, "30")
1733 gcpy       self.writedxf(tn, "0.0")
1734 gcpy       self.writedxf(tn, "11")
1735 gcpy       self.writedxf(tn, str(xend))
1736 gcpy       self.writedxf(tn, "21")

```

---

```

1737 gcpy      self.writedxf(tn, str(yend))
1738 gcpy      self.writedxf(tn, "31")
1739 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:

---

```

1741 gcpy      def beginpolyline(self, tn):#, xbegin, ybegin
1742 gcpy      self.writedxf(tn, "0")
1743 gcpy      self.writedxf(tn, "POLYLINE")
1744 gcpy      self.writedxf(tn, "8")
1745 gcpy      self.writedxf(tn, "default")
1746 gcpy      self.writedxf(tn, "66")
1747 gcpy      self.writedxf(tn, "1")
1748 gcpy #
1749 gcpy      self.writedxfcolor(tn)
1750 gcpy #
1751 gcpy #
1752 gcpy #
1753 gcpy #
1754 gcpy #
1755 gcpy #
1756 gcpy #
1757 gcpy      self.writedxf(tn, "70")
1758 gcpy      self.writedxf(tn, "0")

```

---

then add as many vertexes as are wanted:

---

```

1760 gcpy      def addvertex(self, tn, xend, yend):
1761 gcpy      self.writedxf(tn, "0")
1762 gcpy      self.writedxf(tn, "VERTEX")
1763 gcpy      self.writedxf(tn, "8")
1764 gcpy      self.writedxf(tn, "default")
1765 gcpy      self.writedxf(tn, "70")
1766 gcpy      self.writedxf(tn, "32")
1767 gcpy      self.writedxf(tn, "10")
1768 gcpy      self.writedxf(tn, str(xend))
1769 gcpy      self.writedxf(tn, "20")
1770 gcpy      self.writedxf(tn, str(yend))
1771 gcpy      self.writedxf(tn, "30")
1772 gcpy      self.writedxf(tn, "0.0")

```

---

then end the sequence:

---

```

1774 gcpy      def closepolyline(self, tn):
1775 gcpy      self.writedxf(tn, "0")
1776 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).

---

```

1778 gcpy      def dxfarcs(self, tn, xcenter, ycenter, radius, anglebegin,
                           endangle):
1779 gcpy      if (self.generatedxf == True):
1780 gcpy          self.writedxf(tn, "0")
1781 gcpy          self.writedxf(tn, "ARC")
1782 gcpy #
1783 gcpy          self.writedxfcolor(tn)
1784 gcpy #
1785 gcpy          self.writedxf(tn, "10")
1786 gcpy          self.writedxf(tn, str(xcenter))
1787 gcpy          self.writedxf(tn, "20")
1788 gcpy          self.writedxf(tn, str(ycenter))
1789 gcpy          self.writedxf(tn, "40")
1790 gcpy          self.writedxf(tn, str(radius))
1791 gcpy          self.writedxf(tn, "50")
1792 gcpy          self.writedxf(tn, str(anglebegin))
1793 gcpy          self.writedxf(tn, "51")
1794 gcpy          self.writedxf(tn, str(endangle))
1795 gcpy
1796 gcpy      def gcodearc(self, tn, xcenter, ycenter, radius, anglebegin,
                           endangle):
1797 gcpy          if (self.generategcode == True):

```

---

 1798 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.

```

1800 gcpy     def cutarcNECCdx (self, ex, ey, ez, xcenter, ycenter, radius):
1801 gcpy #         global toolpath
1802 gcpy #         toolpath = self.currenttool()
1803 gcpy #         toolpath = toolpath.translate([self.xpos(), self.ypos(),
1804 gcpy             self.zpos()])
1805 gcpy         self.dxfarc(self.currenttoolnumber(), xcenter, ycenter,
1806 gcpy             radius, 0, 90)
1807 gcpy         if (self.zpos == ez):
1808 gcpy             self.setzpos(0)
1809 gcpy         else:
1810 gcpy             self.setzpos((self.zpos()-ez)/90)
1811 gcpy         self.setxpos(ex)
1812 gcpy         self.setypos(ey)
1813 gcpy         self.setzpos(ez)
1814 gcpy         if self.generatepaths == True:
1815 gcpy             print("Unioning cutarcNECCdx toolpath")
1816 gcpy             self.arcloop(1, 90, xcenter, ycenter, radius)
1817 gcpy             self.toolpaths = self.toolpaths.union(toolpath)
1818 gcpy         else:
1819 gcpy             toolpath = self.arcloop(1, 90, xcenter, ycenter,
1820 gcpy                 radius)
1821 gcpy             print("Returning cutarcNECCdx toolpath")
1822 gcpy         return toolpath
1823 gcpy
1824 gcpy     def cutarcNWCCdx (self, ex, ey, ez, xcenter, ycenter, radius):
1825 gcpy #         global toolpath
1826 gcpy #         toolpath = self.currenttool()
1827 gcpy #         toolpath = toolpath.translate([self.xpos(), self.ypos(),
1828 gcpy             self.zpos()])
1829 gcpy         self.dxfarc(self.currenttoolnumber(), xcenter, ycenter,
1830 gcpy             radius, 90, 180)
1831 gcpy         if (self.zpos == ez):
1832 gcpy             self.setzpos(0)
1833 gcpy         else:
1834 gcpy             self.setzpos((self.zpos()-ez)/90)
1835 gcpy         self.setxpos(ex)
1836 gcpy         self.setypos(ey)
1837 gcpy         self.setzpos(ez)
1838 gcpy         if self.generatepaths == True:
1839 gcpy             self.arcloop(91, 180, xcenter, ycenter, radius)
1840 gcpy             self.toolpaths = self.toolpaths.union(toolpath)
1841 gcpy         else:
1842 gcpy             toolpath = self.arcloop(91, 180, xcenter, ycenter, radius)
1843 gcpy         return toolpath
1844 gcpy
1845 gcpy     def cutarcSWCCdx (self, ex, ey, ez, xcenter, ycenter, radius):
1846 gcpy #         global toolpath
1847 gcpy #         toolpath = self.currenttool()
1848 gcpy #         toolpath = toolpath.translate([self.xpos(), self.ypos(),
1849 gcpy             self.zpos()])
1850 gcpy         self.dxfarc(self.currenttoolnumber(), xcenter, ycenter,
1851 gcpy             radius, 180, 270)
1852 gcpy         if (self.zpos == ez):
1853 gcpy             self.setzpos(0)
1854 gcpy         else:
1855 gcpy             self.setzpos((self.zpos()-ez)/90)
1856 gcpy         self.setxpos(ex)
1857 gcpy         self.setypos(ey)
1858 gcpy         self.setzpos(ez)
1859 gcpy         if self.generatepaths == True:
1860 gcpy             self.arcloop(181, 270, xcenter, ycenter, radius)
1861 gcpy             self.toolpaths = self.toolpaths.union(toolpath)
1862 gcpy         else:
1863 gcpy             toolpath = self.arcloop(181, 270, xcenter, ycenter,
1864 gcpy                 radius)
1865 gcpy         return toolpath
1866 gcpy
1867 gcpy     def cutarcSECCdx (self, ex, ey, ez, xcenter, ycenter, radius):
1868 gcpy #         global toolpath
1869 gcpy #         toolpath = self.currenttool()
1870 gcpy #         toolpath = toolpath.translate([self.xpos(), self.ypos(),
1871 gcpy             self.zpos()])
1872 gcpy         self.dxfarc(self.currenttoolnumber(), xcenter, ycenter,

```

```

    radius, 270, 360)
1864 gcpy     if (self.zpos == ez):
1865 gcpy         self.setzpos(0)
1866 gcpy     else:
1867 gcpy         self.setzpos((self.zpos()-ez)/90)
1868 gcpy #     self.setxpos(ex)
1869 gcpy #     self.setypos(ey)
1870 gcpy #
1871 gcpy     if self.generatepaths == True:
1872 gcpy         self.arcloop(271, 360, xcenter, ycenter, radius)
1873 gcpy #         self.toolpaths = self.toolpaths.union(toolpath)
1874 gcpy #
1875 gcpy     toolpath = self.arcloop(271, 360, xcenter, ycenter,
1876 gcpy             radius)
1877 gcpy     return toolpath
1878 gcpy     def cutarcNECWdxr(self, ex, ey, ez, xcenter, ycenter, radius):
1879 gcpy #         global toolpath
1880 gcpy #         toolpath = self.currenttool()
1881 gcpy #         toolpath = toolpath.translate([self.xpos(), self.ypos(),
1882 gcpy             self.zpos()])
1883 gcpy         self.dxfarc(self.currenttoolnumber(), xcenter, ycenter,
1884 gcpy             radius, 0, 90)
1885 gcpy     if (self.zpos == ez):
1886 gcpy         self.setzpos(0)
1887 gcpy     else:
1888 gcpy         self.setzpos((self.zpos()-ez)/90)
1889 gcpy #
1890 gcpy     if self.generatepaths == True:
1891 gcpy         self.narcloop(89, 0, xcenter, ycenter, radius)
1892 gcpy #         self.toolpaths = self.toolpaths.union(toolpath)
1893 gcpy #
1894 gcpy     toolpath = self.narcloop(89, 0, xcenter, ycenter,
1895 gcpy             radius)
1896 gcpy     return toolpath
1897 gcpy     def cutarcSECWdxr(self, ex, ey, ez, xcenter, ycenter, radius):
1898 gcpy #         global toolpath
1899 gcpy #         toolpath = self.currenttool()
1900 gcpy #         toolpath = toolpath.translate([self.xpos(), self.ypos(),
1901 gcpy             self.zpos()])
1902 gcpy         self.dxfarc(self.currenttoolnumber(), xcenter, ycenter,
1903 gcpy             radius, 270, 360)
1904 gcpy     if (self.zpos == ez):
1905 gcpy         self.setzpos(0)
1906 gcpy     else:
1907 gcpy         self.setzpos((self.zpos()-ez)/90)
1908 gcpy #
1909 gcpy     if self.generatepaths == True:
1910 gcpy         self.narcloop(359, 270, xcenter, ycenter, radius)
1911 gcpy #         self.toolpaths = self.toolpaths.union(toolpath)
1912 gcpy #
1913 gcpy     toolpath = self.narcloop(359, 270, xcenter, ycenter,
1914 gcpy             radius)
1915 gcpy     return toolpath
1916 gcpy     def cutarcSWCWdxr(self, ex, ey, ez, xcenter, ycenter, radius):
1917 gcpy #         global toolpath
1918 gcpy #         toolpath = self.currenttool()
1919 gcpy #         toolpath = toolpath.translate([self.xpos(), self.ypos(),
1920 gcpy             self.zpos()])
1921 gcpy         self.dxfarc(self.currenttoolnumber(), xcenter, ycenter,
1922 gcpy             radius, 180, 270)
1923 gcpy     if (self.zpos == ez):
1924 gcpy         self.setzpos(0)
1925 gcpy     else:
1926 gcpy         self.setzpos((self.zpos()-ez)/90)
1927 gcpy #
1928 gcpy     if self.generatepaths == True:
1929 gcpy         self.narcloop(269, 180, xcenter, ycenter, radius)
1930 gcpy #         self.toolpaths = self.toolpaths.union(toolpath)
1931 gcpy

```

```

1932 gcpy          toolpath = self.narcloop(269, 180, xcenter, ycenter,
1933 gcpy          radius)
1934 gcpy          return toolpath
1935 gcpy      def cutarcNWCWdxr(self, ex, ey, ez, xcenter, ycenter, radius):
1936 gcpy #        global toolpath
1937 gcpy #        toolpath = self.currenttool()
1938 gcpy #        toolpath = toolpath.translate([self.xpos(), self.ypos(),
1939 gcpy          self.zpos()])
1940 gcpy          self.dxfarc(self.currenttoolnumber(), xcenter, ycenter,
1941 gcpy          radius, 90, 180)
1942 gcpy          if (self.zpos == ez):
1943 gcpy          self.setzpos(0)
1944 gcpy #        else:
1945 gcpy #          self.setzpos((self.zpos()-ez)/90)
1946 gcpy #        self.setxpos(ex)
1947 gcpy #        self.setypos(ey)
1948 gcpy #        self.setzpos(ez)
1949 gcpy #        if self.generatepaths == True:
1950 gcpy          self.narcloop(179, 90, xcenter, ycenter, radius)
1951 gcpy          self.toolpaths = self.toolpaths.union(toolpath)
1952 gcpy          return toolpath

```

---

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

```

cutarcNECCdxr(-stockXwidth/4, stockYheight/4+stockYheight/16, -stockZthickness, -stockXwidth/4, stockYh
cutarcNWCdxr(-(stockXwidth/4+stockYheight/16), stockYheight/4, -stockZthickness, -stockXwidth/4, stockYh
cutarcSWCdxr(-stockXwidth/4, stockYheight/4-stockYheight/16, -stockZthickness, -stockXwidth/4, stockYh
cutarcSECCdxr(-(stockXwidth/4-stockYheight/16), stockYheight/4, -stockZthickness, -stockXwidth/4, stockYh

```

---

```

1954 gcpy      def arcCCgc(self, ex, ey, ez, xcenter, ycenter, radius):
1955 gcpy          self.writegc("G03\u00d7X", str(ex), "\u00d7Y", str(ey), "\u00d7Z", str(ez),
1956 gcpy          , "\u00d7R", str(radius))
1957 gcpy      def arcCWgc(self, ex, ey, ez, xcenter, ycenter, radius):
1958 gcpy          self.writegc("G02\u00d7X", str(ex), "\u00d7Y", str(ey), "\u00d7Z", str(ez),
1959 gcpy          , "\u00d7R", str(radius))

```

---

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

---

```

1960 gcpy      def cutarcNECCdxfgc(self, ex, ey, ez, xcenter, ycenter, radius):
1961 gcpy          :
1962 gcpy          self.arcCCgc(ex, ey, ez, xcenter, ycenter, radius)
1963 gcpy          if self.generatepaths == True:
1964 gcpy          self.cutarcNECCdxr(ex, ey, ez, xcenter, ycenter, radius)
1965 gcpy          else:
1966 gcpy          return self.cutarcNECCdxr(ex, ey, ez, xcenter, ycenter,
1967 gcpy          radius)
1968 gcpy      def cutarcNWCdxfgc(self, ex, ey, ez, xcenter, ycenter, radius):
1969 gcpy          :
1970 gcpy          self.arcCCgc(ex, ey, ez, xcenter, ycenter, radius)
1971 gcpy          if self.generatepaths == False:
1972 gcpy          return self.cutarcNWCdxr(ex, ey, ez, xcenter, ycenter,
1973 gcpy          radius)
1974 gcpy      def cutarcSWCdxfgc(self, ex, ey, ez, xcenter, ycenter, radius):
1975 gcpy          :
1976 gcpy          self.arcCCgc(ex, ey, ez, xcenter, ycenter, radius)
1977 gcpy          if self.generatepaths == False:
1978 gcpy          return self.cutarcSWCdxr(ex, ey, ez, xcenter, ycenter,
1979 gcpy          radius)
1980 gcpy      def cutarcSECCdxfgc(self, ex, ey, ez, xcenter, ycenter, radius):
1981 gcpy          :
1982 gcpy          self.arcCCgc(ex, ey, ez, xcenter, ycenter, radius)
1983 gcpy          if self.generatepaths == False:
1984 gcpy          return self.cutarcSECCdxr(ex, ey, ez, xcenter, ycenter,
1985 gcpy          radius)
1986 gcpy      def cutarcNECWdxfgc(self, ex, ey, ez, xcenter, ycenter, radius):
1987 gcpy          :
1988 gcpy          self.arcCWgc(ex, ey, ez, xcenter, ycenter, radius)

```

```

1984 gcpy           if self.generatepaths == False:
1985 gcpy           return self.cutarcNECWdxf(ex, ey, ez, xcenter, ycenter,
1986 gcpy           radius)
1987 gcpy           def cutarcSECWdxfgc(self, ex, ey, ez, xcenter, ycenter, radius):
1988 gcpy           :
1989 gcpy           self.arcCWgc(ex, ey, ez, xcenter, ycenter, radius)
1990 gcpy           if self.generatepaths == False:
1991 gcpy           return self.cutarcSECWdxf(ex, ey, ez, xcenter, ycenter,
1992 gcpy           radius)
1993 gcpy           def cutarcSWCWdxfgc(self, ex, ey, ez, xcenter, ycenter, radius):
1994 gcpy           :
1995 gcpy           self.arcCWgc(ex, ey, ez, xcenter, ycenter, radius)
1996 gcpy           if self.generatepaths == False:
1997 gcpy           return self.cutarcSWCWdxf(ex, ey, ez, xcenter, ycenter,
1998 gcpy           radius)
1999 gcpy           def cutarcNWCWdxfgc(self, ex, ey, ez, xcenter, ycenter, radius):
1999 gcpy           :
2000 gcpy           self.arcCWgc(ex, ey, ez, xcenter, ycenter, radius)
2000 gcpy           if self.generatepaths == False:
2000 gcpy           return self.cutarcNWCWdxf(ex, ey, ez, xcenter, ycenter,
2000 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 }

```

---

### 3.7.3 G-code Overview

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

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

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

G-code	Command/Module
(Design File: )	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)

**3.7.3.1 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.

---

```

2002 gcpy      def dxfpostamble(self, tn):
2003 gcpy #          self.writedxf(tn, str(tn))
2004 gcpy          self.writedxf(tn, "0")
2005 gcpy          self.writedxf(tn, "ENDSEC")
2006 gcpy          self.writedxf(tn, "0")
2007 gcpy          self.writedxf(tn, "EOF")

```

---



---

```

2009 gcpy      def gcodepostamble(self):
2010 gcpy          self.writegc("Z12.700")
2011 gcpy          self.writegc("M05")
2012 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.

---

```

2014 gcpy      def closegcodefile(self):
2015 gcpy          if self.generategcode == True:
2016 gcpy              self.gcodepostamble()
2017 gcpy              self.gc.close()
2018 gcpy

```

```

2019 gcpy      def closedxfffile(self):
2020 gcpy          if self.generateddxf == True:
2021 gcpy #              global dxfclosed
2022 gcpy #                  self.dxfpostamble(-1)
2023 gcpy #                      self.dxfclosed = True
2024 gcpy #                          self.dxf.close()
2025 gcpy
2026 gcpy      def closedxfffiles(self):
2027 gcpy          if self.generateddxfs == True:
2028 gcpy              if (self.large_square_tool_num > 0):
2029 gcpy                  self.dxfpostamble(self.large_square_tool_num)
2030 gcpy              if (self.small_square_tool_num > 0):
2031 gcpy                  self.dxfpostamble(self.small_square_tool_num)
2032 gcpy              if (self.large_ball_tool_num > 0):
2033 gcpy                  self.dxfpostamble(self.large_ball_tool_num)
2034 gcpy              if (self.small_ball_tool_num > 0):
2035 gcpy                  self.dxfpostamble(self.small_ball_tool_num)
2036 gcpy              if (self.large_V_tool_num > 0):
2037 gcpy                  self.dxfpostamble(self.large_V_tool_num)
2038 gcpy              if (self.small_V_tool_num > 0):
2039 gcpy                  self.dxfpostamble(self.small_V_tool_num)
2040 gcpy              if (self.DT_tool_num > 0):
2041 gcpy                  self.dxfpostamble(self.DT_tool_num)
2042 gcpy              if (self.KH_tool_num > 0):
2043 gcpy                  self.dxfpostamble(self.KH_tool_num)
2044 gcpy              if (self.Roundover_tool_num > 0):
2045 gcpy                  self.dxfpostamble(self.Roundover_tool_num)
2046 gcpy              if (self.MISC_tool_num > 0):
2047 gcpy                  self.dxfpostamble(self.MISC_tool_num)
2048 gcpy
2049 gcpy          if (self.large_square_tool_num > 0):
2050 gcpy              self.dxflgsq.close()
2051 gcpy          if (self.small_square_tool_num > 0):
2052 gcpy              self.dfxmsq.close()
2053 gcpy          if (self.large_ball_tool_num > 0):
2054 gcpy              self.dxflgbl.close()
2055 gcpy          if (self.small_ball_tool_num > 0):
2056 gcpy              self.dfxsmbl.close()
2057 gcpy          if (self.large_V_tool_num > 0):
2058 gcpy              self.dxflgV.close()
2059 gcpy          if (self.small_V_tool_num > 0):
2060 gcpy              self.dfxsmV.close()
2061 gcpy          if (self.DT_tool_num > 0):
2062 gcpy              self.dxfDT.close()
2063 gcpy          if (self.KH_tool_num > 0):
2064 gcpy              self.dxfKH.close()
2065 gcpy          if (self.Roundover_tool_num > 0):
2066 gcpy              self.dxfRt.close()
2067 gcpy          if (self.MISC_tool_num > 0):
2068 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 }

```

---

### 3.8 Cutting shapes and expansion

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

It is most expedient to test out new features in a new/separate file insofar as the file structures will allow (tool definitions for example will need to consolidated in [3.4.1.1](#)) which will need to be included in the projects which will make use of said features until such time as they are added into the main `gcodepreview.scad` file.

A basic requirement for two-dimensional regions will be to define them so as to cut them out. Two different geometric treatments will be necessary: modeling the geometry which defines the region to be cut out (output as a DXF); and modeling the movement of the tool, the toolpath which will be used in creating the 3D model and outputting the G-code.

### 3.8.1 Building blocks

The outlines of shapes will be defined using:

- lines — dxfline
- arcs — dxfarc

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

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

- 0
  - **circle** — dxfcircle
  - ellipse (oval) (requires some sort of non-arc curve)
    - \* egg-shaped
  - **annulus** (one circle within another, forming a ring) — handled by nested circles
  - superellipse (see astroid below)
- 1
  - **cone with rounded end (arc)**—see also “sector” under 3 below
- 2
  - **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?

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

---

```

2070 gcpy      def dxfcircle(self, tool_num, xcenter, ycenter, radius):
2071 gcpy          self.dxfarc(tool_num, xcenter, ycenter, radius, 0, 90)
2072 gcpy          self.dxfarc(tool_num, xcenter, ycenter, radius, 90, 180)
2073 gcpy          self.dxfarc(tool_num, xcenter, ycenter, radius, 180, 270)
2074 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.

---

```

2076 gcpy     def cutcircleCC(self, xcenter, ycenter, bz, ez, radius):
2077 gcpy         self.setzpos(bz)
2078 gcpy         self.cutquarterCCNE(xcenter, ycenter + radius, self.zpos()
2079 gcpy             + ez/4, radius)
2080 gcpy         self.cutquarterCCNW(xcenter - radius, ycenter, self.zpos()
2081 gcpy             + ez/4, radius)
2082 gcpy         self.cutquarterCCSW(xcenter, ycenter - radius, self.zpos()
2083 gcpy             + ez/4, radius)
2084 gcpy     def cutcircleCCdx(self, xcenter, ycenter, bz, ez, radius):
2085 gcpy         self.cutcircleCC(self, xcenter, ycenter, bz, ez, radius)
2086 gcpy         self.dxfcircle(self, tool_num, xcenter, ycenter, radius)

```

---

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

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

---

```

2087 gcpy     def dxfrectangle(self, tool_num, xorigin, yorigin, xwidth,
2088 gcpy         yheight, corners = "Square", radius = 6):
2089 gcpy         if corners == "Square":
2090 gcpy             self.dxfline(tool_num, xorigin, yorigin, xorigin +
2091 gcpy                 xwidth, yorigin)
2092 gcpy             self.dxfline(tool_num, xorigin + xwidth, yorigin,
2093 gcpy                 xorigin + xwidth, yorigin + yheight)
2094 gcpy             self.dxfline(tool_num, xorigin + xwidth, yorigin +
2095 gcpy                 yheight, xorigin, yorigin + yheight)
2096 gcpy             self.dxfline(tool_num, xorigin, yorigin + yheight,
2097 gcpy                 xorigin, yorigin)
2098 gcpy         elif corners == "Fillet":
2099 gcpy             self.dxfrectangleround(tool_num, xorigin, yorigin,
2100 gcpy                 xwidth, yheight, radius)
2101 gcpy         elif corners == "Chamfer":
2102 gcpy             self.dxfrectanglechamfer(tool_num, xorigin, yorigin,
2103 gcpy                 xwidth, yheight, radius)
2104 gcpy         elif corners == "Flipped_Fillet":
2105 gcpy             self.dxfrectangleflippedfillet(tool_num, xorigin,
2106 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.

---

```

2100 gcpy     def dxfrectangleround(self, tool_num, xorigin, yorigin, xwidth,
2101 gcpy         yheight, radius):
2102 gcpy # begin section
2103 gcpy     self.writedxf(tool_num, "0")
2104 gcpy     self.writedxf(tool_num, "SECTION")
2105 gcpy     self.writedxf(tool_num, "2")
2106 gcpy     self.writedxf(tool_num, "ENTITIES")
2107 gcpy     self.writedxf(tool_num, "0")
2108 gcpy     self.writedxf(tool_num, "LWPOLYLINE")
2109 gcpy     self.writedxf(tool_num, "5")
2110 gcpy     self.writedxf(tool_num, "4E")
2111 gcpy     self.writedxf(tool_num, "100")
2112 gcpy     self.writedxf(tool_num, "AcDbEntity")
2113 gcpy     self.writedxf(tool_num, "8")
2114 gcpy     self.writedxf(tool_num, "0")
2115 gcpy     self.writedxf(tool_num, "6")
2116 gcpy     self.writedxf(tool_num, "ByLayer")
2117 gcpy #
2118 gcpy     self.writedxfcolor(tool_num)
2119 gcpy #
2120 gcpy     self.writedxf(tool_num, "370")
2121 gcpy     self.writedxf(tool_num, "-1")
2122 gcpy     self.writedxf(tool_num, "100")
2123 gcpy     self.writedxf(tool_num, "AcDbPolyline")
2124 gcpy     self.writedxf(tool_num, "90")
2125 gcpy     self.writedxf(tool_num, "8")
2126 gcpy     self.writedxf(tool_num, "70")
2127 gcpy     self.writedxf(tool_num, "1")
2128 gcpy     self.writedxf(tool_num, "43")
2129 gcpy     self.writedxf(tool_num, "0")
2130 gcpy #1 upper right corner before arc (counter-clockwise)
2131 gcpy     self.writedxf(tool_num, "10")

```

---

```

2131 gcpy      self.writedxf(tool_num, str(xorigin + xwidth))
2132 gcpy      self.writedxf(tool_num, "20")
2133 gcpy      self.writedxf(tool_num, str(yorigin + yheight - radius))
2134 gcpy      self.writedxf(tool_num, "42")
2135 gcpy      self.writedxf(tool_num, "0.414213562373095")
2136 gcpy #2 upper right corner after arc
2137 gcpy      self.writedxf(tool_num, "10")
2138 gcpy      self.writedxf(tool_num, str(xorigin + xwidth - radius))
2139 gcpy      self.writedxf(tool_num, "20")
2140 gcpy      self.writedxf(tool_num, str(yorigin + yheight))
2141 gcpy #3 upper left corner before arc (counter-clockwise)
2142 gcpy      self.writedxf(tool_num, "10")
2143 gcpy      self.writedxf(tool_num, str(xorigin + radius))
2144 gcpy      self.writedxf(tool_num, "20")
2145 gcpy      self.writedxf(tool_num, str(yorigin + yheight))
2146 gcpy      self.writedxf(tool_num, "42")
2147 gcpy      self.writedxf(tool_num, "0.414213562373095")
2148 gcpy #4 upper left corner after arc
2149 gcpy      self.writedxf(tool_num, "10")
2150 gcpy      self.writedxf(tool_num, str(xorigin))
2151 gcpy      self.writedxf(tool_num, "20")
2152 gcpy      self.writedxf(tool_num, str(yorigin + yheight - radius))
2153 gcpy #5 lower left corner before arc (counter-clockwise)
2154 gcpy      self.writedxf(tool_num, "10")
2155 gcpy      self.writedxf(tool_num, str(xorigin))
2156 gcpy      self.writedxf(tool_num, "20")
2157 gcpy      self.writedxf(tool_num, str(yorigin + radius))
2158 gcpy      self.writedxf(tool_num, "42")
2159 gcpy      self.writedxf(tool_num, "0.414213562373095")
2160 gcpy #6 lower left corner after arc
2161 gcpy      self.writedxf(tool_num, "10")
2162 gcpy      self.writedxf(tool_num, str(xorigin + radius))
2163 gcpy      self.writedxf(tool_num, "20")
2164 gcpy      self.writedxf(tool_num, str(yorigin))
2165 gcpy #7 lower right corner before arc (counter-clockwise)
2166 gcpy      self.writedxf(tool_num, "10")
2167 gcpy      self.writedxf(tool_num, str(xorigin + xwidth - radius))
2168 gcpy      self.writedxf(tool_num, "20")
2169 gcpy      self.writedxf(tool_num, str(yorigin))
2170 gcpy      self.writedxf(tool_num, "42")
2171 gcpy      self.writedxf(tool_num, "0.414213562373095")
2172 gcpy #8 lower right corner after arc
2173 gcpy      self.writedxf(tool_num, "10")
2174 gcpy      self.writedxf(tool_num, str(xorigin + xwidth))
2175 gcpy      self.writedxf(tool_num, "20")
2176 gcpy      self.writedxf(tool_num, str(yorigin + radius))
2177 gcpy # end current section
2178 gcpy      self.writedxf(tool_num, "0")
2179 gcpy      self.writedxf(tool_num, "SEQEND")

```

---

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

```

2181 gcpy      def dxfrctanglechamfer(self, tool_num, xorigin, yorigin,
2182 gcpy          xwidth, yheight, radius):
2183 gcpy          self.dxfline(tool_num, xorigin + radius, yorigin, xorigin,
2184 gcpy              yorigin + radius)
2185 gcpy          self.dxfline(tool_num, xorigin, yorigin + yheight - radius,
2186 gcpy              xorigin + radius, yorigin + yheight)
2187 gcpy          self.dxfline(tool_num, xorigin + xwidth - radius, yorigin +
2188 gcpy              yheight, xorigin + xwidth, yorigin + yheight - radius)
2189 gcpy          self.dxfline(tool_num, xorigin + xwidth - radius, yorigin +
2190 gcpy              yheight, xorigin + radius, yorigin + yheight)

```

---

Flipped Fillet:

```

2192 gcpy      def dxfrctangleflippedfillet(self, tool_num, xorigin, yorigin,
2193 gcpy          xwidth, yheight, radius):

```

---

---

```

2193 gcpy      self.dxfarc(tool_num, xorigin, yorigin, radius, 0, 90)
2194 gcpy      self.dxfarc(tool_num, xorigin + xwidth, yorigin, radius,
                           90, 180)
2195 gcpy      self.dxfarc(tool_num, xorigin + xwidth, yorigin + yheight,
                           radius, 180, 270)
2196 gcpy      self.dxfarc(tool_num, xorigin, yorigin + yheight, radius,
                           270, 360)
2197 gcpy      self.dxfline(tool_num, xorigin + radius, yorigin, xorigin +
                           xwidth - radius, yorigin)
2198 gcpy      self.dxfline(tool_num, xorigin + xwidth, yorigin + radius,
                           xorigin + xwidth, yorigin + yheight - radius)
2199 gcpy      self.dxfline(tool_num, xorigin + xwidth - radius, yorigin +
                           yheight, xorigin + radius, yorigin + yheight)
2200 gcpy      self.dxfline(tool_num, xorigin, yorigin + yheight - radius,
                           xorigin, yorigin + radius)
2201 gcpy

```

---

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.

---

```

2203 gcpy      def cutrectangle(self, tool_num, bx, by, bz, xwidth, yheight,
                           zdepth):
2204 gcpy          self.cutline(bx, by, bz)
2205 gcpy          self.cutline(bx, by, bz - zdepth)
2206 gcpy          self.cutline(bx + xwidth, by, bz - zdepth)
2207 gcpy          self.cutline(bx + xwidth, by + yheight, bz - zdepth)
2208 gcpy          self.cutline(bx, by + yheight, bz - zdepth)
2209 gcpy          self.cutline(bx, by, bz - zdepth)
2210 gcpy
2211 gcpy      def cutrectangledxf(self, tool_num, bx, by, bz, xwidth, yheight,
                           zdepth):
2212 gcpy          self.cutrectangle(tool_num, bx, by, bz, xwidth, yheight,
                           zdepth)
2213 gcpy          self.dxfrectangle(tool_num, bx, by, xwidth, yheight, "Square")

```

---

The rounded forms instantiate a radius:

---

```

2215 gcpy      def cutrectangleround(self, tool_num, bx, by, bz, xwidth,
                           yheight, zdepth, radius):
2216 gcpy      #           self.rapid(bx + radius, by, bz)
2217 gcpy          self.cutline(bx + radius, by, bz + zdepth)
2218 gcpy          self.cutline(bx + xwidth - radius, by, bz + zdepth)
2219 gcpy          self.cutquarterCCSE(bx + xwidth, by + radius, bz + zdepth,
                           radius)
2220 gcpy          self.cutline(bx + xwidth, by + yheight - radius, bz +
                           zdepth)
2221 gcpy          self.cutquarterCCNE(bx + xwidth - radius, by + yheight, bz
                           + zdepth, radius)
2222 gcpy          self.cutline(bx + radius, by + yheight, bz + zdepth)
2223 gcpy          self.cutquarterCCNW(bx, by + yheight - radius, bz + zdepth,
                           radius)
2224 gcpy          self.cutline(bx, by + radius, bz + zdepth)
2225 gcpy          self.cutquarterCCSW(bx + radius, by, bz + zdepth, radius)
2226 gcpy
2227 gcpy      def cutrectanglerounddxf(self, tool_num, bx, by, bz, xwidth,
                           yheight, zdepth, radius):
2228 gcpy          self.cutrectangleround(tool_num, bx, by, bz, xwidth,
                           yheight, zdepth, radius)
2229 gcpy          self.dxfrectangleround(tool_num, bx, by, xwidth, yheight,
                           radius)

```

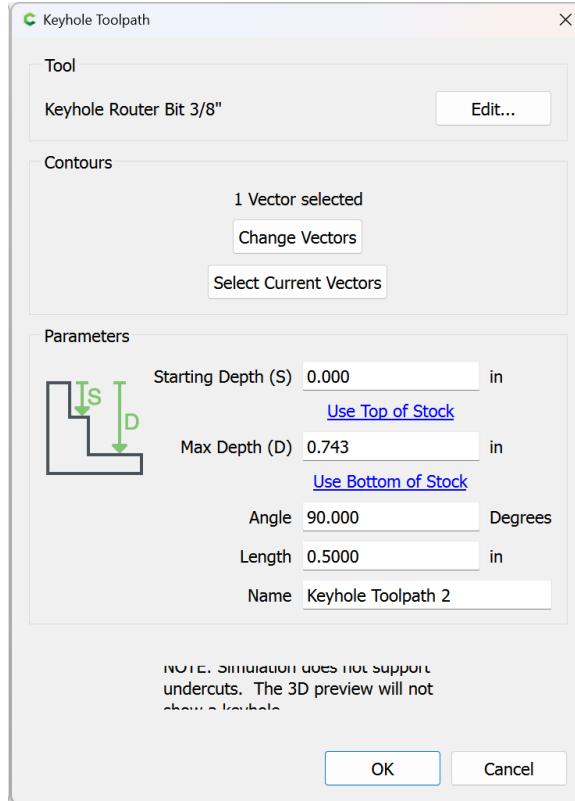
---

**3.8.1.1.3 Keyhole toolpath and undercut tooling** The first topologically unusual toolpath is `cutkeyhole` toolpath — where other toolpaths have a direct correspondence between the associated geometry and the area cut, that Keyhole toolpaths may be used with tooling which undercuts

and which will result in the creation of two different physical regions: the visible surface matching the union of the tool perimeter at the entry point and the linear movement of the shaft and the larger region of the tool perimeter at the depth which the tool is plunged to and moved along.

Tooling for such toolpaths is defined at paragraph [3.5.1](#)

The interface which is being modeled is that of Carbide Create:



Hence the parameters:

- Starting Depth == kh\_start\_depth
- Max Depth == kh\_max\_depth
- Angle == kht\_direction
- Length == kh\_distance
- Tool == kh\_tool\_num

Due to the possibility of rotation, for the in-between positions there are more cases than one would think — for each quadrant there are the following possibilities:

- one node on the clockwise side is outside of the quadrant
- two nodes on the clockwise side are outside of the quadrant
- all nodes are w/in the quadrant
- one node on the counter-clockwise side is outside of the quadrant
- two nodes on the counter-clockwise side are outside of the quadrant

Supporting all of these would require trigonometric comparisons in the if...else blocks, so only the 4 quadrants, N, S, E, and W will be supported in the initial version. This will be done by wrapping the command with a version which only accepts those options:

---

```

2231 gcpy      def cutkeyholegcdxf(self, kh_tool_num, kh_start_depth,
2232 gcpy          kh_max_depth, kht_direction, kh_distance):
2233 gcpy          if (kht_direction == "N"):
2234 gcpy              toolpath = self.cutKHgcdxf(kh_tool_num, kh_start_depth,
2235 gcpy                  kh_max_depth, 90, kh_distance)
2236 gcpy          elif (kht_direction == "S"):
2237 gcpy              toolpath = self.cutKHgcdxf(kh_tool_num, kh_start_depth,
2238 gcpy                  kh_max_depth, 270, kh_distance)
2239 gcpy          elif (kht_direction == "E"):
2240 gcpy              toolpath = self.cutKHgcdxf(kh_tool_num, kh_start_depth,
2241 gcpy                  kh_max_depth, 0, kh_distance)
2242 gcpy          elif (kht_direction == "W"):
2243 gcpy              toolpath = self.cutKHgcdxf(kh_tool_num, kh_start_depth,
2244 gcpy                  kh_max_depth, 180, kh_distance)
2245 gcpy      if self.generatepaths == True:

```

---

```

2241 gcpy #           self.toolpaths = union([self.toolpaths, toolpath])
2242 gcpy     return toolpath
2243 gcpy #
2244 gcpy #           else:
2245 gcpy #               return cube([0.01, 0.01, 0.01])

```

---



---

```

174 gpcscad module cutkeyholegcdxf(kh_tool_num, kh_start_depth, kh_max_depth,
175 gpcscad         kht_direction, kh_distance){
176 gpcscad     gcp.cutkeyholegcdxf(kh_tool_num, kh_start_depth, kh_max_depth,
176 gpcscad         kht_direction, kh_distance);

```

---

`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:

---

```

2246 gcpy     def cutKHgcdxf(self, kh_tool_num, kh_start_depth, kh_max_depth,
2247 gcpy         kh_angle, kh_distance):
2248 gcpy         oXpos = self.xpos()
2249 gcpy         oYpos = self.ypos()
2249 gcpy         self.dxfKH(kh_tool_num, self.xpos(), self.ypos(),
2250 gcpy             kh_start_depth, kh_max_depth, kh_angle, kh_distance)
2250 gcpy         toolpath = self.cutline(self.xpos(), self.ypos(), -
2251 gcpy             kh_max_depth)
2251 gcpy         self.setxpos(oXpos)
2252 gcpy         self.setypos(oYpos)
2253 gcpy #
2254 gcpy         if self.generatepaths == False:
2255 gcpy #
2256 gcpy         else:
2256 gcpy #             return toolpath
2256 gcpy #             return cube([0.001, 0.001, 0.001])

```

---



---

```

2258 gcpy     def dxfKH(self, kh_tool_num, oXpos, oYpos, kh_start_depth,
2259 gcpy         kh_max_depth, kh_angle, kh_distance):
2260 gcpy #         oXpos = self.xpos()
2261 gcpy #         oYpos = self.ypos()
2261 gcpy #Circle at entry hole
2262 gcpy         self.dxfarc(kh_tool_num, oXpos, oYpos, self.tool_radius(
2262 gcpy             kh_tool_num, 7), 0, 90)
2263 gcpy         self.dxfarc(kh_tool_num, oXpos, oYpos, self.tool_radius(
2263 gcpy             kh_tool_num, 7), 90, 180)
2264 gcpy         self.dxfarc(kh_tool_num, oXpos, oYpos, self.tool_radius(
2264 gcpy             kh_tool_num, 7), 180, 270)
2265 gcpy         self.dxfarc(kh_tool_num, oXpos, oYpos, self.tool_radius(
2265 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:

---

```

2266 gcpy #pre-calculate needed values
2267 gcpy         r = self.tool_radius(kh_tool_num, 7)
2268 gcpy #
2268 gcpy         print(r)
2269 gcpy         rt = self.tool_radius(kh_tool_num, 1)
2270 gcpy #
2270 gcpy         print(rt)
2271 gcpy         ro = math.sqrt((self.tool_radius(kh_tool_num, 1))**2-(self.
2271 gcpy             tool_radius(kh_tool_num, 7))**2)
2272 gcpy #
2272 gcpy         print(ro)
2273 gcpy         angle = math.degrees(math.acos(ro/rt))
2274 gcpy #Outlines of entry hole and slot
2275 gcpy         if (kh_angle == 0):
2276 gcpy #Lower left of entry hole
2277 gcpy             self.dxfarc(kh_tool_num, self.xpos(), self.ypos(), self
2277 gcpy                 .tool_radius(kh_tool_num, 1), 180, 270)
2278 gcpy #Upper left of entry hole
2279 gcpy             self.dxfarc(kh_tool_num, self.xpos(), self.ypos(), self
2279 gcpy                 .tool_radius(kh_tool_num, 1), 90, 180)
2280 gcpy #Upper right of entry hole
2281 gcpy             self.dxfarc(kh_tool_num, self.xpos(), self.ypos(), rt,
2281 gcpy                 41.810, 90)
2282 gcpy             self.dxfarc(kh_tool_num, self.xpos(), self.ypos(), rt,
2282 gcpy                 angle, 90)
2283 gcpy #Lower right of entry hole
2284 gcpy             self.dxfarc(kh_tool_num, self.xpos(), self.ypos(), rt,
2284 gcpy                 270, 360-angle)
2285 gcpy #
2285 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)
        ))
2286 gcpy #Actual line of cut
2287 gcpy #           self.dxfline(kh_tool_num, self.xpos(), self.ypos(),
        self.xpos()+kh_distance, self.ypos())
2288 gcpy #upper right of end of slot (kh_max_depth+4.36))/2
2289 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)
2290 gcpy #lower right of end of slot
2291 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)
2292 gcpy #upper right slot
2293 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))
2294 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)
2295 gcpy #end position at top of slot
2296 gcpy #lower right slot
2297 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))
2298 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)
2299 gcpy #end position at top of slot
2300 gcpy #
           hull(){
2301 gcpy #
           translate([xpos(), ypos(), zpos()]){
2302 gcpy #
           keyhole_shaft(6.35, 9.525);
2303 gcpy #
           }
2304 gcpy #
           translate([xpos(), ypos(), zpos()-kh_max_depth]){
2305 gcpy #
           keyhole_shaft(6.35, 9.525);
2306 gcpy #
           }
2307 gcpy #
           }
2308 gcpy #
           hull(){
2309 gcpy #
           translate([xpos(), ypos(), zpos()-kh_max_depth]){
2310 gcpy #
           keyhole_shaft(6.35, 9.525);
2311 gcpy #
           }
2312 gcpy #
           translate([xpos()+kh_distance, ypos(), zpos()-kh_max_depth])
           {
2313 gcpy #
           keyhole_shaft(6.35, 9.525);
2314 gcpy #
           }
2315 gcpy #
           }
2316 gcpy #
           cutwithfeed(getxpos(), getypos(), -kh_max_depth, feed);
2317 gcpy #
           cutwithfeed(getxpos()+kh_distance, getypos(), -kh_max_depth,
           feed);
2318 gcpy #
           setxpos(getxpos()-kh_distance);
2319 gcpy #
           } else if (kh_angle > 0 && kh_angle < 90) {
2320 gcpy //echo(kh_angle);
2321 gcpy #
           dxfarc(getxpos(), getypos(), tool_diameter(KH_tool_num, (
           kh_max_depth))/2, 90+kh_angle, 180+kh_angle, KH_tool_num);
2322 gcpy #
           dxfarc(getxpos(), getypos(), tool_diameter(KH_tool_num, (
           kh_max_depth))/2, 180+kh_angle, 270+kh_angle, KH_tool_num);
2323 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);
2324 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);
2325 gcpy #dxfarc(getxpos()+(kh_distance*cos(kh_angle)),
2326 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);

```

```

2327 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);  

2328 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))),  

2329 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))),  

2330 gcpy # getxpos() +(kh_distance*cos(kh_angle))-((tool_diameter(KH_tool_num  

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

2331 gcpy # getypos() +(kh_distance*sin(kh_angle))+((tool_diameter(KH_tool_num  

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

2332 gcpy #//echo("a", tool_diameter(KH_tool_num, (kh_max_depth+4.36))/2);  

2333 gcpy #//echo("c", tool_diameter(KH_tool_num, (kh_max_depth))/2);  

2334 gcpy #echo("Aangle", asin((tool_diameter(KH_tool_num, (kh_max_depth  

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

2335 gcpy #//echo(kh_angle);  

2336 gcpy # cutwithfeed(getxpos()+(kh_distance*cos(kh_angle)), getypos()+(  

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

2337 gcpy #           toolpath = toolpath.union(self.cutline(self.xpos() +  

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

2338 gcpy     elif (kh_angle == 90):  

2339 gcpy #Lower left of entry hole  

2340 gcpy         self.dxfarc(kh_tool_num, oXpos, oYpos, self.tool_radius  

    (kh_tool_num, 1), 180, 270)  

2341 gcpy #Lower right of entry hole  

2342 gcpy         self.dxfarc(kh_tool_num, oXpos, oYpos, self.tool_radius  

    (kh_tool_num, 1), 270, 360)  

2343 gcpy #left slot  

2344 gcpy         self.dxfline(kh_tool_num, oXpos-r, oYpos+ro, oXpos-r,  

    oYpos+kh_distance)  

2345 gcpy #right slot  

2346 gcpy         self.dxfline(kh_tool_num, oXpos+r, oYpos+ro, oXpos+r,  

    oYpos+kh_distance)  

2347 gcpy #upper left of end of slot  

2348 gcpy         self.dxfarc(kh_tool_num, oXpos, oYpos+kh_distance, r,  

    90, 180)  

2349 gcpy #upper right of end of slot  

2350 gcpy         self.dxfarc(kh_tool_num, oXpos, oYpos+kh_distance, r,  

    0, 90)  

2351 gcpy #Upper right of entry hole  

2352 gcpy         self.dxfarc(kh_tool_num, oXpos, oYpos, rt, 0, 90-angle)  

2353 gcpy #Upper left of entry hole  

2354 gcpy         self.dxfarc(kh_tool_num, oXpos, oYpos, rt, 90+angle,  

    180)  

2355 gcpy #           toolpath = toolpath.union(self.cutline(oXpos, oYpos +  

    kh_distance, -kh_max_depth))  

2356 gcpy     elif (kh_angle == 180):  

2357 gcpy #Lower right of entry hole  

2358 gcpy         self.dxfarc(kh_tool_num, oXpos, oYpos, self.tool_radius  

    (kh_tool_num, 1), 270, 360)  

2359 gcpy #Upper right of entry hole  

2360 gcpy         self.dxfarc(kh_tool_num, oXpos, oYpos, self.tool_radius  

    (kh_tool_num, 1), 0, 90)  

2361 gcpy #Upper left of entry hole  

2362 gcpy         self.dxfarc(kh_tool_num, oXpos, oYpos, rt, 90, 180-  

    angle)  

2363 gcpy #Lower left of entry hole  

2364 gcpy         self.dxfarc(kh_tool_num, oXpos, oYpos, rt, 180+angle,  

    270)  

2365 gcpy #upper slot  

2366 gcpy         self.dxfline(kh_tool_num, oXpos-ro, oYpos-r, oXpos -  

    kh_distance, oYpos-r)  

2367 gcpy #lower slot  

2368 gcpy         self.dxfline(kh_tool_num, oXpos-ro, oYpos+r, oXpos -  

    kh_distance, oYpos+r)  

2369 gcpy #upper left of end of slot  

2370 gcpy         self.dxfarc(kh_tool_num, oXpos-kh_distance, oYpos, r,  

    90, 180)  

2371 gcpy #lower left of end of slot  

2372 gcpy         self.dxfarc(kh_tool_num, oXpos-kh_distance, oYpos, r,  

    180, 270)  

2373 gcpy #           toolpath = toolpath.union(self.cutline(oXpos -  

    kh_distance, oYpos, -kh_max_depth))  

2374 gcpy     elif (kh_angle == 270):  

2375 gcpy #Upper left of entry hole  

2376 gcpy         self.dxfarc(kh_tool_num, oXpos, oYpos, self.tool_radius

```

```

            (kh_tool_num, 1), 90, 180)
2377 gcpy #Upper right of entry hole
2378 gcpy           self.dxfarc(kh_tool_num, oXpos, oYpos, self.tool_radius
                           (kh_tool_num, 1), 0, 90)
2379 gcpy #left slot
2380 gcpy           self.dxfline(kh_tool_num, oXpos-r, oYpos-ro, oXpos-r,
                           oYpos-kh_distance)
2381 gcpy #right slot
2382 gcpy           self.dxfline(kh_tool_num, oXpos+r, oYpos-ro, oXpos+r,
                           oYpos-kh_distance)
2383 gcpy #lower left of end of slot
2384 gcpy           self.dxfarc(kh_tool_num, oXpos, oYpos-kh_distance, r,
                           180, 270)
2385 gcpy #lower right of end of slot
2386 gcpy           self.dxfarc(kh_tool_num, oXpos, oYpos-kh_distance, r,
                           270, 360)
2387 gcpy #lower right of entry hole
2388 gcpy           self.dxfarc(kh_tool_num, oXpos, oYpos, rt, 180, 270-
                           angle)
2389 gcpy #lower left of entry hole
2390 gcpy           self.dxfarc(kh_tool_num, oXpos, oYpos, rt, 270+angle,
                           360)
2391 gcpy #
           toolpath = toolpath.union(self.cutline(oXpos, oYpos-
                           kh_distance, -kh_max_depth))
2392 gcpy #
           print(self.zpos())
2393 gcpy #
           self.setxpos(oXpos)
2394 gcpy #
           self.setypos(oYpos)
2395 gcpy #
           if self.generatepaths == False:
2396 gcpy #
           return toolpath
2397 gcpy
2398 gcpy # } else if (kh_angle == 90) {
2399 gcpy #
           //Lower left of entry hole
2400 gcpy #
           dxfarc(getxpos(), getypos(), 9.525/2, 180, 270, KH_tool_num);
2401 gcpy #
           //Lower right of entry hole
2402 gcpy #
           dxfarc(getxpos(), getypos(), 9.525/2, 270, 360, KH_tool_num);
2403 gcpy #
           //Upper right of entry hole
2404 gcpy #
           dxfarc(getxpos(), getypos(), 9.525/2, 0, acos(tool_diameter(
                           KH_tool_num, 5)/tool_diameter(KH_tool_num, 1)), KH_tool_num);
2405 gcpy #
           //Upper left of entry hole
2406 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
                           );
2407 gcpy #
           //Actual line of cut
2408 gcpy #
           dxfline(getxpos(), getypos(), getxpos(), getypos()+kh_distance
                           );
2409 gcpy #
           //upper right of slot
2410 gcpy #
           dxfarc(getxpos(), getypos()+kh_distance, tool_diameter(
                           KH_tool_num, (kh_max_depth+4.36))/2, 0, 90, KH_tool_num);
2411 gcpy #
           //upper left of slot
2412 gcpy #
           dxfarc(getxpos(), getypos()+kh_distance, tool_diameter(
                           KH_tool_num, (kh_max_depth+6.35))/2, 90, 180, KH_tool_num);
2413 gcpy #
           //right of slot
2414 gcpy #
           dxfline(
2415 gcpy #
               getxpos() + tool_diameter(KH_tool_num, (kh_max_depth))/2,
2416 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-(tool_diameter(KH_tool_num, (kh_max_depth-6.34))/2)^2,
2417 gcpy #
               getxpos() + tool_diameter(KH_tool_num, (kh_max_depth))/2,
2418 gcpy #
               //end position at top of slot
2419 gcpy #
               getypos() + kh_distance,
2420 gcpy #
               KH_tool_num);
2421 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);
2422 gcpy #
           hull(){
2423 gcpy #
               translate([xpos(), ypos(), zpos()]){
2424 gcpy #
                   keyhole_shaft(6.35, 9.525);
2425 gcpy #
               }
2426 gcpy #
               translate([xpos(), ypos(), zpos()-kh_max_depth]){
2427 gcpy #
                   keyhole_shaft(6.35, 9.525);
2428 gcpy #
               }
2429 gcpy #
           hull(){
2430 gcpy #
               translate([xpos(), ypos(), zpos()-kh_max_depth]){
2431 gcpy #
                   keyhole_shaft(6.35, 9.525);
2432 gcpy #
               }
2433 gcpy #
           }
```

```

2434 gcpy #      translate([xpos(), ypos()+kh_distance, zpos()-kh_max_depth])
{
2435 gcpy #      keyhole_shaft(6.35, 9.525);
2436 gcpy #
2437 gcpy #
2438 gcpy #      cutwithfeed(getxpos(), getypos(), -kh_max_depth, feed);
2439 gcpy #      cutwithfeed(getxpos(), getypos()+kh_distance, -kh_max_depth,
feed);
2440 gcpy #      setypos(getypos()-kh_distance);
2441 gcpy # } else if (kh_angle == 180) {
2442 gcpy # //Lower right of entry hole
2443 gcpy # dxfarc(getxpos(), getypos(), 9.525/2, 270, 360, KH_tool_num);
2444 gcpy # //Upper right of entry hole
2445 gcpy # dxfarc(getxpos(), getypos(), 9.525/2, 0, 90, KH_tool_num);
2446 gcpy # //Upper left of entry hole
2447 gcpy # dxfarc(getxpos(), getypos(), 9.525/2, 90, 90+acos(
tool_diameter(KH_tool_num, 5)/tool_diameter(KH_tool_num, 1)),
KH_tool_num);
2448 gcpy # //Lower left of entry hole
2449 gcpy # dxfarc(getxpos(), getypos(), 9.525/2, 270-acos(tool_diameter(
KH_tool_num, 5)/tool_diameter(KH_tool_num, 1)), 270, KH_tool_num);
2450 gcpy # //upper left of slot
2451 gcpy # dxfarc(getxpos()-kh_distance, getypos(), tool_diameter(
KH_tool_num, (kh_max_depth+6.35))/2, 90, 180, KH_tool_num);
2452 gcpy # //lower left of slot
2453 gcpy # dxfarc(getxpos()-kh_distance, getypos(), tool_diameter(
KH_tool_num, (kh_max_depth+6.35))/2, 180, 270, KH_tool_num);
2454 gcpy # //Actual line of cut
2455 gcpy # dxfline(getxpos(), getypos(), getxpos()-kh_distance, getypos())
;
2456 gcpy # //upper left slot
2457 gcpy # dxfline(
2458 gcpy #     getxpos()-(math.sqrt((tool_diameter(KH_tool_num, 1)^2)-
tool_diameter(KH_tool_num, 5)^2))/2),
2459 gcpy #     getypos()+tool_diameter(KH_tool_num, (kh_max_depth))/2,
//((kh_max_depth-6.34))/2)^2-(tool_diameter(KH_tool_num, (
kh_max_depth-6.34))/2)^2,
2460 gcpy #     getxpos()-kh_distance,
2461 gcpy # //end position at top of slot
2462 gcpy #     getypos()+tool_diameter(KH_tool_num, (kh_max_depth))/2,
2463 gcpy #     KH_tool_num);
2464 gcpy # //lower right slot
2465 gcpy # dxfline(
2466 gcpy #     getxpos()-(math.sqrt((tool_diameter(KH_tool_num, 1)^2)-
tool_diameter(KH_tool_num, 5)^2))/2),
2467 gcpy #     getypos()-tool_diameter(KH_tool_num, (kh_max_depth))/2,
//((kh_max_depth-6.34))/2)^2-(tool_diameter(KH_tool_num, (
kh_max_depth-6.34))/2)^2,
2468 gcpy #     getxpos()-kh_distance,
2469 gcpy # //end position at top of slot
2470 gcpy #     getypos()-tool_diameter(KH_tool_num, (kh_max_depth))/2,
2471 gcpy #     KH_tool_num);
2472 gcpy # hull(){
2473 gcpy #     translate([xpos(), ypos(), zpos()]){
2474 gcpy #         keyhole_shaft(6.35, 9.525);
2475 gcpy #
2476 gcpy #         translate([xpos(), ypos(), zpos()-kh_max_depth]){
2477 gcpy #             keyhole_shaft(6.35, 9.525);
2478 gcpy #
2479 gcpy #
2480 gcpy #         hull(){
2481 gcpy #             translate([xpos(), ypos(), zpos()-kh_max_depth]){
2482 gcpy #                 keyhole_shaft(6.35, 9.525);
2483 gcpy #
2484 gcpy #                 translate([xpos()-kh_distance, ypos(), zpos()-kh_max_depth])
{
2485 gcpy #                     keyhole_shaft(6.35, 9.525);
2486 gcpy #
2487 gcpy #
2488 gcpy #                     cutwithfeed(getxpos(), getypos(), -kh_max_depth, feed);
2489 gcpy #                     cutwithfeed(getxpos()-kh_distance, getypos(), -kh_max_depth,
feed);
2490 gcpy #                     setypos(getxpos()+kh_distance);
2491 gcpy # } else if (kh_angle == 270) {
2492 gcpy # //Upper right of entry hole
2493 gcpy # dxfarc(getxpos(), getypos(), 9.525/2, 0, 90, KH_tool_num);
2494 gcpy # //Upper left of entry hole

```

```

2495 gcpy #      dxfarc(getxpos(), getypos(), 9.525/2, 90, 180, KH_tool_num);
2496 gcpy #      //lower right of slot
2497 gcpy #      dxfarc(getxpos(), getypos()-kh_distance, tool_diameter(
2498 gcpy #          KH_tool_num, (kh_max_depth+4.36))/2, 270, 360, KH_tool_num);
2499 gcpy #      //lower left of slot
2500 gcpy #      dxfarc(getxpos(), getypos()-kh_distance, tool_diameter(
2501 gcpy #          KH_tool_num, (kh_max_depth+4.36))/2, 180, 270, KH_tool_num);
2502 gcpy #      //Actual line of cut
2503 gcpy #      dxfline(getxpos(), getypos(), getxpos(), getypos()-kh_distance
2504 gcpy #          );
2505 gcpy #      //right of slot
2506 gcpy #      dxfline(
2507 gcpy #          getxpos() + tool_diameter(KH_tool_num, (kh_max_depth))/2,
2508 gcpy #          getypos() - (math.sqrt((tool_diameter(KH_tool_num, 1)^2) -
2509 gcpy #              tool_diameter(KH_tool_num, 5)^2))/2, //((kh_max_depth-6.34))
2510 gcpy #              /2)^2 - (tool_diameter(KH_tool_num, (kh_max_depth-6.34))/2)^2,
2511 gcpy #              getxpos() + tool_diameter(KH_tool_num, (kh_max_depth))/2,
2512 gcpy #              //end position at top of slot
2513 gcpy #              getypos() - kh_distance,
2514 gcpy #              KH_tool_num);
2515 gcpy #      //left of slot
2516 gcpy #      dxfline(
2517 gcpy #          getxpos() - tool_diameter(KH_tool_num, (kh_max_depth))/2,
2518 gcpy #          getypos() - (math.sqrt((tool_diameter(KH_tool_num, 1)^2) -
2519 gcpy #              tool_diameter(KH_tool_num, 5)^2))/2, //((kh_max_depth-6.34))
2520 gcpy #              /2)^2 - (tool_diameter(KH_tool_num, (kh_max_depth-6.34))/2)^2,
2521 gcpy #              getxpos() - tool_diameter(KH_tool_num, (kh_max_depth))/2,
2522 gcpy #              //end position at top of slot
2523 gcpy #              getypos() - kh_distance,
2524 gcpy #              KH_tool_num);
2525 gcpy #      //Lower right of entry hole
2526 gcpy #      dxfarc(getxpos(), getypos(), 9.525/2, 360-acos(tool_diameter(
2527 gcpy #          KH_tool_num, 5)/tool_diameter(KH_tool_num, 1)), 360, KH_tool_num)
2528 gcpy #      );
2529 gcpy #      hull(){
2530 gcpy #          translate([xpos(), ypos(), zpos()]){
2531 gcpy #              keyhole_shaft(6.35, 9.525);
2532 gcpy #          }
2533 gcpy #          translate([xpos(), ypos(), zpos()-kh_max_depth]){
2534 gcpy #              keyhole_shaft(6.35, 9.525);
2535 gcpy #          }
2536 gcpy #          translate([xpos(), ypos(), zpos()-kh_max_depth]){
2537 gcpy #              keyhole_shaft(6.35, 9.525);
2538 gcpy #              cutwithfeed(getxpos(), getypos(), -kh_max_depth, feed);
2539 gcpy #              cutwithfeed(getxpos(), getypos()-kh_distance, -kh_max_depth,
2540 gcpy #                  feed);
2541 gcpy #              setypos(getypos() + kh_distance);
2542 gcpy #      }

```

---

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

BlocksCAD project page at: <https://www.blockscad3d.com/community/projects/1941456> and discussion at: <https://community.carbide3d.com/t/tool-paths-for-different-sized-dovetail-bit-89098>

Making such cuts will require dovetail tooling such as:

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

Two commands are required:

---

```

2544 gcpy     def cut_pins(self, Joint_Width, stockZthickness,
2545 gcpy         Number_of_Dovetails, Spacing, Proportion, DTT_diameter,
2546 gcpy         DTT_angle):
2547 gcpy         DTO = Tan(DTT_angle) * (stockZthickness * Proportion)
2548 gcpy         DTR = DTT_diameter/2 - DTO
2549 gcpy         cpr = self.rapidXY(0, stockZthickness + Spacing/2)
2550 gcpy         ctp = self.cutlinedxfgc(self.xpos(), self.ypos(), -
2551 gcpy             stockZthickness * Proportion)
2552 gcpy         #           ctp = ctp.union(self.cutlinedxfgc(Joint_Width / (
2553 gcpy             Number_of_Dovetails * 2), self.ypos(), -stockZthickness * 
2554 gcpy             Proportion))
2555 gcpy         i = 1
2556 gcpy         while i < Number_of_Dovetails * 2:
2557 gcpy             print(i)
2558 gcpy             ctp = ctp.union(self.cutlinedxfgc(i * (Joint_Width / (
2559 gcpy                 Number_of_Dovetails * 2)), self.ypos(), -
2560 gcpy                 stockZthickness * Proportion))
2561 gcpy             ctp = ctp.union(self.cutlinedxfgc(i * (Joint_Width / (
2562 gcpy                 Number_of_Dovetails * 2)), (stockZthickness +
2563 gcpy                 Spacing) + (stockZthickness * Proportion) - (
2564 gcpy                     DTT_diameter/2), -(stockZthickness * Proportion)))
2565 gcpy             ctp = ctp.union(self.cutlinedxfgc(i * (Joint_Width / (
2566 gcpy                 Number_of_Dovetails * 2)), stockZthickness + Spacing
2567 gcpy                 / 2, -(stockZthickness * Proportion)))
2568 gcpy             ctp = ctp.union(self.cutlinedxfgc((i + 1) * (
2569 gcpy                 Joint_Width / (Number_of_Dovetails * 2)),
2570 gcpy                 stockZthickness + Spacing/2, -(stockZthickness *
2571 gcpy                 Proportion)))
2572 gcpy             self.dxrectangleround(self.currenttoolnumber(),
2573 gcpy                 i * (Joint_Width / (Number_of_Dovetails * 2))-DTR,
2574 gcpy                 stockZthickness + (Spacing/2) - DTR,
2575 gcpy                 DTR * 2,
2576 gcpy                 (stockZthickness * Proportion) + Spacing/2 + DTR *
2577 gcpy                 2 - (DTT_diameter/2),
2578 gcpy                 DTR)
2579 gcpy             i += 2
2580 gcpy             self.rapidZ(0)
2581 gcpy             return ctp

```

---

and

---

```

2582 gcpy     def cut_tails(self, Joint_Width, stockZthickness,
2583 gcpy         Number_of_Dovetails, Spacing, Proportion, DTT_diameter,
2584 gcpy         DTT_angle):
2585 gcpy         DTO = Tan(DTT_angle) * (stockZthickness * Proportion)
2586 gcpy         DTR = DTT_diameter/2 - DTO
2587 gcpy         cpr = self.rapidXY(0, 0)
2588 gcpy         ctp = self.cutlinedxfgc(self.xpos(), self.ypos(), -
2589 gcpy             stockZthickness * Proportion)
2590 gcpy         ctp = ctp.union(self.cutlinedxfgc(
2591 gcpy             Joint_Width / (Number_of_Dovetails * 2) - (DTT_diameter
2592 gcpy             - DTO),
2593 gcpy             self.ypos(),
2594 gcpy             -stockZthickness * Proportion))
2595 gcpy         i = 1
2596 gcpy         while i < Number_of_Dovetails * 2:
2597 gcpy             ctp = ctp.union(self.cutlinedxfgc(
2598 gcpy                 i * (Joint_Width / (Number_of_Dovetails * 2)) - (
2599 gcpy                     DTT_diameter - DTO),
2600 gcpy                     stockZthickness * Proportion - DTT_diameter / 2,
2601 gcpy                     -(stockZthickness * Proportion)))
2602 gcpy             ctp = ctp.union(self.cutarcCWdx(180, 90,
2603 gcpy                 i * (Joint_Width / (Number_of_Dovetails * 2)),
2604 gcpy                 stockZthickness * Proportion - DTT_diameter / 2,
2605 gcpy                 self.ypos(),
2606 gcpy                 DTT_diameter - DTO, 0, 1))
2607 gcpy             ctp = ctp.union(self.cutarcCWdx(90, 0,
2608 gcpy                 i * (Joint_Width / (Number_of_Dovetails * 2)),
2609 gcpy                 stockZthickness * Proportion - DTT_diameter / 2,
2610 gcpy                 DTT_diameter - DTO, 0, 1))
2611 gcpy             ctp = ctp.union(self.cutlinedxfgc(
2612 gcpy                 i * (Joint_Width / (Number_of_Dovetails * 2)) + (
2613 gcpy                     DTT_diameter - DTO),
2614 gcpy                     0,
2615 gcpy                     -(stockZthickness * Proportion)))
2616 gcpy             ctp = ctp.union(self.cutlinedxfgc(
2617 gcpy                 (i + 2) * (Joint_Width / (Number_of_Dovetails * 2)))

```

---

```

2597 gcpy           - (DTT_diameter - DTO),
2598 gcpy           0,
2599 gcpy           -(stockZthickness * Proportion)))
2600 gcpy           i += 2
2601 gcpy           self.rapidZ(0)
2602 gcpy           self.rapidXY(0, 0)
2603 gcpy           ctp = ctp.union(self.cutlinedxfgc(self.xpos(), self.ypos(),
2604 gcpy           -stockZthickness * Proportion))
2605 gcpy           self.dxfarc(self.currenttoolnumber(), 0, 0, DTR, 180, 270)
2606 gcpy           self.dxfline(self.currenttoolnumber(), -DTR, 0, -DTR,
2607 gcpy           stockZthickness + DTR)
2608 gcpy           self.dxfarc(self.currenttoolnumber(), 0, stockZthickness +
2609 gcpy           DTR, DTR, 90, 180)
2610 gcpy           self.dxfline(self.currenttoolnumber(), 0, stockZthickness +
2611 gcpy           DTR * 2, Joint_Width, stockZthickness + DTR * 2)
2612 gcpy           i = 0
2613 gcpy           while i < Number_of_Dovetails * 2:
2614 gcpy           ctp = ctp.union(self.cutline(i * (Joint_Width /
2615 gcpy           Number_of_Dovetails * 2)), stockZthickness + DTO, -( stockZthickness * Proportion))
2616 gcpy           ctp = ctp.union(self.cutline((i+2) * (Joint_Width /
2617 gcpy           Number_of_Dovetails * 2)), stockZthickness + DTO, -( stockZthickness * Proportion))
2618 gcpy           ctp = ctp.union(self.cutline((i+2) * (Joint_Width /
2619 gcpy           Number_of_Dovetails * 2)), 0, -(stockZthickness *
2620 gcpy           Proportion))
2621 gcpy           self.dxfarc(self.currenttoolnumber(), i * (Joint_Width
2622 gcpy           / (Number_of_Dovetails * 2)), 0, DTR, 270, 360)
2623 gcpy           self.dxfline(self.currenttoolnumber(),
2624 gcpy           i * (Joint_Width / (Number_of_Dovetails * 2)) + DTR
2625 gcpy           ,
2626 gcpy           0,
2627 gcpy           (i + 2) * (Joint_Width / (Number_of_Dovetails * 2)) -
2628 gcpy           - DTR,
2629 gcpy           self.dxfarc(self.currenttoolnumber(), (i + 2) * (
2630 gcpy           Joint_Width / (Number_of_Dovetails * 2)),
2631 gcpy           stockZthickness * Proportion - DTT_diameter / 2,
2632 gcpy           (Joint_Width / (Number_of_Dovetails * 2)) - DTR, 90,
2633 gcpy           180)
2634 gcpy           self.dxfarc(self.currenttoolnumber(), (i + 1) * (
2635 gcpy           Joint_Width / (Number_of_Dovetails * 2)),
2636 gcpy           stockZthickness * Proportion - DTT_diameter / 2,
2637 gcpy           (Joint_Width / (Number_of_Dovetails * 2)) - DTR, 0,
2638 gcpy           90)
2639 gcpy           self.dxfline(self.currenttoolnumber(),
2640 gcpy           (i + 2) * (Joint_Width / (Number_of_Dovetails * 2)) -
2641 gcpy           - DTR,
2642 gcpy           0,
2643 gcpy           (i + 2) * (Joint_Width / (Number_of_Dovetails * 2)) -
2644 gcpy           - DTR, stockZthickness * Proportion -
2645 gcpy           DTT_diameter / 2)
2646 gcpy           self.dxfarc(self.currenttoolnumber(), (i + 2) * (
2647 gcpy           Joint_Width / (Number_of_Dovetails * 2)), 0, DTR,
2648 gcpy           180, 270)
2649 gcpy           i += 2
2650 gcpy           self.dxfarc(self.currenttoolnumber(), Joint_Width,
2651 gcpy           stockZthickness + DTR, DTR, 0, 90)
2652 gcpy           self.dxfline(self.currenttoolnumber(), Joint_Width + DTR,
2653 gcpy           stockZthickness + DTR, Joint_Width + DTR, 0)
2654 gcpy           self.dxfarc(self.currenttoolnumber(), Joint_Width, 0, DTR,
2655 gcpy           270, 360)
2656 gcpy           return ctp

```

---

which are used as:

```

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

```

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

**3.8.1.1.5 Full-blind box joints** BlocksCAD project page at: <https://www.blockscad3d.com/community/projects/1943966> and discussion at: <https://community.carbide3d.com/t/full-blind-box-joints-in-carbide-create/53329>

Full-blind box joints will require 3 separate tools:

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

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

- horizontal
- vertical

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

---

```

2630 gcpy      def Full_Blind_Finger_Joint_square(self, bx, by, orientation,
2631 gcpy          side, width, thickness, Number_of_Pins, largeVdiameter,
2632 gcpy          smallDiameter, normalormirror = "Default"):
2633 gcpy      # Joint_Orientation = "Horizontal" "Even" == "Lower", "Odd" ==
2634 gcpy          "Upper"
2635 gcpy      # Joint_Orientation = "Vertical" "Even" == "Left", "Odd" == "
2636 gcpy          "Right"
2637 gcpy      if (orientation == "Vertical"):
2638 gcpy          if (normalormirror == "Default" and side != "Both"):
2639 gcpy              if (side == "Left"):
2640 gcpy                  normalormirror = "Even"
2641 gcpy              if (side == "Right"):
2642 gcpy                  normalormirror = "Odd"
2643 gcpy      if (orientation == "Horizontal"):
2644 gcpy          if (normalormirror == "Default" and side != "Both"):
2645 gcpy              if (side == "Lower"):
2646 gcpy                  normalormirror = "Even"
2647 gcpy              if (side == "Upper"):
2648 gcpy                  normalormirror = "Odd"
2649 gcpy      Finger_Width = ((Number_of_Pins * 2) - 1) * smallDiameter *
2650 gcpy          1.1
2651 gcpy      Finger_Origin = width/2 - Finger_Width/2
2652 gcpy      rapid = self.rapidZ(0)
2653 gcpy      self.setdxfcolor("Cyan")
2654 gcpy      rapid = rapid.union(self.rapidXY(bx, by))
2655 gcpy      toolpath = (self.Finger_Joint_square(bx, by, orientation,
2656 gcpy          side, width, thickness, Number_of_Pins, Finger_Origin,
2657 gcpy          smallDiameter))
2658 gcpy      if (orientation == "Vertical"):
2659 gcpy          if (side == "Both"):
2660 gcpy              toolpath = self.cutrectangleroundddxf(self.
2661 gcpy                  currenttoolnum, bx - (thickness - smallDiameter
2662 gcpy                      /2), by-smallDiameter/2, 0, (thickness * 2) -
2663 gcpy                      smallDiameter, width+smallDiameter,
2664 gcpy                      (smallDiameter / 2) / Tan(45), smallDiameter/2)
2665 gcpy          if (side == "Left"):
2666 gcpy              toolpath = self.cutrectangleroundddxf(self.
2667 gcpy                  currenttoolnum, bx - (smallDiameter/2), by-
2668 gcpy                  smallDiameter/2, 0, thickness, width+
2669 gcpy                  smallDiameter, ((smallDiameter / 2) / Tan(45)),
2670 gcpy                  smallDiameter/2)
2671 gcpy          if (side == "Right"):
2672 gcpy              toolpath = self.cutrectangleroundddxf(

```

```

2673 gcpy          bx - (smallDiameter/2),
2674 gcpy          by - smallDiameter/2,
2675 gcpy          0,
2676 gcpy          width+smallDiameter,
2677 gcpy          thickness,
2678 gcpy          ((smallDiameter / 2) / Tan(45)),
2679 gcpy          smallDiameter/2)
2680 gcpy          if (side == "Upper"):
2681 gcpy          toolpath = self.cutrectanglerounddxf(
2682 gcpy          self.currenttoolnum,
2683 gcpy          bx - smallDiameter/2,
2684 gcpy          by - (thickness - smallDiameter/2),
2685 gcpy          0,
2686 gcpy          width+smallDiameter,
2687 gcpy          thickness,
2688 gcpy          ((smallDiameter / 2) / Tan(45)),
2689 gcpy          smallDiameter/2)
2690 gcpy          toolpath = toolpath.union(self.Finger_Joint_square(bx, by,
2691 gcpy          orientation, side, width, thickness, Number_of_Pins,
2692 gcpy          Finger_Origin, smallDiameter))
2693 gcpy          return toolpath

def Finger_Joint_square(self, bx, by, orientation, side, width,
2694 gcpy          thickness, Number_of_Pins, Finger_Origin, smallDiameter,
2695 gcpy          normalormirror = "Default"):
2696 gcpy          jointdepth = -(thickness - (smallDiameter / 2) / Tan(45))
# Joint_Orientation = "Horizontal" "Even" == "Lower", "Odd" ==
2697 gcpy          "Upper"
# Joint_Orientation = "Vertical" "Even" == "Left", "Odd" == "
2698 gcpy          Right"
2699 gcpy          if (orientation == "Vertical"):
2700 gcpy          if (normalormirror == "Default" and side != "Both"):
2701 gcpy          if (side == "Left"):
2702 gcpy          normalormirror = "Even"
2703 gcpy          if (side == "Right"):
2704 gcpy          normalormirror = "Odd"
2705 gcpy          if (orientation == "Horizontal"):
2706 gcpy          if (normalormirror == "Default" and side != "Both"):
2707 gcpy          if (side == "Lower"):
2708 gcpy          normalormirror = "Even"
2709 gcpy          if (side == "Upper"):
2710 gcpy          normalormirror = "Odd"
2711 gcpy          radius = smallDiameter/2
2712 gcpy          jointwidth = thickness - smallDiameter
2713 gcpy          toolpath = self.currenttool()
2714 gcpy          rapid = self.rapidZ(0)
2715 gcpy          self.setdxfcolor("Blue")
2716 gcpy          toolpath = toolpath.union(self.cutlineZgcfeed(jointdepth
2717 gcpy          ,1000))
2718 gcpy          self.beginpolyline(self.currenttool())
2719 gcpy          if (orientation == "Vertical"):
2720 gcpy          rapid = rapid.union(self.rapidXY(bx, by + Finger_Origin
2721 gcpy          ))
2722 gcpy          self.addvertex(self.currenttoolnumber(), self.xpos(),
2723 gcpy          self.ypos())
2724 gcpy          toolpath = toolpath.union(self.cutlineZgcfeed(
2725 gcpy          jointdepth,1000))
2726 gcpy          i = 0
2727 gcpy          while i <= Number_of_Pins - 1:
2728 gcpy          if (side == "Right"):
2729 gcpy          toolpath = toolpath.union(self.cutvertexdxf(

```

```

                jointdepth))
2730 gcpy   if (side == "Left"):
2731 gcpy     toolpath = toolpath.union(self.cutvertexdxf(
2732 gcpy       self.xpos(), self.ypos() + smallDiameter +
2733 gcpy         radius/5, jointdepth))
2734 gcpy   #   if (side == "Right" or side == "Both"):
2735 gcpy     i < (Number_of_Pins - 1)):
2736 gcpy       print(i)
2737 gcpy     toolpath = toolpath.union(self.cutvertexdxf(
2738 gcpy       (self.xpos(), self.ypos() + radius,
2739 gcpy         jointdepth))
2740 gcpy     toolpath = toolpath.union(self.cutvertexdxf(
2741 gcpy       (self.xpos() - jointwidth, self.ypos(),
2742 gcpy         jointdepth))
2743 gcpy     toolpath = toolpath.union(self.cutvertexdxf(
2744 gcpy       (self.xpos() + jointwidth, self.ypos(),
2745 gcpy         jointdepth))
2746 gcpy     toolpath = toolpath.union(self.cutvertexdxf(
2747 gcpy       (self.xpos() + radius, self.ypos(),
2748 gcpy         jointdepth))
2749 gcpy     i += 1
# Joint_Orientation = "Horizontal" "Even" == "Lower", "Odd" ==
2750 gcpy     "Upper"
2751 gcpy   if (orientation == "Horizontal"):
2752 gcpy     rapid = rapid.union(self.rapidXY(bx + Finger_Origin, by
2753 gcpy       ))
2754 gcpy     self.addvertex(self.currenttoolnumber(), self.xpos(),
2755 gcpy       self.ypos())
2756 gcpy     toolpath = toolpath.union(self.cutlineZgcfeed(
2757 gcpy       jointdepth,1000))
2758 gcpy     i = 0
2759 gcpy   while i <= Number_of_Pins - 1:
2760 gcpy     if (side == "Upper"):
2761 gcpy       toolpath = toolpath.union(self.cutvertexdxf(
2762 gcpy         self.xpos() + smallDiameter + radius/5, self
2763 gcpy           .ypos(), jointdepth))
2764 gcpy     if (side == "Lower" or side == "Both"):
2765 gcpy       toolpath = toolpath.union(self.cutvertexdxf(
2766 gcpy         self.xpos() + radius, self.ypos(),
2767 gcpy           jointdepth))
2768 gcpy     if (side == "Lower"):
2769 gcpy       toolpath = toolpath.union(self.cutvertexdxf(
2770 gcpy         self.xpos() + smallDiameter + radius/5, self
2771 gcpy           .ypos(), jointdepth))
2772 gcpy     if (side == "Upper" or side == "Both"):
2773 gcpy       i < (Number_of_Pins - 1)):
2774 gcpy         print(i)
2775 gcpy       toolpath = toolpath.union(self.cutvertexdxf(
2776 gcpy         (self.xpos() + radius, self.ypos(),
2777 gcpy           jointdepth))
2778 gcpy     toolpath = toolpath.union(self.cutvertexdxf(
2779 gcpy       (self.xpos(), self.ypos() - jointwidth,
2780 gcpy         jointdepth))
2781 gcpy     toolpath = toolpath.union(self.cutvertexdxf(
2782 gcpy       (self.xpos() + radius, self.ypos(),
2783 gcpy         jointdepth))
2784 gcpy     if (side == "Lower"):
2785 gcpy       toolpath = toolpath.union(self.cutvertexdxf(
2786 gcpy         self.xpos() + smallDiameter + radius/5, self
2787 gcpy           .ypos(), jointdepth))
2788 gcpy     if (side == "Upper" or side == "Both"):
2789 gcpy       i < (Number_of_Pins - 1)):
2790 gcpy         print(i)
2791 gcpy       toolpath = toolpath.union(self.cutvertexdxf(
2792 gcpy         (self.xpos() + radius, self.ypos(),
2793 gcpy           jointdepth))
2794 gcpy     toolpath = toolpath.union(self.cutvertexdxf(
2795 gcpy       (self.xpos(), self.ypos() + jointwidth,
2796 gcpy         jointdepth))
2797 gcpy     toolpath = toolpath.union(self.cutvertexdxf(
2798 gcpy       (self.xpos() + radius, self.ypos(),
2799 gcpy         jointdepth)))
2800 gcpy     i += 1

```

```

2767 gcpy         self.closepolyline(self.currenttoolnumber())
2768 gcpy         return toolpath
2769 gcpy
2770 gcpy     def Full_Blind_Finger_Joint_smallV(self, bx, by, orientation,
2771 gcpy             side, width, thickness, Number_of_Pins, largeVdiameter,
2772 gcpy             smallDiameter):
2773 gcpy             rapid = self.rapidZ(0)
2774 gcpy             #   rapid = rapid.union(self.rapidXY(bx, by))
2775 gcpy             self.setdxfcolor("Red")
2776 gcpy             if (orientation == "Vertical"):
2777 gcpy                 rapid = rapid.union(self.rapidXY(bx, by - smallDiameter
2778 gcpy                     /6))
2779 gcpy                 toolpath = self.cutlineZgcfeed(-thickness,1000)
2780 gcpy                 toolpath = self.cutlinedxfgc(bx, by + width +
2781 gcpy                     smallDiameter/6, - thickness)
2782 gcpy             if (orientation == "Horizontal"):
2783 gcpy                 rapid = rapid.union(self.rapidXY(bx - smallDiameter/6,
2784 gcpy                     by))
2785 gcpy                 toolpath = self.cutlineZgcfeed(-thickness,1000)
2786 gcpy                 toolpath = self.cutlinedxfgc(bx + width + smallDiameter
2787 gcpy                     /6, by, -thickness)
2788 gcpy             #   rapid = self.rapidZ(0)
2789 gcpy
2790 gcpy             return toolpath
2791 gcpy
2792 gcpy     def Full_Blind_Finger_Joint_largeV(self, bx, by, orientation,
2793 gcpy             side, width, thickness, Number_of_Pins, largeVdiameter,
2794 gcpy             smallDiameter):
2795 gcpy             radius = smallDiameter/2
2796 gcpy             rapid = self.rapidZ(0)
2797 gcpy             Finger_Width = ((Number_of_Pins * 2) - 1) * smallDiameter *
2798 gcpy                 1.1
2799 gcpy             Finger_Origin = width/2 - Finger_Width/2
2800 gcpy             #   rapid = rapid.union(self.rapidXY(bx, by))
2801 gcpy             # Joint_Orientation = "Horizontal" "Even" == "Lower", "Odd" ==
2802 gcpy                 "Upper"
2803 gcpy             # Joint_Orientation = "Vertical" "Even" == "Left", "Odd" == "
2804 gcpy                 "Right"
2805 gcpy             if (orientation == "Vertical"):
2806 gcpy                 rapid = rapid.union(self.rapidXY(bx, by))
2807 gcpy                 toolpath = self.cutlineZgcfeed(-thickness,1000)
2808 gcpy                 toolpath = toolpath.union(self.cutlinedxfgc(bx, by +
2809 gcpy                     Finger_Origin, -thickness))
2810 gcpy                 rapid = self.rapidZ(0)
2811 gcpy                 rapid = rapid.union(self.rapidXY(bx, by + width -
2812 gcpy                     Finger_Origin))
2813 gcpy                 self.setdxfcolor("Blue")
2814 gcpy                 toolpath = toolpath.union(self.cutlineZgcfeed(-
2815 gcpy                     thickness,1000))
2816 gcpy                 toolpath = toolpath.union(self.cutlinedxfgc(bx,
2817 gcpy                     by + width, -thickness))
2818 gcpy             if (side == "Left" or side == "Both"):
2819 gcpy                 rapid = self.rapidZ(0)
2820 gcpy                 self.setdxfcolor("DarkGray")
2821 gcpy                 rapid = rapid.union(self.rapidXY(bx+thickness -(
2822 gcpy                     smallDiameter / 2) / Tan(45), by - radius/2))
2823 gcpy                 toolpath = toolpath.union(self.cutlineZgcfeed(-(
2824 gcpy                     smallDiameter / 2) / Tan(45),10000))
2825 gcpy                 toolpath = toolpath.union(self.cutlinedxfgc(bx+
2826 gcpy                     thickness-(smallDiameter / 2) / Tan(45), by +
2827 gcpy                     width + radius/2, -(smallDiameter / 2) / Tan(45)
2828 gcpy                     ))
2829 gcpy                 rapid = self.rapidZ(0)
2830 gcpy                 self.setdxfcolor("Green")
2831 gcpy                 rapid = rapid.union(self.rapidXY(bx+thickness/2, by
2832 gcpy                     +width))
2833 gcpy                 toolpath = toolpath.union(self.cutlineZgcfeed(-
2834 gcpy                     thickness/2,1000))
2835 gcpy                 toolpath = toolpath.union(self.cutlinedxfgc(bx+
2836 gcpy                     thickness/2, by + width -thickness, -thickness
2837 gcpy                     /2))
2838 gcpy                 rapid = self.rapidZ(0)
2839 gcpy                 rapid = rapid.union(self.rapidXY(bx+thickness/2, by
2840 gcpy                     ))
2841 gcpy                 toolpath = toolpath.union(self.cutlineZgcfeed(-
2842 gcpy                     thickness/2,1000))
2843 gcpy                 toolpath = toolpath.union(self.cutlinedxfgc(bx+
2844 gcpy                     thickness/2, by +thickness, -thickness/2))

```

```

2818 gcpy
2819 gcpy
2820 gcpy
2821 gcpy
2822 gcpy
2823 gcpy
2824 gcpy
2825 gcpy
2826 gcpy
2827 gcpy
2828 gcpy
2829 gcpy
2830 gcpy
2831 gcpy
2832 gcpy
2833 gcpy
# 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)))
        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)))
        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)

```

```

2865 gcpy           self.setdxfcolor("Green")
2866 gcpy           rapid = rapid.union(self.rapidXY(bx+width, by-
2867 gcpy           thickness/2))
2868 gcpy           toolpath = toolpath.union(self.cutlineZgcfeed(-
2869 gcpy           thickness/2,1000))
2870 gcpy           toolpath = toolpath.union(self.cutlinedxfgc(bx +
2871 gcpy           width -thickness, by-thickness/2, -thickness/2))
2872 gcpy           rapid = self.rapidZ(0)
2873 gcpy           rapid = rapid.union(self.rapidXY(bx, by-thickness
2874 gcpy           /2))
2875 gcpy           toolpath = toolpath.union(self.cutlineZgcfeed(-
2876 gcpy           thickness/2,1000))
2877 gcpy           toolpath = toolpath.union(self.cutlinedxfgc(bx +
2878 gcpy           thickness, by-thickness/2, -thickness/2))
2879 gcpy           rapid = self.rapidZ(0)
2880 gcpy           return toolpath

2881 gcpy       def Full_Blind_Finger_Joint(self, bx, by, orientation, side,
2882 gcpy           width, thickness, largeVdiameter, smallDiameter,
2883 gcpy           normalormirror = "Default", squaretool = 102, smallV = 390,
2884 gcpy           largeV = 301):
2885 gcpy           Number_of_Pins = int((width - thickness * 2) / (
2886 gcpy               smallDiameter * 2.2) / 2) + 0.0) * 2 + 1
2887 gcpy           print("Number of Pins: ", Number_of_Pins)
2888 gcpy           self.movetosafeZ()
2889 gcpy           self.toolchange(squaretool, 17000)
2890 gcpy           toolpath = self.Full_Blind_Finger_Joint_square(bx, by,
2891 gcpy               orientation, side, width, thickness, Number_of_Pins,
2892 gcpy               largeVdiameter, smallDiameter)
2893 gcpy           self.movetosafeZ()
2894 gcpy           self.toolchange(smallV, 17000)
2895 gcpy           toolpath = toolpath.union(self.
2896 gcpy               Full_Blind_Finger_Joint_smallV(bx, by, orientation, side
2897 gcpy               , width, thickness, Number_of_Pins, largeVdiameter,
2898 gcpy               smallDiameter))
2899 gcpy           self.toolchange(largeV, 17000)
2900 gcpy           toolpath = toolpath.union(self.
2901 gcpy               Full_Blind_Finger_Joint_largeV(bx, by, orientation, side
2902 gcpy               , width, thickness, Number_of_Pins, largeVdiameter,
2903 gcpy               smallDiameter))
2904 gcpy           return toolpath

```

---

### 3.9 (Reading) G-code Files

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

First, a template file will be necessary:

```

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

---

```

2889 gcpy      def previewgcodefile(self, gc_file):
2890 gcpy          gc_file = open(gc_file, 'r')
2891 gcpy          gcfilecontents = []
2892 gcpy          with gc_file as file:
2893 gcpy              for line in file:
2894 gcpy                  command = line
2895 gcpy                  gcfilecontents.append(line)
2896 gcpy
2897 gcpy          numlinesfound = 0
2898 gcpy          for line in gcfilecontents:
2899 gcpy              print(line)
2900 gcpy              if line[:10] == "(stockMin:":
2901 gcpy                  subdivisions = line.split()
2902 gcpy                  extentleft = float(subdivisions[0][10:-3])
2903 gcpy                  extentfb = float(subdivisions[1][-3:])
2904 gcpy                  extentd = float(subdivisions[2][-3:])
2905 gcpy                  numlinesfound = numlinesfound + 1
2906 gcpy              if line[:13] == "(STOCK/BLOCK,":
2907 gcpy                  subdivisions = line.split()
2908 gcpy                  sizeX = float(subdivisions[0][13:-1])
2909 gcpy                  sizeY = float(subdivisions[1][-1])
2910 gcpy                  sizeZ = float(subdivisions[4][-1])
2911 gcpy                  numlinesfound = numlinesfound + 1
2912 gcpy              if line[:3] == "G21":
2913 gcpy                  units = "mm"
2914 gcpy                  numlinesfound = numlinesfound + 1
2915 gcpy              if numlinesfound >=3:
2916 gcpy                  break
2917 gcpy      print(numlinesfound)

```

---

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

```

2919 gcpy          self.setupcuttingarea(sizeX, sizeY, sizeZ, extentleft,
2920 gcpy                      extentfb, extentd)
2921 gcpy          commands = []
2922 gcpy          for line in gcfilecontents:
2923 gcpy              Xc = 0
2924 gcpy              Yc = 0
2925 gcpy              Zc = 0
2926 gcpy              Fc = 0
2927 gcpy              Xp = 0.0
2928 gcpy              Yp = 0.0
2929 gcpy              Zp = 0.0
2930 gcpy              if line == "G53G0Z-5.000\n":
2931 gcpy                  self.movetosafeZ()
2932 gcpy              if line[:3] == "M6T":
2933 gcpy                  tool = int(line[3:])
2934 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)

```

---

```

2935 gcpy           if line[:2] == "G0":
2936 gcpy           machinestate = "rapid"
2937 gcpy           if line[:2] == "G1":
2938 gcpy           machinestate = "cutline"
2939 gcpy           if line[:2] == "G0" or line[:2] == "G1" or line[:1] ==
2940 gcpy             "X" or line[:1] == "Y" or line[:1] == "Z":
2941 gcpy             if "F" in line:
2942 gcpy               Fplus = line.split("F")
2943 gcpy               Fc = 1
2944 gcpy               fr = float(Fplus[1])
2945 gcpy               line = Fplus[0]
2946 gcpy           if "Z" in line:
2947 gcpy               Zplus = line.split("Z")
2948 gcpy               Zc = 1
2949 gcpy               Zp = float(Zplus[1])
2950 gcpy               line = Zplus[0]
2951 gcpy           if "Y" in line:
2952 gcpy               Yplus = line.split("Y")
2953 gcpy               Yc = 1
2954 gcpy               Yp = float(Yplus[1])
2955 gcpy               line = Yplus[0]
2956 gcpy           if "X" in line:
2957 gcpy               Xplus = line.split("X")
2958 gcpy               Xc = 1
2959 gcpy               Xp = float(Xplus[1])
2960 gcpy           if Zc == 1:
2961 gcpy             if Yc == 1:
2962 gcpy               if machinestate == "rapid":
2963 gcpy                 command = "rapidXYZ(" + str(Xp) + "
2964 gcpy                   ", " + str(Yp) + ", " + str(Zp) +
2965 gcpy                   ")"
2966 gcpy                 self.rapidXYZ(Xp, Yp, Zp)
2967 gcpy             else:
2968 gcpy               command = "cutlineXYZ(" + str(Xp) +
2969 gcpy                 ", " + str(Yp) + ", " + str(Zp) +
2970 gcpy                 ")"
2971 gcpy               self.cutlineXYZ(Xp, Yp, Zp)
2972 gcpy             else:
2973 gcpy               if machinestate == "rapid":
2974 gcpy                 command = "rapidYZ(" + str(Yp) + ",
2975 gcpy                   ", " + str(Zp) + ")"
2976 gcpy                 self.rapidYZ(Yp, Zp)
2977 gcpy             else:
2978 gcpy               command = "cutlineYZ(" + str(Yp) +
2979 gcpy                 ", " + str(Zp) + ")"
2980 gcpy               self.cutlineYZ(Yp, Zp)
2981 gcpy             else:
2982 gcpy               if Xc == 1:
2983 gcpy                 if machinestate == "rapid":
2984 gcpy                   command = "rapidXZ(" + str(Xp) + ",
2985 gcpy                     ", " + str(Zp) + ")"
2986 gcpy                     self.rapidXZ(Xp, Zp)
2987 gcpy             else:
2988 gcpy               command = "cutlineXZ(" + str(Xp) +
2989 gcpy                 ", " + str(Zp) + ")"
2990 gcpy               self.cutlineXZ(Xp, Zp)
2991 gcpy             else:
2992 gcpy               if Yc == 1:
2993 gcpy                 if machinestate == "rapid":
2994 gcpy                   command = "rapidXY(" + str(Xp) + ",
2995 gcpy                     ", " + str(Yp) + ")"
2996 gcpy                     self.rapidXY(Xp, Yp)
2997 gcpy             else:
2998 gcpy               command = "cutlineXY(" + str(Xp) +
2999 gcpy                 ", " + str(Yp) + ")"
3000 gcpy               self.cutlineXY(Xp, Yp)

```

```

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 #
3017 gcpy #
3018 gcpy #
3019 gcpy #
3020 gcpy
3021 gcpy #     for command in commands:
3022 gcpy #         print(command)
3023 gcpy
3024 gcpy #     show(self.stockandtoolpaths())
3025 gcpy     self.stockandtoolpaths()

```

---

## 4 Notes

### 4.1 Other Resources

#### 4.1.1 Coding Style

A notable influence on the coding style in this project is John Ousterhout's *A Philosophy of Software Design*[SoftwareDesign]. Complexity is managed by the overall design and structure of the code, structuring it so that each component may be worked with on an individual basis, hiding the maximum information, and exposing the maximum functionality, with names selected so as to express their functionality/usage.

Red Flags to avoid include:

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

#### 4.1.2 Coding References

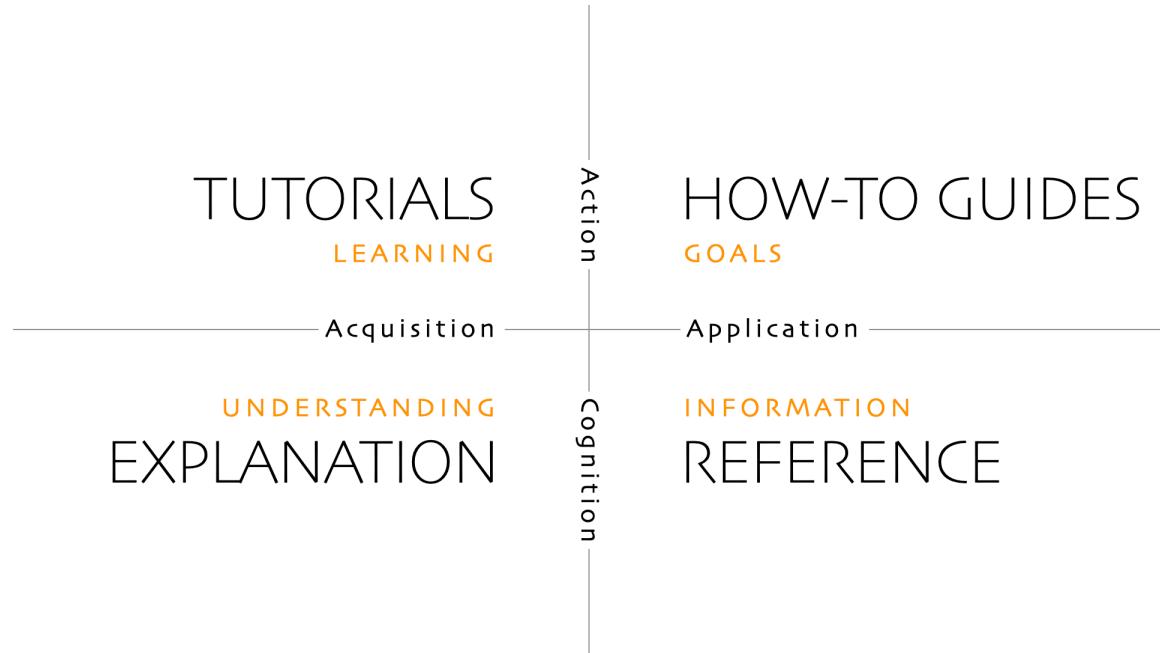
<https://thewhitetulip.gitbook.io/py/06-file-handling>

### 4.1.3 Documentation Style

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

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

resulting in a matrix of:



where:

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

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

### 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:

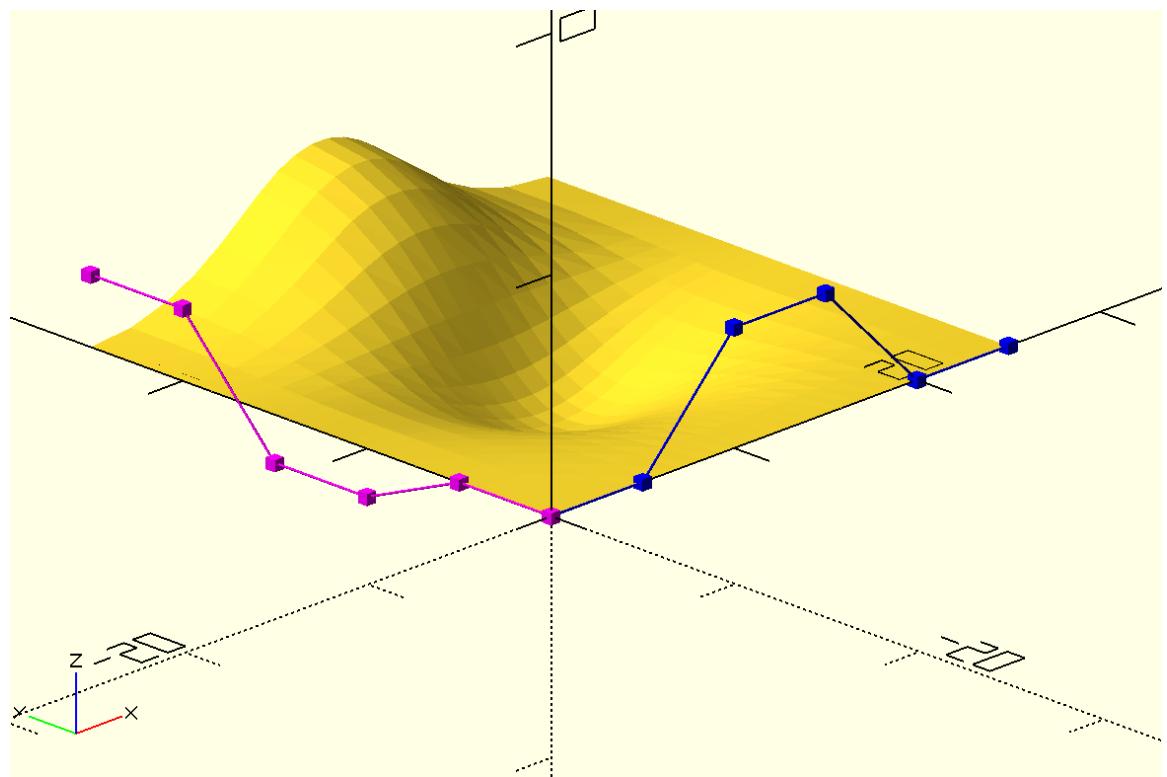
3 planes \* 3 Béziers \* (2 on-curve + 2 off-curve points) == 36 coordinate pairs

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  
closedxffile, 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, 85  
dxfwrite, 84  
endmill square, 54  
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  
opendxffile, 81  
opengcodefile, 81, 82  
plunge, 69  
previewgcodefile, 113  
ra, 34  
rapid, 60  
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, 85  
writedxfKH, 84  
writedxflbl, 84  
writedxfllsq, 84  
writedxfllV, 84  
writedxfsmbl, 84  
writedxfsmfq, 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, 85  
dxfwrite, 84  
endmill square, 54  
endmill v, 54  
gcodepreview, 31, 34  
gcp.setupstock, 34  
init, 31  
opendxffile, 81  
opengcodefile, 81, 82  
previewgcodefile, 113  
rapid, 60  
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, 85  
writedxfKH, 84  
writedxflbl, 84  
writedxfgsq, 84  
writedxfV, 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