The gcodepreview PythonSCAD library[*]

Author: William F. Adams

`willadams at aol dot com`

2025/07/4

Abstract

The gcodepreview library allows using PythonSCAD (OpenPythonSCAD) to move a tool in lines and arcs and output DXF and G-code files so as to work as a CAD/CAM program for CNC.
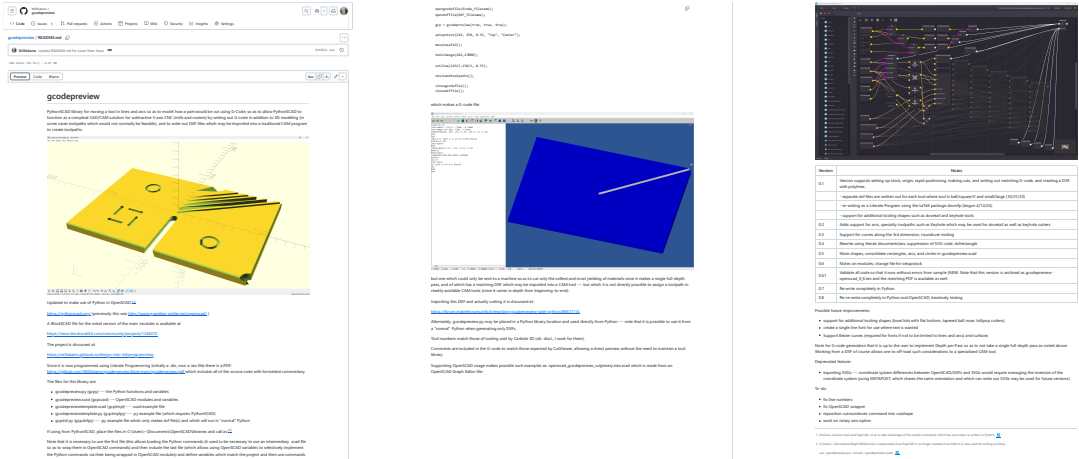
Contents

---

[*]This file (`gcodepreview`) has version number v0.9, last revised 2025/07/4.

1   readme.md


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

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

```
 86 rdme
 87 rdme | Version      | Notes         |
 88 rdme | ------------ | ------------- |
 89 rdme | 0.1          | Version  supports setting up stock, origin, rapid
         positioning, making cuts, and writing out matching G-code, and
         creating a DXF with polylines.                          |
 90 rdme |              | - separate dxf files are written out for each
         tool where tool is ball/square/V and small/large (10/31/23)
         |
 91 rdme |              | - re-writing as a Literate Program using the
         LaTeX package docmfp (begun 4/12/24)
         |
 92 rdme |              | - support for additional tooling shapes such as
         dovetail and keyhole tools
         |
 93 rdme | 0.2          | Adds support for arcs, specialty toolpaths such
         as Keyhole which may be used for dovetail as well as keyhole
         cutters
                                                                 |
 94 rdme | 0.3          | Support for curves along the 3rd dimension,
         roundover tooling
         |
 95 rdme | 0.4          | Rewrite using literati documentclass, suppression
          of SVG code, dxfrectangle
         |
 96 rdme | 0.5          | More shapes, consolidate rectangles, arcs, and
         circles in gcodepreview.scad
         |
 97 rdme | 0.6          | Notes on modules, change file for setupstock
         |
 98 rdme | 0.61         | Validate all code so that it runs without errors
         from sample (NEW: Note that this version is archived as
         gcodepreview-openscad_0_6.tex and the matching PDF is available
         as well|
 99 rdme | 0.7          | Re-write completely in Python
         |
100 rdme | 0.8          | Re-re-write completely in Python and OpenSCAD,
         iteratively testing
         |
101 rdme | 0.801        | Add support for bowl bits with flat bottom
         |
102 rdme | 0.802        | Add support for tapered ball-nose and  V tools
         with flat bottom
         |
103 rdme | 0.803        | Implement initial color support and joinery
         modules (dovetail and full blind box joint modules)
         |
104 rdme | 0.9          | Re-write to use Python lists for 3D shapes for
         toolpaths and rapids.                                   |
105 rdme
106 rdme Possible future improvements:
107 rdme
108 rdme  - support for post-processors
109 rdme  - support for 4th-axis
110 rdme  - support for two-sided machining (import an STL or other file to
         use for stock, or possibly preserve the state after one cut and
          then rotate the cut stock/part)
111 rdme  - support for additional tooling shapes (lollipop cutters)
112 rdme  - create a single line font for use where text is wanted
113 rdme  - Support Bézier curves (required for fonts if not to be limited
          to lines and arcs) and surfaces
114 rdme
115 rdme Note for G-code generation that it is up to the user to implement
         Depth per Pass so as to not take a single full-depth pass as
         noted above. Working from a DXF of course allows one to off-load
          such considerations to a specialized CAM tool.
```

```
116 rdme
117 rdme To-do:
118 rdme
119 rdme  - clock-wise arcs
120 rdme  - add toolpath for cutting countersinks using ball-nose tool from
              inside working out
121 rdme  - verify OpenSCAD wrapper and add any missing commands for Python
122 rdme  - verify support for shaft on tooling
123 rdme  - create a folder of template and sample files
124 rdme  - clean up/comment out all mentions of previous versions/features/
              implementations/deprecated features
125 rdme  - fully implement/verify describing/saving/loading tools using
              CutViewer comments
126 rdme
127 rdme Deprecated feature:
128 rdme
129 rdme  - exporting SVGs --- coordinate system differences between
              OpenSCAD/DXFs and SVGs would require managing the inversion of
              the coordinate system (using METAPOST, which shares the same
              orientation and which can write out SVGs may be used for future
              versions)
```

2 Usage and Templates

The gcodepreview library allows the modeling of 2D geometry and 3D shapes using Python or by calling Python from within Open(Python)SCAD, enabling the creation of 2D DXFs, G-code (which cuts a 2D or 3D part), or 3D models as a preview of how the file will cut. These abilities may be accessed in "plain" Python (to make DXFs), or Python or OpenSCAD in PythonSCAD (to make DXFs, and/or G-code with 3D modeling) for a preview. Providing them in a programmatic context allows making parts or design elements of parts (*e.g.*, joinery) which would be tedious or difficult (or verging on impossible) to draw by hand in a traditional CAD or vector drawing application. A further consideration is that this is "Design for Manufacture" taken to its ultimate extreme, and that a part so designed is inherently manufacturable (so long as the dimensions and radii allows for reasonable tool (and toolpath) geometries).

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

It should be that the readme at the project page which serves as an overview, and this section (which serves as a collection of templates and a tutorial) are all the documentation which most users will need (and arguably is still too much). The balance of the document after this section shows all the code and implementation details, and will where appropriate show examples of usage excerpted from the template files (serving as a how-to guide as well as documenting the code) as well as Indices (which serve as a front-end for reference).
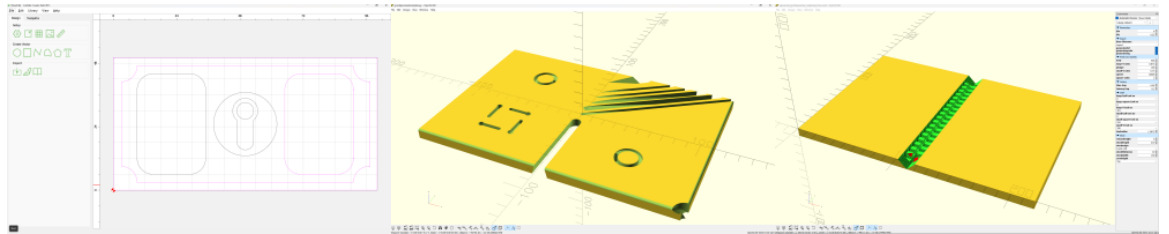
Some comments on the templates:

- minimal — each is intended as a framework for a minimal working example (MWE) — it should be possible to comment out unused/unneeded portions and so arrive at code which tests any aspect of this project and which may be used as a starting point for a new part/project

- compleat — a quite wide variety of tools are listed (and probably more will be added in the future), but pre-defining them and having these "hooks" seems the easiest mechanism to handle the requirements of subtractive machining.

- shortcuts — as the various examples show, while in real life it is necessary to make many passes with a tool, an expedient shortcut is to forgo the `loop` operation and just use a `hull()` operation and avoid the requirement of implementing Depth per Pass (but note that this will lose the previewing of scalloped tool marks in places where they might appear otherwise)

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

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

2.1 gcpdxf.py

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

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

```
 66 gcpdxfpy
 67 gcpdxfpy gcp.setdxfcolor("Black")
 68 gcpdxfpy
 69 gcpdxfpy gcp.beginpolyline(large_square_tool_num)
 70 gcpdxfpy gcp.addvertex(large_square_tool_num, stockXwidth*0.75+radius*1.5,
               stockYheight/4-radius/2)
 71 gcpdxfpy gcp.addvertex(large_square_tool_num, stockXwidth*0.75+radius,
               stockYheight/4-radius/2)
 72 gcpdxfpy gcp.addvertex(large_square_tool_num, stockXwidth*0.75+radius,
               stockYheight*0.75+radius/2)
 73 gcpdxfpy gcp.addvertex(large_square_tool_num, stockXwidth*0.75+radius*1.5,
               stockYheight*0.75+radius/2)
 74 gcpdxfpy gcp.closepolyline(large_square_tool_num)
 75 gcpdxfpy
 76 gcpdxfpy gcp.dxfarc(large_square_tool_num, stockXwidth*0.75+radius*1.5,
               stockYheight*0.75, radius/2,  0, 90)
 77 gcpdxfpy
 78 gcpdxfpy gcp.beginpolyline(large_square_tool_num)
 79 gcpdxfpy gcp.addvertex(large_square_tool_num, stockXwidth*0.75+radius*2,
               stockYheight*0.75)
 80 gcpdxfpy gcp.addvertex(large_square_tool_num, stockXwidth*0.75+radius*2,
               stockYheight/4)
 81 gcpdxfpy gcp.closepolyline(large_square_tool_num)
 82 gcpdxfpy
 83 gcpdxfpy gcp.dxfarc(large_square_tool_num, stockXwidth*0.75+radius*1.5,
               stockYheight/4, radius/2,  270, 360)
 84 gcpdxfpy
 85 gcpdxfpy gcp.setdxfcolor("White")
 86 gcpdxfpy
 87 gcpdxfpy gcp.beginpolyline(large_square_tool_num)
 88 gcpdxfpy gcp.addvertex(large_square_tool_num, stockXwidth*0.25-radius*1.5,
               stockYheight/4-radius/2)
 89 gcpdxfpy gcp.addvertex(large_square_tool_num, stockXwidth*0.25-radius,
               stockYheight/4-radius/2)
 90 gcpdxfpy gcp.addvertex(large_square_tool_num, stockXwidth*0.25-radius,
               stockYheight*0.75+radius/2)
 91 gcpdxfpy gcp.addvertex(large_square_tool_num, stockXwidth*0.25-radius*1.5,
               stockYheight*0.75+radius/2)
 92 gcpdxfpy gcp.closepolyline(large_square_tool_num)
 93 gcpdxfpy
 94 gcpdxfpy gcp.dxfarc(large_square_tool_num, stockXwidth*0.25-radius*1.5,
               stockYheight*0.75, radius/2,  90, 180)
 95 gcpdxfpy
 96 gcpdxfpy gcp.beginpolyline(large_square_tool_num)
 97 gcpdxfpy gcp.addvertex(large_square_tool_num, stockXwidth*0.25-radius*2,
               stockYheight*0.75)
 98 gcpdxfpy gcp.addvertex(large_square_tool_num, stockXwidth*0.25-radius*2,
               stockYheight/4)
 99 gcpdxfpy gcp.closepolyline(large_square_tool_num)
100 gcpdxfpy
101 gcpdxfpy gcp.dxfarc(large_square_tool_num, stockXwidth*0.25-radius*1.5,
               stockYheight/4, radius/2,  180, 270)
102 gcpdxfpy
103 gcpdxfpy gcp.setdxfcolor("Yellow")
104 gcpdxfpy gcp.dxfcircle(large_square_tool_num, stockXwidth/4+1+radius/2,
               stockYheight/4, radius/2)
105 gcpdxfpy
106 gcpdxfpy gcp.setdxfcolor("Green")
107 gcpdxfpy gcp.dxfcircle(large_square_tool_num, stockXwidth*0.75-(1+radius/2),
                stockYheight*0.75, radius/2)
108 gcpdxfpy
109 gcpdxfpy gcp.setdxfcolor("Cyan")
110 gcpdxfpy gcp.dxfcircle(large_square_tool_num, stockXwidth/4+1+radius/2,
               stockYheight*0.75, radius/2)
111 gcpdxfpy
112 gcpdxfpy gcp.setdxfcolor("Magenta")
113 gcpdxfpy gcp.dxfcircle(large_square_tool_num, stockXwidth*0.75-(1+radius/2),
                stockYheight/4, radius/2)
114 gcpdxfpy
115 gcpdxfpy gcp.setdxfcolor("Dark␣Gray")
116 gcpdxfpy
117 gcpdxfpy gcp.dxfcircle(large_square_tool_num, stockXwidth/2, stockYheight/2,
                radius * 2)
118 gcpdxfpy
119 gcpdxfpy gcp.setdxfcolor("Light␣Gray")
120 gcpdxfpy
121 gcpdxfpy gcp.dxfKH(374, stockXwidth/2, stockYheight/5*3, 0, -7, 270,
               11.5875)
```
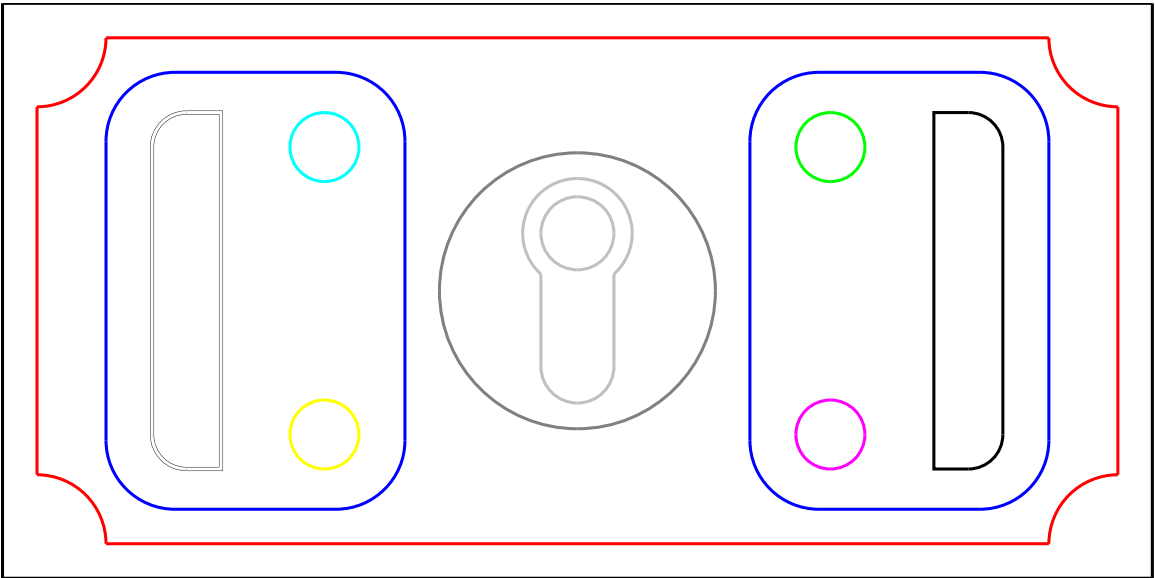
```
122 gcpdxfpy
123 gcpdxfpy gcp.closedxffile()
```
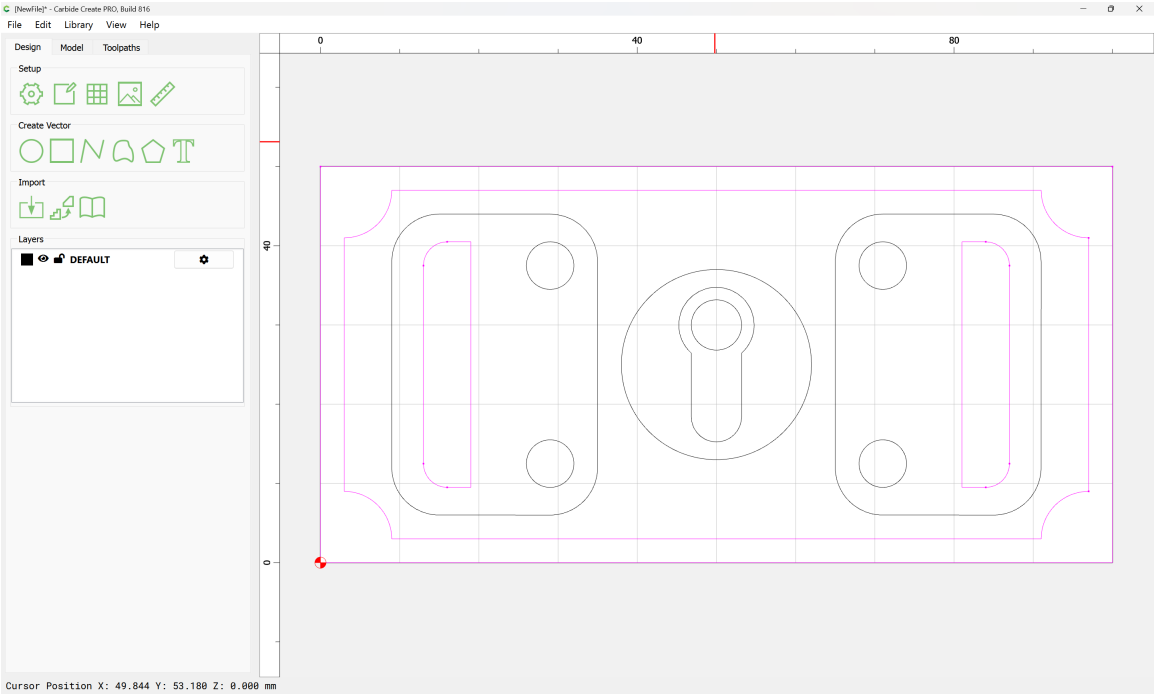
which creates a `.dxf` file which may be imported into any CAD program:
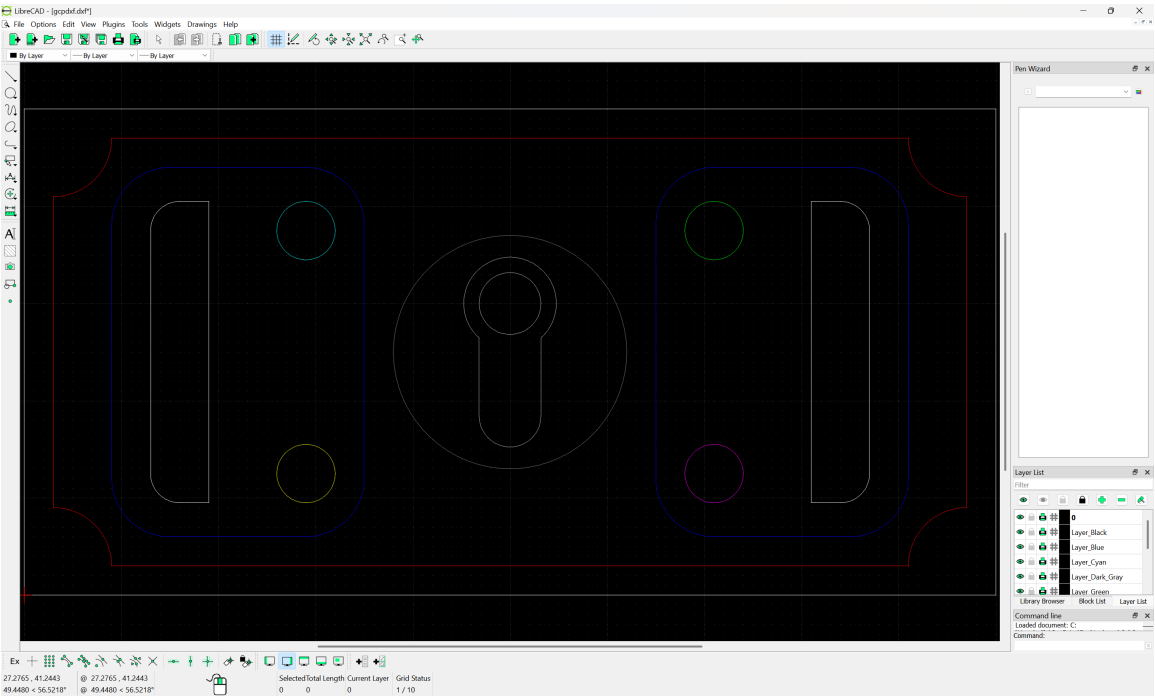
with the appearance (once converted into a `.svg` and then re-saved as a `.pdf` and edited so as to show the white elements):

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

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

- `dxfrectangle` (specify lower-left and upper-right corners)

   `dxfrectangleround` (specified as "Fillet" and radius for the round option)

   `dxfrectanglechamfer` (specified as "Chamfer" and radius for the round option)

   `dxfrectangleflippedfillet` (specified as "Flipped Fillet" and radius for the option)

- `dxfcircle` (specifying their center and radius)

- `dxfline` (specifying begin/end points)

- `dxfarc` (specifying arc center, radius, and beginning/ending angles)

- `dxfKH` (specifying origin, depth, angle, distance)

2.2   gcpcutdxf.py

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

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

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

```
 95 gcpcutdxfpy
 96 gcpcutdxfpy gcp.rapid(stockXwidth/2 + (radius * 2) + inset + radius, radius, 0,
                    "laser")
 97 gcpcutdxfpy
 98 gcpcutdxfpy gcp.cutrectanglerounddxf(large_square_tool_num, stockXwidth/2 + (
                    radius * 2) + inset, radius, 0, stockXwidth/2 - (radius * 4),
                    stockYheight - (radius * 2), -stockZthickness/4, radius)
 99 gcpcutdxfpy
100 gcpcutdxfpy gcp.setdxfcolor("Red")
101 gcpcutdxfpy
102 gcpcutdxfpy gcp.rapid(stockXwidth/2 + radius, stockYheight/2, 0, "laser")
103 gcpcutdxfpy
104 gcpcutdxfpy gcp.toolchange(large_square_tool_num)
105 gcpcutdxfpy
106 gcpcutdxfpy gcp.cutcircleCC(stockXwidth/2, stockYheight/2, 0, -stockZthickness,
                    radius)
107 gcpcutdxfpy
108 gcpcutdxfpy gcp.cutcircleCC(stockXwidth/2, stockYheight/2, -stockZthickness, -
                    stockZthickness, radius*1.5)
109 gcpcutdxfpy
110 gcpcutdxfpy gcp.closedxffile()
111 gcpcutdxfpy
112 gcpcutdxfpy gcp.stockandtoolpaths()
```

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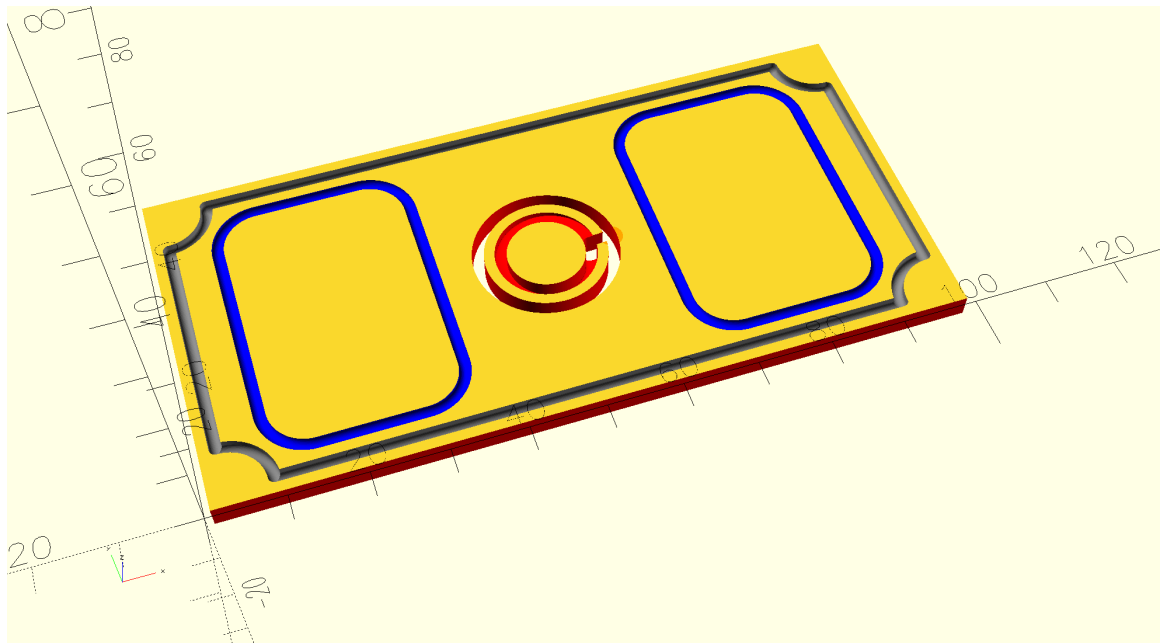
and which allows an interactive usage in working up a design such as for lasercutting, and which incorporates and option to the rapid(x,y,z) command which simulates turning a laser off, repositioning, then powering up the laser after.

2.3   gcodepreviewtemplate.py

Note that since the v0.7 re-write, it is possible to directly use the underlying Python code. Using Python to generate 3D previews of how DXFs or G-code will cut requires the use of PythonSCAD.

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

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

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

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

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

Which generates a 3D model which previews in PythonSCAD as:
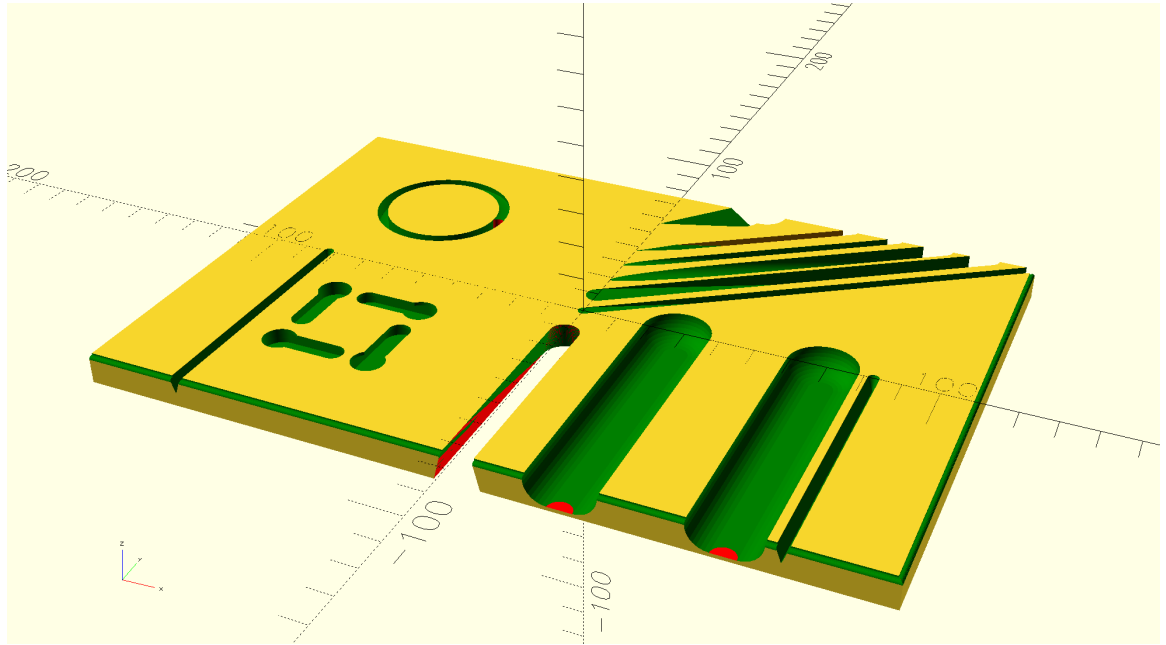
2.4   gcodepreviewtemplate.scad

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

```
 1 gcptmpl //!OpenSCAD
 2 gcptmpl
 3 gcptmpl use <gcodepreview.py>
 4 gcptmpl include <gcodepreview.scad>
 5 gcptmpl
 6 gcptmpl $fa = 2;
 7 gcptmpl $fs = 0.125;
 8 gcptmpl fa = 2;
 9 gcptmpl fs = 0.125;
10 gcptmpl
11 gcptmpl /* [Stock] */
12 gcptmpl stockXwidth = 220;
13 gcptmpl /* [Stock] */
14 gcptmpl stockYheight = 150;
15 gcptmpl /* [Stock] */
16 gcptmpl stockZthickness = 8.35;
17 gcptmpl /* [Stock] */
18 gcptmpl zeroheight = "Top"; // [Top, Bottom]
19 gcptmpl /* [Stock] */
20 gcptmpl stockzero = "Center"; // [Lower-Left, Center-Left, Top-Left, Center
           ]
21 gcptmpl /* [Stock] */
22 gcptmpl retractheight = 9;
23 gcptmpl
24 gcptmpl /* [Export] */
25 gcptmpl Base_filename = "export";
26 gcptmpl /* [Export] */
27 gcptmpl generatedxf = true;
28 gcptmpl /* [Export] */
29 gcptmpl generategcode = true;
30 gcptmpl
31 gcptmpl /* [CAM] */
32 gcptmpl toolradius = 1.5875;
33 gcptmpl /* [CAM] */
34 gcptmpl large_square_tool_num = 0; // [0:0, 112:112, 102:102, 201:201]
35 gcptmpl /* [CAM] */
36 gcptmpl small_square_tool_num = 102; // [0:0, 122:122, 112:112, 102:102]
37 gcptmpl /* [CAM] */
38 gcptmpl large_ball_tool_num = 0; // [0:0, 111:111, 101:101, 202:202]
39 gcptmpl /* [CAM] */
40 gcptmpl small_ball_tool_num = 0; // [0:0, 121:121, 111:111, 101:101]
41 gcptmpl /* [CAM] */
42 gcptmpl large_V_tool_num = 0; // [0:0, 301:301, 690:690]
43 gcptmpl /* [CAM] */
```

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

```
120 gcptmpl
121 gcptmpl rapidZ(retractheight);
122 gcptmpl toolchange(202, 10000);
123 gcptmpl rapidXY(0, stockYheight/8);
124 gcptmpl rapidZ(0);
125 gcptmpl cutlinedxfgc(stockXwidth/16*6, stockYheight/2, -stockZthickness);
126 gcptmpl
127 gcptmpl rapidZ(retractheight);
128 gcptmpl toolchange(101, 10000);
129 gcptmpl rapidXY(0, stockYheight/16*3);
130 gcptmpl rapidZ(0);
131 gcptmpl cutlinedxfgc(stockXwidth/16*5, stockYheight/2, -stockZthickness);
132 gcptmpl
133 gcptmpl rapidZ(retractheight);
134 gcptmpl toolchange(390, 10000);
135 gcptmpl rapidXY(0, stockYheight/16*4);
136 gcptmpl rapidZ(0);
137 gcptmpl
138 gcptmpl cutlinedxfgc(stockXwidth/16*4, stockYheight/2, -stockZthickness);
139 gcptmpl rapidZ(retractheight);
140 gcptmpl
141 gcptmpl toolchange(301, 10000);
142 gcptmpl rapidXY(0, stockYheight/16*6);
143 gcptmpl rapidZ(0);
144 gcptmpl
145 gcptmpl cutlinedxfgc(stockXwidth/16*2, stockYheight/2, -stockZthickness);
146 gcptmpl
147 gcptmpl
148 gcptmpl movetosafeZ();
149 gcptmpl rapid(gcp.xpos(), gcp.ypos(), retractheight);
150 gcptmpl toolchange(102, 10000);
151 gcptmpl
152 gcptmpl //rapidXY(stockXwidth/4+stockYheight/8+stockYheight/16, +
              stockYheight/8);
153 gcptmpl rapidXY(-stockXwidth/4+stockXwidth/16, (stockYheight/4));//+
              stockYheight/16
154 gcptmpl rapidZ(0);
155 gcptmpl
156 gcptmpl //cutarcCW(360, 270, gcp.xpos()-stockYheight/16, gcp.ypos(),
              stockYheight/16, -stockZthickness);
157 gcptmpl //gcp.cutarcCW(270, 180, gcp.xpos(), gcp.ypos()+stockYheight/16,
              stockYheight/16))
158 gcptmpl //cutarcCC(0, 90, gcp.xpos()-stockYheight/16, gcp.ypos(),
              stockYheight/16, -stockZthickness/4);
159 gcptmpl //cutarcCC(90, 180, gcp.xpos(), gcp.ypos()-stockYheight/16,
              stockYheight/16, -stockZthickness/4);
160 gcptmpl //cutarcCC(180, 270, gcp.xpos()+stockYheight/16, gcp.ypos(),
              stockYheight/16, -stockZthickness/4);
161 gcptmpl //cutarcCC(270, 360, gcp.xpos(), gcp.ypos()+stockYheight/16,
              stockYheight/16, -stockZthickness/4);
162 gcptmpl
163 gcptmpl movetosafeZ();
164 gcptmpl //rapidXY(stockXwidth/4+stockYheight/8-stockYheight/16, -
              stockYheight/8);
165 gcptmpl rapidXY(stockXwidth/4-stockYheight/16, -(stockYheight/4));
166 gcptmpl rapidZ(0);
167 gcptmpl
168 gcptmpl //cutarcCW(180, 90, gcp.xpos()+stockYheight/16, gcp.ypos(),
              stockYheight/16, -stockZthickness/4);
169 gcptmpl //cutarcCW(90, 0, gcp.xpos(), gcp.ypos()-stockYheight/16,
              stockYheight/16, -stockZthickness/4);
170 gcptmpl //cutarcCW(360, 270, gcp.xpos()-stockYheight/16, gcp.ypos(),
              stockYheight/16, -stockZthickness/4);
171 gcptmpl //cutarcCW(270, 180, gcp.xpos(), gcp.ypos()+stockYheight/16,
              stockYheight/16, -stockZthickness/4);
172 gcptmpl
173 gcptmpl movetosafeZ();
174 gcptmpl
175 gcptmpl rapidXY(-stockXwidth/4 + stockYheight/8, (stockYheight/4));
176 gcptmpl rapidZ(0);
177 gcptmpl
178 gcptmpl cutquarterCCNEdxf(xpos() - stockYheight/8, ypos() + stockYheight/8,
              -stockZthickness/4, stockYheight/8);
179 gcptmpl cutquarterCCNWdxf(xpos() - stockYheight/8, ypos() - stockYheight/8,
              -stockZthickness/2, stockYheight/8);
180 gcptmpl cutquarterCCSWdxf(xpos() + stockYheight/8, ypos() - stockYheight/8,
              -stockZthickness * 0.75, stockYheight/8);
181 gcptmpl //cutquarterCCSEdxf(xpos() + stockYheight/8, ypos() + stockYheight
```

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

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

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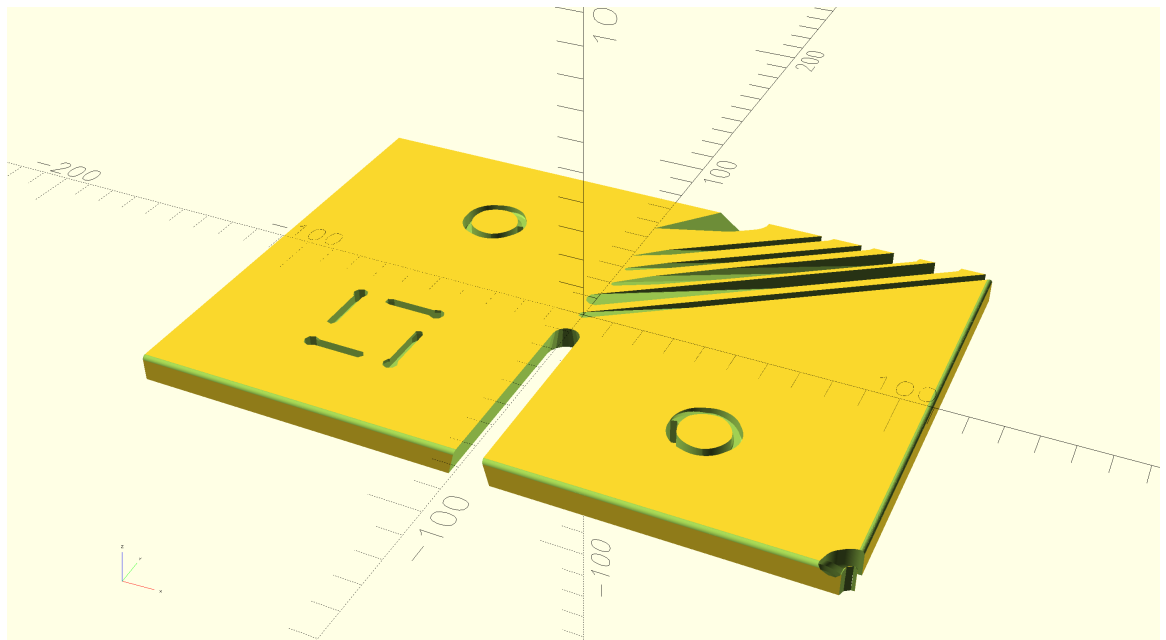
3   **gcodepreview**

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

- A Python file: gcodepreview.py (`gcpy`) — this has variables in the traditional sense which are used for tracking machine position and so forth. Note that where it is placed/loaded from will depend on whether it is imported into a Python file:
  `import gcodepreview_standalone as gcp`
  or used in an OpenSCAD file:
  `use <gcodepreview.py>`
  with an additional OpenSCAD module which allows accessing it and that there is an option for loading directly from the Github repository implemented in PythonSCAD

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

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

In earlier versions there were several possible ways to work with the 3D models of the cuts, either directly displaying the returned 3D model when explicitly called for after storing it in a variable or calling it up as a calculation (Python command `ouput(<foo>)` or OpenSCAD returning a model, or calling an appropriate OpenSCAD command), however as-of v0.9 the tool movements are modeled as lists of `hull()` operations which must be processed as such and are `differenced` from the stock. The templates set up these options as noted, and ensure that `True == true`.

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

3.1   **Module Naming Convention**

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

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

  - `rapid` (action — create tool movement of 3D object so as to show any collision or rubbing)

  - – `open` (action (file))

  - – `close` (action (file))

  - – `set` (action/function) — note that the matching `get` is implicit in functions which return variables, e.g., `xpos()`

  - – `current`

- Nouns (shapes)

  - – `arc`

  - – `line`

  - – `rectangle`

  - – `circle`

- Suffixes

  - – `feed` (parameter)

  - – `gcode`/`gc` (filetype)

  - – `pos` — position

  - – `tool`

  - – `loop`

  - – `CC`/`CW`

  - – `number`/`num` — note that `num` is used internally for variable names, while `number` will be used for module/function names, making it straight-forward to ensure that functions and variables have different names for purposes of scope

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

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

	line			arc		
	cut	dxf	gcode	cut	dxf	gcode
cut	cutline		cutlinegc	cutarc		cutarcgc
dxf	cutlinedxf	dxfline		cutarcdxf	dxfarc	
gcode	cutlinegc		linegc	cutarcgc		arcgc
		cutlinedxfgc			cutarcdxfgc	

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

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

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

Principles for naming modules (and variables):

- minimize use of underscores (for convenience sake, underscores are not used for index entries)

- identify which aspect of the project structure is being worked with (cut(ting), `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

- OpenSCAD — see: https://openscad.org/documentation.html

- Programming in OpenSCAD with variables and calling Python — this requires 3 files and was originally used in the project as written up at: https://github.com/WillAdams/gcodepreview/blob/main/gcodepreview-openscad_0_6.pdf (for further details see below, notably various commented out lines in the source .tex file)

- Programming in OpenSCAD and calling Python where all variables as variables are held in Python classes (this is the technique used as of v0.8)

- Programming in Python and calling OpenSCAD — https://old.reddit.com/r/OpenPythonSCAD/comments/1heczmi/finally_using_scad_modules/

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

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


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

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


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

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

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


which in turn calls the internal Python 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.

3.1.1 Parameters and Default Values

Ideally, there would be *no* hard-coded values — every value used for calculation will be parameterized, and subject to control/modification. Fortunately, Python affords a feature which specifically addresses this, optional arguments with default values:
https://stackoverflow.com/questions/9539921/how-do-i-define-a-function-with-optional-argument

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

3.2   Implementation files and gcodepreview class

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

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

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

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

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

- Mandatory
    - stocksetup:
        * stockXwidth, stockYheight, stockZthickness, zeroheight, stockzero, retractheight
    - gcpfiles:
        * basefilename, generatedxf, generategcode
    - largesquaretool:

* large_square_tool_num, toolradius, plunge, feed, speed
        – currenttoolnum
            * endmilltype
            * diameter
            * flute
            * shaftdiameter
            * shaftheight
            * shaftlength
            * toolnumber
            * cutcolor
            * rapidcolor
            * shaftcolor
* Optional
        – smallsquaretool:
            * small_square_tool_num, small_square_ratio
        – largeballtool:
            * large_ball_tool_num, large_ball_ratio
        – largeVtool:
            * large_V_tool_num, large_V_ratio
        – smallballtool:
            * small_ball_tool_num, small_ball_ratio
        – smallVtool:
            * small_V_tool_num, small_V_ratio
        – DTtool:
            * DT_tool_num, DT_ratio
        – KHtool:
            * KH_tool_num, KH_ratio
        – Roundovertool:
            * Roundover_tool_num, RO_ratio
        – misctool:
            * MISC_tool_num, MISC_ratio

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

```
23 gcpy  class gcodepreview:
24 gcpy
25 gcpy      def __init__(self,
26 gcpy                   generategcode = False,
27 gcpy                   generatedxf = False,
28 gcpy                   gcpfa = 2,
29 gcpy                   gcpfs = 0.125,
30 gcpy                   steps = 10
31 gcpy                   ):
32 gcpy          """
33 gcpy          Initialize gcodepreview object.
34 gcpy
35 gcpy          Parameters
36 gcpy          ----------
37 gcpy          generategcode : boolean
38 gcpy                          Enables writing out G-code.
39 gcpy          generatedxf   : boolean
40 gcpy                          Enables writing out DXF file(s).
41 gcpy
42 gcpy          Returns
43 gcpy          -------
44 gcpy          object
45 gcpy              The initialized gcodepreview object.
46 gcpy          """
47 gcpy          if generategcode == 1:
48 gcpy              self.generategcode = True
49 gcpy          elif generategcode == 0:
50 gcpy              self.generategcode = False
51 gcpy          else:
52 gcpy              self.generategcode = generategcode
```

```
53 gcpy          if generatedxf == 1:
54 gcpy              self.generatedxf = True
55 gcpy          elif generatedxf == 0:
56 gcpy              self.generatedxf = False
57 gcpy          else:
58 gcpy              self.generatedxf = generatedxf
59 gcpy # unless multiple dxfs are enabled, the check for them is of course
          False
60 gcpy          self.generatedxfs = False
61 gcpy # set up 3D previewing parameters
62 gcpy          fa = gcpfa
63 gcpy          fs = gcpfs
64 gcpy          self.steps = steps
65 gcpy # initialize the machine state
66 gcpy          self.mc = "Initialized"
67 gcpy          self.mpx = float(0)
68 gcpy          self.mpy = float(0)
69 gcpy          self.mpz = float(0)
70 gcpy          self.tpz = float(0)
71 gcpy # initialize the toolpath state
72 gcpy          self.retractheight = 5
73 gcpy # initialize the DEFAULT tool
74 gcpy          self.currenttoolnum = 102
75 gcpy          self.endmilltype = "square"
76 gcpy          self.diameter = 3.175
77 gcpy          self.flute = 12.7
78 gcpy          self.shaftdiameter = 3.175
79 gcpy          self.shaftheight = 12.7
80 gcpy          self.shaftlength = 19.5
81 gcpy          self.toolnumber = "100036"
82 gcpy          self.cutcolor = "green"
83 gcpy          self.rapidcolor = "orange"
84 gcpy          self.shaftcolor = "red"
85 gcpy # the variables for holding 3D models must be initialized as empty
          lists so as to ensure that only append or extend commands are
          used with them
86 gcpy          self.rapids = []
87 gcpy          self.toolpaths = []
88 gcpy
89 gcpy #    def myfunc(self, var):
90 gcpy #        self.vv = var * var
91 gcpy #        return self.vv
92 gcpy #
93 gcpy #    def getvv(self):
94 gcpy #        return self.vv
95 gcpy #
96 gcpy #    def checkint(self):
97 gcpy #        return self.mc
98 gcpy #
99 gcpy #    def makecube(self, xdim, ydim, zdim):
100 gcpy #        self.c=cube([xdim, ydim, zdim])
101 gcpy #
102 gcpy #    def placecube(self):
103 gcpy #        show(self.c)
104 gcpy #
105 gcpy #    def instantiatecube(self):
106 gcpy #        return self.c
```

3.2.1 Position and Variables

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

The first such variables are for xyz position:

mpx • mpx

mpy • mpy

mpz • mpz

Similarly, for some toolpaths it will be necessary to track the depth along the Z-axis as the toolpath 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, a 3D model of the tool was available as `currenttool` itself and used where appropriate, but in v0.9, this was changed to using lists for concatenating the `hulled` shapes of tool movements, so the module, `toolmovement` which given begin/end position returns the appropriate shape(s) as a list.

`toolmovement`

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

`xpos`
`ypos`
`zpos`

```
108 gcpy      def xpos(self):
109 gcpy          return self.mpx
110 gcpy
111 gcpy      def ypos(self):
112 gcpy          return self.mpy
113 gcpy
114 gcpy      def zpos(self):
115 gcpy          return self.mpz
```

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

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

`setxpos`
`setypos`
`setzpos`

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

```
117 gcpy      def setxpos(self, newxpos):
118 gcpy          self.mpx = newxpos
119 gcpy
120 gcpy      def setypos(self, newypos):
121 gcpy          self.mpy = newypos
122 gcpy
123 gcpy      def setzpos(self, newzpos):
124 gcpy          self.mpz = newzpos
```

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

3.2.2   Initial Modules

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

- mpx

- mpy

- mpz

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

`toolmovement`
`endmilltype`
`diameter`
`flute`
`ra`
`tip`
`toolmovement`
`rapids`
`toolpaths`

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

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

`gcodepreview`
`setupstock`

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

`gcp.setupstock`

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

```
126 gcpy      def setupstock(self, stockXwidth,
127 gcpy                          stockYheight,
128 gcpy                          stockZthickness,
```

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

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

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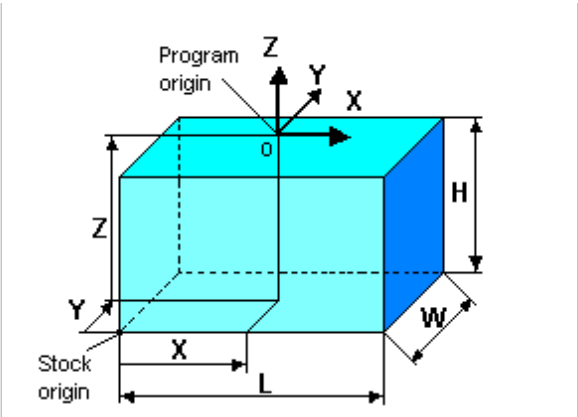


```
162 gcpy           if self.zeroheight == "Top":
163 gcpy               if self.stockzero == "Lower-Left":
164 gcpy                   self.stock = self.stock.translate([0, 0, -self.
                               stockZthickness])
165 gcpy                   if self.generategcode == True:
166 gcpy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

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

```
205 gcpy                                      , "mm)")
                                    self.writegc("(STOCK/BLOCK,␣", str(self.
                                        stockXwidth), ",␣", str(self.stockYheight),
                                        ",␣", str(self.stockZthickness), ",␣0.00,␣",
                                         str(self.stockYheight), ",␣0.00)")
206 gcpy                  if self.stockzero == "Center":
207 gcpy                      self.stock = self.stock.translate([-self.
                                    stockXwidth / 2, -self.stockYheight / 2, 0])
208 gcpy                      if self.generategcode == True:
209 gcpy                          self.writegc("(stockMin:␣-", str(self.
                                        stockXwidth/2), ",␣-", str(self.stockYheight
                                        /2), "mm,␣0.00mm)")
210 gcpy                          self.writegc("(stockMax:", str(self.stockXwidth
                                        /2), "mm,␣", str(self.stockYheight/2), "mm,␣
                                        ", str(self.stockZthickness), "mm)")
211 gcpy                          self.writegc("(STOCK/BLOCK,␣", str(self.
                                        stockXwidth), ",␣", str(self.stockYheight),
                                        ",␣", str(self.stockZthickness), ",␣", str(
                                        self.stockXwidth/2), ",␣", str(self.
                                        stockYheight/2), ",␣0.00)")
212 gcpy              if self.generategcode == True:
213 gcpy                  self.writegc("G90");
214 gcpy                  self.writegc("G21");
```

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

The OpenSCAD version is simply a descriptor:

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

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

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

3.2.3 Adjustments and Additions

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

Shifting the stock is simple:

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

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

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

```
227 gcpy      def addtostock(self, stockXwidth, stockYheight, stockZthickness
                 ,
228 gcpy                         shiftX = 0,
229 gcpy                         shiftY = 0,
230 gcpy                         shiftZ = 0):
```

```
231 gcpy            addedpart = cube([stockXwidth, stockYheight,
                       stockZthickness])
232 gcpy            addedpart = addedpart.translate([shiftX, shiftY, shiftZ])
233 gcpy            self.stock = self.stock.union(addedpart)
```

the OpenSCAD module is a descriptor as expected:

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

3.3   Tools and Changes

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

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

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

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

3.3.1   Numbering for Tools

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

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

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

1	2−3	4−5	6
endmill type	radius/angle	cutting diameter(and tip radius for tapered ball nose)	cutting flute length

- 1st digit: endmill type:
    - 0 - "O"-flute
    - 1 - square
    - 2 - ball
    - 3 - V
    - 4 - bowl
    - 5 - tapered ball
    - 6 - roundover
    - 7 - thread-cutting
    - 8 - dovetail
    - 9 - other (e.g., keyhole, lollipop, or manufacturer number — if manufacturer number is used, then the 9 and any padding zeroes will be removed from the G-code or DXF when writing out file(s))

- 2nd and 3rd digits shape radius (ball/roundover) or angle (V), 2nd and 3rd digit together 10–99 indicate measurement in tenth of a millimeter. 2nd digit:
    - 0 - Imperial (00 indicates n/a or square)
    - any other value for both the 2nd and 3rd digits together indicate a metric measurement or an angle in degrees

- 3rd digit (if 2nd is 0 indicating Imperial)
    - 1 - $1/32^{nd}$
    - 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 #326 (Amana 46200-K))

  #201 == 100047 (also #251 and #322 (Amana 46202-K))

  #205 == 100048

  #324 == 100048 (Amana 46170-K)

- Ball

    #121 == 201012

    #111 == 202024

    #101 == 203036

    #202 == 204047

    #325 == 204048 (Amana 46376-K)

- V

    #301 == 390074

    #302 == 360071

    #327 == 360098 (Amana RC-1148)

- Single (O) flute

    #282 == 000204

    #274 == 000036

    #278 == 000047

- Tapered Ball Nose

    #501 == 530131

    #502 == 540131

(note that some dimensions were rounded off/approximated)
Extending that to the non-Carbide 3D tooling thus implemented:

- Dovetail

    814 == 814071

    45828 == 808071

- Keyhole Tool

    374 == 906043

    375 == 906053

    376 == 907040

    378 == 907050

- Roundover Tool

    56142 == 602032

    56125 == 603042

    1568 == 603032

    1570

    1572 == 604042

    1574

- Threadmill

    648 == 7

- Bowl bit

    45981

    45982

    1370

    1372

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

- No way to indicate flute geometry beyond O-flute

- Lack of precision for metric tooling/limited support for Imperial sizes, notably, the dimensions used are scaled for smaller tooling and are not suited to larger scale tooling such as bowl bits

- No way to indicate several fairly common shapes including keyhole, lollipop, and flat-bottomed V/chamfer tools (except of course for using 9#####)

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

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

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

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

```
235 gcpy    def currenttoolnumber(self):
236 gcpy        return(self.currenttoolnum)
```

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

- endmilltype

    O-flute

    square

    ball

    V

    keyhole

    dovetail

    roundover

    tapered ball

- diameter

- flute

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

- radius

- angle

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

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

Note that the comments written out in G-code correspond to those used by the G-code 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.

```
238 gcpy    def toolchange(self, tool_number, speed = 10000):
239 gcpy        self.currenttoolnum = tool_number
240 gcpy
241 gcpy        if (self.generategcode == True):
242 gcpy            self.writegc("(Toolpath)")
243 gcpy            self.writegc("M05")
```

toolchange   The Python definition for `toolchange` requires the tool number (used to write out the G-code comment description for CutViewer and also expects the speed for the current tool since this is passed into the G-code tool change command as part of the spindle on command. A simple if–then structure, the variables necessary for defining the toolshape are (re)defined each time the
toolmovement command is called so that they may be used by the command `toolmovement` for actually modeling the shapes and the path and the resultant material removal.

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

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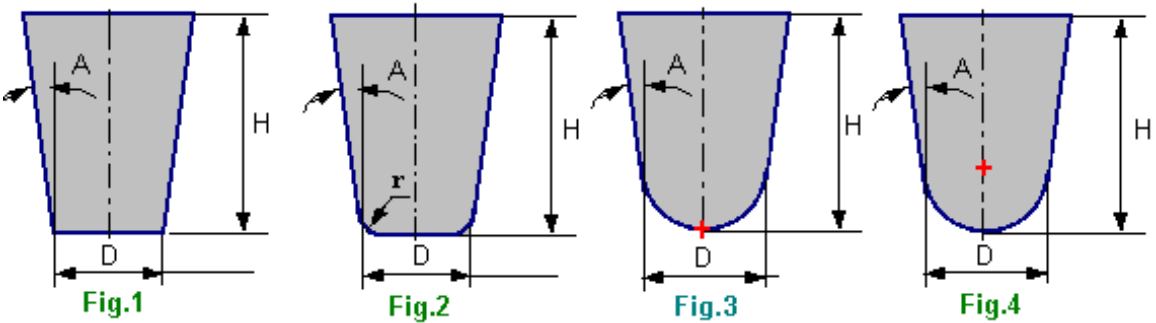


```
245 gcpy          if (tool_number == 201): #201/251/322 (Amana 46202-K) ==
                  100047
246 gcpy            self.writegc("(TOOL/MILL,␣6.35,␣0.00,␣0.00,␣0.00)")
247 gcpy            self.endmilltype = "square"
248 gcpy            self.diameter = 6.35
249 gcpy            self.flute = 19.05
250 gcpy            self.shaftdiameter = 6.35
251 gcpy            self.shaftheight = 19.05
252 gcpy            self.shaftlength = 20.0
253 gcpy            self.toolnumber = "100047"
254 gcpy          elif (tool_number == 102): #102/326 == 100036
255 gcpy            self.writegc("(TOOL/MILL,␣3.175,␣0.00,␣0.00,␣0.00)")
256 gcpy            self.endmilltype = "square"
257 gcpy            self.diameter = 3.175
258 gcpy            self.flute = 12.7
259 gcpy            self.shaftdiameter = 3.175
260 gcpy            self.shaftheight = 12.7
261 gcpy            self.shaftlength = 20.0
262 gcpy            self.toolnumber = 100036
263 gcpy          elif (tool_number == 112): #112 == 100024
264 gcpy            self.writegc("(TOOL/MILL,␣1.5875,␣0.00,␣0.00,␣0.00)")
265 gcpy            self.endmilltype = "square"
266 gcpy            self.diameter = 1.5875
267 gcpy            self.flute = 6.35
268 gcpy            self.shaftdiameter = 3.175
269 gcpy            self.shaftheight = 6.35
270 gcpy            self.shaftlength = 12.0
271 gcpy            self.toolnumber = "100024"
272 gcpy          elif (tool_number == 122): #122 == 100012
273 gcpy            self.writegc("(TOOL/MILL,␣0.79375,␣0.00,␣0.00,␣0.00)")
274 gcpy            self.endmilltype = "square"
275 gcpy            self.diameter = 0.79375
276 gcpy            self.flute = 1.5875
277 gcpy            self.shaftdiameter = 3.175
278 gcpy            self.shaftheight = 1.5875
279 gcpy            self.shaftlength = 12.0
280 gcpy            self.toolnumber = "100012"
281 gcpy          elif (tool_number == 324): #324 (Amana 46170-K) == 100048
282 gcpy            self.writegc("(TOOL/MILL,␣6.35,␣0.00,␣0.00,␣0.00)")
283 gcpy            self.endmilltype = "square"
284 gcpy            self.diameter = 6.35
285 gcpy            self.flute = 22.225
286 gcpy            self.shaftdiameter = 6.35
287 gcpy            self.shaftheight = 22.225
288 gcpy            self.shaftlength = 20.0
289 gcpy            self.toolnumber = "100048"
290 gcpy          elif (tool_number == 205): #205 == 100048
291 gcpy            self.writegc("(TOOL/MILL,␣6.35,␣0.00,␣0.00,␣0.00)")
292 gcpy            self.endmilltype = "square"
293 gcpy            self.diameter = 6.35
294 gcpy            self.flute = 25.4
295 gcpy            self.shaftdiameter = 6.35
296 gcpy            self.shaftheight = 25.4
297 gcpy            self.shaftlength = 20.0
298 gcpy            self.toolnumber = "100048"
299 gcpy #
```

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

```
300 gcpy              elif (tool_number == 282): #282 == 000204
301 gcpy                  self.writegc("(TOOL/MILL,␣2.0,␣0.00,␣0.00,␣0.00)")
302 gcpy                  self.endmilltype = "O-flute"
303 gcpy                  self.diameter = 2.0
304 gcpy                  self.flute = 6.35
305 gcpy                  self.shaftdiameter = 6.35
306 gcpy                  self.shaftheight = 6.35
307 gcpy                  self.shaftlength = 12.0
308 gcpy                  self.toolnumber = "000204"
309 gcpy              elif (tool_number == 274): #274 == 000036
310 gcpy                  self.writegc("(TOOL/MILL,␣3.175,␣0.00,␣0.00,␣0.00)")
311 gcpy                  self.endmilltype = "O-flute"
312 gcpy                  self.diameter = 3.175
313 gcpy                  self.flute = 12.7
314 gcpy                  self.shaftdiameter = 3.175
315 gcpy                  self.shaftheight = 12.7
316 gcpy                  self.shaftlength = 20.0
317 gcpy                  self.toolnumber = "000036"
318 gcpy              elif (tool_number == 278): #278 == 000047
319 gcpy                  self.writegc("(TOOL/MILL,␣6.35,␣0.00,␣0.00,␣0.00)")
320 gcpy                  self.endmilltype = "O-flute"
321 gcpy                  self.diameter = 6.35
322 gcpy                  self.flute = 19.05
323 gcpy                  self.shaftdiameter = 3.175
324 gcpy                  self.shaftheight = 19.05
325 gcpy                  self.shaftlength = 20.0
326 gcpy                  self.toolnumber = "000047"
327 gcpy  #
```

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

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

```
373 gcpy  #
```

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

```
374 gcpy          elif (tool_number == 301): #301 == 390074
375 gcpy              self.writegc("(TOOL/MILL,␣0.10,␣0.05,␣6.35,␣45.00)")
376 gcpy              self.endmilltype = "V"
377 gcpy              self.diameter = 12.7
378 gcpy              self.flute = 6.35
379 gcpy              self.angle = 90
380 gcpy              self.shaftdiameter = 6.35
381 gcpy              self.shaftheight = 6.35
382 gcpy              self.shaftlength = 20.0
383 gcpy              self.toolnumber = "390074"
384 gcpy          elif (tool_number == 302): #302 == 360071
385 gcpy              self.writegc("(TOOL/MILL,␣0.10,␣0.05,␣6.35,␣30.00)")
386 gcpy              self.endmilltype = "V"
387 gcpy              self.diameter = 12.7
388 gcpy              self.flute = 11.067
389 gcpy              self.angle = 60
390 gcpy              self.shaftdiameter = 6.35
391 gcpy              self.shaftheight = 11.067
392 gcpy              self.shaftlength = 20.0
393 gcpy              self.toolnumber = "360071"
394 gcpy          elif (tool_number == 390): #390 == 390032
395 gcpy              self.writegc("(TOOL/MILL,␣0.03,␣0.00,␣1.5875,␣45.00)")
396 gcpy              self.endmilltype = "V"
397 gcpy              self.diameter = 3.175
398 gcpy              self.flute = 1.5875
399 gcpy              self.angle = 90
400 gcpy              self.shaftdiameter = 3.175
401 gcpy              self.shaftheight = 1.5875
402 gcpy              self.shaftlength = 20.0
403 gcpy              self.toolnumber = "390032"
404 gcpy          elif (tool_number == 327): #327 (Amana RC-1148) == 360098
405 gcpy              self.writegc("(TOOL/MILL,␣0.03,␣0.00,␣13.4874,␣30.00)")
406 gcpy              self.endmilltype = "V"
407 gcpy              self.diameter = 25.4
408 gcpy              self.flute = 22.134
409 gcpy              self.angle = 60
410 gcpy              self.shaftdiameter = 6.35
411 gcpy              self.shaftheight = 22.134
412 gcpy              self.shaftlength = 20.0
413 gcpy              self.toolnumber = "360098"
414 gcpy          elif (tool_number == 323): #323 == 330041 30 degree V Amana
                    , 45771-K
415 gcpy              self.writegc("(TOOL/MILL,␣0.10,␣0.05,␣11.18,␣15.00)")
416 gcpy              self.endmilltype = "V"
417 gcpy              self.diameter = 6.35
418 gcpy              self.flute = 11.849
419 gcpy              self.angle = 30
420 gcpy              self.shaftdiameter = 6.35
421 gcpy              self.shaftheight = 11.849
422 gcpy              self.shaftlength = 20.0
423 gcpy              self.toolnumber = "330041"
424 gcpy  #
```

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

```
425 gcpy          elif (tool_number == 374): #374 == 906043
426 gcpy              self.writegc("(TOOL/MILL,␣9.53,␣0.00,␣3.17,␣0.00)")
427 gcpy              self.endmilltype = "keyhole"
428 gcpy              self.diameter = 9.525
429 gcpy              self.flute = 3.175
430 gcpy              self.radius = 6.35
431 gcpy              self.shaftdiameter = 6.35
432 gcpy              self.shaftheight = 3.175
433 gcpy              self.shaftlength = 20.0
434 gcpy              self.toolnumber = "906043"
435 gcpy          elif (tool_number == 375): #375 == 906053
436 gcpy              self.writegc("(TOOL/MILL,␣9.53,␣0.00,␣3.17,␣0.00)")
437 gcpy              self.endmilltype = "keyhole"
```

```
438 gcpy                    self.diameter = 9.525
439 gcpy                    self.flute = 3.175
440 gcpy                    self.radius = 8
441 gcpy                    self.shaftdiameter = 6.35
442 gcpy                    self.shaftheight = 3.175
443 gcpy                    self.shaftlength = 20.0
444 gcpy                    self.toolnumber = "906053"
445 gcpy           elif (tool_number == 376): #376 == 907040
446 gcpy                    self.writegc("(TOOL/MILL,␣12.7,␣0.00,␣4.77,␣0.00)")
447 gcpy                    self.endmilltype = "keyhole"
448 gcpy                    self.diameter = 12.7
449 gcpy                    self.flute = 4.7625
450 gcpy                    self.radius = 6.35
451 gcpy                    self.shaftdiameter = 6.35
452 gcpy                    self.shaftheight = 4.7625
453 gcpy                    self.shaftlength = 20.0
454 gcpy                    self.toolnumber = "907040"
455 gcpy           elif (tool_number == 378): #378 == 907050
456 gcpy                    self.writegc("(TOOL/MILL,␣12.7,␣0.00,␣4.77,␣0.00)")
457 gcpy                    self.endmilltype = "keyhole"
458 gcpy                    self.diameter = 12.7
459 gcpy                    self.flute = 4.7625
460 gcpy                    self.radius = 8
461 gcpy                    self.shaftdiameter = 6.35
462 gcpy                    self.shaftheight = 4.7625
463 gcpy                    self.shaftlength = 20.0
464 gcpy                    self.toolnumber = "907050"
465 gcpy #
```

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

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

```
508 gcpy                    self.shaftheight = 15.875
509 gcpy                    self.shaftlength = 20.0
510 gcpy                    self.toolnumber = "401372"
511 gcpy #
```

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

```
512 gcpy              elif (tool_number == 501): #501 == 530131
513 gcpy                  self.writegc("(TOOL/MILL,0.03,␣0.00,␣10.00,␣30.00)")
514 gcpy #                self.currenttoolshape = self.toolshapes("tapered ball
         ", 3.175, 5.561, 30, 0.254)
515 gcpy                  self.endmilltype = "tapered␣ball"
516 gcpy                  self.diameter = 3.175
517 gcpy                  self.flute = 5.561
518 gcpy                  self.angle = 30
519 gcpy                  self.tip = 0.254
520 gcpy                  self.shaftdiameter = 3.175
521 gcpy                  self.shaftheight = 5.561
522 gcpy                  self.shaftlength = 10.0
523 gcpy                  self.toolnumber = "530131"
524 gcpy            elif (tool_number == 502): #502 == 540131
525 gcpy                  self.writegc("(TOOL/MILL,0.03,␣0.00,␣10.00,␣20.00)")
526 gcpy #                self.currenttoolshape = self.toolshapes("tapered ball
         ", 3.175, 4.117, 40, 0.254)
527 gcpy                  self.endmilltype = "tapered␣ball"
528 gcpy                  self.diameter = 3.175
529 gcpy                  self.flute = 4.117
530 gcpy                  self.angle = 40
531 gcpy                  self.tip = 0.254
532 gcpy                  self.shaftdiameter = 3.175
533 gcpy                  self.shaftheight = 4.117
534 gcpy                  self.shaftlength = 10.0
535 gcpy                  self.toolnumber = "540131"
536 gcpy #          elif (tool_number == 204):#
537 gcpy #              self.writegc("()")
538 gcpy #              self.currenttoolshape = self.tapered_ball(1.5875,
         6.35, 38.1, 3.6)
539 gcpy #          elif (tool_number == 304):#
540 gcpy #              self.writegc("()")
541 gcpy #              self.currenttoolshape = self.tapered_ball(3.175, 6.35,
         38.1, 2.4)
542 gcpy #
```

3.3.1.8 Roundover (corner rounding) Note that the parameters will need to incorporate the tip diameter into the overall diameter. CutViewer uses:

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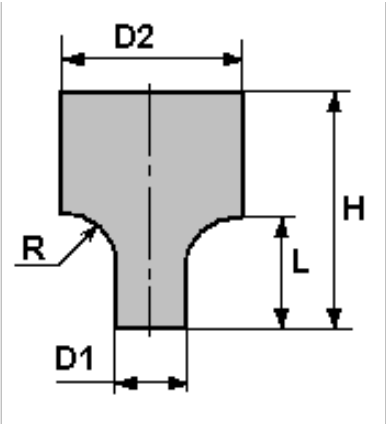


```
543 gcpy              elif (tool_number == 56125):#0.508/2, 1.531 56125 == 603042
544 gcpy                  self.writegc("(TOOL/CRMILL,␣0.508,␣6.35,␣3.175,␣7.9375,
                         ␣3.175)")
545 gcpy                  self.endmilltype = "roundover"
546 gcpy                  self.tip = 0.508
```

```
547 gcpy                    self.diameter = 6.35 - self.tip
548 gcpy                    self.flute = 8 - self.tip
549 gcpy                    self.radius = 3.175 - self.tip
550 gcpy                    self.shaftdiameter = 6.35
551 gcpy                    self.shaftheight = 8
552 gcpy                    self.shaftlength = 10.0
553 gcpy                    self.toolnumber = "603042"
554 gcpy                elif (tool_number == 56142):#0.508/2, 2.921 56142 == 602032
555 gcpy                    self.writegc("(TOOL/CRMILL,␣0.508,␣3.571875,␣1.5875,␣
                             5.55625,␣1.5875)")
556 gcpy                    self.endmilltype = "roundover"
557 gcpy                    self.tip = 0.508
558 gcpy                    self.diameter = 3.175 - self.tip
559 gcpy                    self.flute = 4.7625 - self.tip
560 gcpy                    self.radius = 1.5875 - self.tip
561 gcpy                    self.shaftdiameter = 3.175
562 gcpy                    self.shaftheight = 4.7625
563 gcpy                    self.shaftlength = 10.0
564 gcpy                    self.toolnumber = "602032"
565 gcpy #         elif (tool_number == 312):#1.524/2, 3.175
566 gcpy #             self.writegc("(TOOL/CRMILL, Diameter1, Diameter2,
          Radius, Height, Length)")
567 gcpy #         elif (tool_number == 1568):#0.507/2, 4.509 1568 == 603032
568 gcpy ##FIX         self.writegc("(TOOL/CRMILL, 0.17018, 9.525,
          4.7625, 12.7, 4.7625)")
569 gcpy ##            self.currenttoolshape = self.toolshapes("roundover",
          3.175, 6.35, 3.175, 0.396875)
570 gcpy #             self.endmilltype = "roundover"
571 gcpy #             self.diameter = 3.175
572 gcpy #             self.flute = 6.35
573 gcpy #             self.radius = 3.175
574 gcpy #             self.tip = 0.396875
575 gcpy #             self.toolnumber = "603032"
576 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
577 gcpy #         elif (tool_number == 1570):#0.507/2, 4.509 1570 == 600002
          ?!?
578 gcpy #             self.writegc("(TOOL/CRMILL, 0.17018, 9.525, 4.7625,
          12.7, 4.7625)")
579 gcpy ##            self.currenttoolshape = self.toolshapes("roundover",
          4.7625, 9.525, 4.7625, 0.396875)
580 gcpy #             self.endmilltype = "roundover"
581 gcpy #             self.diameter = 4.7625
582 gcpy #             self.flute = 9.525
583 gcpy #             self.radius = 4.7625
584 gcpy #             self.tip = 0.396875
585 gcpy #             self.toolnumber = "600002"
586 gcpy #         elif (tool_number == 1572): #1572 = 604042
587 gcpy ##FIX         self.writegc("(TOOL/CRMILL, 0.17018, 9.525,
          4.7625, 12.7, 4.7625)")
588 gcpy ##            self.currenttoolshape = self.toolshapes("roundover",
          6.35, 12.7, 6.35, 0.396875)
589 gcpy #             self.endmilltype = "roundover"
590 gcpy #             self.diameter = 6.35
591 gcpy #             self.flute = 12.7
592 gcpy #             self.radius = 6.35
593 gcpy #             self.tip = 0.396875
594 gcpy #             self.toolnumber = "604042"
595 gcpy #         elif (tool_number == 1574): #1574 == 600062
596 gcpy ##FIX         self.writegc("(TOOL/CRMILL, 0.17018, 9.525,
          4.7625, 12.7, 4.7625)")
597 gcpy ##            self.currenttoolshape = self.toolshapes("roundover",
          9.525, 19.5, 9.515, 0.396875)
598 gcpy #             self.endmilltype = "roundover"
599 gcpy #             self.diameter = 9.525
600 gcpy #             self.flute = 19.5
601 gcpy #             self.radius = 9.515
602 gcpy #             self.tip = 0.396875
603 gcpy #             self.toolnumber = "600062"
604 gcpy #
```

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

```
605 gcpy                elif (tool_number == 814): #814 == 814071
```

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

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

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

As was discussed in the PythonSCAD Google Group (https://groups.google.com/g/pythonscad/c/rtiYa38W8tY), if a list is used instead, then the contents of the list are added all at once at a single level when processed.

An example file which shows this concept:

```
from openscad import *
fn=200

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

features = []

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

part = difference(box, features)

show(part)
```

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

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

For example:

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

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

3.3.1.10 closing G-code With the tools delineated, the module is closed out and the toolchange information written into the G-code as well as the command to start the spindle at the specified speed.

```
626 gcpy        self.writegc("M6T", str(tool_number))
627 gcpy        self.writegc("M03S", str(speed))
```

3.3.2   Laser support

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

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

```
M3 S0
S0
G0X0Y16
S1000
G1X100F1200
S0
M5 S0
M3 S0
S0
G0X0Y12
S1000
G1X100F1000
S0
M5 S0
M3 S0
S0
G0X0Y8
S1000
G1X100F800
S0
M5 S0
M3 S0
S0
G0X0Y4
S1000
G1X100F600
S0
M5 S0
M3 S0
S0
G0X0Y0
S1000
G1X100F400
S0
M5 S0
```

3.4   Shapes and tool movement

With all the scaffolding in place, it is possible to model the tool and `hull()` between copies of the
cut...   3D model of the tool, or a cross-section of it for both `cut...` and `rapid...` operations.
rapid...   The majority of commands will be more general, focusing on tooling which is generally supported by this library, moving in lines and arcs so as to describe shapes which lend themselves to representation with those tools and which match up with both toolpaths and supported geometry in Carbide Create, and the usage requirements of the typical user.

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

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

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

- *shaft*

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

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

- Keyhole tools — intended to cut slots for retaining hardware used for picture hanging, they may be used to create slots for other purposes Note that it will be necessary to model these thrice, once for the actual keyhole cutting, second for the fluted portion of the shaft, and then the shaft should be modeled for collision https://assetssc.leevalley.com/en-gb/shop/tools/power-tool-accessories/router-bits/30113-keyhole-router-bits

- Dovetail cutters — used for the joinery of the same name, they cut a large area at the bottom which slants up to a narrower region at a defined angle

- Lollipop cutters — normally used for 3D work, as their name suggests they are essentially a (cutting) ball on a narrow stick (the tool shaft), they are mentioned here only for compleatness' sake and are not (at this time) implemented

- Threadmill — used for cutting threads, normally a single form geometry is used on a CNC.

3.4.1 Generalized commands and cuts

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

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

3.4.2 Movement and color

toolmovement 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.

```
629 gcpy     def setcolor(self,
630 gcpy                 cutcolor = "green",
631 gcpy                 rapidcolor = "orange",
632 gcpy                 shaftcolor = "red"):
633 gcpy         self.cutcolor = cutcolor
634 gcpy         self.rapidcolor = rapidcolor
635 gcpy         self.shaftcolor = shaftcolor
```

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

Table 1: Colors in OpenSCAD and DXF

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

(note that the names are not case-sensitive)

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

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

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

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

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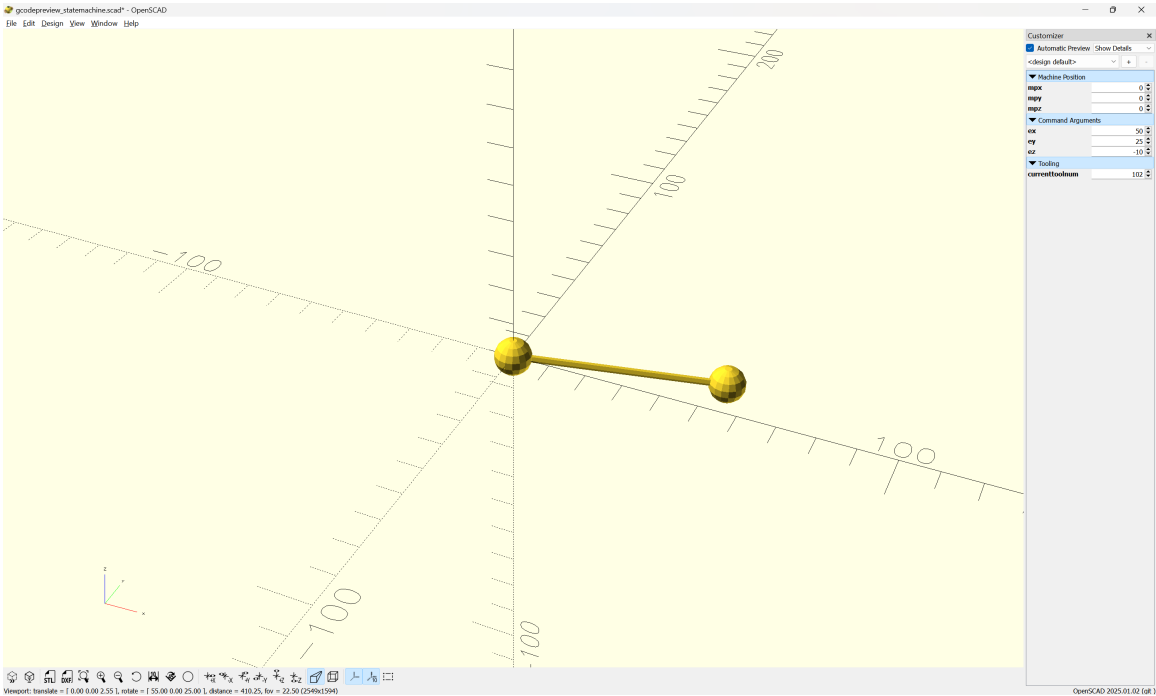

```
637 gcpy      def toolmovement(self, bx, by, bz, ex, ey, ez, step = 0):
638 gcpy          tslist = []
639 gcpy          if step > 0:
640 gcpy              steps = step
641 gcpy          else:
642 gcpy              steps = self.steps
```

3.4.2.2   Normal Tooling/toolshapes   Most tooling has quite standard shapes and are defined by their profile as defined in a class which simply defines/declares their shape and `hull()`s them together:
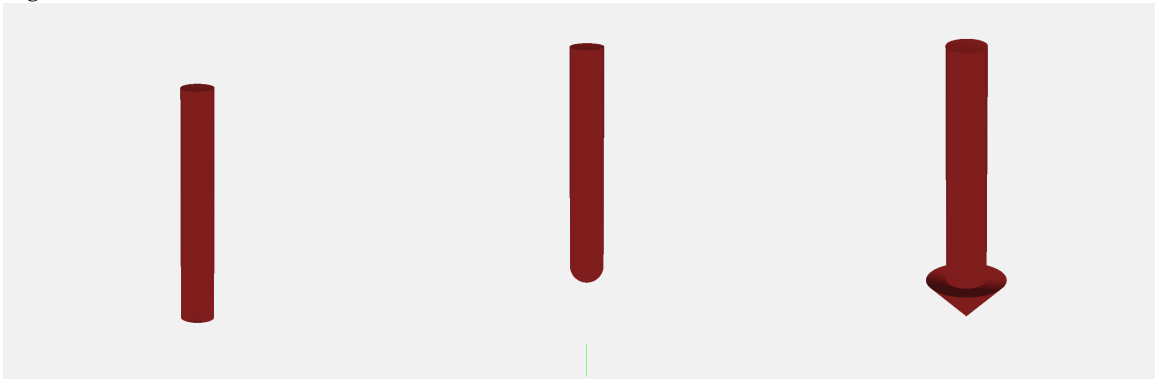
- Square (#201 and 102) — able to cut a flat bottom, perpendicular side and right angle, their simple and easily understood geometry makes them a standard choice

- Ballnose (#202 and 101) — rounded, they are the standard choice for concave and organic shapes

- V tooling (#301, 302 and 390) — pointed at the tip, they are available in a variety of angles and diameters and may be used for decorative V carving, or for chamfering or cutting specific angles

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

movement describes.

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

```
644 gcpy          if self.endmilltype == "square":
645 gcpy              ts = cylinder(r1=(self.diameter / 2), r2=(self.diameter
                          / 2), h=self.flute, center = False)
646 gcpy              tslist.append(hull(ts.translate([bx, by, bz]), ts.
                          translate([ex, ey, ez])))
647 gcpy              return tslist
648 gcpy
649 gcpy          if self.endmilltype == "O-flute":
650 gcpy              ts = cylinder(r1=(self.diameter / 2), r2=(self.diameter
                          / 2), h=self.flute, center = False)
651 gcpy              tslist.append(hull(ts.translate([bx, by, bz]), ts.
                          translate([ex, ey, ez])))
652 gcpy              return tslist
```

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

```
654 gcpy          if self.endmilltype == "ball":
655 gcpy              b = sphere(r=(self.diameter / 2))
656 gcpy              s = cylinder(r1=(self.diameter / 2), r2=(self.diameter
                          / 2), h=self.flute, center=False)
657 gcpy              bs = union(b, s)
658 gcpy              bs = bs.translate([0, 0, (self.diameter / 2)])
659 gcpy              tslist.append(hull(bs.translate([bx, by, bz]), bs.
                          translate([ex, ey, ez])))
660 gcpy              return tslist
661 gcpy #
```

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

```
662 gcpy          if self.endmilltype == "bowl":
663 gcpy              inner = cylinder(r1 = self.diameter/2 - self.radius, r2
                          = self.diameter/2 - self.radius, h = self.flute)
664 gcpy              outer = cylinder(r1 = self.diameter/2, r2 = self.
                          diameter/2, h = self.flute - self.radius)
665 gcpy              outer = outer.translate([0,0, self.radius])
666 gcpy              slices = hull(outer, inner)
667 gcpy #     slices = cylinder(r1 = 0.0001, r2 = 0.0001, h = 0.0001, center
                  =False)
668 gcpy              for i in range(1, 90 - self.steps, self.steps):
669 gcpy                  slice = cylinder(r1 = self.diameter / 2 - self.
                              radius + self.radius * Sin(i), r2 = self.
                              diameter / 2 - self.radius + self.radius * Sin(i
                              +self.steps), h = self.radius/90, center=False)
670 gcpy                  slices = hull(slices, slice.translate([0, 0, self.
                              radius - self.radius * Cos(i+self.steps)]))
671 gcpy              tslist.append(hull(slices.translate([bx, by, bz]),
                          slices.translate([ex, ey, ez])))
672 gcpy              return tslist
673 gcpy #
```

3.4.2.6 V The `endmill v` is modeled as a cylinder with a zero width base and a second cylinder for the shaft (note that Python's `math` defaults to radians, hence the need to convert from degrees):

```
674 gcpy          if self.endmilltype == "V":
675 gcpy              v = cylinder(r1=0, r2=(self.diameter / 2), h=((self.
                          diameter / 2) / Tan((self.angle / 2))), center=False
                          )
676 gcpy #               s = cylinder(r1=(self.diameter / 2), r2=(self.
                  diameter / 2), h=self.flute, center=False)
677 gcpy #               sh = s.translate([0, 0, ((self.diameter / 2) / Tan
                  ((self.angle / 2)))])
678 gcpy              tslist.append(hull(v.translate([bx, by, bz]), v.
                          translate([ex, ey, ez])))
679 gcpy              return tslist
```

3.4.2.7 Keyhole Keyhole toolpaths (see: subsection 3.7.0.2.3 are intended for use with tooling which projects beyond the narrower shaft and so will cut usefully underneath the visible surface. Also described as "undercut" tooling, but see below.
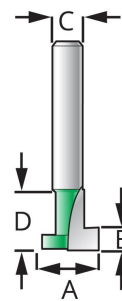
Keyhole Router Bits

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

```
681 gcpy          if self.endmilltype == "keyhole":
682 gcpy              kh = cylinder(r1=(self.diameter / 2), r2=(self.diameter
                          / 2), h=self.flute, center=False)
683 gcpy              sh = (cylinder(r1=(self.radius / 2), r2=(self.radius /
                          2), h=self.flute*2, center=False))
684 gcpy              tslist.append(hull(kh.translate([bx, by, bz]), kh.
                          translate([ex, ey, ez])))
685 gcpy              tslist.append(hull(sh.translate([bx, by, bz]), sh.
                          translate([ex, ey, ez])))
686 gcpy              return tslist
```

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

```
688 gcpy          if self.endmilltype == "tapered ball":
689 gcpy              b = sphere(r=(self.tip / 2))
690 gcpy              s = cylinder(r1=(self.tip / 2), r2=(self.diameter / 2),
                          h=self.flute, center=False)
691 gcpy              bshape = union(b, s)
692 gcpy              tslist.append(hull(bshape.translate([bx, by, bz]),
                          bshape.translate([ex, ey, ez])))
693 gcpy              return tslist
```

dovetail **3.4.2.9 Dovetails** The `dovetail` is modeled as a cylinder with the differing bottom and top diameters determining the angle (though `dt_angle` is still required as a parameter)

```
695 gcpy          if self.endmilltype == "dovetail":
696 gcpy              dt = cylinder(r1=(self.diameter / 2), r2=(self.diameter
                          / 2) - self.flute * Tan(self.angle), h= self.flute,
                          center=False)
697 gcpy              tslist.append(hull(dt.translate([bx, by, bz]), dt.
                          translate([ex, ey, ez])))
698 gcpy              return tslist
699 gcpy          if self.endmilltype == "other":
700 gcpy              tslist = []
701 gcpy #    def dovetail(self, dt_bottomdiameter, dt_topdiameter,
            dt_height, dt_angle):
702 gcpy #        return cylinder(r1=(dt_bottomdiameter / 2), r2=(
            dt_topdiameter / 2), h= dt_height, center=False)
```

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

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

3.4.2.11 Roundover tooling It is not possible to represent all tools using tool changes as coded above which require using a hull operation between 3D representations of the tools at the beginning and end points. Tooling which cannot be so represented will be implemented separately roundover below, see paragraph 3.4.2.10 — roundover tooling will need to generate a list of slices of the tool shape hulled together.

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

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

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

```
723 gcpy        def shaftmovement(self, bx, by, bz, ex, ey, ez):
724 gcpy            tslist = []
725 gcpy            ts = cylinder(r1=(self.shaftdiameter / 2), r2=(self.
                       shaftdiameter / 2), h=self.shaftlength, center = False)
726 gcpy            ts = ts.translate([0, 0, self.shaftheight])
727 gcpy            tslist.append(hull(ts.translate([bx, by, bz]), ts.translate
                       ([ex, ey, ez])))
728 gcpy            return tslist
```

rapid **3.4.2.13 rapid and cut (lines)** A matching pair of commands is made for these, and rapid is used as the basis for a series of commands which match typical usages of G0.
Note the addition of a Laser mode which simulates the tool having been turned off — likely further changes will be required.

```
730 gcpy        def rapid(self, ex, ey, ez, laser = 0):
731 gcpy #           print(self.rapidcolor)
732 gcpy            if laser == 0:
733 gcpy                tm = self.toolmovement(self.xpos(), self.ypos(), self.
                           zpos(), ex, ey, ez)
734 gcpy                tm = color(tm, self.shaftcolor)
735 gcpy                ts = self.shaftmovement(self.xpos(), self.ypos(), self.
                           zpos(), ex, ey, ez)
736 gcpy                ts = color(ts, self.rapidcolor)
737 gcpy                self.toolpaths.extend([tm, ts])
738 gcpy            self.setxpos(ex)
739 gcpy            self.setypos(ey)
740 gcpy            self.setzpos(ez)
741 gcpy
742 gcpy        def cutline(self, ex, ey, ez):
```

```
743 gcpy #          print(self.cutcolor)
744 gcpy #          print(ex, ey, ez)
745 gcpy           tm = self.toolmovement(self.xpos(), self.ypos(), self.zpos
                       (), ex, ey, ez)
746 gcpy           tm = color(tm, self.cutcolor)
747 gcpy           ts = self.shaftmovement(self.xpos(), self.ypos(), self.zpos
                       (), ex, ey, ez)
748 gcpy           ts = color(ts, self.rapidcolor)
749 gcpy           self.setxpos(ex)
750 gcpy           self.setypos(ey)
751 gcpy           self.setzpos(ez)
752 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:

```
754 gcpy     def movetosafeZ(self):
755 gcpy         rapid = self.rapid(self.xpos(), self.ypos(), self.
                     retractheight)
756 gcpy #        if self.generatepaths == True:
757 gcpy #            rapid = self.rapid(self.xpos(), self.ypos(), self.
           retractheight)
758 gcpy #            self.rapids = self.rapids.union(rapid)
759 gcpy #        else:
760 gcpy #  if (generategcode == true) {
761 gcpy #  //   writecomment("PREPOSITION FOR RAPID PLUNGE");Z25.650
762 gcpy #  //G1Z24.663F381.0, "F", str(plunge)
763 gcpy #        if self.generatepaths == False:
764 gcpy #            return rapid
765 gcpy #        else:
766 gcpy #            return cube([0.001, 0.001, 0.001])
767 gcpy         return rapid
768 gcpy
769 gcpy     def rapidXYZ(self, ex, ey, ez):
770 gcpy         rapid = self.rapid(ex, ey, ez)
771 gcpy #        if self.generatepaths == False:
772 gcpy         return rapid
773 gcpy
774 gcpy     def rapidXY(self, ex, ey):
775 gcpy         rapid = self.rapid(ex, ey, self.zpos())
776 gcpy #        if self.generatepaths == True:
777 gcpy #            self.rapids = self.rapids.union(rapid)
778 gcpy #        else:
779 gcpy #        if self.generatepaths == False:
780 gcpy         return rapid
781 gcpy
782 gcpy     def rapidXZ(self, ex, ez):
783 gcpy         rapid = self.rapid(ex, self.ypos(), ez)
784 gcpy #        if self.generatepaths == False:
785 gcpy         return rapid
786 gcpy
787 gcpy     def rapidYZ(self, ey, ez):
788 gcpy         rapid = self.rapid(self.xpos(), ey, ez)
789 gcpy #        if self.generatepaths == False:
790 gcpy         return rapid
791 gcpy
792 gcpy     def rapidX(self, ex):
793 gcpy         rapid = self.rapid(ex, self.ypos(), self.zpos())
794 gcpy #        if self.generatepaths == False:
795 gcpy         return rapid
796 gcpy
797 gcpy     def rapidY(self, ey):
798 gcpy         rapid = self.rapid(self.xpos(), ey, self.zpos())
799 gcpy #        if self.generatepaths == False:
800 gcpy         return rapid
801 gcpy
802 gcpy     def rapidZ(self, ez):
803 gcpy         rapid = [self.rapid(self.xpos(), self.ypos(), ez)]
804 gcpy #        if self.generatepaths == True:
805 gcpy #            self.rapids = self.rapids.union(rapid)
806 gcpy #        else:
807 gcpy #        if self.generatepaths == False:
```

```
808 gcpy            return rapid
```

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

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

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

cut...   The Python commands `cut...` add the `currenttool` to the `toolpath` `hull`ed 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:

```
810 gcpy      def cutlinedxf(self, ex, ey, ez):
811 gcpy          self.dxfline(self.currenttoolnumber(), self.xpos(), self.
                      ypos(), ex, ey)
812 gcpy          self.cutline(ex, ey, ez)
813 gcpy
814 gcpy      def cutlinedxfgc(self, ex, ey, ez):
815 gcpy          self.dxfline(self.currenttoolnumber(), self.xpos(), self.
                      ypos(), ex, ey)
816 gcpy          self.writegc("G01␣X", str(ex), "␣Y", str(ey), "␣Z", str(ez)
                      )
817 gcpy          self.cutline(ex, ey, ez)
818 gcpy
819 gcpy      def cutvertexdxf(self, ex, ey, ez):
820 gcpy          self.addvertex(self.currenttoolnumber(), ex, ey)
821 gcpy          self.writegc("G01␣X", str(ex), "␣Y", str(ey), "␣Z", str(ez)
                      )
822 gcpy          self.cutline(ex, ey, ez)
823 gcpy
824 gcpy      def cutlineXYZwithfeed(self, ex, ey, ez, feed):
825 gcpy          return self.cutline(ex, ey, ez)
826 gcpy
827 gcpy      def cutlineXYZ(self, ex, ey, ez):
828 gcpy          return self.cutline(ex, ey, ez)
829 gcpy
830 gcpy      def cutlineXYwithfeed(self, ex, ey, feed):
831 gcpy          return self.cutline(ex, ey, self.zpos())
832 gcpy
833 gcpy      def cutlineXY(self, ex, ey):
834 gcpy          return self.cutline(ex, ey, self.zpos())
835 gcpy
836 gcpy      def cutlineXZwithfeed(self, ex, ez, feed):
837 gcpy          return self.cutline(ex, self.ypos(), ez)
838 gcpy
839 gcpy      def cutlineXZ(self, ex, ez):
840 gcpy          return self.cutline(ex, self.ypos(), ez)
841 gcpy
842 gcpy      def cutlineXwithfeed(self, ex, feed):
843 gcpy          return self.cutline(ex, self.ypos(), self.zpos())
844 gcpy
845 gcpy      def cutlineX(self, ex):
846 gcpy          return self.cutline(ex, self.ypos(), self.zpos())
847 gcpy
848 gcpy      def cutlineYZ(self, ey, ez):
849 gcpy          return self.cutline(self.xpos(), ey, ez)
850 gcpy
851 gcpy      def cutlineYwithfeed(self, ey, feed):
```

```
852 gcpy          return self.cutline(self.xpos(), ey, self.zpos())
853 gcpy
854 gcpy      def cutlineY(self, ey):
855 gcpy          return self.cutline(self.xpos(), ey, self.zpos())
856 gcpy
857 gcpy      def cutlineZgcfeed(self, ez, feed):
858 gcpy          self.writegc("G01␣Z", str(ez), "F", str(feed))
859 gcpy          return self.cutline(self.xpos(), self.ypos(), ez)
860 gcpy
861 gcpy      def cutlineZwithfeed(self, ez, feed):
862 gcpy          return self.cutline(self.xpos(), self.ypos(), ez)
863 gcpy
864 gcpy      def cutlineZ(self, ez):
865 gcpy          return self.cutline(self.xpos(), self.ypos(), ez)
```

The matching OpenSCAD command is a descriptor:

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

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

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

- cutarcCC — counter-clock-wise

- cutarcNWCW — cut the upper-left quadrant of a circle moving clockwise

- cutarcNWCC — upper-left quadrant counter-clockwise

- cutarcNECW

- cutarcNECC

- cutarcSECW

- cutarcSECC

- cutarcNECW

- cutarcNECC

- cutcircleCC — while it won't matter for generating a DXF, when G-code is implemented direction of cut will be a consideration for that

- cutcircleCW

- cutcircleCCdxf

- cutcircleCWdxf

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

- G-code — G2 (clockwise) and G3 (counter-clockwise) arcs may be specified, and since the endpoint is the positional requirement, it is most likely best to use the offset to the center (I and J), rather than the radius parameter (K) G2/3 ...

- DXF — dxfarc(xcenter, ycenter, radius, anglebegin, endangle, tn)

- approximation of arc using lines (OpenSCAD) in both clock-wise and counter-clock-wise directions

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

Parameters which will need to be passed in are:

- `ex` — note that the matching origins (`bx`, `by`, `bz`) as well as the (current) toolnumber are accessed using the appropriate commands for machine position

- `ey`

- `ez` — allowing a different Z position will make possible threading and similar helical tool-paths

- `xcenter` — the center position will be specified as an absolute position which will require calculating the offset when it is used for G-code's `IJ`, for which `xctr/yctr` are suggested

- `ycenter`

- `radius` — while this could be calculated, passing it in as a parameter is both convenient and (potentially) could be used as a check on the other parameters

- `tpzreldim` — the relative depth (or increase in height) of the current cutting motion

Since OpenSCAD does not have an arc movement command it is necessary to iterate through a

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

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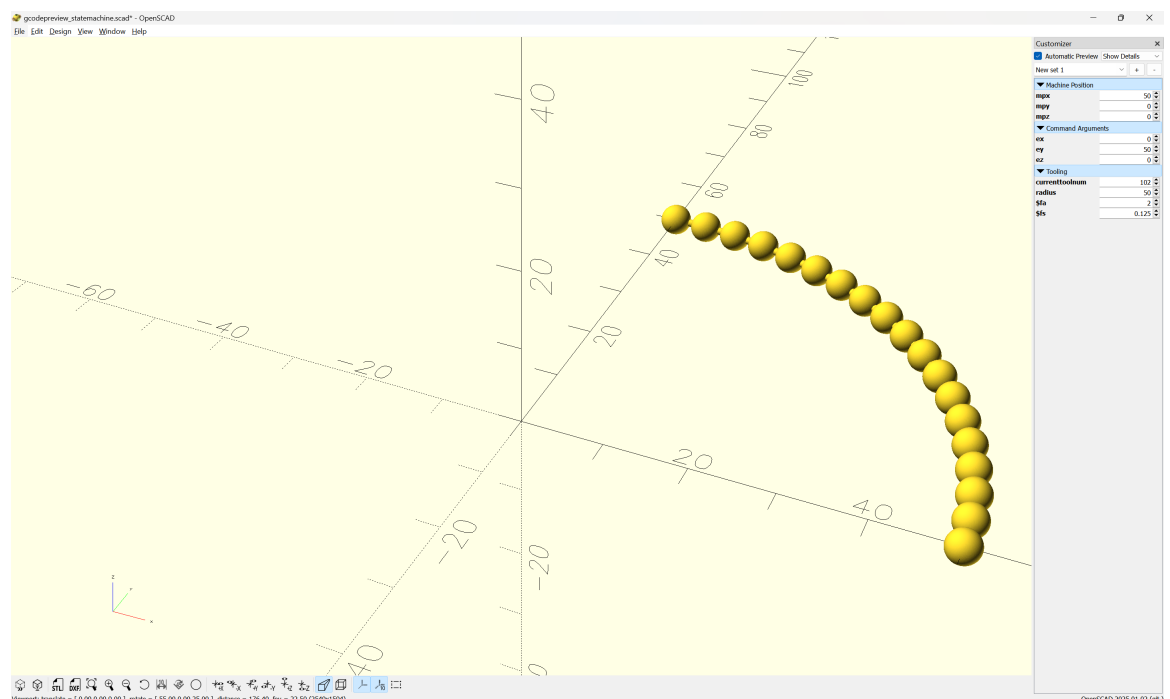
The code for which makes this obvious:

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

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

/* [Tooling] */
```

```
currenttoolnum = 102;

machine_extents();

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

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

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

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

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

```
867 gcpy     def cutarcCC(self, barc, earc, xcenter, ycenter, radius,
                tpzreldim, stepsizearc=1):
868 gcpy         tpzinc = tpzreldim / (earc - barc)
869 gcpy         i = barc
870 gcpy         while i < earc:
871 gcpy             self.cutline(xcenter + radius * Cos(math.radians(i)),
                         ycenter + radius * Sin(math.radians(i)), self.zpos()
                         +tpzinc)
872 gcpy             i += stepsizearc
873 gcpy         self.setxpos(xcenter + radius * Cos(math.radians(earc)))
874 gcpy         self.setypos(ycenter + radius * Sin(math.radians(earc)))
875 gcpy
876 gcpy     def cutarcCW(self, barc, earc, xcenter, ycenter, radius,
                tpzreldim, stepsizearc=1):
877 gcpy #        print(str(self.zpos()))
878 gcpy #        print(str(ez))
879 gcpy #        print(str(barc - earc))
880 gcpy #        tpzinc = ez - self.zpos() / (barc - earc)
881 gcpy #        print(str(tzinc))
882 gcpy #        global toolpath
883 gcpy #        print("Entering n toolpath")
884 gcpy         tpzinc = tpzreldim / (barc - earc)
885 gcpy #        cts = self.currenttoolshape
886 gcpy #        toolpath = cts
```

```
887 gcpy #        toolpath = toolpath.translate([self.xpos(), self.ypos(),
            self.zpos()])
888 gcpy #        toolpath = []
889 gcpy        i = barc
890 gcpy        while i > earc:
891 gcpy            self.cutline(xcenter + radius * Cos(math.radians(i)),
                        ycenter + radius * Sin(math.radians(i)), self.zpos()
                        +tpzinc)
892 gcpy #            self.setxpos(xcenter + radius * Cos(math.radians(i)))
893 gcpy #            self.setypos(ycenter + radius * Sin(math.radians(i)))
894 gcpy #            print(str(self.xpos()), str(self.ypos()), str(self.zpos
            ())))
895 gcpy #            self.setzpos(self.zpos()+tpzinc)
896 gcpy            i += abs(stepsizearc) * -1
897 gcpy #        self.dxfarc(self.currenttoolnumber(), xcenter, ycenter,
            radius, barc, earc)
898 gcpy #        if self.generatepaths == True:
899 gcpy #            print("Unioning n toolpath")
900 gcpy #            self.toolpaths = self.toolpaths.union(toolpath)
901 gcpy #        else:
902 gcpy        self.setxpos(xcenter + radius * Cos(math.radians(earc)))
903 gcpy        self.setypos(ycenter + radius * Sin(math.radians(earc)))
904 gcpy #        self.toolpaths.extend(toolpath)
905 gcpy #        if self.generatepaths == False:
906 gcpy #        return toolpath
907 gcpy #        else:
908 gcpy #            return cube([0.01, 0.01, 0.01])
```

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

```
910 gcpy    def cutarcCWdxf(self, barc, earc, xcenter, ycenter, radius,
                tpzreldim, stepsizearc=1):
911 gcpy        self.cutarcCW(barc, earc, xcenter, ycenter, radius,
                    tpzreldim, stepsizearc=1)
912 gcpy        self.dxfarc(self.currenttoolnumber(), xcenter, ycenter,
                    radius, earc, barc)
913 gcpy #        if self.generatepaths == False:
914 gcpy #        return toolpath
915 gcpy #        else:
916 gcpy #            return cube([0.01, 0.01, 0.01])
917 gcpy
918 gcpy    def cutarcCCdxf(self, barc, earc, xcenter, ycenter, radius,
                tpzreldim, stepsizearc=1):
919 gcpy        self.cutarcCC(barc, earc, xcenter, ycenter, radius,
                    tpzreldim, stepsizearc=1)
920 gcpy        self.dxfarc(self.currenttoolnumber(), xcenter, ycenter,
                    radius, barc, earc)
```

Matching OpenSCAD modules are easily made:

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

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

```
922 gcpy    def cutquarterCCNE(self, ex, ey, ez, radius):
923 gcpy        if self.zpos() == ez:
924 gcpy            tpzinc = 0
925 gcpy        else:
926 gcpy            tpzinc = (ez - self.zpos()) / 90
927 gcpy #        print("tpzinc ", tpzinc)
928 gcpy        i = 1
929 gcpy        while i < 91:
930 gcpy            self.cutline(ex + radius * Cos(i), ey - radius + radius
                        * Sin(i), self.zpos()+tpzinc)
931 gcpy            i += 1
932 gcpy
933 gcpy    def cutquarterCCNW(self, ex, ey, ez, radius):
934 gcpy        if self.zpos() == ez:
935 gcpy            tpzinc = 0
936 gcpy        else:
```

```
937 gcpy                    tpzinc = (ez - self.zpos()) / 90
938 gcpy #                   tpzinc = (self.zpos() + ez) / 90
939 gcpy            print("tpzinc␣", tpzinc)
940 gcpy            i = 91
941 gcpy            while i < 181:
942 gcpy                self.cutline(ex + radius + radius * Cos(i), ey + radius
                           * Sin(i), self.zpos()+tpzinc)
943 gcpy                i += 1
944 gcpy
945 gcpy        def cutquarterCCSW(self, ex, ey, ez, radius):
946 gcpy            if self.zpos() == ez:
947 gcpy                tpzinc = 0
948 gcpy            else:
949 gcpy                tpzinc = (ez - self.zpos()) / 90
950 gcpy #                   tpzinc = (self.zpos() + ez) / 90
951 gcpy            print("tpzinc␣", tpzinc)
952 gcpy            i = 181
953 gcpy            while i < 271:
954 gcpy                self.cutline(ex + radius * Cos(i), ey + radius + radius
                           * Sin(i), self.zpos()+tpzinc)
955 gcpy                i += 1
956 gcpy
957 gcpy        def cutquarterCCSE(self, ex, ey, ez, radius):
958 gcpy            if self.zpos() == ez:
959 gcpy                tpzinc = 0
960 gcpy            else:
961 gcpy                tpzinc = (ez - self.zpos()) / 90
962 gcpy #                   tpzinc = (self.zpos() + ez) / 90
963 gcpy #            print("tpzinc ", tpzinc)
964 gcpy            i = 271
965 gcpy            while i < 361:
966 gcpy                self.cutline(ex - radius + radius * Cos(i), ey + radius
                           * Sin(i), self.zpos()+tpzinc)
967 gcpy                i += 1
968 gcpy
969 gcpy        def cutquarterCCNEdxf(self, ex, ey, ez, radius):
970 gcpy            self.cutquarterCCNE(ex, ey, ez, radius)
971 gcpy            self.dxfarc(self.currenttoolnumber(), ex, ey - radius,
                        radius,  0, 90)
972 gcpy
973 gcpy        def cutquarterCCNWdxf(self, ex, ey, ez, radius):
974 gcpy            self.cutquarterCCNW(ex, ey, ez, radius)
975 gcpy            self.dxfarc(self.currenttoolnumber(), ex + radius, ey,
                        radius, 90, 180)
976 gcpy
977 gcpy        def cutquarterCCSWdxf(self, ex, ey, ez, radius):
978 gcpy            self.cutquarterCCSW(ex, ey, ez, radius)
979 gcpy            self.dxfarc(self.currenttoolnumber(), ex, ey + radius,
                        radius, 180, 270)
980 gcpy
981 gcpy        def cutquarterCCSEdxf(self, ex, ey, ez, radius):
982 gcpy            self.cutquarterCCSE(ex, ey, ez, radius)
983 gcpy            self.dxfarc(self.currenttoolnumber(), ex - radius, ey,
                        radius, 270, 360)
```

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

```
114 gcpscad }
115 gcpscad
116 gcpscad module cutquarterCCSWdxf(ex, ey, ez, radius){
117 gcpscad     gcp.cutquarterCCSWdxf(ex, ey, ez, radius);
118 gcpscad }
119 gcpscad
120 gcpscad module cutquarterCCSEdxf(self, ex, ey, ez, radius){
121 gcpscad     gcp.cutquarterCCSEdxf(ex, ey, ez, radius);
122 gcpscad }
```

3.4.3 tooldiameter

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

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

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

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

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

```
 985 gcpy     def tool_diameter(self, ptd_tool, ptd_depth):
 986 gcpy # Square 122, 112, 102, 201
 987 gcpy         if ptd_tool == 122:
 988 gcpy             return 0.79375
 989 gcpy         if ptd_tool == 112:
 990 gcpy             return 1.5875
 991 gcpy         if ptd_tool == 102:
 992 gcpy             return 3.175
 993 gcpy         if ptd_tool == 201:
 994 gcpy             return 6.35
 995 gcpy # Ball 121, 111, 101, 202
 996 gcpy         if ptd_tool == 122:
 997 gcpy             if ptd_depth > 0.396875:
 998 gcpy                 return 0.79375
 999 gcpy             else:
1000 gcpy                 return ptd_tool
1001 gcpy         if ptd_tool == 112:
1002 gcpy             if ptd_depth > 0.79375:
1003 gcpy                 return 1.5875
1004 gcpy             else:
1005 gcpy                 return ptd_tool
1006 gcpy         if ptd_tool == 101:
1007 gcpy             if ptd_depth > 1.5875:
1008 gcpy                 return 3.175
1009 gcpy             else:
1010 gcpy                 return ptd_tool
1011 gcpy         if ptd_tool == 202:
1012 gcpy             if ptd_depth > 3.175:
1013 gcpy                 return 6.35
1014 gcpy             else:
1015 gcpy                 return ptd_tool
1016 gcpy # V 301, 302, 390
1017 gcpy         if ptd_tool == 301:
1018 gcpy             return ptd_tool
1019 gcpy         if ptd_tool == 302:
1020 gcpy             return ptd_tool
1021 gcpy         if ptd_tool == 390:
1022 gcpy             return ptd_tool
1023 gcpy # Keyhole
1024 gcpy         if ptd_tool == 374:
1025 gcpy             if ptd_depth < 3.175:
1026 gcpy                 return 9.525
1027 gcpy             else:
1028 gcpy                 return 6.35
1029 gcpy         if ptd_tool == 375:
1030 gcpy             if ptd_depth < 3.175:
1031 gcpy                 return 9.525
1032 gcpy             else:
1033 gcpy                 return 8
```

```
1034 gcpy            if ptd_tool == 376:
1035 gcpy                if ptd_depth < 4.7625:
1036 gcpy                    return 12.7
1037 gcpy                else:
1038 gcpy                    return 6.35
1039 gcpy            if ptd_tool == 378:
1040 gcpy                if ptd_depth < 4.7625:
1041 gcpy                    return 12.7
1042 gcpy                else:
1043 gcpy                    return 8
1044 gcpy # Dovetail
1045 gcpy            if ptd_tool == 814:
1046 gcpy                if ptd_depth > 12.7:
1047 gcpy                    return 6.35
1048 gcpy                else:
1049 gcpy                    return ptd_tool
1050 gcpy            if ptd_tool == 808079:
1051 gcpy                if ptd_depth > 20.95:
1052 gcpy                    return 6.816
1053 gcpy                else:
1054 gcpy                    return ptd_tool
1055 gcpy # Bowl Bit
1056 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
1057 gcpy            if ptd_tool == 45982:
1058 gcpy                if ptd_depth > 6.35:
1059 gcpy                    return 15.875
1060 gcpy                else:
1061 gcpy                    return ptd_tool
1062 gcpy # Tapered Ball Nose
1063 gcpy            if ptd_tool == 204:
1064 gcpy                if ptd_depth > 6.35:
1065 gcpy                    return ptd_tool
1066 gcpy            if ptd_tool == 304:
1067 gcpy                if ptd_depth > 6.35:
1068 gcpy                    return ptd_tool
1069 gcpy                else:
1070 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:

```
1072 gcpy    def tool_radius(self, ptd_tool, ptd_depth):
1073 gcpy        tr = self.tool_diameter(ptd_tool, ptd_depth)/2
1074 gcpy        return tr
```

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

3.4.4   Feeds and Speeds

feed   There are several possibilities for handling feeds and speeds. Currently, base values for `feed`,
plunge   `plunge`, and `speed` are used, which may then be adjusted using various `<tooldescriptor>_ratio`
speed   values, as an acknowledgement of the likelihood of a trim router being used as a spindle, the
assumption is that the `speed` will remain unchanged.
   The tools which need to be calculated thus are those in addition to the `large_square` tool:

- `small_square_ratio`

- `small_ball_ratio`

- `large_ball_ratio`

- `small_V_ratio`

- `large_V_ratio`

- `KH_ratio`

- `DT_ratio`

3.5   Difference of Stock, Rapids, and Toolpaths

At the end of cutting it will be necessary to subtract the accumulated toolpaths and rapids from
the stock.
   For Python, the initial 3D model is stored in the variable `stock`:

```
1076 gcpy     def stockandtoolpaths(self, option = "stockandtoolpaths"):
1077 gcpy         if option == "stock":
1078 gcpy             show(self.stock)
1079 gcpy         elif option == "toolpaths":
1080 gcpy             show(self.toolpaths)
1081 gcpy         elif option == "rapids":
1082 gcpy             show(self.rapids)
1083 gcpy         else:
1084 gcpy             part = self.stock.difference(self.rapids)
1085 gcpy             part = self.stock.difference(self.toolpaths)
1086 gcpy             show(part)
```

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

```
1088 gcpy     def returnstockandtoolpaths(self):
1089 gcpy         part = self.stock.difference(self.toolpaths)
1090 gcpy         return part
```

and then make use of that specific command for OpenSCAD:

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

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

3.6   Output files

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

3.6.1   Python and OpenSCAD File Handling

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

```
1092 gcpy     def writegc(self, *arguments):
1093 gcpy         if self.generategcode == True:
1094 gcpy             line_to_write = ""
1095 gcpy             for element in arguments:
1096 gcpy                 line_to_write += element
1097 gcpy             self.gc.write(line_to_write)
1098 gcpy             self.gc.write("\n")
1099 gcpy
1100 gcpy     def writedxf(self, toolnumber, *arguments):
1101 gcpy #       global dxfclosed
1102 gcpy         line_to_write = ""
1103 gcpy         for element in arguments:
1104 gcpy             line_to_write += element
1105 gcpy         if self.generatedxf == True:
1106 gcpy             if self.dxfclosed == False:
1107 gcpy                 self.dxf.write(line_to_write)
1108 gcpy                 self.dxf.write("\n")
1109 gcpy         if self.generatedxfs == True:
1110 gcpy             self.writedxfs(toolnumber, line_to_write)
1111 gcpy
1112 gcpy     def writedxfs(self, toolnumber, line_to_write):
1113 gcpy #       print("Processing writing toolnumber", toolnumber)
1114 gcpy #       line_to_write = ""
1115 gcpy #       for element in arguments:
1116 gcpy #           line_to_write += element
1117 gcpy         if (toolnumber == 0):
1118 gcpy             return
1119 gcpy         elif self.generatedxfs == True:
1120 gcpy             if (self.large_square_tool_num == toolnumber):
1121 gcpy                 self.dxflgsq.write(line_to_write)
1122 gcpy                 self.dxflgsq.write("\n")
1123 gcpy             if (self.small_square_tool_num == toolnumber):
1124 gcpy                 self.dxfsmsq.write(line_to_write)
1125 gcpy                 self.dxfsmsq.write("\n")
```

```
1126 gcpy                    if (self.large_ball_tool_num == toolnumber):
1127 gcpy                        self.dxflgbl.write(line_to_write)
1128 gcpy                        self.dxflgbl.write("\n")
1129 gcpy                    if (self.small_ball_tool_num == toolnumber):
1130 gcpy                        self.dxfsmbl.write(line_to_write)
1131 gcpy                        self.dxfsmbl.write("\n")
1132 gcpy                    if (self.large_V_tool_num == toolnumber):
1133 gcpy                        self.dxflgV.write(line_to_write)
1134 gcpy                        self.dxflgV.write("\n")
1135 gcpy                    if (self.small_V_tool_num == toolnumber):
1136 gcpy                        self.dxfsmV.write(line_to_write)
1137 gcpy                        self.dxfsmV.write("\n")
1138 gcpy                    if (self.DT_tool_num == toolnumber):
1139 gcpy                        self.dxfDT.write(line_to_write)
1140 gcpy                        self.dxfDT.write("\n")
1141 gcpy                    if (self.KH_tool_num == toolnumber):
1142 gcpy                        self.dxfKH.write(line_to_write)
1143 gcpy                        self.dxfKH.write("\n")
1144 gcpy                    if (self.Roundover_tool_num == toolnumber):
1145 gcpy                        self.dxfRt.write(line_to_write)
1146 gcpy                        self.dxfRt.write("\n")
1147 gcpy                    if (self.MISC_tool_num == toolnumber):
1148 gcpy                        self.dxfMt.write(line_to_write)
1149 gcpy                        self.dxfMt.write("\n")
```

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

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

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

```
130 gcpscad module opendxffile(basefilename){
131 gcpscad    gcp.opendxffile(basefilename);
132 gcpscad }
133 gcpscad
134 gcpscad 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 gcpscad    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 gcpscad }
```

opengcodefile   With matching OpenSCAD commands: opengcodefile for OpenSCAD:

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

and Python:

```
1151 gcpy    def opengcodefile(self, basefilename = "export",
1152 gcpy                        currenttoolnum = 102,
1153 gcpy                        toolradius = 3.175,
1154 gcpy                        plunge = 400,
1155 gcpy                        feed = 1600,
1156 gcpy                        speed = 10000
1157 gcpy                        ):
1158 gcpy        self.basefilename = basefilename
1159 gcpy        self.currenttoolnum = currenttoolnum
1160 gcpy        self.toolradius = toolradius
1161 gcpy        self.plunge = plunge
1162 gcpy        self.feed = feed
1163 gcpy        self.speed = speed
1164 gcpy        if self.generategcode == True:
```

```
1165 gcpy                     self.gcodefilename = basefilename + ".nc"
1166 gcpy                     self.gc = open(self.gcodefilename, "w")
1167 gcpy                     self.writegc("(Design␣File:␣" + self.basefilename + ")"
                                 )
1168 gcpy
1169 gcpy        def opendxffile(self, basefilename = "export"):
1170 gcpy            self.basefilename = basefilename
1171 gcpy #            global generatedxfs
1172 gcpy #            global dxfclosed
1173 gcpy            self.dxfclosed = False
1174 gcpy            self.dxfcolor = "Black"
1175 gcpy            if self.generatedxf == True:
1176 gcpy                self.generatedxfs = False
1177 gcpy                self.dxffilename = basefilename + ".dxf"
1178 gcpy                self.dxf = open(self.dxffilename, "w")
1179 gcpy                self.dxfpreamble(-1)
1180 gcpy
1181 gcpy        def opendxffiles(self, basefilename = "export",
1182 gcpy                         large_square_tool_num = 0,
1183 gcpy                         small_square_tool_num = 0,
1184 gcpy                         large_ball_tool_num = 0,
1185 gcpy                         small_ball_tool_num = 0,
1186 gcpy                         large_V_tool_num = 0,
1187 gcpy                         small_V_tool_num = 0,
1188 gcpy                         DT_tool_num = 0,
1189 gcpy                         KH_tool_num = 0,
1190 gcpy                         Roundover_tool_num = 0,
1191 gcpy                         MISC_tool_num = 0):
1192 gcpy #         global generatedxfs
1193 gcpy            self.basefilename = basefilename
1194 gcpy            self.generatedxfs = True
1195 gcpy            self.large_square_tool_num = large_square_tool_num
1196 gcpy            self.small_square_tool_num = small_square_tool_num
1197 gcpy            self.large_ball_tool_num = large_ball_tool_num
1198 gcpy            self.small_ball_tool_num = small_ball_tool_num
1199 gcpy            self.large_V_tool_num = large_V_tool_num
1200 gcpy            self.small_V_tool_num = small_V_tool_num
1201 gcpy            self.DT_tool_num = DT_tool_num
1202 gcpy            self.KH_tool_num = KH_tool_num
1203 gcpy            self.Roundover_tool_num = Roundover_tool_num
1204 gcpy            self.MISC_tool_num = MISC_tool_num
1205 gcpy            if self.generatedxf == True:
1206 gcpy                if (large_square_tool_num > 0):
1207 gcpy                    self.dxflgsqfilename = basefilename + str(
                                 large_square_tool_num) + ".dxf"
1208 gcpy #                   print("Opening ", str(self.dxflgsqfilename))
1209 gcpy                    self.dxflgsq = open(self.dxflgsqfilename, "w")
1210 gcpy                if (small_square_tool_num > 0):
1211 gcpy #                   print("Opening small square")
1212 gcpy                    self.dxfsmsqfilename = basefilename + str(
                                 small_square_tool_num) + ".dxf"
1213 gcpy                    self.dxfsmsq = open(self.dxfsmsqfilename, "w")
1214 gcpy                if (large_ball_tool_num > 0):
1215 gcpy #                   print("Opening large ball")
1216 gcpy                    self.dxflgblfilename = basefilename + str(
                                 large_ball_tool_num) + ".dxf"
1217 gcpy                    self.dxflgbl = open(self.dxflgblfilename, "w")
1218 gcpy                if (small_ball_tool_num > 0):
1219 gcpy #                   print("Opening small ball")
1220 gcpy                    self.dxfsmblfilename = basefilename + str(
                                 small_ball_tool_num) + ".dxf"
1221 gcpy                    self.dxfsmbl = open(self.dxfsmblfilename, "w")
1222 gcpy                if (large_V_tool_num > 0):
1223 gcpy #                   print("Opening large V")
1224 gcpy                    self.dxflgVfilename = basefilename + str(
                                 large_V_tool_num) + ".dxf"
1225 gcpy                    self.dxflgV = open(self.dxflgVfilename, "w")
1226 gcpy                if (small_V_tool_num > 0):
1227 gcpy #                   print("Opening small V")
1228 gcpy                    self.dxfsmVfilename = basefilename + str(
                                 small_V_tool_num) + ".dxf"
1229 gcpy                    self.dxfsmV = open(self.dxfsmVfilename, "w")
1230 gcpy                if (DT_tool_num > 0):
1231 gcpy #                   print("Opening DT")
1232 gcpy                    self.dxfDTfilename = basefilename + str(DT_tool_num
                                 ) + ".dxf"
1233 gcpy                    self.dxfDT = open(self.dxfDTfilename, "w")
1234 gcpy                if (KH_tool_num > 0):
```

```
1235 gcpy #                      print("Opening KH")
1236 gcpy                       self.dxfKHfilename = basefilename + str(KH_tool_num
                                    ) + ".dxf"
1237 gcpy                       self.dxfKH = open(self.dxfKHfilename, "w")
1238 gcpy              if (Roundover_tool_num > 0):
1239 gcpy #                      print("Opening Rt")
1240 gcpy                       self.dxfRtfilename = basefilename + str(
                                    Roundover_tool_num) + ".dxf"
1241 gcpy                       self.dxfRt = open(self.dxfRtfilename, "w")
1242 gcpy              if (MISC_tool_num > 0):
1243 gcpy #                      print("Opening Mt")
1244 gcpy                       self.dxfMtfilename = basefilename + str(
                                    MISC_tool_num) + ".dxf"
1245 gcpy                       self.dxfMt = open(self.dxfMtfilename, "w")
```

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

```
1246 gcpy              if (large_square_tool_num > 0):
1247 gcpy                  self.dxfpreamble(large_square_tool_num)
1248 gcpy              if (small_square_tool_num > 0):
1249 gcpy                  self.dxfpreamble(small_square_tool_num)
1250 gcpy              if (large_ball_tool_num > 0):
1251 gcpy                  self.dxfpreamble(large_ball_tool_num)
1252 gcpy              if (small_ball_tool_num > 0):
1253 gcpy                  self.dxfpreamble(small_ball_tool_num)
1254 gcpy              if (large_V_tool_num > 0):
1255 gcpy                  self.dxfpreamble(large_V_tool_num)
1256 gcpy              if (small_V_tool_num > 0):
1257 gcpy                  self.dxfpreamble(small_V_tool_num)
1258 gcpy              if (DT_tool_num > 0):
1259 gcpy                  self.dxfpreamble(DT_tool_num)
1260 gcpy              if (KH_tool_num > 0):
1261 gcpy                  self.dxfpreamble(KH_tool_num)
1262 gcpy              if (Roundover_tool_num > 0):
1263 gcpy                  self.dxfpreamble(Roundover_tool_num)
1264 gcpy              if (MISC_tool_num > 0):
1265 gcpy                  self.dxfpreamble(MISC_tool_num)
```

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

Future considerations:

- Multiple Preview Modes:

- Fast Preview: Write all movements with both begin and end positions into a list for a specific tool — as this is done, check for a previous movement between those positions and compare depths and tool number — keep only the deepest movement for a given tool.

- Motion Preview: Work up a 3D model of the machine and actually show the stock in relation to it,

3.6.2   DXF Overview

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

```
0
SECTION
2
ENTITIES
0
LWPOLYLINE
90
2
70
0
43
0
10
-31.375
20
-34.9152
10
-31.375
20
-18.75
0
ARC
```

```
10
-54.75
20
-37.5
40
4
50
0
51
90
0
ENDSEC
0
EOF
```

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

dxfwrite

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

writedxflgbl   • Ball nose, large (lgbl) `writedxflgbl`

writedxfsmbl   • Ball nose, small (smbl) `writedxfsmbl`

writedxflgsq   • Square, large (lgsq) `writedxflgsq`

writedxfsmsq   • Square, small (smsq) `writedxfsmsq`

writedxflgV   • V, large (lgV) `writedxflgV`

writedxfsmV   • V, small (smV) `writedxfsmV`

writedxfKH   • Keyhole (KH) `writedxfKH`

writedxfDT   • Dovetail (DT) `writedxfDT`

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

```
1267 gcpy       def dxfpreamble(self, tn):
1268 gcpy #         self.writedxf(tn, str(tn))
1269 gcpy           self.writedxf(tn, "0")
1270 gcpy           self.writedxf(tn, "SECTION")
1271 gcpy           self.writedxf(tn, "2")
1272 gcpy           self.writedxf(tn, "ENTITIES")
```

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

dxfline   • a line `dxfline`

beginpolyline   • connected lines `beginpolyline/addvertex/closepolyline`
addvertex
closepolyline   • arc `dxfarc`
dxfarc
dxfcircle   • circle — a notable option would be for the arc to close on itself, creating a circle `dxfcircle`

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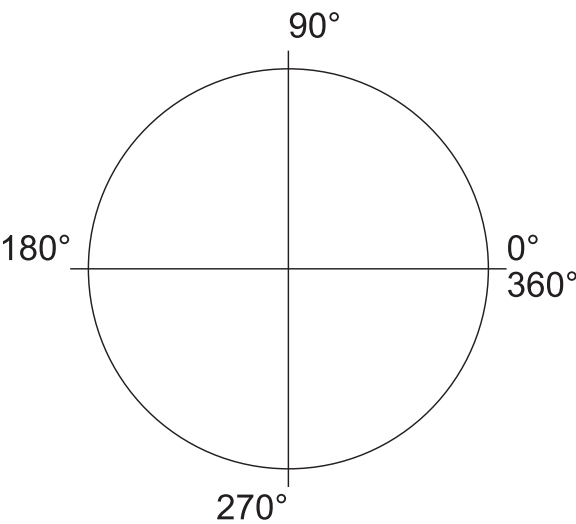
Note that arcs of greater than 90 degrees are not rendered accurately (in certain applications at least), so, for the sake of precision, they should be limited to a swing of 90 degrees or less. Further note that 4 arcs may be stitched together to make a circle:

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

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

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

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

```
1274 gcpy    def setdxfcolor(self, color):
1275 gcpy        self.dxfcolor = color
1276 gcpy        self.cutcolor = color
1277 gcpy
1278 gcpy    def writedxfcolor(self, tn):
1279 gcpy        self.writedxf(tn, "8")
1280 gcpy        if (self.dxfcolor == "Black"):
1281 gcpy            self.writedxf(tn, "Layer_Black")
1282 gcpy        if (self.dxfcolor == "Red"):
1283 gcpy            self.writedxf(tn, "Layer_Red")
1284 gcpy        if (self.dxfcolor == "Yellow"):
1285 gcpy            self.writedxf(tn, "Layer_Yellow")
1286 gcpy        if (self.dxfcolor == "Green"):
1287 gcpy            self.writedxf(tn, "Layer_Green")
1288 gcpy        if (self.dxfcolor == "Cyan"):
1289 gcpy            self.writedxf(tn, "Layer_Cyan")
1290 gcpy        if (self.dxfcolor == "Blue"):
1291 gcpy            self.writedxf(tn, "Layer_Blue")
1292 gcpy        if (self.dxfcolor == "Magenta"):
1293 gcpy            self.writedxf(tn, "Layer_Magenta")
1294 gcpy        if (self.dxfcolor == "White"):
1295 gcpy            self.writedxf(tn, "Layer_White")
1296 gcpy        if (self.dxfcolor == "Dark␣Gray"):
1297 gcpy            self.writedxf(tn, "Layer_Dark_Gray")
1298 gcpy        if (self.dxfcolor == "Light␣Gray"):
1299 gcpy            self.writedxf(tn, "Layer_Light_Gray")
1300 gcpy
1301 gcpy        self.writedxf(tn, "62")
1302 gcpy        if (self.dxfcolor == "Black"):
1303 gcpy            self.writedxf(tn, "0")
1304 gcpy        if (self.dxfcolor == "Red"):
1305 gcpy            self.writedxf(tn, "1")
```

```
1306 gcpy                    if (self.dxfcolor == "Yellow"):
1307 gcpy                        self.writedxf(tn, "2")
1308 gcpy                    if (self.dxfcolor == "Green"):
1309 gcpy                        self.writedxf(tn, "3")
1310 gcpy                    if (self.dxfcolor == "Cyan"):
1311 gcpy                        self.writedxf(tn, "4")
1312 gcpy                    if (self.dxfcolor == "Blue"):
1313 gcpy                        self.writedxf(tn, "5")
1314 gcpy                    if (self.dxfcolor == "Magenta"):
1315 gcpy                        self.writedxf(tn, "6")
1316 gcpy                    if (self.dxfcolor == "White"):
1317 gcpy                        self.writedxf(tn, "7")
1318 gcpy                    if (self.dxfcolor == "Dark␣Gray"):
1319 gcpy                        self.writedxf(tn, "8")
1320 gcpy                    if (self.dxfcolor == "Light␣Gray"):
1321 gcpy                        self.writedxf(tn, "9")
```

```
142 gcpscad module setdxfcolor(color){
143 gcpscad     gcp.setdxfcolor(color);
144 gcpscad }
```

A further refinement would be to connect multiple line segments/arcs into a larger polyline, but since most CAM tools implicitly join elements on import, that is not necessary.

There are three possible interactions for DXF elements and toolpaths:

- describe the motion of the tool

- define a perimeter of an area which will be cut by a tool

- define a centerpoint for a specialty toolpath such as Drill or Keyhole

and it is possible that multiple such elements could be instantiated for a given toolpath.

When writing out to a DXF file there is a pair of commands, a public facing command which takes in a tool number in addition to the coordinates which then writes out to the main DXF file and then calls an internal command to which repeats the call with the tool number so as to write it out to the matching file.

```
1323 gcpy     def dxfline(self, tn, xbegin, ybegin, xend, yend):
1324 gcpy         self.writedxf(tn, "0")
1325 gcpy         self.writedxf(tn, "LINE")
1326 gcpy #
1327 gcpy         self.writedxfcolor(tn)
1328 gcpy #
1329 gcpy         self.writedxf(tn, "10")
1330 gcpy         self.writedxf(tn, str(xbegin))
1331 gcpy         self.writedxf(tn, "20")
1332 gcpy         self.writedxf(tn, str(ybegin))
1333 gcpy         self.writedxf(tn, "30")
1334 gcpy         self.writedxf(tn, "0.0")
1335 gcpy         self.writedxf(tn, "11")
1336 gcpy         self.writedxf(tn, str(xend))
1337 gcpy         self.writedxf(tn, "21")
1338 gcpy         self.writedxf(tn, str(yend))
1339 gcpy         self.writedxf(tn, "31")
1340 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:

```
1342 gcpy     def beginpolyline(self, tn):#, xbegin, ybegin
1343 gcpy         self.writedxf(tn, "0")
1344 gcpy         self.writedxf(tn, "POLYLINE")
1345 gcpy         self.writedxf(tn, "8")
1346 gcpy         self.writedxf(tn, "default")
1347 gcpy         self.writedxf(tn, "66")
1348 gcpy         self.writedxf(tn, "1")
1349 gcpy #
1350 gcpy         self.writedxfcolor(tn)
1351 gcpy #
1352 gcpy #        self.writedxf(tn, "10")
1353 gcpy #        self.writedxf(tn, str(xbegin))
1354 gcpy #        self.writedxf(tn, "20")
1355 gcpy #        self.writedxf(tn, str(ybegin))
1356 gcpy #        self.writedxf(tn, "30")
```

```
1357 gcpy #          self.writedxf(tn, "0.0")
1358 gcpy          self.writedxf(tn, "70")
1359 gcpy          self.writedxf(tn, "0")
```

then add as many vertexes as are wanted:

```
1361 gcpy      def addvertex(self, tn, xend, yend):
1362 gcpy          self.writedxf(tn, "0")
1363 gcpy          self.writedxf(tn, "VERTEX")
1364 gcpy          self.writedxf(tn, "8")
1365 gcpy          self.writedxf(tn, "default")
1366 gcpy          self.writedxf(tn, "70")
1367 gcpy          self.writedxf(tn, "32")
1368 gcpy          self.writedxf(tn, "10")
1369 gcpy          self.writedxf(tn, str(xend))
1370 gcpy          self.writedxf(tn, "20")
1371 gcpy          self.writedxf(tn, str(yend))
1372 gcpy          self.writedxf(tn, "30")
1373 gcpy          self.writedxf(tn, "0.0")
```

then end the sequence:

```
1375 gcpy      def closepolyline(self, tn):
1376 gcpy          self.writedxf(tn, "0")
1377 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).

```
1379 gcpy      def dxfarc(self, tn, xcenter, ycenter, radius, anglebegin,
                  endangle):
1380 gcpy          if (self.generatedxf == True):
1381 gcpy              self.writedxf(tn, "0")
1382 gcpy              self.writedxf(tn, "ARC")
1383 gcpy #
1384 gcpy              self.writedxfcolor(tn)
1385 gcpy #
1386 gcpy              self.writedxf(tn, "10")
1387 gcpy              self.writedxf(tn, str(xcenter))
1388 gcpy              self.writedxf(tn, "20")
1389 gcpy              self.writedxf(tn, str(ycenter))
1390 gcpy              self.writedxf(tn, "40")
1391 gcpy              self.writedxf(tn, str(radius))
1392 gcpy              self.writedxf(tn, "50")
1393 gcpy              self.writedxf(tn, str(anglebegin))
1394 gcpy              self.writedxf(tn, "51")
1395 gcpy              self.writedxf(tn, str(endangle))
1396 gcpy
1397 gcpy      def gcodearc(self, tn, xcenter, ycenter, radius, anglebegin,
                  endangle):
1398 gcpy          if (self.generategcode == True):
1399 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.

```
1401 gcpy      def cutarcNECCdxf(self, ex, ey, ez, xcenter, ycenter, radius):
1402 gcpy #        global toolpath
1403 gcpy #        toolpath = self.currenttool()
1404 gcpy #        toolpath = toolpath.translate([self.xpos(), self.ypos(),
              self.zpos()])
1405 gcpy          self.dxfarc(self.currenttoolnumber(), xcenter, ycenter,
                  radius, 0, 90)
1406 gcpy          if (self.zpos == ez):
1407 gcpy              self.settzpos(0)
1408 gcpy          else:
1409 gcpy              self.settzpos((self.zpos()-ez)/90)
1410 gcpy #        self.setxpos(ex)
1411 gcpy #        self.setypos(ey)
1412 gcpy #        self.setzpos(ez)
1413 gcpy #        if self.generatepaths == True:
1414 gcpy #            print("Unioning cutarcNECCdxf toolpath")
1415 gcpy          self.arcloop(1, 90, xcenter, ycenter, radius)
1416 gcpy #            self.toolpaths = self.toolpaths.union(toolpath)
1417 gcpy #        else:
```

```
1418 gcpy #              toolpath = self.arcloop(1, 90, xcenter, ycenter,
          radius)
1419 gcpy #              print("Returning cutarcNECCdxf toolpath")
1420 gcpy         return toolpath
1421 gcpy
1422 gcpy     def cutarcNWCCdxf(self, ex, ey, ez, xcenter, ycenter, radius):
1423 gcpy #         global toolpath
1424 gcpy #         toolpath = self.currenttool()
1425 gcpy #         toolpath = toolpath.translate([self.xpos(), self.ypos(),
          self.zpos()])
1426 gcpy         self.dxfarc(self.currenttoolnumber(), xcenter, ycenter,
                  radius, 90, 180)
1427 gcpy         if (self.zpos == ez):
1428 gcpy             self.settzpos(0)
1429 gcpy         else:
1430 gcpy             self.settzpos((self.zpos()-ez)/90)
1431 gcpy #         self.setxpos(ex)
1432 gcpy #         self.setypos(ey)
1433 gcpy #         self.setzpos(ez)
1434 gcpy #         if self.generatepaths == True:
1435 gcpy #             self.arcloop(91, 180, xcenter, ycenter, radius)
1436 gcpy #             self.toolpaths = self.toolpaths.union(toolpath)
1437 gcpy #         else:
1438 gcpy         toolpath = self.arcloop(91, 180, xcenter, ycenter, radius)
1439 gcpy         return toolpath
1440 gcpy
1441 gcpy     def cutarcSWCCdxf(self, ex, ey, ez, xcenter, ycenter, radius):
1442 gcpy #         global toolpath
1443 gcpy #         toolpath = self.currenttool()
1444 gcpy #         toolpath = toolpath.translate([self.xpos(), self.ypos(),
          self.zpos()])
1445 gcpy         self.dxfarc(self.currenttoolnumber(), xcenter, ycenter,
                  radius, 180, 270)
1446 gcpy         if (self.zpos == ez):
1447 gcpy             self.settzpos(0)
1448 gcpy         else:
1449 gcpy             self.settzpos((self.zpos()-ez)/90)
1450 gcpy #         self.setxpos(ex)
1451 gcpy #         self.setypos(ey)
1452 gcpy #         self.setzpos(ez)
1453 gcpy         if self.generatepaths == True:
1454 gcpy             self.arcloop(181, 270, xcenter, ycenter, radius)
1455 gcpy #             self.toolpaths = self.toolpaths.union(toolpath)
1456 gcpy         else:
1457 gcpy             toolpath = self.arcloop(181, 270, xcenter, ycenter,
                      radius)
1458 gcpy             return toolpath
1459 gcpy
1460 gcpy     def cutarcSECCdxf(self, ex, ey, ez, xcenter, ycenter, radius):
1461 gcpy #         global toolpath
1462 gcpy #         toolpath = self.currenttool()
1463 gcpy #         toolpath = toolpath.translate([self.xpos(), self.ypos(),
          self.zpos()])
1464 gcpy         self.dxfarc(self.currenttoolnumber(), xcenter, ycenter,
                  radius, 270, 360)
1465 gcpy         if (self.zpos == ez):
1466 gcpy             self.settzpos(0)
1467 gcpy         else:
1468 gcpy             self.settzpos((self.zpos()-ez)/90)
1469 gcpy #         self.setxpos(ex)
1470 gcpy #         self.setypos(ey)
1471 gcpy #         self.setzpos(ez)
1472 gcpy         if self.generatepaths == True:
1473 gcpy             self.arcloop(271, 360, xcenter, ycenter, radius)
1474 gcpy #             self.toolpaths = self.toolpaths.union(toolpath)
1475 gcpy         else:
1476 gcpy             toolpath = self.arcloop(271, 360, xcenter, ycenter,
                      radius)
1477 gcpy             return toolpath
1478 gcpy
1479 gcpy     def cutarcNECWdxf(self, ex, ey, ez, xcenter, ycenter, radius):
1480 gcpy #         global toolpath
1481 gcpy #         toolpath = self.currenttool()
1482 gcpy #         toolpath = toolpath.translate([self.xpos(), self.ypos(),
          self.zpos()])
1483 gcpy         self.dxfarc(self.currenttoolnumber(), xcenter, ycenter,
                  radius, 0, 90)
1484 gcpy         if (self.zpos == ez):
```

```
1485 gcpy                    self.settzpos(0)
1486 gcpy            else:
1487 gcpy                    self.settzpos((self.zpos()-ez)/90)
1488 gcpy #         self.setxpos(ex)
1489 gcpy #         self.setypos(ey)
1490 gcpy #         self.setzpos(ez)
1491 gcpy            if self.generatepaths == True:
1492 gcpy                    self.narcloop(89, 0, xcenter, ycenter, radius)
1493 gcpy #              self.toolpaths = self.toolpaths.union(toolpath)
1494 gcpy            else:
1495 gcpy                    toolpath = self.narcloop(89, 0, xcenter, ycenter,
                                radius)
1496 gcpy                    return toolpath
1497 gcpy
1498 gcpy        def cutarcSECWdxf(self, ex, ey, ez, xcenter, ycenter, radius):
1499 gcpy #          global toolpath
1500 gcpy #          toolpath = self.currenttool()
1501 gcpy #          toolpath = toolpath.translate([self.xpos(), self.ypos(),
            self.zpos()])
1502 gcpy            self.dxfarc(self.currenttoolnumber(), xcenter, ycenter,
                        radius, 270, 360)
1503 gcpy            if (self.zpos == ez):
1504 gcpy                    self.settzpos(0)
1505 gcpy            else:
1506 gcpy                    self.settzpos((self.zpos()-ez)/90)
1507 gcpy #         self.setxpos(ex)
1508 gcpy #         self.setypos(ey)
1509 gcpy #         self.setzpos(ez)
1510 gcpy            if self.generatepaths == True:
1511 gcpy                    self.narcloop(359, 270, xcenter, ycenter, radius)
1512 gcpy #              self.toolpaths = self.toolpaths.union(toolpath)
1513 gcpy            else:
1514 gcpy                    toolpath = self.narcloop(359, 270, xcenter, ycenter,
                                radius)
1515 gcpy                    return toolpath
1516 gcpy
1517 gcpy        def cutarcSWCWdxf(self, ex, ey, ez, xcenter, ycenter, radius):
1518 gcpy #          global toolpath
1519 gcpy #          toolpath = self.currenttool()
1520 gcpy #          toolpath = toolpath.translate([self.xpos(), self.ypos(),
            self.zpos()])
1521 gcpy            self.dxfarc(self.currenttoolnumber(), xcenter, ycenter,
                        radius, 180, 270)
1522 gcpy            if (self.zpos == ez):
1523 gcpy                    self.settzpos(0)
1524 gcpy            else:
1525 gcpy                    self.settzpos((self.zpos()-ez)/90)
1526 gcpy #         self.setxpos(ex)
1527 gcpy #         self.setypos(ey)
1528 gcpy #         self.setzpos(ez)
1529 gcpy            if self.generatepaths == True:
1530 gcpy                    self.narcloop(269, 180, xcenter, ycenter, radius)
1531 gcpy #              self.toolpaths = self.toolpaths.union(toolpath)
1532 gcpy            else:
1533 gcpy                    toolpath = self.narcloop(269, 180, xcenter, ycenter,
                                radius)
1534 gcpy                    return toolpath
1535 gcpy
1536 gcpy        def cutarcNWCWdxf(self, ex, ey, ez, xcenter, ycenter, radius):
1537 gcpy #          global toolpath
1538 gcpy #          toolpath = self.currenttool()
1539 gcpy #          toolpath = toolpath.translate([self.xpos(), self.ypos(),
            self.zpos()])
1540 gcpy            self.dxfarc(self.currenttoolnumber(), xcenter, ycenter,
                        radius, 90, 180)
1541 gcpy            if (self.zpos == ez):
1542 gcpy                    self.settzpos(0)
1543 gcpy            else:
1544 gcpy                    self.settzpos((self.zpos()-ez)/90)
1545 gcpy #         self.setxpos(ex)
1546 gcpy #         self.setypos(ey)
1547 gcpy #         self.setzpos(ez)
1548 gcpy            if self.generatepaths == True:
1549 gcpy                    self.narcloop(179, 90, xcenter, ycenter, radius)
1550 gcpy #              self.toolpaths = self.toolpaths.union(toolpath)
1551 gcpy            else:
1552 gcpy                    toolpath = self.narcloop(179, 90, xcenter, ycenter,
                                radius)
```

```
1553 gcpy            return toolpath
```

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

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

```
1555 gcpy        def arcCCgc(self, ex, ey, ez, xcenter, ycenter, radius):
1556 gcpy            self.writegc("G03␣X", str(ex), "␣Y", str(ey), "␣Z", str(ez)
                        , "␣R", str(radius))
1557 gcpy
1558 gcpy        def arcCWgc(self, ex, ey, ez, xcenter, ycenter, radius):
1559 gcpy            self.writegc("G02␣X", str(ex), "␣Y", str(ey), "␣Z", str(ez)
                        , "␣R", str(radius))
```

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

```
1561 gcpy        def cutarcNECCdxfgc(self, ex, ey, ez, xcenter, ycenter, radius)
                        :
1562 gcpy            self.arcCCgc(ex, ey, ez, xcenter, ycenter, radius)
1563 gcpy            if self.generatepaths == True:
1564 gcpy                self.cutarcNECCdxf(ex, ey, ez, xcenter, ycenter, radius
                            )
1565 gcpy            else:
1566 gcpy                return self.cutarcNECCdxf(ex, ey, ez, xcenter, ycenter,
                            radius)
1567 gcpy
1568 gcpy        def cutarcNWCCdxfgc(self, ex, ey, ez, xcenter, ycenter, radius)
                        :
1569 gcpy            self.arcCCgc(ex, ey, ez, xcenter, ycenter, radius)
1570 gcpy            if self.generatepaths == False:
1571 gcpy                return self.cutarcNWCCdxf(ex, ey, ez, xcenter, ycenter,
                            radius)
1572 gcpy
1573 gcpy        def cutarcSWCCdxfgc(self, ex, ey, ez, xcenter, ycenter, radius)
                        :
1574 gcpy            self.arcCCgc(ex, ey, ez, xcenter, ycenter, radius)
1575 gcpy            if self.generatepaths == False:
1576 gcpy                return self.cutarcSWCCdxf(ex, ey, ez, xcenter, ycenter,
                            radius)
1577 gcpy
1578 gcpy        def cutarcSECCdxfgc(self, ex, ey, ez, xcenter, ycenter, radius)
                        :
1579 gcpy            self.arcCCgc(ex, ey, ez, xcenter, ycenter, radius)
1580 gcpy            if self.generatepaths == False:
1581 gcpy                return self.cutarcSECCdxf(ex, ey, ez, xcenter, ycenter,
                            radius)
1582 gcpy
1583 gcpy        def cutarcNECWdxfgc(self, ex, ey, ez, xcenter, ycenter, radius)
                        :
1584 gcpy            self.arcCWgc(ex, ey, ez, xcenter, ycenter, radius)
1585 gcpy            if self.generatepaths == False:
1586 gcpy                return self.cutarcNECWdxf(ex, ey, ez, xcenter, ycenter,
                            radius)
1587 gcpy
1588 gcpy        def cutarcSECWdxfgc(self, ex, ey, ez, xcenter, ycenter, radius)
                        :
1589 gcpy            self.arcCWgc(ex, ey, ez, xcenter, ycenter, radius)
1590 gcpy            if self.generatepaths == False:
1591 gcpy                return self.cutarcSECWdxf(ex, ey, ez, xcenter, ycenter,
                            radius)
1592 gcpy
1593 gcpy        def cutarcSWCWdxfgc(self, ex, ey, ez, xcenter, ycenter, radius)
                        :
1594 gcpy            self.arcCWgc(ex, ey, ez, xcenter, ycenter, radius)
1595 gcpy            if self.generatepaths == False:
1596 gcpy                return self.cutarcSWCWdxf(ex, ey, ez, xcenter, ycenter,
                            radius)
1597 gcpy
1598 gcpy        def cutarcNWCWdxfgc(self, ex, ey, ez, xcenter, ycenter, radius)
                        :
1599 gcpy            self.arcCWgc(ex, ey, ez, xcenter, ycenter, radius)
1600 gcpy            if self.generatepaths == False:
1601 gcpy                return self.cutarcNWCWdxf(ex, ey, ez, xcenter, ycenter,
                            radius)
```

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

3.6.3 G-code Overview

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

Command/Module	G-code
`opengcodefile(s)(...); setupstock(...)`	`(export.nc)`
	`(stockMin: -109.5, -75mm, -8.35mm)`
	`(stockMax:109.5mm, 75mm, 0.00mm)`
	`(STOCK/BLOCK, 219, 150, 8.35, 109.5, 75, 8.35)`
	`G90`
	`G21`
`movetosafez()`	`(Move to safe Z to avoid workholding)`
	`G53G0Z-5.000`
`toolchange(...);`	`(TOOL/MILL, 3.17, 0.00, 0.00, 0.00)`
	`M6T102`
	`M03S16000`
`cutoneaxis_setfeed(...);`	`(PREPOSITION FOR RAPID PLUNGE)`
	`G0X0Y0`
	`Z0.25`
	`G1Z0F100`
	`G1 X109.5 Y75 Z-8.35F400`
	`Z9`
`cutwithfeed(...);`	
`closegcodefile();`	`M05`
	`M02`

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

G-code	Command/Module
(Design File:) (stockMin:0.00mm, -152.40mm, -34.92mm) (stockMax:109.50mm, -77.40mm, 0.00mm) (STOCK/BLOCK, 109.50, 75.00, 34.92, 0.00, 152.40, 34.92) G90 G21	opengcodefile(s)(...); setupstock(.
(Move to safe Z to avoid workholding) G53G0Z-5.000	movetosafez()
(Toolpath: Contour Toolpath 1) M05 (TOOL/MILL, 3.17, 0.00, 0.00, 0.00) M6T102 M03S10000	toolchange(...);
(PREPOSITION FOR RAPID PLUNGE)	writecomment(...)
G0X0.000Y-152.400 Z0.250	rapid(...) rapid(...)
G1Z-1.000F203.2 X109.500Y-77.400F508.0 X57.918Y16.302Z-0.726 Y22.023Z-1.023 X61.190Z-0.681 Y21.643 X57.681 Z12.700	cutwithfeed(...); cutwithfeed(...);
M05 M02	closegcodefile();

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

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

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

- onOpen

- onClose

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

3.6.3.1 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.

```
1603 gcpy     def dxfpostamble(self, tn):
1604 gcpy #        self.writedxf(tn, str(tn))
1605 gcpy         self.writedxf(tn, "0")
1606 gcpy         self.writedxf(tn, "ENDSEC")
1607 gcpy         self.writedxf(tn, "0")
1608 gcpy         self.writedxf(tn, "EOF")
```

```
1610 gcpy     def gcodepostamble(self):
1611 gcpy         self.writegc("Z12.700")
1612 gcpy         self.writegc("M05")
1613 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.

```
1615 gcpy     def closegcodefile(self):
1616 gcpy         if self.generategcode == True:
1617 gcpy             self.gcodepostamble()
1618 gcpy             self.gc.close()
1619 gcpy
```

```
1620 gcpy      def closedxffile(self):
1621 gcpy          if self.generatedxf == True:
1622 gcpy #            global dxfclosed
1623 gcpy              self.dxfpostamble(-1)
1624 gcpy #            self.dxfclosed = True
1625 gcpy              self.dxf.close()
1626 gcpy
1627 gcpy      def closedxffiles(self):
1628 gcpy          if self.generatedxfs == True:
1629 gcpy              if (self.large_square_tool_num > 0):
1630 gcpy                  self.dxfpostamble(self.large_square_tool_num)
1631 gcpy              if (self.small_square_tool_num > 0):
1632 gcpy                  self.dxfpostamble(self.small_square_tool_num)
1633 gcpy              if (self.large_ball_tool_num > 0):
1634 gcpy                  self.dxfpostamble(self.large_ball_tool_num)
1635 gcpy              if (self.small_ball_tool_num > 0):
1636 gcpy                  self.dxfpostamble(self.small_ball_tool_num)
1637 gcpy              if (self.large_V_tool_num > 0):
1638 gcpy                  self.dxfpostamble(self.large_V_tool_num)
1639 gcpy              if (self.small_V_tool_num > 0):
1640 gcpy                  self.dxfpostamble(self.small_V_tool_num)
1641 gcpy              if (self.DT_tool_num > 0):
1642 gcpy                  self.dxfpostamble(self.DT_tool_num)
1643 gcpy              if (self.KH_tool_num > 0):
1644 gcpy                  self.dxfpostamble(self.KH_tool_num)
1645 gcpy              if (self.Roundover_tool_num > 0):
1646 gcpy                  self.dxfpostamble(self.Roundover_tool_num)
1647 gcpy              if (self.MISC_tool_num > 0):
1648 gcpy                  self.dxfpostamble(self.MISC_tool_num)
1649 gcpy
1650 gcpy              if (self.large_square_tool_num > 0):
1651 gcpy                  self.dxflgsq.close()
1652 gcpy              if (self.small_square_tool_num > 0):
1653 gcpy                  self.dxfsmsq.close()
1654 gcpy              if (self.large_ball_tool_num > 0):
1655 gcpy                  self.dxflgbl.close()
1656 gcpy              if (self.small_ball_tool_num > 0):
1657 gcpy                  self.dxfsmbl.close()
1658 gcpy              if (self.large_V_tool_num > 0):
1659 gcpy                  self.dxflgV.close()
1660 gcpy              if (self.small_V_tool_num > 0):
1661 gcpy                  self.dxfsmV.close()
1662 gcpy              if (self.DT_tool_num > 0):
1663 gcpy                  self.dxfDT.close()
1664 gcpy              if (self.KH_tool_num > 0):
1665 gcpy                  self.dxfKH.close()
1666 gcpy              if (self.Roundover_tool_num > 0):
1667 gcpy                  self.dxfRt.close()
1668 gcpy              if (self.MISC_tool_num > 0):
1669 gcpy                  self.dxfMt.close()
```

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

```
162 gcpscad module closegcodefile(){
163 gcpscad     gcp.closegcodefile();
164 gcpscad }
165 gcpscad
166 gcpscad module closedxffiles(){
167 gcpscad     gcp.closedxffiles();
168 gcpscad }
169 gcpscad
170 gcpscad module closedxffile(){
171 gcpscad     gcp.closedxffile();
172 gcpscad }
```

3.7 Cutting shapes and expansion

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

It is most expedient to test out new features in a new/separate file insofar as the file structures
will allow (tool definitions for example will need to consolidated in 3.3.1.1) which will need to be
included in the projects which will make use of said features until such time as they are added
into the main `gcodepreview.scad` file.

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

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

- lines — `dxfline`

- arcs — `dxfarc`

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

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

- 0
    - circle — `dxfcircle`
    - ellipse (oval) (requires some sort of non-arc curve)
        - egg-shaped
    - annulus (one circle within another, forming a ring) — handled by nested circles
    - superellipse (see astroid below)

- 1
    - cone with rounded end (arc)—see also "sector" under 3 below

- 2
    - semicircle/circular/half-circle segment (arc and a straight line); see also sector below
    - arch—curve possibly smoothly joining a pair of straight lines with a flat bottom
    - lens/vesica piscis (two convex curves)
    - lune/crescent (one convex, one concave curve)
    - heart (two curves)
    - tomoe (comma shape)—non-arc curves

- 3
    - triangle
        - equilateral
        - isosceles
        - right triangle
        - scalene
    - (circular) sector (two straight edges, one convex arc)
        - quadrant (90°)
        - sextants (60°)
        - octants (45°)
    - deltoid curve (three concave arcs)
    - Reuleaux triangle (three convex arcs)
    - arbelos (one convex, two concave arcs)
    - two straight edges, one concave arc—an example is the hyperbolic sector[1]
    - two convex, one concave arc

- 4
    - rectangle (including square) — `dxfrectangle, dxfrectangleround`
    - parallelogram
    - rhombus
    - trapezoid/trapezium
    - kite
    - ring/annulus segment (straight line, concave arc, straight line, convex arc)

---

[1] `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`

– 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 re-placed 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 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 informa-tion 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?

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

```
1671 gcpy      def dxfcircle(self, tool_num, xcenter, ycenter, radius):
1672 gcpy          self.dxfarc(tool_num, xcenter, ycenter, radius,   0,  90)
1673 gcpy          self.dxfarc(tool_num, xcenter, ycenter, radius,  90, 180)
1674 gcpy          self.dxfarc(tool_num, xcenter, ycenter, radius, 180, 270)
1675 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.

```
1677 gcpy      def cutcircleCC(self, xcenter, ycenter, bz, ez, radius):
1678 gcpy          self.setzpos(bz)
1679 gcpy          self.cutquarterCCNE(xcenter, ycenter + radius, self.zpos()
                       + ez/4, radius)
```

```
1680 gcpy              self.cutquarterCCNW(xcenter - radius, ycenter, self.zpos()
                           + ez/4, radius)
1681 gcpy              self.cutquarterCCSW(xcenter, ycenter - radius, self.zpos()
                           + ez/4, radius)
1682 gcpy              self.cutquarterCCSE(xcenter + radius, ycenter, self.zpos()
                           + ez/4, radius)
1683 gcpy
1684 gcpy         def cutcircleCCdxf(self, xcenter, ycenter, bz, ez, radius):
1685 gcpy              self.cutcircleCC(self, xcenter, ycenter, bz, ez, radius)
1686 gcpy              self.dxfcircle(self, tool_num, xcenter, ycenter, radius)
```

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

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

```
1688 gcpy         def dxfrectangle(self, tool_num, xorigin, yorigin, xwidth,
                       yheight, corners = "Square", radius = 6):
1689 gcpy              if corners == "Square":
1690 gcpy                  self.dxfline(tool_num, xorigin, yorigin, xorigin +
                               xwidth, yorigin)
1691 gcpy                  self.dxfline(tool_num, xorigin + xwidth, yorigin,
                               xorigin + xwidth, yorigin + yheight)
1692 gcpy                  self.dxfline(tool_num, xorigin + xwidth, yorigin +
                               yheight, xorigin, yorigin + yheight)
1693 gcpy                  self.dxfline(tool_num, xorigin, yorigin + yheight,
                               xorigin, yorigin)
1694 gcpy              elif corners == "Fillet":
1695 gcpy                  self.dxfrectangleround(tool_num, xorigin, yorigin,
                               xwidth, yheight, radius)
1696 gcpy              elif corners == "Chamfer":
1697 gcpy                  self.dxfrectanglechamfer(tool_num, xorigin, yorigin,
                               xwidth, yheight, radius)
1698 gcpy              elif corners == "Flipped␣Fillet":
1699 gcpy                  self.dxfrectangleflippedfillet(tool_num, xorigin,
                               yorigin, xwidth, yheight, radius)
```

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

```
1701 gcpy         def dxfrectangleround(self, tool_num, xorigin, yorigin, xwidth,
                       yheight, radius):
1702 gcpy # begin section
1703 gcpy              self.writedxf(tool_num, "0")
1704 gcpy              self.writedxf(tool_num, "SECTION")
1705 gcpy              self.writedxf(tool_num, "2")
1706 gcpy              self.writedxf(tool_num, "ENTITIES")
1707 gcpy              self.writedxf(tool_num, "0")
1708 gcpy              self.writedxf(tool_num, "LWPOLYLINE")
1709 gcpy              self.writedxf(tool_num, "5")
1710 gcpy              self.writedxf(tool_num, "4E")
1711 gcpy              self.writedxf(tool_num, "100")
1712 gcpy              self.writedxf(tool_num, "AcDbEntity")
1713 gcpy              self.writedxf(tool_num, "8")
1714 gcpy              self.writedxf(tool_num, "0")
1715 gcpy              self.writedxf(tool_num, "6")
1716 gcpy              self.writedxf(tool_num, "ByLayer")
1717 gcpy #
1718 gcpy              self.writedxfcolor(tool_num)
1719 gcpy #
1720 gcpy              self.writedxf(tool_num, "370")
1721 gcpy              self.writedxf(tool_num, "-1")
1722 gcpy              self.writedxf(tool_num, "100")
1723 gcpy              self.writedxf(tool_num, "AcDbPolyline")
1724 gcpy              self.writedxf(tool_num, "90")
1725 gcpy              self.writedxf(tool_num, "8")
1726 gcpy              self.writedxf(tool_num, "70")
1727 gcpy              self.writedxf(tool_num, "1")
1728 gcpy              self.writedxf(tool_num, "43")
1729 gcpy              self.writedxf(tool_num, "0")
1730 gcpy #1 upper right corner before arc (counter-clockwise)
1731 gcpy              self.writedxf(tool_num, "10")
1732 gcpy              self.writedxf(tool_num, str(xorigin + xwidth))
1733 gcpy              self.writedxf(tool_num, "20")
1734 gcpy              self.writedxf(tool_num, str(yorigin + yheight - radius))
1735 gcpy              self.writedxf(tool_num, "42")
1736 gcpy              self.writedxf(tool_num, "0.414213562373095")
```

```
1737 gcpy  #2 upper right corner after arc
1738 gcpy          self.writedxf(tool_num, "10")
1739 gcpy          self.writedxf(tool_num, str(xorigin + xwidth - radius))
1740 gcpy          self.writedxf(tool_num, "20")
1741 gcpy          self.writedxf(tool_num, str(yorigin + yheight))
1742 gcpy  #3 upper left corner before arc (counter-clockwise)
1743 gcpy          self.writedxf(tool_num, "10")
1744 gcpy          self.writedxf(tool_num, str(xorigin + radius))
1745 gcpy          self.writedxf(tool_num, "20")
1746 gcpy          self.writedxf(tool_num, str(yorigin + yheight))
1747 gcpy          self.writedxf(tool_num, "42")
1748 gcpy          self.writedxf(tool_num, "0.414213562373095")
1749 gcpy  #4 upper left corner after arc
1750 gcpy          self.writedxf(tool_num, "10")
1751 gcpy          self.writedxf(tool_num, str(xorigin))
1752 gcpy          self.writedxf(tool_num, "20")
1753 gcpy          self.writedxf(tool_num, str(yorigin + yheight - radius))
1754 gcpy  #5 lower left corner before arc (counter-clockwise)
1755 gcpy          self.writedxf(tool_num, "10")
1756 gcpy          self.writedxf(tool_num, str(xorigin))
1757 gcpy          self.writedxf(tool_num, "20")
1758 gcpy          self.writedxf(tool_num, str(yorigin + radius))
1759 gcpy          self.writedxf(tool_num, "42")
1760 gcpy          self.writedxf(tool_num, "0.414213562373095")
1761 gcpy  #6 lower left corner after arc
1762 gcpy          self.writedxf(tool_num, "10")
1763 gcpy          self.writedxf(tool_num, str(xorigin + radius))
1764 gcpy          self.writedxf(tool_num, "20")
1765 gcpy          self.writedxf(tool_num, str(yorigin))
1766 gcpy  #7 lower right corner before arc (counter-clockwise)
1767 gcpy          self.writedxf(tool_num, "10")
1768 gcpy          self.writedxf(tool_num, str(xorigin + xwidth - radius))
1769 gcpy          self.writedxf(tool_num, "20")
1770 gcpy          self.writedxf(tool_num, str(yorigin))
1771 gcpy          self.writedxf(tool_num, "42")
1772 gcpy          self.writedxf(tool_num, "0.414213562373095")
1773 gcpy  #8 lower right corner after arc
1774 gcpy          self.writedxf(tool_num, "10")
1775 gcpy          self.writedxf(tool_num, str(xorigin + xwidth))
1776 gcpy          self.writedxf(tool_num, "20")
1777 gcpy          self.writedxf(tool_num, str(yorigin + radius))
1778 gcpy  # end current section
1779 gcpy          self.writedxf(tool_num, "0")
1780 gcpy          self.writedxf(tool_num, "SEQEND")
```

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

```
1782 gcpy      def dxfrectanglechamfer(self, tool_num, xorigin, yorigin,
              xwidth, yheight, radius):
1783 gcpy          self.dxfline(tool_num, xorigin + radius, yorigin, xorigin,
                 yorigin + radius)
1784 gcpy          self.dxfline(tool_num, xorigin, yorigin + yheight - radius,
                 xorigin + radius, yorigin + yheight)
1785 gcpy          self.dxfline(tool_num, xorigin + xwidth - radius, yorigin +
                 yheight, xorigin + xwidth, yorigin + yheight - radius)
1786 gcpy          self.dxfline(tool_num, xorigin + xwidth - radius, yorigin,
                 xorigin + xwidth, yorigin + radius)
1787 gcpy
1788 gcpy          self.dxfline(tool_num, xorigin + radius, yorigin, xorigin +
                 xwidth - radius, yorigin)
1789 gcpy          self.dxfline(tool_num, xorigin + xwidth, yorigin + radius,
                 xorigin + xwidth, yorigin + yheight - radius)
1790 gcpy          self.dxfline(tool_num, xorigin + xwidth - radius, yorigin +
                 yheight, xorigin + radius, yorigin + yheight)
1791 gcpy          self.dxfline(tool_num, xorigin, yorigin + yheight - radius,
                 xorigin, yorigin + radius)
```

Flipped Fillet:

```
1793 gcpy      def dxfrectangleflippedfillet(self, tool_num, xorigin, yorigin,
              xwidth, yheight, radius):
1794 gcpy          self.dxfarc(tool_num, xorigin, yorigin, radius,  0, 90)
1795 gcpy          self.dxfarc(tool_num, xorigin + xwidth, yorigin, radius,
                 90, 180)
1796 gcpy          self.dxfarc(tool_num, xorigin + xwidth, yorigin + yheight,
                 radius, 180, 270)
```

```
1797 gcpy            self.dxfarc(tool_num, xorigin, yorigin + yheight, radius,
                         270, 360)
1798 gcpy
1799 gcpy            self.dxfline(tool_num, xorigin + radius, yorigin, xorigin +
                         xwidth - radius, yorigin)
1800 gcpy            self.dxfline(tool_num, xorigin + xwidth, yorigin + radius,
                         xorigin + xwidth, yorigin + yheight - radius)
1801 gcpy            self.dxfline(tool_num, xorigin + xwidth - radius, yorigin +
                         yheight, xorigin + radius, yorigin + yheight)
1802 gcpy            self.dxfline(tool_num, xorigin, yorigin + yheight - radius,
                         xorigin, yorigin + radius)
```

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

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

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

```
1804 gcpy        def cutrectangle(self, tool_num, bx, by, bz, xwidth, yheight,
                     zdepth):
1805 gcpy            self.cutline(bx, by, bz)
1806 gcpy            self.cutline(bx, by, bz - zdepth)
1807 gcpy            self.cutline(bx + xwidth, by, bz - zdepth)
1808 gcpy            self.cutline(bx + xwidth, by + yheight, bz - zdepth)
1809 gcpy            self.cutline(bx, by + yheight, bz - zdepth)
1810 gcpy            self.cutline(bx, by, bz - zdepth)
1811 gcpy
1812 gcpy        def cutrectangledxf(self, tool_num, bx, by, bz, xwidth, yheight
                     , zdepth):
1813 gcpy            self.cutrectangle(tool_num, bx, by, bz, xwidth, yheight,
                         zdepth)
1814 gcpy            self.dxfrectangle(tool_num, bx, by, xwidth, yheight, "
                         Square")
```

The rounded forms instantiate a radius:

```
1816 gcpy        def cutrectangleround(self, tool_num, bx, by, bz, xwidth,
                     yheight, zdepth, radius):
1817 gcpy #          self.rapid(bx + radius, by, bz)
1818 gcpy            self.cutline(bx + radius, by, bz + zdepth)
1819 gcpy            self.cutline(bx + xwidth - radius, by, bz + zdepth)
1820 gcpy            self.cutquarterCCSE(bx + xwidth, by + radius, bz + zdepth,
                         radius)
1821 gcpy            self.cutline(bx + xwidth, by + yheight - radius, bz +
                         zdepth)
1822 gcpy            self.cutquarterCCNE(bx + xwidth - radius, by + yheight, bz
                         + zdepth, radius)
1823 gcpy            self.cutline(bx + radius, by + yheight, bz + zdepth)
1824 gcpy            self.cutquarterCCNW(bx, by + yheight - radius, bz + zdepth,
                         radius)
1825 gcpy            self.cutline(bx, by + radius, bz + zdepth)
1826 gcpy            self.cutquarterCCSW(bx + radius, by, bz + zdepth, radius)
1827 gcpy
1828 gcpy        def cutrectanglerounddxf(self, tool_num, bx, by, bz, xwidth,
                     yheight, zdepth, radius):
1829 gcpy            self.cutrectangleround(tool_num, bx, by, bz, xwidth,
                         yheight, zdepth, radius)
1830 gcpy            self.dxfrectangleround(tool_num, bx, by, xwidth, yheight,
                         radius)
```

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

Tooling for such toolpaths is defined at paragraph 3.4.0.1

The interface which is being modeled is that of Carbide Create:
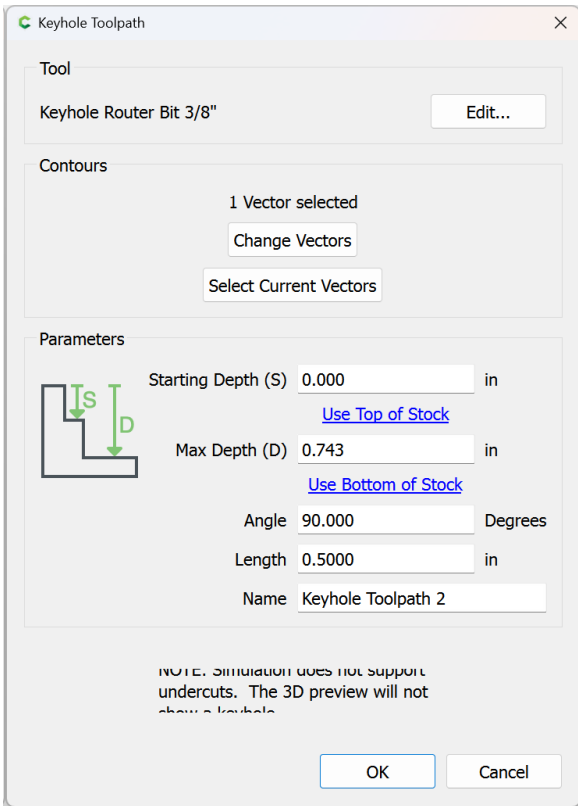
Hence the parameters:

- Starting Depth == `kh_start_depth`

- Max Depth == `kh_max_depth`

- Angle == `kht_direction`

- Length == `kh_distance`

- Tool == `kh_tool_num`

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

- one node on the clockwise side is outside of the quadrant

- two nodes on the clockwise side are outside of the quadrant

- all nodes are w/in the quadrant

- one node on the counter-clockwise side is outside of the quadrant

- two nodes on the counter-clockwise side are outside of the quadrant

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

```
1832 gcpy      def cutkeyholegcdxf(self, kh_tool_num, kh_start_depth,
                   kh_max_depth, kht_direction, kh_distance):
1833 gcpy          if (kht_direction == "N"):
1834 gcpy              toolpath = self.cutKHgcdxf(kh_tool_num, kh_start_depth,
                           kh_max_depth, 90, kh_distance)
1835 gcpy          elif (kht_direction == "S"):
1836 gcpy              toolpath = self.cutKHgcdxf(kh_tool_num, kh_start_depth,
                           kh_max_depth, 270, kh_distance)
1837 gcpy          elif (kht_direction == "E"):
1838 gcpy              toolpath = self.cutKHgcdxf(kh_tool_num, kh_start_depth,
                           kh_max_depth, 0, kh_distance)
1839 gcpy          elif (kht_direction == "W"):
1840 gcpy              toolpath = self.cutKHgcdxf(kh_tool_num, kh_start_depth,
                           kh_max_depth, 180, kh_distance)
1841 gcpy #        if self.generatepaths == True:
1842 gcpy #            self.toolpaths = union([self.toolpaths, toolpath])
1843 gcpy          return toolpath
1844 gcpy #        else:
1845 gcpy #            return cube([0.01, 0.01, 0.01])
```

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

cutKHgcdxf   The original version of the command, `cutKHgcdxf` retains an interface which allows calling it for arbitrary beginning and ending points of an arc.

Note that code is still present for the partial calculation of one quadrant (for the case of all nodes within the quadrant). The first task is to place a circle at the origin which is invariant of angle:

```
1847 gcpy      def cutKHgcdxf(self, kh_tool_num, kh_start_depth, kh_max_depth,
                   kh_angle, kh_distance):
1848 gcpy          oXpos = self.xpos()
1849 gcpy          oYpos = self.ypos()
1850 gcpy          self.dxfKH(kh_tool_num, self.xpos(), self.ypos(),
                       kh_start_depth, kh_max_depth, kh_angle, kh_distance)
1851 gcpy          toolpath = self.cutline(self.xpos(), self.ypos(), -
                       kh_max_depth)
1852 gcpy          self.setxpos(oXpos)
1853 gcpy          self.setypos(oYpos)
1854 gcpy #          if self.generatepaths == False:
1855 gcpy          return toolpath
1856 gcpy #          else:
1857 gcpy #              return cube([0.001, 0.001, 0.001])
```

```
1859 gcpy      def dxfKH(self, kh_tool_num, oXpos, oYpos, kh_start_depth,
                   kh_max_depth, kh_angle, kh_distance):
1860 gcpy #          oXpos = self.xpos()
1861 gcpy #          oYpos = self.ypos()
1862 gcpy #Circle at entry hole
1863 gcpy          self.dxfarc(kh_tool_num, oXpos, oYpos, self.tool_radius(
                       kh_tool_num, 7), 0, 90)
1864 gcpy          self.dxfarc(kh_tool_num, oXpos, oYpos, self.tool_radius(
                       kh_tool_num, 7), 90, 180)
1865 gcpy          self.dxfarc(kh_tool_num, oXpos, oYpos, self.tool_radius(
                       kh_tool_num, 7), 180, 270)
1866 gcpy          self.dxfarc(kh_tool_num, oXpos, oYpos, self.tool_radius(
                       kh_tool_num, 7), 270, 360)
```

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

```
1867 gcpy #pre-calculate needed values
1868 gcpy          r = self.tool_radius(kh_tool_num, 7)
1869 gcpy #          print(r)
1870 gcpy          rt = self.tool_radius(kh_tool_num, 1)
1871 gcpy #          print(rt)
1872 gcpy          ro = math.sqrt((self.tool_radius(kh_tool_num, 1))**2-(self.
                       tool_radius(kh_tool_num, 7))**2)
1873 gcpy #          print(ro)
1874 gcpy          angle = math.degrees(math.acos(ro/rt))
1875 gcpy #Outlines of entry hole and slot
1876 gcpy          if (kh_angle == 0):
1877 gcpy #Lower left of entry hole
1878 gcpy              self.dxfarc(kh_tool_num, self.xpos(), self.ypos(), self
                           .tool_radius(kh_tool_num, 1), 180, 270)
1879 gcpy #Upper left of entry hole
1880 gcpy              self.dxfarc(kh_tool_num, self.xpos(), self.ypos(), self
                           .tool_radius(kh_tool_num, 1), 90, 180)
1881 gcpy #Upper right of entry hole
1882 gcpy #              self.dxfarc(kh_tool_num, self.xpos(), self.ypos(), rt,
                   41.810, 90)
1883 gcpy              self.dxfarc(kh_tool_num, self.xpos(), self.ypos(), rt,
                           angle, 90)
1884 gcpy #Lower right of entry hole
1885 gcpy              self.dxfarc(kh_tool_num, self.xpos(), self.ypos(), rt,
                           270, 360-angle)
1886 gcpy #              self.dxfarc(kh_tool_num, self.xpos(), self.ypos(),
                   self.tool_radius(kh_tool_num, 1), 270, 270+math.acos(math.
                   radians(self.tool_diameter(kh_tool_num, 5)/self.tool_diameter(
                   kh_tool_num, 1))))
1887 gcpy #Actual line of cut
1888 gcpy #              self.dxfline(kh_tool_num, self.xpos(), self.ypos(),
                   self.xpos()+kh_distance, self.ypos())
```

```
1889 gcpy #upper right of end of slot (kh_max_depth+4.36))/2
1890 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)
1891 gcpy #lower right of end of slot
1892 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)
1893 gcpy #upper right slot
1894 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))
1895 gcpy #          self.dxfline(kh_tool_num, self.xpos()+(math.sqrt((self
          .tool_diameter(kh_tool_num, 1)^2)-(self.tool_diameter(
          kh_tool_num, 5)^2))/2), self.ypos()+self.tool_diameter(
          kh_tool_num, (kh_max_depth))/2, ( (kh_max_depth-6.34))/2)^2-(
          self.tool_diameter(kh_tool_num, (kh_max_depth-6.34))/2)^2, self.
          xpos()+kh_distance, self.ypos()+self.tool_diameter(kh_tool_num,
          (kh_max_depth))/2, kh_tool_num)
1896 gcpy #end position at top of slot
1897 gcpy #lower right slot
1898 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))
1899 gcpy #        dxfline(kh_tool_num, self.xpos()+(math.sqrt((self.
          tool_diameter(kh_tool_num, 1)^2)-(self.tool_diameter(kh_tool_num
          , 5)^2))/2), self.ypos()-self.tool_diameter(kh_tool_num, (
          kh_max_depth))/2, ( (kh_max_depth-6.34))/2)^2-(self.
          tool_diameter(kh_tool_num, (kh_max_depth-6.34))/2)^2, self.xpos
          ()+kh_distance, self.ypos()-self.tool_diameter(kh_tool_num, (
          kh_max_depth))/2, KH_tool_num)
1900 gcpy #end position at top of slot
1901 gcpy #    hull(){
1902 gcpy #       translate([xpos(), ypos(), zpos()]){
1903 gcpy #          keyhole_shaft(6.35, 9.525);
1904 gcpy #       }
1905 gcpy #       translate([xpos(), ypos(), zpos()-kh_max_depth]){
1906 gcpy #          keyhole_shaft(6.35, 9.525);
1907 gcpy #       }
1908 gcpy #    }
1909 gcpy #    hull(){
1910 gcpy #       translate([xpos(), ypos(), zpos()-kh_max_depth]){
1911 gcpy #          keyhole_shaft(6.35, 9.525);
1912 gcpy #       }
1913 gcpy #       translate([xpos()+kh_distance, ypos(), zpos()-kh_max_depth])
          {
1914 gcpy #          keyhole_shaft(6.35, 9.525);
1915 gcpy #       }
1916 gcpy #    }
1917 gcpy #    cutwithfeed(getxpos(), getypos(), -kh_max_depth, feed);
1918 gcpy #    cutwithfeed(getxpos()+kh_distance, getypos(), -kh_max_depth,
          feed);
1919 gcpy #    setxpos(getxpos()-kh_distance);
1920 gcpy #  } else if (kh_angle > 0 && kh_angle < 90) {
1921 gcpy #//echo(kh_angle);
1922 gcpy #  dxfarc(getxpos(), getypos(), tool_diameter(KH_tool_num, (
          kh_max_depth))/2, 90+kh_angle, 180+kh_angle, KH_tool_num);
1923 gcpy #  dxfarc(getxpos(), getypos(), tool_diameter(KH_tool_num, (
          kh_max_depth))/2, 180+kh_angle, 270+kh_angle, KH_tool_num);
1924 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);
1925 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);
1926 gcpy #dxfarc(getxpos()+(kh_distance*cos(kh_angle)),
1927 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);
1928 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);
1929 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))),
```

```
1930 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))),
1931 gcpy # getxpos()+(kh_distance*cos(kh_angle))-((tool_diameter(KH_tool_num
              , (kh_max_depth+4.36))/2)*sin(kh_angle)),
1932 gcpy # getypos()+(kh_distance*sin(kh_angle))+((tool_diameter(KH_tool_num
              , (kh_max_depth+4.36))/2)*cos(kh_angle)), KH_tool_num);
1933 gcpy #//echo("a", tool_diameter(KH_tool_num, (kh_max_depth+4.36))/2);
1934 gcpy #//echo("c", tool_diameter(KH_tool_num, (kh_max_depth))/2);
1935 gcpy #echo("Aangle", asin((tool_diameter(KH_tool_num, (kh_max_depth
              +4.36))/2)/(tool_diameter(KH_tool_num, (kh_max_depth))/2)));
1936 gcpy #//echo(kh_angle);
1937 gcpy # cutwithfeed(getxpos()+(kh_distance*cos(kh_angle)), getypos()+(
              kh_distance*sin(kh_angle)), -kh_max_depth, feed);
1938 gcpy #             toolpath = toolpath.union(self.cutline(self.xpos()+
              kh_distance, self.ypos(), -kh_max_depth))
1939 gcpy          elif (kh_angle == 90):
1940 gcpy #Lower left of entry hole
1941 gcpy              self.dxfarc(kh_tool_num, oXpos, oYpos, self.tool_radius
                      (kh_tool_num, 1), 180, 270)
1942 gcpy #Lower right of entry hole
1943 gcpy              self.dxfarc(kh_tool_num, oXpos, oYpos, self.tool_radius
                      (kh_tool_num, 1), 270, 360)
1944 gcpy #left slot
1945 gcpy              self.dxfline(kh_tool_num, oXpos-r, oYpos+ro, oXpos-r,
                      oYpos+kh_distance)
1946 gcpy #right slot
1947 gcpy              self.dxfline(kh_tool_num, oXpos+r, oYpos+ro, oXpos+r,
                      oYpos+kh_distance)
1948 gcpy #upper left of end of slot
1949 gcpy              self.dxfarc(kh_tool_num, oXpos, oYpos+kh_distance, r,
                      90, 180)
1950 gcpy #upper right of end of slot
1951 gcpy              self.dxfarc(kh_tool_num, oXpos, oYpos+kh_distance, r,
                      0, 90)
1952 gcpy #Upper right of entry hole
1953 gcpy              self.dxfarc(kh_tool_num, oXpos, oYpos, rt, 0, 90-angle)
1954 gcpy #Upper left of entry hole
1955 gcpy              self.dxfarc(kh_tool_num, oXpos, oYpos, rt, 90+angle,
                      180)
1956 gcpy #             toolpath = toolpath.union(self.cutline(oXpos, oYpos+
              kh_distance, -kh_max_depth))
1957 gcpy          elif (kh_angle == 180):
1958 gcpy #Lower right of entry hole
1959 gcpy              self.dxfarc(kh_tool_num, oXpos, oYpos, self.tool_radius
                      (kh_tool_num, 1), 270, 360)
1960 gcpy #Upper right of entry hole
1961 gcpy              self.dxfarc(kh_tool_num, oXpos, oYpos, self.tool_radius
                      (kh_tool_num, 1), 0, 90)
1962 gcpy #Upper left of entry hole
1963 gcpy              self.dxfarc(kh_tool_num, oXpos, oYpos, rt, 90, 180-
                      angle)
1964 gcpy #Lower left of entry hole
1965 gcpy              self.dxfarc(kh_tool_num, oXpos, oYpos, rt, 180+angle,
                      270)
1966 gcpy #upper slot
1967 gcpy              self.dxfline(kh_tool_num, oXpos-ro, oYpos-r, oXpos-
                      kh_distance, oYpos-r)
1968 gcpy #lower slot
1969 gcpy              self.dxfline(kh_tool_num, oXpos-ro, oYpos+r, oXpos-
                      kh_distance, oYpos+r)
1970 gcpy #upper left of end of slot
1971 gcpy              self.dxfarc(kh_tool_num, oXpos-kh_distance, oYpos, r,
                      90, 180)
1972 gcpy #lower left of end of slot
1973 gcpy              self.dxfarc(kh_tool_num, oXpos-kh_distance, oYpos, r,
                      180, 270)
1974 gcpy #             toolpath = toolpath.union(self.cutline(oXpos-
              kh_distance, oYpos, -kh_max_depth))
1975 gcpy          elif (kh_angle == 270):
1976 gcpy #Upper left of entry hole
1977 gcpy              self.dxfarc(kh_tool_num, oXpos, oYpos, self.tool_radius
                      (kh_tool_num, 1), 90, 180)
1978 gcpy #Upper right of entry hole
1979 gcpy              self.dxfarc(kh_tool_num, oXpos, oYpos, self.tool_radius
                      (kh_tool_num, 1), 0, 90)
1980 gcpy #left slot
1981 gcpy              self.dxfline(kh_tool_num, oXpos-r, oYpos-ro, oXpos-r,
```

```
                                     oYpos-kh_distance)
1982 gcpy #right slot
1983 gcpy                self.dxfline(kh_tool_num, oXpos+r, oYpos-ro, oXpos+r,
                           oYpos-kh_distance)
1984 gcpy #lower left of end of slot
1985 gcpy                self.dxfarc(kh_tool_num, oXpos, oYpos-kh_distance, r,
                           180, 270)
1986 gcpy #lower right of end of slot
1987 gcpy                self.dxfarc(kh_tool_num, oXpos, oYpos-kh_distance, r,
                           270, 360)
1988 gcpy #lower right of entry hole
1989 gcpy                self.dxfarc(kh_tool_num, oXpos, oYpos, rt, 180, 270-
                           angle)
1990 gcpy #lower left of entry hole
1991 gcpy                self.dxfarc(kh_tool_num, oXpos, oYpos, rt, 270+angle,
                           360)
1992 gcpy #            toolpath = toolpath.union(self.cutline(oXpos, oYpos-
         kh_distance, -kh_max_depth))
1993 gcpy #       print(self.zpos())
1994 gcpy #       self.setxpos(oXpos)
1995 gcpy #       self.setypos(oYpos)
1996 gcpy #       if self.generatepaths == False:
1997 gcpy #            return toolpath
1998 gcpy
1999 gcpy #  } else if (kh_angle == 90) {
2000 gcpy #    //Lower left of entry hole
2001 gcpy #    dxfarc(getxpos(), getypos(), 9.525/2, 180, 270, KH_tool_num);
2002 gcpy #    //Lower right of entry hole
2003 gcpy #    dxfarc(getxpos(), getypos(), 9.525/2, 270, 360, KH_tool_num);
2004 gcpy #    //Upper right of entry hole
2005 gcpy #    dxfarc(getxpos(), getypos(), 9.525/2, 0, acos(tool_diameter(
         KH_tool_num, 5)/tool_diameter(KH_tool_num, 1)), KH_tool_num);
2006 gcpy #    //Upper left of entry hole
2007 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
         );
2008 gcpy #    //Actual line of cut
2009 gcpy #    dxfline(getxpos(), getypos(), getxpos(), getypos()+kh_distance
         );
2010 gcpy #    //upper right of slot
2011 gcpy #    dxfarc(getxpos(), getypos()+kh_distance, tool_diameter(
         KH_tool_num, (kh_max_depth+4.36))/2, 0, 90, KH_tool_num);
2012 gcpy #    //upper left of slot
2013 gcpy #    dxfarc(getxpos(), getypos()+kh_distance, tool_diameter(
         KH_tool_num, (kh_max_depth+6.35))/2, 90, 180, KH_tool_num);
2014 gcpy #    //right of slot
2015 gcpy #    dxfline(
2016 gcpy #        getxpos()+tool_diameter(KH_tool_num, (kh_max_depth))/2,
2017 gcpy #        getypos()+(math.sqrt((tool_diameter(KH_tool_num, 1)^2)-(
         tool_diameter(KH_tool_num, 5)^2))/2), //( (kh_max_depth-6.34))
         /2)^2-(tool_diameter(KH_tool_num, (kh_max_depth-6.34))/2)^2,
2018 gcpy #        getxpos()+tool_diameter(KH_tool_num, (kh_max_depth))/2,
2019 gcpy #    //end position at top of slot
2020 gcpy #        getypos()+kh_distance,
2021 gcpy #        KH_tool_num);
2022 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);
2023 gcpy #    hull(){
2024 gcpy #      translate([xpos(), ypos(), zpos()]){
2025 gcpy #        keyhole_shaft(6.35, 9.525);
2026 gcpy #      }
2027 gcpy #      translate([xpos(), ypos(), zpos()-kh_max_depth]){
2028 gcpy #        keyhole_shaft(6.35, 9.525);
2029 gcpy #      }
2030 gcpy #    }
2031 gcpy #    hull(){
2032 gcpy #      translate([xpos(), ypos(), zpos()-kh_max_depth]){
2033 gcpy #        keyhole_shaft(6.35, 9.525);
2034 gcpy #      }
2035 gcpy #      translate([xpos(), ypos()+kh_distance, zpos()-kh_max_depth])
         {
2036 gcpy #        keyhole_shaft(6.35, 9.525);
2037 gcpy #      }
2038 gcpy #    }
2039 gcpy #    cutwithfeed(getxpos(), getypos(), -kh_max_depth, feed);
```

```
2040 gcpy #    cutwithfeed(getxpos(), getypos()+kh_distance, -kh_max_depth,
              feed);
2041 gcpy #    setypos(getypos()-kh_distance);
2042 gcpy #  } else if (kh_angle == 180) {
2043 gcpy #    //Lower right of entry hole
2044 gcpy #    dxfarc(getxpos(), getypos(), 9.525/2, 270, 360, KH_tool_num);
2045 gcpy #    //Upper right of entry hole
2046 gcpy #    dxfarc(getxpos(), getypos(), 9.525/2, 0, 90, KH_tool_num);
2047 gcpy #    //Upper left of entry hole
2048 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);
2049 gcpy #    //Lower left of entry hole
2050 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
              );
2051 gcpy #    //upper left of slot
2052 gcpy #    dxfarc(getxpos()-kh_distance, getypos(), tool_diameter(
              KH_tool_num, (kh_max_depth+6.35))/2, 90, 180, KH_tool_num);
2053 gcpy #    //lower left of slot
2054 gcpy #    dxfarc(getxpos()-kh_distance, getypos(), tool_diameter(
              KH_tool_num, (kh_max_depth+6.35))/2, 180, 270, KH_tool_num);
2055 gcpy #    //Actual line of cut
2056 gcpy #    dxfline(getxpos(), getypos(), getxpos()-kh_distance, getypos()
              );
2057 gcpy #    //upper left slot
2058 gcpy #    dxfline(
2059 gcpy #        getxpos()-(math.sqrt((tool_diameter(KH_tool_num, 1)^2)-(
              tool_diameter(KH_tool_num, 5)^2))/2),
2060 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,
2061 gcpy #        getxpos()-kh_distance,
2062 gcpy #    //end position at top of slot
2063 gcpy #        getypos()+tool_diameter(KH_tool_num, (kh_max_depth))/2,
2064 gcpy #        KH_tool_num);
2065 gcpy #    //lower right slot
2066 gcpy #    dxfline(
2067 gcpy #        getxpos()-(math.sqrt((tool_diameter(KH_tool_num, 1)^2)-(
              tool_diameter(KH_tool_num, 5)^2))/2),
2068 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,
2069 gcpy #        getxpos()-kh_distance,
2070 gcpy #    //end position at top of slot
2071 gcpy #        getypos()-tool_diameter(KH_tool_num, (kh_max_depth))/2,
2072 gcpy #        KH_tool_num);
2073 gcpy #    hull(){
2074 gcpy #      translate([xpos(), ypos(), zpos()]){
2075 gcpy #        keyhole_shaft(6.35, 9.525);
2076 gcpy #      }
2077 gcpy #      translate([xpos(), ypos(), zpos()-kh_max_depth]){
2078 gcpy #        keyhole_shaft(6.35, 9.525);
2079 gcpy #      }
2080 gcpy #    }
2081 gcpy #    hull(){
2082 gcpy #      translate([xpos(), ypos(), zpos()-kh_max_depth]){
2083 gcpy #        keyhole_shaft(6.35, 9.525);
2084 gcpy #      }
2085 gcpy #      translate([xpos()-kh_distance, ypos(), zpos()-kh_max_depth])
              {
2086 gcpy #        keyhole_shaft(6.35, 9.525);
2087 gcpy #      }
2088 gcpy #    }
2089 gcpy #    cutwithfeed(getxpos(), getypos(), -kh_max_depth, feed);
2090 gcpy #    cutwithfeed(getxpos()-kh_distance, getypos(), -kh_max_depth,
              feed);
2091 gcpy #    setxpos(getxpos()+kh_distance);
2092 gcpy #  } else if (kh_angle == 270) {
2093 gcpy #    //Upper right of entry hole
2094 gcpy #    dxfarc(getxpos(), getypos(), 9.525/2, 0, 90, KH_tool_num);
2095 gcpy #    //Upper left of entry hole
2096 gcpy #    dxfarc(getxpos(), getypos(), 9.525/2, 90, 180, KH_tool_num);
2097 gcpy #    //lower right of slot
2098 gcpy #    dxfarc(getxpos(), getypos()-kh_distance, tool_diameter(
              KH_tool_num, (kh_max_depth+4.36))/2, 270, 360, KH_tool_num);
2099 gcpy #    //lower left of slot
2100 gcpy #    dxfarc(getxpos(), getypos()-kh_distance, tool_diameter(
```

```
            KH_tool_num, (kh_max_depth+4.36))/2, 180, 270, KH_tool_num);
2101 gcpy #    //Actual line of cut
2102 gcpy #    dxfline(getxpos(), getypos(), getxpos(), getypos()-kh_distance
          );
2103 gcpy #    //right of slot
2104 gcpy #    dxfline(
2105 gcpy #        getxpos()+tool_diameter(KH_tool_num, (kh_max_depth))/2,
2106 gcpy #        getypos()-(math.sqrt((tool_diameter(KH_tool_num, 1)^2)-(
          tool_diameter(KH_tool_num, 5)^2))/2), //( (kh_max_depth-6.34))
          /2)^2-(tool_diameter(KH_tool_num, (kh_max_depth-6.34))/2)^2,
2107 gcpy #        getxpos()+tool_diameter(KH_tool_num, (kh_max_depth))/2,
2108 gcpy #    //end position at top of slot
2109 gcpy #        getypos()-kh_distance,
2110 gcpy #        KH_tool_num);
2111 gcpy #    //left of slot
2112 gcpy #    dxfline(
2113 gcpy #        getxpos()-tool_diameter(KH_tool_num, (kh_max_depth))/2,
2114 gcpy #        getypos()-(math.sqrt((tool_diameter(KH_tool_num, 1)^2)-(
          tool_diameter(KH_tool_num, 5)^2))/2), //( (kh_max_depth-6.34))
          /2)^2-(tool_diameter(KH_tool_num, (kh_max_depth-6.34))/2)^2,
2115 gcpy #        getxpos()-tool_diameter(KH_tool_num, (kh_max_depth))/2,
2116 gcpy #    //end position at top of slot
2117 gcpy #        getypos()-kh_distance,
2118 gcpy #        KH_tool_num);
2119 gcpy #    //Lower right of entry hole
2120 gcpy #    dxfarc(getxpos(), getypos(), 9.525/2, 360-acos(tool_diameter(
          KH_tool_num, 5)/tool_diameter(KH_tool_num, 1)), 360, KH_tool_num
          );
2121 gcpy #    //Lower left of entry hole
2122 gcpy #    dxfarc(getxpos(), getypos(), 9.525/2, 180, 180+acos(
          tool_diameter(KH_tool_num, 5)/tool_diameter(KH_tool_num, 1)),
          KH_tool_num);
2123 gcpy #    hull(){
2124 gcpy #      translate([xpos(), ypos(), zpos()]){
2125 gcpy #        keyhole_shaft(6.35, 9.525);
2126 gcpy #      }
2127 gcpy #      translate([xpos(), ypos(), zpos()-kh_max_depth]){
2128 gcpy #        keyhole_shaft(6.35, 9.525);
2129 gcpy #      }
2130 gcpy #    }
2131 gcpy #    hull(){
2132 gcpy #      translate([xpos(), ypos(), zpos()-kh_max_depth]){
2133 gcpy #        keyhole_shaft(6.35, 9.525);
2134 gcpy #      }
2135 gcpy #      translate([xpos(), ypos()-kh_distance, zpos()-kh_max_depth])
          {
2136 gcpy #        keyhole_shaft(6.35, 9.525);
2137 gcpy #      }
2138 gcpy #    }
2139 gcpy #    cutwithfeed(getxpos(), getypos(), -kh_max_depth, feed);
2140 gcpy #    cutwithfeed(getxpos(), getypos()-kh_distance, -kh_max_depth,
          feed);
2141 gcpy #    setypos(getypos()+kh_distance);
2142 gcpy #  }
2143 gcpy #}
```

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

BlocksCAD project page at: https://www.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:

```
2145 gcpy     def cut_pins(self, Joint_Width, stockZthickness,
                 Number_of_Dovetails, Spacing, Proportion, DTT_diameter,
                 DTT_angle):
2146 gcpy         DTO = Tan(math.radians(DTT_angle)) * (stockZthickness *
                     Proportion)
```

```
2147 gcpy            DTR = DTT_diameter/2 - DTO
2148 gcpy            cpr = self.rapidXY(0, stockZthickness + Spacing/2)
2149 gcpy            ctp = self.cutlinedxfgc(self.xpos(), self.ypos(), -
                        stockZthickness * Proportion)
2150 gcpy #          ctp = ctp.union(self.cutlinedxfgc(Joint_Width / (
             Number_of_Dovetails * 2), self.ypos(), -stockZthickness *
             Proportion))
2151 gcpy            i = 1
2152 gcpy            while i < Number_of_Dovetails * 2:
2153 gcpy #              print(i)
2154 gcpy                ctp = ctp.union(self.cutlinedxfgc(i * (Joint_Width / (
                            Number_of_Dovetails * 2)), self.ypos(), -
                            stockZthickness * Proportion))
2155 gcpy                ctp = ctp.union(self.cutlinedxfgc(i * (Joint_Width / (
                            Number_of_Dovetails * 2)), (stockZthickness +
                            Spacing) + (stockZthickness * Proportion) - (
                            DTT_diameter/2), -(stockZthickness * Proportion)))
2156 gcpy                ctp = ctp.union(self.cutlinedxfgc(i * (Joint_Width / (
                            Number_of_Dovetails * 2)), stockZthickness + Spacing
                            /2, -(stockZthickness * Proportion)))
2157 gcpy                ctp = ctp.union(self.cutlinedxfgc((i + 1) * (
                            Joint_Width / (Number_of_Dovetails * 2)),
                            stockZthickness + Spacing/2,-(stockZthickness *
                            Proportion)))
2158 gcpy                self.dxfrectangleround(self.currenttoolnumber(),
2159 gcpy                    i * (Joint_Width / (Number_of_Dovetails * 2))-DTR,
2160 gcpy                    stockZthickness + (Spacing/2) - DTR,
2161 gcpy                    DTR * 2,
2162 gcpy                    (stockZthickness * Proportion) + Spacing/2 + DTR *
                                2 - (DTT_diameter/2),
2163 gcpy                    DTR)
2164 gcpy                i += 2
2165 gcpy            self.rapidZ(0)
2166 gcpy            return ctp
```

and

```
2168 gcpy        def cut_tails(self, Joint_Width, stockZthickness,
                    Number_of_Dovetails, Spacing, Proportion, DTT_diameter,
                    DTT_angle):
2169 gcpy            DTO = Tan(math.radians(DTT_angle)) * (stockZthickness *
                        Proportion)
2170 gcpy            DTR = DTT_diameter/2 - DTO
2171 gcpy            cpr = self.rapidXY(0, 0)
2172 gcpy            ctp = self.cutlinedxfgc(self.xpos(), self.ypos(), -
                        stockZthickness * Proportion)
2173 gcpy            ctp = ctp.union(self.cutlinedxfgc(
2174 gcpy                Joint_Width / (Number_of_Dovetails * 2) - (DTT_diameter
                            - DTO),
2175 gcpy                self.ypos(),
2176 gcpy                -stockZthickness * Proportion))
2177 gcpy            i = 1
2178 gcpy            while i < Number_of_Dovetails * 2:
2179 gcpy                ctp = ctp.union(self.cutlinedxfgc(
2180 gcpy                    i * (Joint_Width / (Number_of_Dovetails * 2)) - (
                                DTT_diameter - DTO),
2181 gcpy                    stockZthickness * Proportion - DTT_diameter / 2,
2182 gcpy                    -(stockZthickness * Proportion)))
2183 gcpy                ctp = ctp.union(self.cutarcCWdxf(180, 90,
2184 gcpy                    i * (Joint_Width / (Number_of_Dovetails * 2)),
2185 gcpy                    stockZthickness * Proportion - DTT_diameter / 2,
2186 gcpy #                  self.ypos(),
2187 gcpy                    DTT_diameter - DTO, 0, 1))
2188 gcpy                ctp = ctp.union(self.cutarcCWdxf(90, 0,
2189 gcpy                    i * (Joint_Width / (Number_of_Dovetails * 2)),
2190 gcpy                    stockZthickness * Proportion - DTT_diameter / 2,
2191 gcpy                    DTT_diameter - DTO, 0, 1))
2192 gcpy                ctp = ctp.union(self.cutlinedxfgc(
2193 gcpy                    i * (Joint_Width / (Number_of_Dovetails * 2)) + (
                                DTT_diameter - DTO),
2194 gcpy                    0,
2195 gcpy                    -(stockZthickness * Proportion)))
2196 gcpy                ctp = ctp.union(self.cutlinedxfgc(
2197 gcpy                    (i + 2) * (Joint_Width / (Number_of_Dovetails * 2))
                                - (DTT_diameter - DTO),
2198 gcpy                    0,
2199 gcpy                    -(stockZthickness * Proportion)))
2200 gcpy                i += 2
```

```
2201 gcpy          self.rapidZ(0)
2202 gcpy          self.rapidXY(0, 0)
2203 gcpy          ctp = ctp.union(self.cutlinedxfgc(self.xpos(), self.ypos(),
                      -stockZthickness * Proportion))
2204 gcpy          self.dxfarc(self.currenttoolnumber(), 0, 0, DTR, 180, 270)
2205 gcpy          self.dxfline(self.currenttoolnumber(), -DTR, 0, -DTR,
                      stockZthickness + DTR)
2206 gcpy          self.dxfarc(self.currenttoolnumber(), 0, stockZthickness +
                      DTR, DTR, 90, 180)
2207 gcpy          self.dxfline(self.currenttoolnumber(), 0, stockZthickness +
                      DTR * 2, Joint_Width, stockZthickness + DTR * 2)
2208 gcpy          i = 0
2209 gcpy          while i < Number_of_Dovetails * 2:
2210 gcpy              ctp = ctp.union(self.cutline(i * (Joint_Width / (
                          Number_of_Dovetails * 2)), stockZthickness + DTO, -(
                          stockZthickness * Proportion)))
2211 gcpy              ctp = ctp.union(self.cutline((i+2) * (Joint_Width / (
                          Number_of_Dovetails * 2)), stockZthickness + DTO, -(
                          stockZthickness * Proportion)))
2212 gcpy              ctp = ctp.union(self.cutline((i+2) * (Joint_Width / (
                          Number_of_Dovetails * 2)), 0, -(stockZthickness *
                          Proportion)))
2213 gcpy              self.dxfarc(self.currenttoolnumber(), i * (Joint_Width
                          / (Number_of_Dovetails * 2)), 0, DTR, 270, 360)
2214 gcpy              self.dxfline(self.currenttoolnumber(),
2215 gcpy                  i * (Joint_Width / (Number_of_Dovetails * 2)) + DTR
                              ,
2216 gcpy                  0,
2217 gcpy                  i * (Joint_Width / (Number_of_Dovetails * 2)) + DTR
                              , stockZthickness * Proportion - DTT_diameter /
                              2)
2218 gcpy              self.dxfarc(self.currenttoolnumber(), (i + 1) * (
                          Joint_Width / (Number_of_Dovetails * 2)),
                          stockZthickness * Proportion - DTT_diameter / 2, (
                          Joint_Width / (Number_of_Dovetails * 2)) - DTR, 90,
                          180)
2219 gcpy              self.dxfarc(self.currenttoolnumber(), (i + 1) * (
                          Joint_Width / (Number_of_Dovetails * 2)),
                          stockZthickness * Proportion - DTT_diameter / 2, (
                          Joint_Width / (Number_of_Dovetails * 2)) - DTR, 0,
                          90)
2220 gcpy              self.dxfline(self.currenttoolnumber(),
2221 gcpy                  (i + 2) * (Joint_Width / (Number_of_Dovetails * 2))
                              - DTR,
2222 gcpy                  0,
2223 gcpy                  (i + 2) * (Joint_Width / (Number_of_Dovetails * 2))
                              - DTR, stockZthickness * Proportion -
                              DTT_diameter / 2)
2224 gcpy              self.dxfarc(self.currenttoolnumber(), (i + 2) * (
                          Joint_Width / (Number_of_Dovetails * 2)), 0, DTR,
                          180, 270)
2225 gcpy              i += 2
2226 gcpy          self.dxfarc(self.currenttoolnumber(), Joint_Width,
                      stockZthickness + DTR, DTR, 0, 90)
2227 gcpy          self.dxfline(self.currenttoolnumber(), Joint_Width + DTR,
                      stockZthickness + DTR, Joint_Width + DTR, 0)
2228 gcpy          self.dxfarc(self.currenttoolnumber(), Joint_Width, 0, DTR,
                      270, 360)
2229 gcpy          return ctp
```

which are used as:

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

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

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

3.7.0.2.5   Full-blind box joints   BlocksCAD project page at: https://www.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:

```
2231 gcpy    def Full_Blind_Finger_Joint_square(self, bx, by, orientation,
                 side, width, thickness, Number_of_Pins, largeVdiameter,
                 smallDiameter, normalormirror = "Default"):
2232 gcpy    # Joint_Orientation = "Horizontal" "Even" == "Lower", "Odd" ==
                 "Upper"
2233 gcpy    # Joint_Orientation = "Vertical" "Even" == "Left", "Odd" == "
                 Right"
2234 gcpy        if (orientation == "Vertical"):
2235 gcpy            if (normalormirror == "Default" and side != "Both"):
2236 gcpy                if (side == "Left"):
2237 gcpy                    normalormirror = "Even"
2238 gcpy                if (side == "Right"):
2239 gcpy                    normalormirror = "Odd"
2240 gcpy        if (orientation == "Horizontal"):
2241 gcpy            if (normalormirror == "Default" and side != "Both"):
2242 gcpy                if (side == "Lower"):
2243 gcpy                    normalormirror = "Even"
2244 gcpy                if (side == "Upper"):
2245 gcpy                    normalormirror = "Odd"
2246 gcpy        Finger_Width = ((Number_of_Pins * 2) - 1) * smallDiameter *
                 1.1
2247 gcpy        Finger_Origin = width/2 - Finger_Width/2
2248 gcpy        rapid = self.rapidZ(0)
2249 gcpy        self.setdxfcolor("Cyan")
2250 gcpy        rapid = rapid.union(self.rapidXY(bx, by))
2251 gcpy        toolpath = (self.Finger_Joint_square(bx, by, orientation,
                 side, width, thickness, Number_of_Pins, Finger_Origin,
                 smallDiameter))
2252 gcpy        if (orientation == "Vertical"):
2253 gcpy            if (side == "Both"):
2254 gcpy                toolpath = self.cutrectanglerounddxf(self.
                     currenttoolnum, bx - (thickness - smallDiameter
                     /2), by-smallDiameter/2, 0, (thickness * 2) -
                     smallDiameter, width+smallDiameter, (
                     smallDiameter / 2) / Tan(math.radians(45)),
                     smallDiameter/2)
2255 gcpy            if (side == "Left"):
2256 gcpy                toolpath = self.cutrectanglerounddxf(self.
                     currenttoolnum, bx - (smallDiameter/2), by-
                     smallDiameter/2, 0, thickness, width+
                     smallDiameter, ((smallDiameter / 2) / Tan(math.
                     radians(45))), smallDiameter/2)
2257 gcpy            if (side == "Right"):
2258 gcpy                toolpath = self.cutrectanglerounddxf(self.
                     currenttoolnum, bx - (thickness - smallDiameter
                     /2), by-smallDiameter/2, 0, thickness, width+
                     smallDiameter, ((smallDiameter / 2) / Tan(math.
                     radians(45))), smallDiameter/2)
2259 gcpy        toolpath = toolpath.union(self.Finger_Joint_square(bx, by,
                 orientation, side, width, thickness, Number_of_Pins,
                 Finger_Origin, smallDiameter))
2260 gcpy        if (orientation == "Horizontal"):
2261 gcpy            if (side == "Both"):
2262 gcpy                toolpath = self.cutrectanglerounddxf(
2263 gcpy                    self.currenttoolnum,
2264 gcpy                    bx-smallDiameter/2,
2265 gcpy                    by - (thickness - smallDiameter/2),
2266 gcpy                    0,
2267 gcpy                    width+smallDiameter,
2268 gcpy                    (thickness * 2) - smallDiameter,
2269 gcpy                    (smallDiameter / 2) / Tan(math.radians(45)),
2270 gcpy                    smallDiameter/2)
2271 gcpy            if (side == "Lower"):
2272 gcpy                toolpath = self.cutrectanglerounddxf(
2273 gcpy                    self.currenttoolnum,
2274 gcpy                    bx - (smallDiameter/2),
2275 gcpy                    by - smallDiameter/2,
2276 gcpy                    0,
```

```
2277 gcpy                              width+smallDiameter,
2278 gcpy                              thickness,
2279 gcpy                              ((smallDiameter / 2) / Tan(math.radians(45))),
2280 gcpy                              smallDiameter/2)
2281 gcpy                  if (side == "Upper"):
2282 gcpy                      toolpath = self.cutrectanglerounddxf(
2283 gcpy                          self.currenttoolnum,
2284 gcpy                          bx - smallDiameter/2,
2285 gcpy                          by - (thickness - smallDiameter/2),
2286 gcpy                          0,
2287 gcpy                          width+smallDiameter,
2288 gcpy                          thickness,
2289 gcpy                          ((smallDiameter / 2) / Tan(math.radians(45))),
2290 gcpy                          smallDiameter/2)
2291 gcpy              toolpath = toolpath.union(self.Finger_Joint_square(bx, by,
                          orientation, side, width, thickness, Number_of_Pins,
                          Finger_Origin, smallDiameter))
2292 gcpy              return toolpath
2293 gcpy
2294 gcpy          def Finger_Joint_square(self, bx, by, orientation, side, width,
                      thickness, Number_of_Pins, Finger_Origin, smallDiameter,
                      normalormirror = "Default"):
2295 gcpy              jointdepth = -(thickness - (smallDiameter / 2) / Tan(math.
                          radians(45)))
2296 gcpy      # Joint_Orientation = "Horizontal" "Even" == "Lower", "Odd" ==
                  "Upper"
2297 gcpy      # Joint_Orientation = "Vertical" "Even" == "Left", "Odd" == "
                  Right"
2298 gcpy              if (orientation == "Vertical"):
2299 gcpy                  if (normalormirror == "Default" and side != "Both"):
2300 gcpy                      if (side == "Left"):
2301 gcpy                          normalormirror = "Even"
2302 gcpy                      if (side == "Right"):
2303 gcpy                          normalormirror = "Odd"
2304 gcpy              if (orientation == "Horizontal"):
2305 gcpy                  if (normalormirror == "Default" and side != "Both"):
2306 gcpy                      if (side == "Lower"):
2307 gcpy                          normalormirror = "Even"
2308 gcpy                      if (side == "Upper"):
2309 gcpy                          normalormirror = "Odd"
2310 gcpy              radius = smallDiameter/2
2311 gcpy              jointwidth = thickness - smallDiameter
2312 gcpy              toolpath = self.currenttool()
2313 gcpy              rapid = self.rapidZ(0)
2314 gcpy              self.setdxfcolor("Blue")
2315 gcpy              toolpath = toolpath.union(self.cutlineZgcfeed(jointdepth
                          ,1000))
2316 gcpy              self.beginpolyline(self.currenttool())
2317 gcpy              if (orientation == "Vertical"):
2318 gcpy                  rapid = rapid.union(self.rapidXY(bx, by + Finger_Origin
                              ))
2319 gcpy                  self.addvertex(self.currenttoolnumber(), self.xpos(),
                              self.ypos())
2320 gcpy                  toolpath = toolpath.union(self.cutlineZgcfeed(
                              jointdepth,1000))
2321 gcpy                  i = 0
2322 gcpy                  while i <= Number_of_Pins - 1:
2323 gcpy                      if (side == "Right"):
2324 gcpy                          toolpath = toolpath.union(self.cutvertexdxf(
                                      self.xpos(), self.ypos() + smallDiameter +
                                      radius/5, jointdepth))
2325 gcpy                      if (side == "Left" or side == "Both"):
2326 gcpy                          toolpath = toolpath.union(self.cutvertexdxf(
                                      self.xpos(), self.ypos() + radius,
                                      jointdepth))
2327 gcpy                          toolpath = toolpath.union(self.cutvertexdxf(
                                      self.xpos() + jointwidth, self.ypos(),
                                      jointdepth))
2328 gcpy                          toolpath = toolpath.union(self.cutvertexdxf(
                                      self.xpos(), self.ypos() + radius/5,
                                      jointdepth))
2329 gcpy                          toolpath = toolpath.union(self.cutvertexdxf(
                                      self.xpos() - jointwidth, self.ypos(),
                                      jointdepth))
2330 gcpy                          toolpath = toolpath.union(self.cutvertexdxf(
                                      self.xpos(), self.ypos() + radius,
                                      jointdepth))
2331 gcpy                      if (side == "Left"):
```

```
2332 gcpy                                     toolpath = toolpath.union(self.cutvertexdxf(
                                                 self.xpos(), self.ypos() + smallDiameter +
                                                 radius/5, jointdepth))
2333 gcpy                         if (side == "Right" or side == "Both"):
2334 gcpy                             if (i < (Number_of_Pins - 1)):
2335 gcpy          #                     print(i)
2336 gcpy                                 toolpath = toolpath.union(self.cutvertexdxf
                                             (self.xpos(), self.ypos() + radius,
                                             jointdepth))
2337 gcpy                                 toolpath = toolpath.union(self.cutvertexdxf
                                             (self.xpos() - jointwidth, self.ypos(),
                                             jointdepth))
2338 gcpy                                 toolpath = toolpath.union(self.cutvertexdxf
                                             (self.xpos(), self.ypos() + radius/5,
                                             jointdepth))
2339 gcpy                                 toolpath = toolpath.union(self.cutvertexdxf
                                             (self.xpos() + jointwidth, self.ypos(),
                                             jointdepth))
2340 gcpy                                 toolpath = toolpath.union(self.cutvertexdxf
                                             (self.xpos(), self.ypos() + radius,
                                             jointdepth))
2341 gcpy                         i += 1
2342 gcpy          # Joint_Orientation = "Horizontal" "Even" == "Lower", "Odd" ==
                      "Upper"
2343 gcpy             if (orientation == "Horizontal"):
2344 gcpy                 rapid = rapid.union(self.rapidXY(bx + Finger_Origin, by
                             ))
2345 gcpy                 self.addvertex(self.currenttoolnumber(), self.xpos(),
                             self.ypos())
2346 gcpy                 toolpath = toolpath.union(self.cutlineZgcfeed(
                             jointdepth,1000))
2347 gcpy                 i = 0
2348 gcpy                 while i <= Number_of_Pins - 1:
2349 gcpy                     if (side == "Upper"):
2350 gcpy                         toolpath = toolpath.union(self.cutvertexdxf(
                                     self.xpos() + smallDiameter + radius/5, self
                                     .ypos(), jointdepth))
2351 gcpy                     if (side == "Lower" or side == "Both"):
2352 gcpy                         toolpath = toolpath.union(self.cutvertexdxf(
                                     self.xpos() + radius, self.ypos(),
                                     jointdepth))
2353 gcpy                         toolpath = toolpath.union(self.cutvertexdxf(
                                     self.xpos(), self.ypos() + jointwidth,
                                     jointdepth))
2354 gcpy                         toolpath = toolpath.union(self.cutvertexdxf(
                                     self.xpos() + radius/5, self.ypos(),
                                     jointdepth))
2355 gcpy                         toolpath = toolpath.union(self.cutvertexdxf(
                                     self.xpos(), self.ypos() - jointwidth,
                                     jointdepth))
2356 gcpy                         toolpath = toolpath.union(self.cutvertexdxf(
                                     self.xpos() + radius, self.ypos(),
                                     jointdepth))
2357 gcpy                     if (side == "Lower"):
2358 gcpy                         toolpath = toolpath.union(self.cutvertexdxf(
                                     self.xpos() + smallDiameter + radius/5, self
                                     .ypos(), jointdepth))
2359 gcpy                     if (side == "Upper" or side == "Both"):
2360 gcpy                         if (i < (Number_of_Pins - 1)):
2361 gcpy          #                     print(i)
2362 gcpy                                 toolpath = toolpath.union(self.cutvertexdxf
                                             (self.xpos() + radius, self.ypos(),
                                             jointdepth))
2363 gcpy                                 toolpath = toolpath.union(self.cutvertexdxf
                                             (self.xpos(), self.ypos() - jointwidth,
                                             jointdepth))
2364 gcpy                                 toolpath = toolpath.union(self.cutvertexdxf
                                             (self.xpos() + radius/5, self.ypos(),
                                             jointdepth))
2365 gcpy                                 toolpath = toolpath.union(self.cutvertexdxf
                                             (self.xpos(), self.ypos() + jointwidth,
                                             jointdepth))
2366 gcpy                                 toolpath = toolpath.union(self.cutvertexdxf
                                             (self.xpos() + radius, self.ypos(),
                                             jointdepth))
2367 gcpy                         i += 1
2368 gcpy             self.closepolyline(self.currenttoolnumber())
2369 gcpy         return toolpath
```

```
2370 gcpy
2371 gcpy      def Full_Blind_Finger_Joint_smallV(self, bx, by, orientation,
                   side, width, thickness, Number_of_Pins, largeVdiameter,
                   smallDiameter):
2372 gcpy          rapid = self.rapidZ(0)
2373 gcpy      #    rapid = rapid.union(self.rapidXY(bx, by))
2374 gcpy          self.setdxfcolor("Red")
2375 gcpy          if (orientation == "Vertical"):
2376 gcpy              rapid = rapid.union(self.rapidXY(bx, by - smallDiameter
                          /6))
2377 gcpy              toolpath = self.cutlineZgcfeed(-thickness,1000)
2378 gcpy              toolpath = self.cutlinedxfgc(bx, by + width +
                          smallDiameter/6, - thickness)
2379 gcpy          if (orientation == "Horizontal"):
2380 gcpy              rapid = rapid.union(self.rapidXY(bx - smallDiameter/6,
                          by))
2381 gcpy              toolpath = self.cutlineZgcfeed(-thickness,1000)
2382 gcpy              toolpath = self.cutlinedxfgc(bx + width + smallDiameter
                          /6, by, -thickness)
2383 gcpy      #        rapid = self.rapidZ(0)
2384 gcpy
2385 gcpy          return toolpath
2386 gcpy
2387 gcpy      def Full_Blind_Finger_Joint_largeV(self, bx, by, orientation,
                   side, width, thickness, Number_of_Pins, largeVdiameter,
                   smallDiameter):
2388 gcpy          radius = smallDiameter/2
2389 gcpy          rapid = self.rapidZ(0)
2390 gcpy          Finger_Width = ((Number_of_Pins * 2) - 1) * smallDiameter *
                       1.1
2391 gcpy          Finger_Origin = width/2 - Finger_Width/2
2392 gcpy      #    rapid = rapid.union(self.rapidXY(bx, by))
2393 gcpy      # Joint_Orientation = "Horizontal" "Even" == "Lower", "Odd" ==
                   "Upper"
2394 gcpy      # Joint_Orientation = "Vertical" "Even" == "Left", "Odd" == "
                   Right"
2395 gcpy          if (orientation == "Vertical"):
2396 gcpy              rapid = rapid.union(self.rapidXY(bx, by))
2397 gcpy              toolpath = self.cutlineZgcfeed(-thickness,1000)
2398 gcpy              toolpath = toolpath.union(self.cutlinedxfgc(bx, by +
                          Finger_Origin, -thickness))
2399 gcpy              rapid = self.rapidZ(0)
2400 gcpy              rapid = rapid.union(self.rapidXY(bx, by + width -
                          Finger_Origin))
2401 gcpy              self.setdxfcolor("Blue")
2402 gcpy              toolpath = toolpath.union(self.cutlineZgcfeed(-
                          thickness,1000))
2403 gcpy              toolpath = toolpath.union(self.cutlinedxfgc(bx, by +
                          width, -thickness))
2404 gcpy              if (side == "Left" or side == "Both"):
2405 gcpy                  rapid = self.rapidZ(0)
2406 gcpy                  self.setdxfcolor("Dark Gray")
2407 gcpy                  rapid = rapid.union(self.rapidXY(bx+thickness-(
                              smallDiameter / 2) / Tan(math.radians(45)), by -
                              radius/2))
2408 gcpy                  toolpath = toolpath.union(self.cutlineZgcfeed(-(
                              smallDiameter / 2) / Tan(math.radians(45))
                              ,10000))
2409 gcpy                  toolpath = toolpath.union(self.cutlinedxfgc(bx+
                              thickness-(smallDiameter / 2) / Tan(math.radians
                              (45)), by + width + radius/2, -(smallDiameter /
                              2) / Tan(math.radians(45))))
2410 gcpy                  rapid = self.rapidZ(0)
2411 gcpy                  self.setdxfcolor("Green")
2412 gcpy                  rapid = rapid.union(self.rapidXY(bx+thickness/2, by
                              +width))
2413 gcpy                  toolpath = toolpath.union(self.cutlineZgcfeed(-
                              thickness/2,1000))
2414 gcpy                  toolpath = toolpath.union(self.cutlinedxfgc(bx+
                              thickness/2, by + width -thickness, -thickness
                              /2))
2415 gcpy                  rapid = self.rapidZ(0)
2416 gcpy                  rapid = rapid.union(self.rapidXY(bx+thickness/2, by
                              ))
2417 gcpy                  toolpath = toolpath.union(self.cutlineZgcfeed(-
                              thickness/2,1000))
2418 gcpy                  toolpath = toolpath.union(self.cutlinedxfgc(bx+
                              thickness/2, by +thickness, -thickness/2))
```

```
2419 gcpy                  if (side == "Right" or side == "Both"):
2420 gcpy                      rapid = self.rapidZ(0)
2421 gcpy                      self.setdxfcolor("Dark␣Gray")
2422 gcpy                      rapid = rapid.union(self.rapidXY(bx-(thickness-(
                                  smallDiameter / 2) / Tan(math.radians(45))), by
                                  - radius/2))
2423 gcpy                      toolpath = toolpath.union(self.cutlineZgcfeed(-(
                                  smallDiameter / 2) / Tan(math.radians(45))
                                  ,10000))
2424 gcpy                      toolpath = toolpath.union(self.cutlinedxfgc(bx-(
                                  thickness-(smallDiameter / 2) / Tan(math.radians
                                  (45))), by + width + radius/2, -(smallDiameter /
                                   2) / Tan(math.radians(45))))
2425 gcpy                      rapid = self.rapidZ(0)
2426 gcpy                      self.setdxfcolor("Green")
2427 gcpy                      rapid = rapid.union(self.rapidXY(bx-thickness/2, by
                                  +width))
2428 gcpy                      toolpath = toolpath.union(self.cutlineZgcfeed(-
                                  thickness/2,1000))
2429 gcpy                      toolpath = toolpath.union(self.cutlinedxfgc(bx-
                                  thickness/2, by + width -thickness, -thickness
                                  /2))
2430 gcpy                      rapid = self.rapidZ(0)
2431 gcpy                      rapid = rapid.union(self.rapidXY(bx-thickness/2, by
                                  ))
2432 gcpy                      toolpath = toolpath.union(self.cutlineZgcfeed(-
                                  thickness/2,1000))
2433 gcpy                      toolpath = toolpath.union(self.cutlinedxfgc(bx-
                                  thickness/2, by +thickness, -thickness/2))
2434 gcpy          # Joint_Orientation = "Horizontal" "Even" == "Lower", "Odd" ==
                  "Upper"
2435 gcpy              if (orientation == "Horizontal"):
2436 gcpy                  rapid = rapid.union(self.rapidXY(bx, by))
2437 gcpy                  self.setdxfcolor("Blue")
2438 gcpy                  toolpath = self.cutlineZgcfeed(-thickness,1000)
2439 gcpy                  toolpath = toolpath.union(self.cutlinedxfgc(bx +
                              Finger_Origin, by, -thickness))
2440 gcpy                  rapid = rapid.union(self.rapidZ(0))
2441 gcpy                  rapid = rapid.union(self.rapidXY(bx + width -
                              Finger_Origin, by))
2442 gcpy                  toolpath = toolpath.union(self.cutlineZgcfeed(-
                              thickness,1000))
2443 gcpy                  toolpath = toolpath.union(self.cutlinedxfgc(bx + width,
                               by, -thickness))
2444 gcpy                  if (side == "Lower" or side == "Both"):
2445 gcpy                      rapid = self.rapidZ(0)
2446 gcpy                      self.setdxfcolor("Dark␣Gray")
2447 gcpy                      rapid = rapid.union(self.rapidXY(bx - radius, by+
                                  thickness-(smallDiameter / 2) / Tan(math.radians
                                  (45))))
2448 gcpy                      toolpath = toolpath.union(self.cutlineZgcfeed(-(
                                  smallDiameter / 2) / Tan(math.radians(45))
                                  ,10000))
2449 gcpy                      toolpath = toolpath.union(self.cutlinedxfgc(bx +
                                  width + radius, by+thickness-(smallDiameter / 2)
                                   / Tan(math.radians(45)), -(smallDiameter / 2) /
                                   Tan(math.radians(45))))
2450 gcpy                      rapid = self.rapidZ(0)
2451 gcpy                      self.setdxfcolor("Green")
2452 gcpy                      rapid = rapid.union(self.rapidXY(bx+width, by+
                                  thickness/2))
2453 gcpy                      toolpath = toolpath.union(self.cutlineZgcfeed(-
                                  thickness/2,1000))
2454 gcpy                      toolpath = toolpath.union(self.cutlinedxfgc(bx +
                                  width -thickness, by+thickness/2, -thickness/2))
2455 gcpy                      rapid = self.rapidZ(0)
2456 gcpy                      rapid = rapid.union(self.rapidXY(bx, by+thickness
                                  /2))
2457 gcpy                      toolpath = toolpath.union(self.cutlineZgcfeed(-
                                  thickness/2,1000))
2458 gcpy                      toolpath = toolpath.union(self.cutlinedxfgc(bx +
                                  thickness, by+thickness/2, -thickness/2))
2459 gcpy                  if (side == "Upper" or side == "Both"):
2460 gcpy                      rapid = self.rapidZ(0)
2461 gcpy                      self.setdxfcolor("Dark␣Gray")
2462 gcpy                      rapid = rapid.union(self.rapidXY(bx - radius, by-(
                                  thickness-(smallDiameter / 2) / Tan(math.radians
                                  (45))))
```

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

3.8   (Reading) G-code Files

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

First, a template file will be necessary:

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

previewgcodefile   Which simply needs to call the previewgcodefile command:

```
2490 gcpy          def previewgcodefile(self, gc_file):
2491 gcpy              gc_file = open(gc_file, 'r')
2492 gcpy              gcfilecontents = []
2493 gcpy              with gc_file as file:
2494 gcpy                  for line in file:
2495 gcpy                      command = line
```

```
2496 gcpy                       gcfilecontents.append(line)
2497 gcpy
2498 gcpy            numlinesfound = 0
2499 gcpy            for line in gcfilecontents:
2500 gcpy #                print(line)
2501 gcpy                if line[:10] == "(stockMin:":
2502 gcpy                    subdivisions = line.split()
2503 gcpy                    extentleft = float(subdivisions[0][10:-3])
2504 gcpy                    extentfb = float(subdivisions[1][:-3])
2505 gcpy                    extentd = float(subdivisions[2][:-3])
2506 gcpy                    numlinesfound = numlinesfound + 1
2507 gcpy                if line[:13] == "(STOCK/BLOCK,":
2508 gcpy                    subdivisions = line.split()
2509 gcpy                    sizeX = float(subdivisions[0][13:-1])
2510 gcpy                    sizeY = float(subdivisions[1][:-1])
2511 gcpy                    sizeZ = float(subdivisions[4][:-1])
2512 gcpy                    numlinesfound = numlinesfound + 1
2513 gcpy                if line[:3] == "G21":
2514 gcpy                    units = "mm"
2515 gcpy                    numlinesfound = numlinesfound + 1
2516 gcpy                if numlinesfound >=3:
2517 gcpy                    break
2518 gcpy #                print(numlinesfound)
2519 gcpy
2520 gcpy            self.setupcuttingarea(sizeX, sizeY, sizeZ, extentleft,
                        extentfb, extentd)
2521 gcpy
2522 gcpy            commands = []
2523 gcpy            for line in gcfilecontents:
2524 gcpy                Xc = 0
2525 gcpy                Yc = 0
2526 gcpy                Zc = 0
2527 gcpy                Fc = 0
2528 gcpy                Xp = 0.0
2529 gcpy                Yp = 0.0
2530 gcpy                Zp = 0.0
2531 gcpy                if line == "G53G0Z-5.000\n":
2532 gcpy                    self.movetosafeZ()
2533 gcpy                if line[:3] == "M6T":
2534 gcpy                    tool = int(line[3:])
2535 gcpy                    self.toolchange(tool)
2536 gcpy                if line[:2] == "G0":
2537 gcpy                    machinestate = "rapid"
2538 gcpy                if line[:2] == "G1":
2539 gcpy                    machinestate = "cutline"
2540 gcpy                if line[:2] == "G0" or line[:2] == "G1" or line[:1] ==
                        "X" or line[:1] == "Y" or line[:1] == "Z":
2541 gcpy                    if "F" in line:
2542 gcpy                        Fplus = line.split("F")
2543 gcpy                        Fc = 1
2544 gcpy                        fr = float(Fplus[1])
2545 gcpy                        line = Fplus[0]
2546 gcpy                    if "Z" in line:
2547 gcpy                        Zplus = line.split("Z")
2548 gcpy                        Zc = 1
2549 gcpy                        Zp = float(Zplus[1])
2550 gcpy                        line = Zplus[0]
2551 gcpy                    if "Y" in line:
2552 gcpy                        Yplus = line.split("Y")
2553 gcpy                        Yc = 1
2554 gcpy                        Yp = float(Yplus[1])
2555 gcpy                        line = Yplus[0]
2556 gcpy                    if "X" in line:
2557 gcpy                        Xplus = line.split("X")
2558 gcpy                        Xc = 1
2559 gcpy                        Xp = float(Xplus[1])
2560 gcpy                    if Zc == 1:
2561 gcpy                        if Yc == 1:
2562 gcpy                            if Xc == 1:
2563 gcpy                                if machinestate == "rapid":
2564 gcpy                                    command = "rapidXYZ(" + str(Xp) + "
                                            ,␣" + str(Yp) + ",␣" + str(Zp) +
                                            ")"
2565 gcpy                                    self.rapidXYZ(Xp, Yp, Zp)
2566 gcpy                                else:
2567 gcpy                                    command = "cutlineXYZ(" + str(Xp) +
                                            ",␣" + str(Yp) + ",␣" + str(Zp)
                                            + ")"
```

```
2568 gcpy                                          self.cutlineXYZ(Xp, Yp, Zp)
2569 gcpy                              else:
2570 gcpy                                  if machinestate == "rapid":
2571 gcpy                                      command = "rapidYZ(" + str(Yp) + ",
                                                  ␣" + str(Zp) + ")"
2572 gcpy                                      self.rapidYZ(Yp, Zp)
2573 gcpy                                  else:
2574 gcpy                                      command = "cutlineYZ(" + str(Yp) +
                                                  ",␣" + str(Zp) + ")"
2575 gcpy                                      self.cutlineYZ(Yp, Zp)
2576 gcpy                          else:
2577 gcpy                              if Xc == 1:
2578 gcpy                                  if machinestate == "rapid":
2579 gcpy                                      command = "rapidXZ(" + str(Xp) + ",
                                                  ␣" + str(Zp) + ")"
2580 gcpy                                      self.rapidXZ(Xp, Zp)
2581 gcpy                                  else:
2582 gcpy                                      command = "cutlineXZ(" + str(Xp) +
                                                  ",␣" + str(Zp) + ")"
2583 gcpy                                      self.cutlineXZ(Xp, Zp)
2584 gcpy                              else:
2585 gcpy                                  if machinestate == "rapid":
2586 gcpy                                      command = "rapidZ(" + str(Zp) + ")"
2587 gcpy                                      self.rapidZ(Zp)
2588 gcpy                                  else:
2589 gcpy                                      command = "cutlineZ(" + str(Zp) + "
                                                  )"
2590 gcpy                                      self.cutlineZ(Zp)
2591 gcpy                      else:
2592 gcpy                          if Yc == 1:
2593 gcpy                              if Xc == 1:
2594 gcpy                                  if machinestate == "rapid":
2595 gcpy                                      command = "rapidXY(" + str(Xp) + ",
                                                  ␣" + str(Yp) + ")"
2596 gcpy                                      self.rapidXY(Xp, Yp)
2597 gcpy                                  else:
2598 gcpy                                      command = "cutlineXY(" + str(Xp) +
                                                  ",␣" + str(Yp) + ")"
2599 gcpy                                      self.cutlineXY(Xp, Yp)
2600 gcpy                              else:
2601 gcpy                                  if machinestate == "rapid":
2602 gcpy                                      command = "rapidY(" + str(Yp) + ")"
2603 gcpy                                      self.rapidY(Yp)
2604 gcpy                                  else:
2605 gcpy                                      command = "cutlineY(" + str(Yp) + "
                                                  )"
2606 gcpy                                      self.cutlineY(Yp)
2607 gcpy                          else:
2608 gcpy                              if Xc == 1:
2609 gcpy                                  if machinestate == "rapid":
2610 gcpy                                      command = "rapidX(" + str(Xp) + ")"
2611 gcpy                                      self.rapidX(Xp)
2612 gcpy                                  else:
2613 gcpy                                      command = "cutlineX(" + str(Xp) + "
                                                  )"
2614 gcpy                                      self.cutlineX(Xp)
2615 gcpy                      commands.append(command)
2616 gcpy #                 print(line)
2617 gcpy #                 print(command)
2618 gcpy #                 print(machinestate, Xc, Yc, Zc)
2619 gcpy #                 print(Xp, Yp, Zp)
2620 gcpy #                 print("/n")
2621 gcpy
2622 gcpy #          for command in commands:
2623 gcpy #              print(command)
2624 gcpy
2625 gcpy          show(self.stockandtoolpaths())
```

4   Notes

4.1   Other Resources

4.1.1   Coding Style

A notable influence on the coding style in this project is John Ousterhout's *A Philosophy of Software Design*[SoftwareDesign]. Complexity is managed by the overall design and structure of the code,

structuring it so that each component may be worked with on an individual basis, hiding the maximum information, and exposing the maximum functionality, with names selected so as to express their functionality/usage.

Red Flags to avoid include:

- Shallow Module

- Information Leakage

- Temporal Decomposition

- Overexposure

- Pass-Through Method

- Repetition

- Special-General Mixture

- Conjoined Methods

- Comment Repeats Code

- Implementation Documentation Contaminates Interface

- Vague Name

- Hard to Pick Name

- Hard to Describe

- Nonobvious Code

4.1.2   Coding References

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

4.1.3   Documentation Style

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

- Action (Practical) vs. Cognition (Theoretical)

- Acquisition (Studying) vs. Application (Working)

resulting in a matrix of:
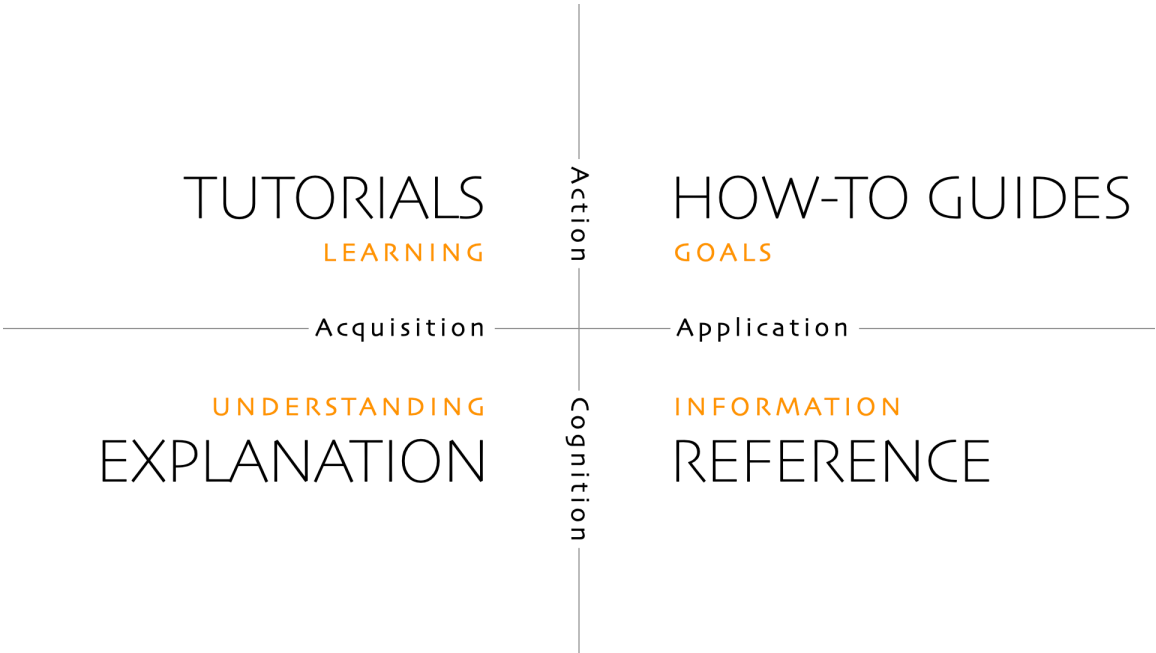
where:

1. readme.md — (Overview) Explanation (understanding-oriented)

2. Templates — Tutorials (learning-oriented)

3. gcodepreview — How-to Guides (problem-oriented)

4. Index — Reference (information-oriented)

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

4.1.4   Holidays

Holidays are from https://nationaltoday.com/

4.1.5   DXFs

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

4.2   Future

4.2.1   Images

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

4.2.2   Bézier curves in 2 dimensions

Take a Bézier curve definition and approximate it as arcs and write them into a DXF?
   https://pomax.github.io/bezierinfo/
   https://ciechanow.ski/curves-and-surfaces/
   https://www.youtube.com/watch?v=aVwxzDHniEw
   c.f., https://linuxcnc.org/docs/html/gcode/g-code.html#gcode:g5

4.2.3   Bézier curves in 3 dimensions

One question is how many Bézier curves would it be necessary to have to define a surface in 3 dimensions. Attributes for this which are desirable/necessary:

- concise — a given Bézier curve should be represented by just the point coordinates, so two on-curve points, two off-curve points, each with a pair of coordinates

- For a given shape/region it will need to be possible to have a matching definition exactly match up with it so that one could piece together a larger more complex shape from smaller/simpler regions

- similarly it will be necessary for it to be possible to sub-divide a defined region — for example it should be possible if one had 4 adjacent regions, then the four quadrants at the intersection of the four regions could be used to construct a new region — is it possible to derive a new Bézier curve from half of two other curves?

For the three planes:

- XY

- XZ

- ZY

it should be possible to have three Bézier curves (left-most/right-most or front-back or top/bottom for two, and a mid-line for the third), so a region which can be so represented would be definable by:

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

which is a marked contrast to representations such as:
   https://github.com/DavidPhillipOster/Teapot
and regions which could not be so represented could be sub-divided until the representation is workable.
   Or, it may be that fewer (only two?) curves are needed:
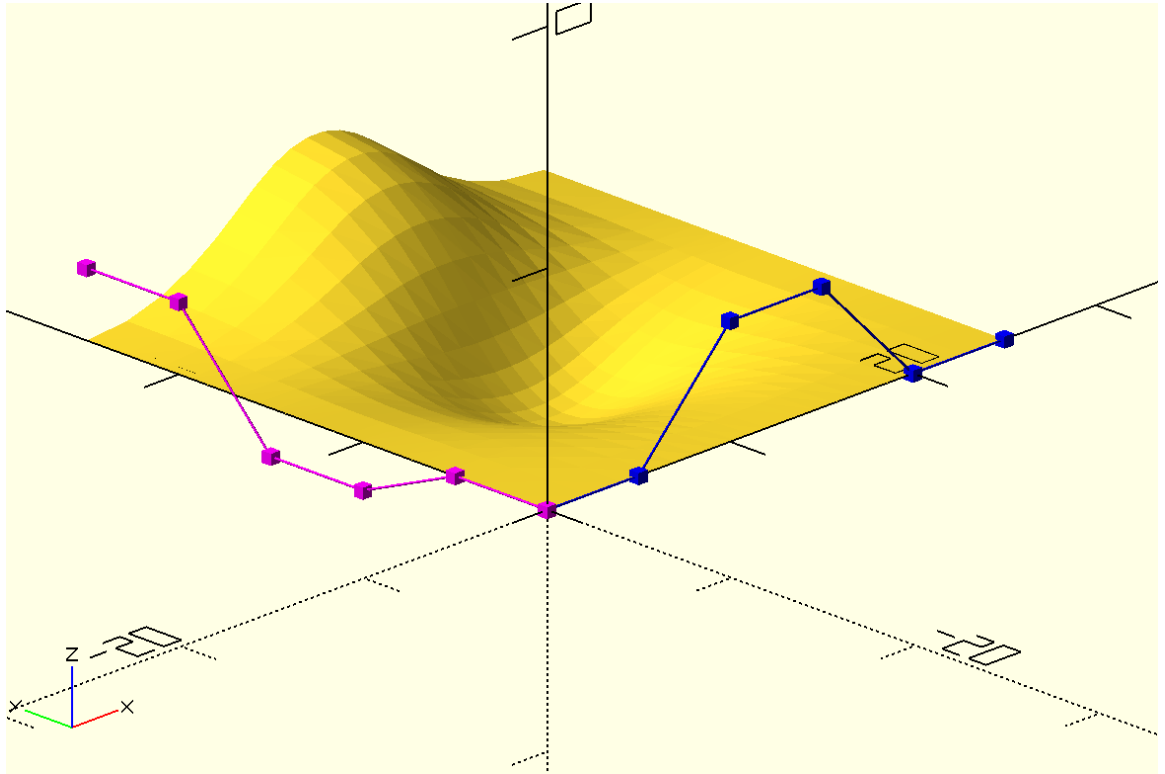
https://pages.mtu.edu/~shene/COURSES/cs3621/NOTES/notes.html
c.f., https://github.com/BelfrySCAD/BOSL2/wiki/nurbs.scad and https://old.reddit.com/r/OpenPythonSCAD/comments/1gjcz4z/pythonscad_will_get_a_new_spline_function/

4.2.4   Mathematics

https://elementsofprogramming.com/

References

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

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

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

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

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

[RS274]   Thomas R. Kramer, Frederick M. Proctor, Elena R. Messina. https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=823374 https://www.nist.gov/publications/nist-rs274ngc-interpreter-version-3

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

Command Glossary

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

Index

Routines

Variables