The gcodepreview PythonSCAD library*

Author: William F. Adams
willadams at aol dot com

2025/01/29

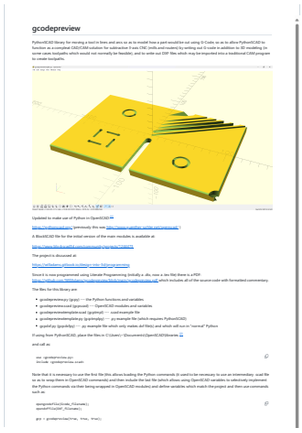
Abstract

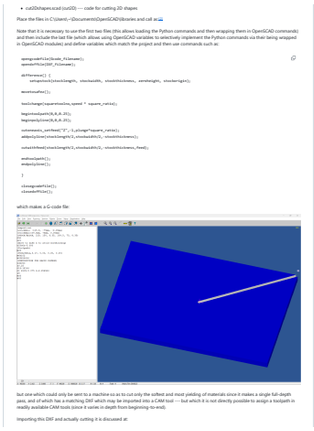
The gcodepreview library allows using PythonSCAD (OpenPythonSCAD) to move a tool in lines and arcs and output DXF and G-code files so as to work as a CAD/CAM program for CNC.

Contents

*This file (gcodepreview) has version number v0.8, last revised 2025/01/29.

1 readme.md





```
rdme # gcodepreview
rdme
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 and routers) by writing out G-code in addition to 3D modeling (in some 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.
rdme
rdme ![OpenSCAD gcodepreview Unit Tests](https://raw.githubusercontent.com/WillAdams/gcodepreview/main/gcodepreview_unittests.png?raw=true)
rdme
rdme Updated to make use of Python in OpenSCAD:[^rapcad]
rdme
rdme [^rapcad]: Previous versions had used RapCAD, so as to take advantage of the writeln command, which has since been re-written in Python.
rdme
rdme https://pythonscad.org/ (previously this was http://www.guenther-sohler.net/openscad/ )
rdme
rdme A BlockSCAD file for the initial version of the
rdme main modules is available at:
rdme
rdme https://www.blockscad3d.com/community/projects/1244473
rdme
rdme The project is discussed at:
rdme
rdme https://willadams.gitbook.io/design-into-3d/programming
rdme
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 commentary.
rdme
rdme The files for this library are:
rdme
rdme  - gcodepreview.py (gcpy) --- the Python functions and variables
rdme  - gcodepreview.scad (gcpscad) --- OpenSCAD modules and variables
rdme  - gcodepreviewtemplate.scad (gcptmpl) --- .scad example file
rdme  - gcodepreviewtemplate.py (gcptmplpy) --- .py example file (which requires PythonSCAD)
rdme  - gcpdxf.py (gcpdxfpy) --- .py example file which only makes dxf file(s) and which will run in "normal" Python
rdme
rdme If using from PythonSCAD, place the files in C:\Users\\\~\Documents\OpenSCAD\libraries [^libraries]
rdme
rdme [^libraries]: C:\Users\\\~\Documents\RapCAD\libraries is deprecated since RapCAD is no longer needed since Python is now used for writing out files.
rdme
rdme and call as:
rdme
rdme      use <gcodepreview.py>
rdme      include <gcodepreview.scad>
```

```
40 rdme
41 rdme Note that it is necessary to use the first file (this allows
          loading the Python commands (it used to be necessary to use an
          intermediary .scad file so as to wrap them in OpenSCAD 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:
42 rdme
43 rdme      opengcodefile(Gcode_filename);
44 rdme      opendxffile(DXF_filename);
45 rdme
46 rdme      gcp = gcodepreview(true, true, true);
47 rdme
48 rdme      setupstock(219, 150, 8.35, "Top", "Center");
49 rdme
50 rdme      movetosafeZ();
51 rdme
52 rdme      toolchange(102,17000);
53 rdme
54 rdme      cutline(219/2,150/2,-8.35);
55 rdme
56 rdme      stockandtoolpaths();
57 rdme
58 rdme      closegcodefile();
59 rdme      closedxffile();
60 rdme
61 rdme which makes a G-code file:
62 rdme
63 rdme ![OpenSCAD template G-code file](https://raw.githubusercontent.com/
          WillAdams/gcodepreview/main/gcodepreview_template.png?raw=true)
64 rdme
65 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 of 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).
66 rdme
67 rdme Importing this DXF and actually cutting it is discussed at:
68 rdme
69 rdme https://forum.makerforums.info/t/rewriting-gcodepreview-with-python
          /88617/14
70 rdme
71 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.
72 rdme
73 rdme Tool numbers match those of tooling sold by Carbide 3D (ob. discl.,
           I work for them).
74 rdme
75 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.
76 rdme
77 rdme Supporting OpenSCAD usage makes possible such examples as:
          openscad_gcodepreview_cutjoinery.tres.scad which is made from an
           OpenSCAD Graph Editor file:
78 rdme
79 rdme ![OpenSCAD Graph Editor Cut Joinery File](https://raw.
          githubusercontent.com/WillAdams/gcodepreview/main/
          OSGE_cutjoinery.png?raw=true)
80 rdme
81 rdme | Version       | Notes         |
82 rdme | ------------- | ------------- |
83 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. |
84 rdme |     |  - separate dxf files are written out for each tool where
          tool is ball/square/V and small/large (10/31/23)           |
85 rdme |     |  - re-writing as a Literate Program using the LaTeX package
           docmfp (begun 4/12/24)                                    |
86 rdme |     |  - support for additional tooling shapes such as dovetail
          and keyhole tools                                          |
87 rdme | 0.2 | Adds support for arcs, specialty toolpaths such as Keyhole
          which may be used for dovetail as well as keyhole cutters |
88 rdme | 0.3 | Support for curves along the 3rd dimension, roundover
```

```
          tooling                                                          |
 89 rdme | 0.4 | Rewrite using literati documentclass, suppression of SVG
          code, dxfrectangle                                               |
 90 rdme | 0.5 | More shapes, consolidate rectangles, arcs, and circles in
          gcodepreview.scad                                                |
 91 rdme | 0.6 | Notes on modules, change file for setupstock

          |
 92 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|
 93 rdme | 0.7 | Re-write completely in Python

          |
 94 rdme | 0.8 | Re-re-write completely in Python and OpenSCAD, iteratively
          testing |
 95 rdme
 96 rdme Possible future improvements:
 97 rdme
 98 rdme  - support for additional tooling shapes (bowl bits with flat
           bottom, tapered ball nose, lollipop cutters)
 99 rdme  - create a single line font for use where text is wanted
100 rdme  - Support Bézier curves (required for fonts if not to be limited
           to lines and arcs) and surfaces
101 rdme
102 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.
103 rdme
104 rdme Deprecated feature:
105 rdme
106 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)PythonSCAD, enabling the creation of 2D DXFs, G-code, 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 G-code and/or for 3D modeling)

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 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 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 tutorial) is all the documentation which most users will need (and arguably is still too much). A command glossary may be added in a future version. 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
- 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 everything.
- 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 implementing Depth per Pass (but note that this will lose the previewing of scalloped tool marks in places where they might appear otherwise)

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

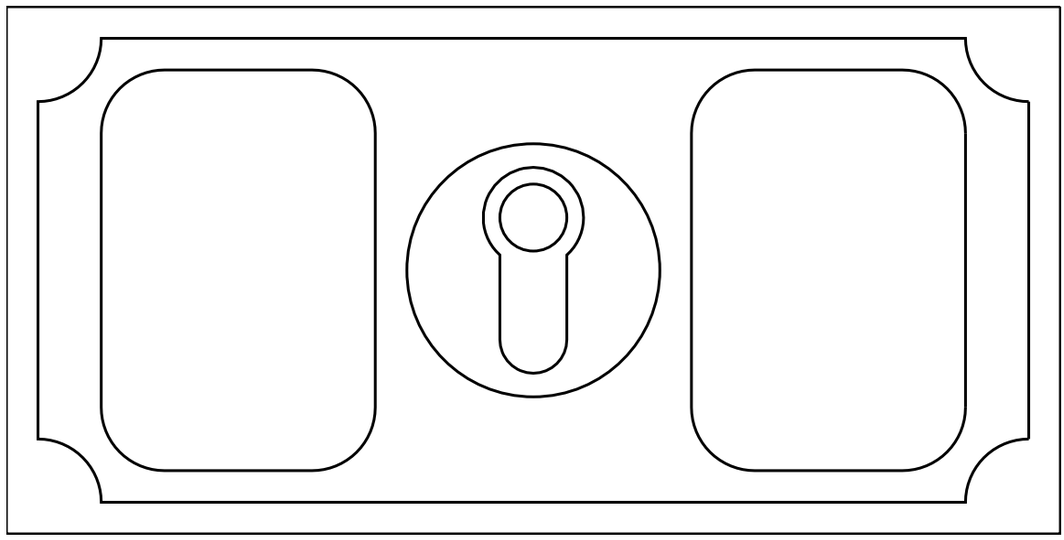
2.1 gcpdxf.py

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 `(openscad.)nimport(<URL>)` which if enabled/uncommented (and the following line commented out), will import the library from Github, sidestepping the need to download and install the library.

```
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, #generatepaths
6 gcpdxfpy                     False, #generategcode
7 gcpdxfpy                     True #generatedxf
8 gcpdxfpy                     )
9 gcpdxfpy
10 gcpdxfpy # [Stock] */
11 gcpdxfpy stockXwidth = 100
12 gcpdxfpy # [Stock] */
13 gcpdxfpy stockYheight = 50
14 gcpdxfpy
15 gcpdxfpy # [Export] */
16 gcpdxfpy Base_filename = "dxfexport"
17 gcpdxfpy
18 gcpdxfpy
19 gcpdxfpy # [CAM] */
20 gcpdxfpy large_square_tool_num = 102
21 gcpdxfpy # [CAM] */
22 gcpdxfpy small_square_tool_num = 0
23 gcpdxfpy # [CAM] */
24 gcpdxfpy large_ball_tool_num = 0
25 gcpdxfpy # [CAM] */
26 gcpdxfpy small_ball_tool_num = 0
27 gcpdxfpy # [CAM] */
28 gcpdxfpy large_V_tool_num = 0
29 gcpdxfpy # [CAM] */
30 gcpdxfpy small_V_tool_num = 0
```

```
gcpdxfpy # [CAM] */
gcpdxfpy DT_tool_num = 374
gcpdxfpy # [CAM] */
gcpdxfpy KH_tool_num = 0
gcpdxfpy # [CAM] */
gcpdxfpy Roundover_tool_num = 0
gcpdxfpy # [CAM] */
gcpdxfpy MISC_tool_num = 0
gcpdxfpy
gcpdxfpy # [Design] */
gcpdxfpy inset = 3
gcpdxfpy # [Design] */
gcpdxfpy radius = 6
gcpdxfpy # [Design] */
gcpdxfpy cornerstyle = "Fillet" # "Chamfer", "Flipped Fillet"
gcpdxfpy
gcpdxfpy gcp.opendxffile(Base_filename)
gcpdxfpy #gcp.opendxffiles(Base_filename,
gcpdxfpy #                  large_square_tool_num,
gcpdxfpy #                  small_square_tool_num,
gcpdxfpy #                  large_ball_tool_num,
gcpdxfpy #                  small_ball_tool_num,
gcpdxfpy #                  large_V_tool_num,
gcpdxfpy #                  small_V_tool_num,
gcpdxfpy #                  DT_tool_num,
gcpdxfpy #                  KH_tool_num,
gcpdxfpy #                  Roundover_tool_num,
gcpdxfpy #                  MISC_tool_num)
gcpdxfpy
gcpdxfpy gcp.dxfrectangle(large_square_tool_num, 0, 0, stockXwidth,
             stockYheight)
gcpdxfpy
gcpdxfpy gcp.dxfarc(large_square_tool_num, inset,  inset, radius,   0,  90)
gcpdxfpy gcp.dxfarc(large_square_tool_num, stockXwidth - inset,  inset,
             radius,  90, 180)
gcpdxfpy gcp.dxfarc(large_square_tool_num, stockXwidth - inset, stockYheight
              - inset, radius, 180, 270)
gcpdxfpy gcp.dxfarc(large_square_tool_num, inset, stockYheight - inset,
             radius, 270, 360)
gcpdxfpy
gcpdxfpy gcp.dxfline(large_square_tool_num, inset, inset + radius, inset,
             stockYheight - (inset + radius))
gcpdxfpy gcp.dxfline(large_square_tool_num, inset + radius, inset,
             stockXwidth - (inset + radius), inset)
gcpdxfpy gcp.dxfline(large_square_tool_num, stockXwidth - inset, inset +
             radius,stockXwidth - inset,stockYheight - (inset + radius))
gcpdxfpy gcp.dxfline(large_square_tool_num, inset + radius,stockYheight-
             inset,stockXwidth - (inset + radius),stockYheight - inset)
gcpdxfpy
gcpdxfpy gcp.dxfrectangle(large_square_tool_num, radius +inset, radius,
             stockXwidth/2 - (radius * 4), stockYheight - (radius * 2),
             cornerstyle, radius)
gcpdxfpy gcp.dxfrectangle(large_square_tool_num, stockXwidth/2 + (radius *
             2) + inset, radius, stockXwidth/2 - (radius * 4), stockYheight -
              (radius * 2), cornerstyle, radius)
gcpdxfpy #gcp.dxfrectangleround(large_square_tool_num, 64, 7, 24, 36, radius
             )
gcpdxfpy #gcp.dxfrectanglechamfer(large_square_tool_num, 64, 7, 24, 36,
             radius)
gcpdxfpy #gcp.dxfrectangleflippedfillet(large_square_tool_num, 64, 7, 24,
             36, radius)
gcpdxfpy
gcpdxfpy gcp.dxfcircle(large_square_tool_num, stockXwidth/2, stockYheight/2,
              radius * 2)
gcpdxfpy
gcpdxfpy gcp.dxfKH(374, stockXwidth/2, stockYheight/5*3, 0, -7, 270,
             11.5875)
gcpdxfpy
gcpdxfpy #gcp.closedxffiles()
gcpdxfpy gcp.closedxffile()
```

which creates:



Note that the lines referencing multiple files (`open/closedxffiles`) may be uncommented if the project wants separate `dxf` files for different tools.

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 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 generatepaths = False
20 gcptmplpy # [Export] */
21 gcptmplpy generatedxf = True
22 gcptmplpy # [Export] */
23 gcptmplpy generategcode = True
24 gcptmplpy
25 gcptmplpy # [Stock] */
26 gcptmplpy stockXwidth = 220
27 gcptmplpy # [Stock] */
28 gcptmplpy stockYheight = 150
29 gcptmplpy # [Stock] */
30 gcptmplpy stockZthickness = 8.35
31 gcptmplpy # [Stock] */
32 gcptmplpy zeroheight = "Top" # [Top, Bottom]
33 gcptmplpy # [Stock] */
```

```
gcptmplpy stockzero = "Center" # [Lower-Left, Center-Left, Top-Left, Center]
gcptmplpy # [Stock] */
gcptmplpy retractheight = 9
gcptmplpy
gcptmplpy # [CAM] */
gcptmplpy toolradius = 1.5875
gcptmplpy # [CAM] */
gcptmplpy large_square_tool_num = 201 # [0:0,112:112,102:102,201:201]
gcptmplpy # [CAM] */
gcptmplpy small_square_tool_num = 102 # [0:0,122:122,112:112,102:102]
gcptmplpy # [CAM] */
gcptmplpy large_ball_tool_num = 202 # [0:0,111:111,101:101,202:202]
gcptmplpy # [CAM] */
gcptmplpy small_ball_tool_num = 101 # [0:0,121:121,111:111,101:101]
gcptmplpy # [CAM] */
gcptmplpy large_V_tool_num = 301 # [0:0,301:301,690:690]
gcptmplpy # [CAM] */
gcptmplpy small_V_tool_num = 390 # [0:0,390:390,301:301]
gcptmplpy # [CAM] */
gcptmplpy DT_tool_num = 814 # [0:0,814:814]
gcptmplpy # [CAM] */
gcptmplpy KH_tool_num = 374 # [0:0,374:374,375:375,376:376,378]
gcptmplpy # [CAM] */
gcptmplpy Roundover_tool_num = 56142 # [56142:56142, 56125:56125, 1570:1570]
gcptmplpy # [CAM] */
gcptmplpy MISC_tool_num = 0 #
gcptmplpy
gcptmplpy # [Feeds and Speeds] */
gcptmplpy plunge = 100
gcptmplpy # [Feeds and Speeds] */
gcptmplpy feed = 400
gcptmplpy # [Feeds and Speeds] */
gcptmplpy speed = 16000
gcptmplpy # [Feeds and Speeds] */
gcptmplpy small_square_ratio = 0.75 # [0.25:2]
gcptmplpy # [Feeds and Speeds] */
gcptmplpy large_ball_ratio = 1.0 # [0.25:2]
gcptmplpy # [Feeds and Speeds] */
gcptmplpy small_ball_ratio = 0.75 # [0.25:2]
gcptmplpy # [Feeds and Speeds] */
gcptmplpy large_V_ratio = 0.875 # [0.25:2]
gcptmplpy # [Feeds and Speeds] */
gcptmplpy small_V_ratio = 0.625 # [0.25:2]
gcptmplpy # [Feeds and Speeds] */
gcptmplpy DT_ratio = 0.75 # [0.25:2]
gcptmplpy # [Feeds and Speeds] */
gcptmplpy KH_ratio = 0.75 # [0.25:2]
gcptmplpy # [Feeds and Speeds] */
gcptmplpy RO_ratio = 0.5 # [0.25:2]
gcptmplpy # [Feeds and Speeds] */
gcptmplpy MISC_ratio = 0.5 # [0.25:2]
gcptmplpy
gcptmplpy gcp = gcodepreview(generatepaths,
gcptmplpy                    generategcode,
gcptmplpy                    generatedxf,
gcptmplpy                    )
gcptmplpy
gcptmplpy gcp.opengcodefile(Base_filename)
gcptmplpy gcp.opendxffile(Base_filename)
gcptmplpy gcp.opendxffiles(Base_filename,
gcptmplpy                  large_square_tool_num,
gcptmplpy                  small_square_tool_num,
gcptmplpy                  large_ball_tool_num,
gcptmplpy                  small_ball_tool_num,
gcptmplpy                  large_V_tool_num,
gcptmplpy                  small_V_tool_num,
gcptmplpy                  DT_tool_num,
gcptmplpy                  KH_tool_num,
gcptmplpy                  Roundover_tool_num,
gcptmplpy                  MISC_tool_num)
gcptmplpy gcp.setupstock(stockXwidth,stockYheight,stockZthickness,"Top","
              Center",retractheight)
gcptmplpy
gcptmplpy #print(pygcpversion())
gcptmplpy
gcptmplpy #print(gcp.myfunc(4))
gcptmplpy
gcptmplpy #print(gcp.getvv())
```

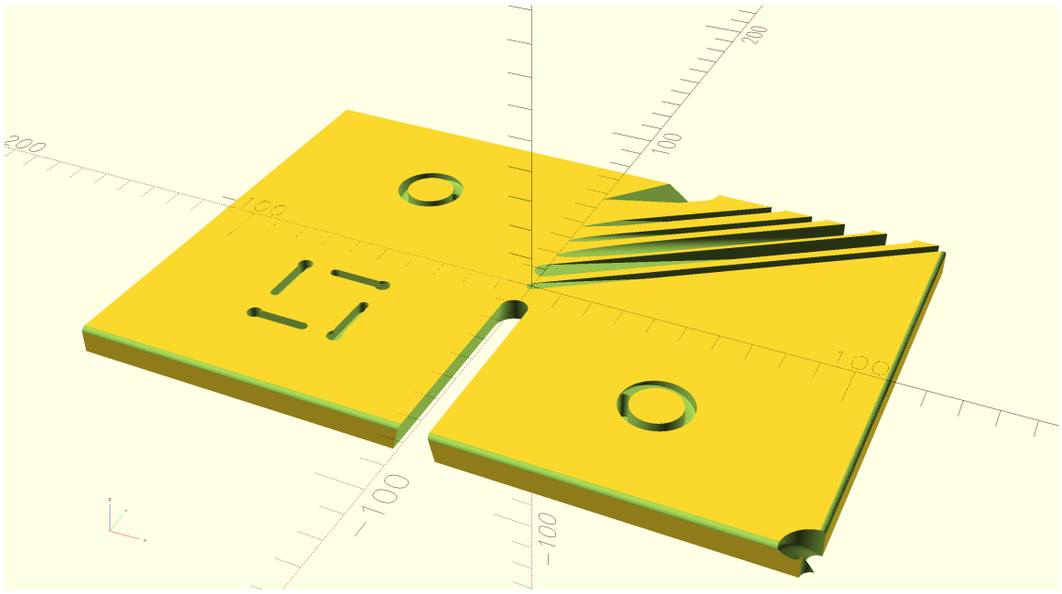
```
111 gcptmplpy
112 gcptmplpy #ts = cylinder(12.7, 1.5875, 1.5875)
113 gcptmplpy #toolpaths = gcp.cutshape(stockXwidth/2,stockYheight/2,-
                stockZthickness)
114 gcptmplpy
115 gcptmplpy gcp.movetosafeZ()
116 gcptmplpy
117 gcptmplpy gcp.toolchange(102,10000)
118 gcptmplpy
119 gcptmplpy #gcp.rapidXY(6,12)
120 gcptmplpy gcp.rapidZ(0)
121 gcptmplpy
122 gcptmplpy #print (gcp.xpos())
123 gcptmplpy #print (gcp.ypos())
124 gcptmplpy #psetzpos(7)
125 gcptmplpy #gcp.setzpos(-12)
126 gcptmplpy #print (gcp.zpos())
127 gcptmplpy
128 gcptmplpy #print ("X", str(gcp.xpos()))
129 gcptmplpy #print ("Y", str(gcp.ypos()))
130 gcptmplpy #print ("Z", str(gcp.zpos()))
131 gcptmplpy
132 gcptmplpy toolpaths = gcp.currenttool()
133 gcptmplpy
134 gcptmplpy #toolpaths = gcp.cutline(stockXwidth/2,stockYheight/2,-
                stockZthickness)
135 gcptmplpy toolpaths = toolpaths.union(gcp.cutlinedxfgc(stockXwidth/2,
                stockYheight/2, -stockZthickness))
136 gcptmplpy
137 gcptmplpy gcp.rapidZ(retractheight)
138 gcptmplpy gcp.toolchange(201,10000)
139 gcptmplpy gcp.rapidXY(0, stockYheight/16)
140 gcptmplpy gcp.rapidZ(0)
141 gcptmplpy toolpaths = toolpaths.union(gcp.cutlinedxfgc(stockXwidth/16*7,
                stockYheight/2, -stockZthickness))
142 gcptmplpy
143 gcptmplpy gcp.rapidZ(retractheight)
144 gcptmplpy gcp.toolchange(202,10000)
145 gcptmplpy gcp.rapidXY(0, stockYheight/8)
146 gcptmplpy gcp.rapidZ(0)
147 gcptmplpy toolpaths = toolpaths.union(gcp.cutlinedxfgc(stockXwidth/16*6,
                stockYheight/2, -stockZthickness))
148 gcptmplpy
149 gcptmplpy gcp.rapidZ(retractheight)
150 gcptmplpy gcp.toolchange(101,10000)
151 gcptmplpy gcp.rapidXY(0, stockYheight/16*3)
152 gcptmplpy gcp.rapidZ(0)
153 gcptmplpy toolpaths = toolpaths.union(gcp.cutlinedxfgc(stockXwidth/16*5,
                stockYheight/2, -stockZthickness))
154 gcptmplpy
155 gcptmplpy gcp.setzpos(retractheight)
156 gcptmplpy gcp.toolchange(390,10000)
157 gcptmplpy gcp.rapidXY(0, stockYheight/16*4)
158 gcptmplpy gcp.rapidZ(0)
159 gcptmplpy toolpaths = toolpaths.union(gcp.cutlinedxfgc(stockXwidth/16*4,
                stockYheight/2, -stockZthickness))
160 gcptmplpy gcp.rapidZ(retractheight)
161 gcptmplpy
162 gcptmplpy gcp.toolchange(301,10000)
163 gcptmplpy gcp.rapidXY(0, stockYheight/16*6)
164 gcptmplpy gcp.rapidZ(0)
165 gcptmplpy toolpaths = toolpaths.union(gcp.cutlinedxfgc(stockXwidth/16*2,
                stockYheight/2, -stockZthickness))
166 gcptmplpy
167 gcptmplpy rapids = gcp.rapid(gcp.xpos(),gcp.ypos(),retractheight)
168 gcptmplpy gcp.toolchange(102,10000)
169 gcptmplpy
170 gcptmplpy rapids = gcp.rapid(-stockXwidth/4+stockYheight/16, +stockYheight
                /4,0)
171 gcptmplpy
172 gcptmplpy toolpaths = toolpaths.union(gcp.cutarcCC(0,90, gcp.xpos()-
                stockYheight/16, gcp.ypos(), stockYheight/16, -stockZthickness
                /4))
173 gcptmplpy toolpaths = toolpaths.union(gcp.cutarcCC(90,180, gcp.xpos(), gcp.
                ypos()-stockYheight/16, stockYheight/16, -stockZthickness/4))
174 gcptmplpy toolpaths = toolpaths.union(gcp.cutarcCC(180,270, gcp.xpos()+
                stockYheight/16, gcp.ypos(), stockYheight/16, -stockZthickness
                /4))
```

```
175 gcptmplpy toolpaths = toolpaths.union(gcp.cutarcCC(270,360, gcp.xpos(), gcp.
                ypos()+stockYheight/16, stockYheight/16, -stockZthickness/4))
176 gcptmplpy
177 gcptmplpy rapids = gcp.movetosafeZ()
178 gcptmplpy rapids = gcp.rapidXY(stockXwidth/4-stockYheight/16, -stockYheight
                /4)
179 gcptmplpy rapids = gcp.rapidZ(0)
180 gcptmplpy
181 gcptmplpy toolpaths = toolpaths.union(gcp.cutarcCW(180,90, gcp.xpos()+
                stockYheight/16, gcp.ypos(), stockYheight/16, -stockZthickness
                /4))
182 gcptmplpy toolpaths = toolpaths.union(gcp.cutarcCW(90,0, gcp.xpos(), gcp.ypos
                ()-stockYheight/16, stockYheight/16, -stockZthickness/4))
183 gcptmplpy toolpaths = toolpaths.union(gcp.cutarcCW(360,270, gcp.xpos()-
                stockYheight/16, gcp.ypos(), stockYheight/16, -stockZthickness
                /4))
184 gcptmplpy toolpaths = toolpaths.union(gcp.cutarcCW(270,180, gcp.xpos(), gcp.
                ypos()+stockYheight/16, stockYheight/16, -stockZthickness/4))
185 gcptmplpy
186 gcptmplpy rapids = gcp.movetosafeZ()
187 gcptmplpy gcp.toolchange(201,10000)
188 gcptmplpy rapids = gcp.rapidXY(stockXwidth/2, -stockYheight/2)
189 gcptmplpy rapids = gcp.rapidZ(0)
190 gcptmplpy
191 gcptmplpy toolpaths = toolpaths.union(gcp.cutlinedxfgc(gcp.xpos(), gcp.ypos()
                , -stockZthickness))
192 gcptmplpy #test = gcp.cutlinedxfgc(gcp.xpos(), gcp.ypos(), -stockZthickness)
193 gcptmplpy
194 gcptmplpy rapids = gcp.movetosafeZ()
195 gcptmplpy rapids = gcp.rapidXY(stockXwidth/2-6.34, -stockYheight/2)
196 gcptmplpy rapids = gcp.rapidZ(0)
197 gcptmplpy
198 gcptmplpy toolpaths = toolpaths.union(gcp.cutarcCW(180,90, stockXwidth/2, -
                stockYheight/2, 6.34, -stockZthickness))
199 gcptmplpy
200 gcptmplpy rapids = gcp.movetosafeZ()
201 gcptmplpy gcp.toolchange(814,10000)
202 gcptmplpy rapids = gcp.rapidXY(0, -(stockYheight/2+12.7))
203 gcptmplpy rapids = gcp.rapidZ(0)
204 gcptmplpy
205 gcptmplpy toolpaths = toolpaths.union(gcp.cutlinedxfgc(gcp.xpos(), gcp.ypos()
                , -stockZthickness))
206 gcptmplpy toolpaths = toolpaths.union(gcp.cutlinedxfgc(gcp.xpos(), -12.7, -
                stockZthickness))
207 gcptmplpy
208 gcptmplpy rapids = gcp.rapidXY(0, -(stockYheight/2+12.7))
209 gcptmplpy rapids = gcp.movetosafeZ()
210 gcptmplpy gcp.toolchange(374,10000)
211 gcptmplpy rapids = gcp.rapidXY(stockXwidth/4-stockXwidth/16, -(stockYheight
                /4+stockYheight/16))
212 gcptmplpy rapids = gcp.rapidZ(0)
213 gcptmplpy
214 gcptmplpy gcp.rapidZ(retractheight)
215 gcptmplpy gcp.toolchange(374,10000)
216 gcptmplpy gcp.rapidXY(-stockXwidth/4-stockXwidth/16, -(stockYheight/4+
                stockYheight/16))
217 gcptmplpy gcp.rapidZ(0)
218 gcptmplpy
219 gcptmplpy toolpaths = toolpaths.union(gcp.cutline(gcp.xpos(), gcp.ypos(), -
                stockZthickness/2))
220 gcptmplpy toolpaths = toolpaths.union(gcp.cutlinedxfgc(gcp.xpos()+
                stockYheight/9, gcp.ypos(), gcp.zpos()))
221 gcptmplpy #below should probably be cutlinegc
222 gcptmplpy toolpaths = toolpaths.union(gcp.cutline(gcp.xpos()-stockYheight/9,
                gcp.ypos(), gcp.zpos()))
223 gcptmplpy toolpaths = toolpaths.union(gcp.cutline(gcp.xpos(), gcp.ypos(), 0))
224 gcptmplpy
225 gcptmplpy #key = gcp.cutkeyholegcdxf(KH_tool_num, 0, stockZthickness*0.75, "E
                ", stockYheight/9)
226 gcptmplpy #key = gcp.cutKHgcdxf(374, 0, stockZthickness*0.75, 90,
                stockYheight/9)
227 gcptmplpy #toolpaths = toolpaths.union(key)
228 gcptmplpy
229 gcptmplpy gcp.rapidZ(retractheight)
230 gcptmplpy gcp.rapidXY(-stockXwidth/4+stockXwidth/16, -(stockYheight/4+
                stockYheight/16))
231 gcptmplpy gcp.rapidZ(0)
232 gcptmplpy #toolpaths = toolpaths.union(gcp.cutkeyholegcdxf(KH_tool_num, 0,
```

```
                stockZthickness*0.75, "N", stockYheight/9))
233 gcptmplpy toolpaths = toolpaths.union(gcp.cutline(gcp.xpos(), gcp.ypos(), -
                stockZthickness/2))
234 gcptmplpy toolpaths = toolpaths.union(gcp.cutlinedxfgc(gcp.xpos(), gcp.ypos()
                +stockYheight/9, gcp.zpos()))
235 gcptmplpy #below should probably be cutlinegc
236 gcptmplpy toolpaths = toolpaths.union(gcp.cutline(gcp.xpos(), gcp.ypos()-
                stockYheight/9, gcp.zpos()))
237 gcptmplpy toolpaths = toolpaths.union(gcp.cutline(gcp.xpos(), gcp.ypos(), 0))
238 gcptmplpy
239 gcptmplpy gcp.rapidZ(retracttheight)
240 gcptmplpy gcp.rapidXY(-stockXwidth/4+stockXwidth/16, -(stockYheight/4-
                stockYheight/8))
241 gcptmplpy gcp.rapidZ(0)
242 gcptmplpy #toolpaths = toolpaths.union(gcp.cutkeyholegcdxf(KH_tool_num, 0,
                stockZthickness*0.75, "W", stockYheight/9))
243 gcptmplpy toolpaths = toolpaths.union(gcp.cutline(gcp.xpos(), gcp.ypos(), -
                stockZthickness/2))
244 gcptmplpy toolpaths = toolpaths.union(gcp.cutlinedxfgc(gcp.xpos()-
                stockYheight/9, gcp.ypos(), gcp.zpos()))
245 gcptmplpy #below should probably be cutlinegc
246 gcptmplpy toolpaths = toolpaths.union(gcp.cutline(gcp.xpos()+stockYheight/9,
                gcp.ypos(), gcp.zpos()))
247 gcptmplpy toolpaths = toolpaths.union(gcp.cutline(gcp.xpos(), gcp.ypos(), 0))
248 gcptmplpy
249 gcptmplpy gcp.rapidZ(retracttheight)
250 gcptmplpy gcp.rapidXY(-stockXwidth/4-stockXwidth/16, -(stockYheight/4-
                stockYheight/8))
251 gcptmplpy gcp.rapidZ(0)
252 gcptmplpy #toolpaths = toolpaths.union(gcp.cutkeyholegcdxf(KH_tool_num, 0,
                stockZthickness*0.75, "S", stockYheight/9))
253 gcptmplpy toolpaths = toolpaths.union(gcp.cutline(gcp.xpos(), gcp.ypos(), -
                stockZthickness/2))
254 gcptmplpy toolpaths = toolpaths.union(gcp.cutlinedxfgc(gcp.xpos(), gcp.ypos()
                -stockYheight/9, gcp.zpos()))
255 gcptmplpy #below should probably be cutlinegc
256 gcptmplpy toolpaths = toolpaths.union(gcp.cutline(gcp.xpos(), gcp.ypos()+
                stockYheight/9, gcp.zpos()))
257 gcptmplpy toolpaths = toolpaths.union(gcp.cutline(gcp.xpos(), gcp.ypos(), 0))
258 gcptmplpy
259 gcptmplpy gcp.rapidZ(retracttheight)
260 gcptmplpy gcp.toolchange(56142,10000)
261 gcptmplpy gcp.rapidXY(-stockXwidth/2, -(stockYheight/2+0.508/2))
262 gcptmplpy #gcp.cutZgcfeed(-1.531,plunge)
263 gcptmplpy toolpaths = toolpaths.union(gcp.cutline(gcp.xpos(), gcp.ypos(),
                -1.531))
264 gcptmplpy toolpaths = toolpaths.union(gcp.cutlinedxfgc(stockXwidth/2+0.508/2,
                 -(stockYheight/2+0.508/2), -1.531))
265 gcptmplpy
266 gcptmplpy gcp.rapidZ(retracttheight)
267 gcptmplpy #gcp.toolchange(56125,10000)
268 gcptmplpy #gcp.cutZgcfeed(-1.531,plunge)
269 gcptmplpy toolpaths = toolpaths.union(gcp.cutline(gcp.xpos(), gcp.ypos(),
                -1.531))
270 gcptmplpy toolpaths = toolpaths.union(gcp.cutlinedxfgc(stockXwidth/2+0.508/2,
                 (stockYheight/2+0.508/2), -1.531))
271 gcptmplpy
272 gcptmplpy
273 gcptmplpy part = gcp.stock.difference(toolpaths)
274 gcptmplpy
275 gcptmplpy output (part)
276 gcptmplpy #output(test)
277 gcptmplpy #output (key)
278 gcptmplpy #output(dt)
279 gcptmplpy #gcp.stockandtoolpaths()
280 gcptmplpy #gcp.stockandtoolpaths("stock")
281 gcptmplpy #output (gcp.stock)
282 gcptmplpy #output (gcp.toolpaths)
283 gcptmplpy #output (toolpaths)
284 gcptmplpy
285 gcptmplpy #gcp.makecube(3, 2, 1)
286 gcptmplpy #
287 gcptmplpy #gcp.placecube()
288 gcptmplpy #
289 gcptmplpy #c = gcp.instantiatecube()
290 gcptmplpy #
291 gcptmplpy #output(c)
292 gcptmplpy
```

```
293 gcptmplpy gcp.closegcodefile()
294 gcptmplpy gcp.closedxffiles()
295 gcptmplpy gcp.closedxffile()
```

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



2.3 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. A further consideration worth noting is that when called from OpenSCAD, Python will not halt for errors, but will run through to the end which is an expedient thing for viewing the end result of in-process code.

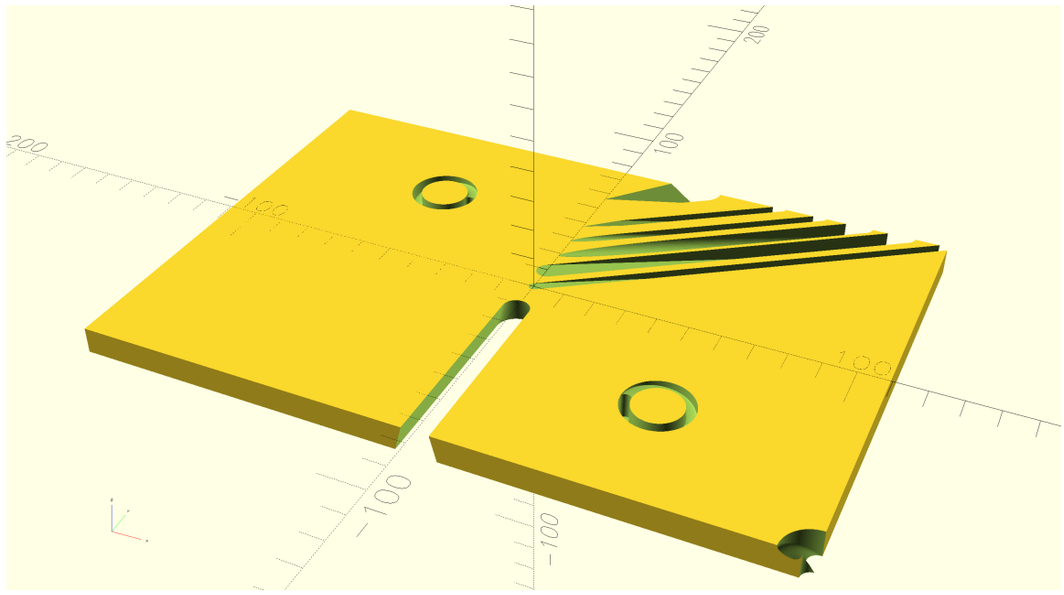
```
 1 gcptmpl //!OpenSCAD
 2 gcptmpl
 3 gcptmpl use <gcodepreview.py>
 4 gcptmpl include <gcodepreview.scad>
 5 gcptmpl
 6 gcptmpl $fa = 2;
 7 gcptmpl $fs = 0.125;
 8 gcptmpl fa = 2;
 9 gcptmpl fs = 0.125;
10 gcptmpl
11 gcptmpl /* [Stock] */
12 gcptmpl stockXwidth = 219;
13 gcptmpl /* [Stock] */
14 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 generatepaths = true;
28 gcptmpl /* [Export] */
29 gcptmpl generatedxf = true;
30 gcptmpl /* [Export] */
31 gcptmpl generategcode = true;
32 gcptmpl
33 gcptmpl /* [CAM] */
```

```
gcptmpl toolradius = 1.5875;
gcptmpl /* [CAM] */
gcptmpl large_square_tool_num = 0; // [0:0,112:112,102:102,201:201]
gcptmpl /* [CAM] */
gcptmpl small_square_tool_num = 102; // [0:0,122:122,112:112,102:102]
gcptmpl /* [CAM] */
gcptmpl large_ball_tool_num = 0; // [0:0,111:111,101:101,202:202]
gcptmpl /* [CAM] */
gcptmpl small_ball_tool_num = 0; // [0:0,121:121,111:111,101:101]
gcptmpl /* [CAM] */
gcptmpl large_V_tool_num = 0; // [0:0,301:301,690:690]
gcptmpl /* [CAM] */
gcptmpl small_V_tool_num = 0; // [0:0,390:390,301:301]
gcptmpl /* [CAM] */
gcptmpl DT_tool_num = 0; // [0:0,814:814]
gcptmpl /* [CAM] */
gcptmpl KH_tool_num = 0; // [0:0,374:374,375:375,376:376,378]
gcptmpl /* [CAM] */
gcptmpl Roundover_tool_num = 0; // [56142:56142, 56125:56125, 1570:1570]
gcptmpl /* [CAM] */
gcptmpl MISC_tool_num = 0; // [648:648]
gcptmpl
gcptmpl /* [Feeds and Speeds] */
gcptmpl plunge = 100;
gcptmpl /* [Feeds and Speeds] */
gcptmpl feed = 400;
gcptmpl /* [Feeds and Speeds] */
gcptmpl speed = 16000;
gcptmpl /* [Feeds and Speeds] */
gcptmpl small_square_ratio = 0.75; // [0.25:2]
gcptmpl /* [Feeds and Speeds] */
gcptmpl large_ball_ratio = 1.0; // [0.25:2]
gcptmpl /* [Feeds and Speeds] */
gcptmpl small_ball_ratio = 0.75; // [0.25:2]
gcptmpl /* [Feeds and Speeds] */
gcptmpl large_V_ratio = 0.875; // [0.25:2]
gcptmpl /* [Feeds and Speeds] */
gcptmpl small_V_ratio = 0.625; // [0.25:2]
gcptmpl /* [Feeds and Speeds] */
gcptmpl DT_ratio = 0.75; // [0.25:2]
gcptmpl /* [Feeds and Speeds] */
gcptmpl KH_ratio = 0.75; // [0.25:2]
gcptmpl /* [Feeds and Speeds] */
gcptmpl RO_ratio = 0.5; // [0.25:2]
gcptmpl /* [Feeds and Speeds] */
gcptmpl MISC_ratio = 0.5; // [0.25:2]
gcptmpl
gcptmpl thegeneratepaths = generatepaths == true ? 1 : 0;
gcptmpl thegeneratedxf = generatedxf == true ? 1 : 0;
gcptmpl thegenerategcode = generategcode == true ? 1 : 0;
gcptmpl
gcptmpl gcp = gcodepreview(thegeneratepaths,
gcptmpl                    thegenerategcode,
gcptmpl                    thegeneratedxf,
gcptmpl                    );
gcptmpl
gcptmpl opengcodefile(Base_filename);
gcptmpl opendxffile(Base_filename);
gcptmpl opendxffiles(Base_filename,
gcptmpl                  large_square_tool_num,
gcptmpl                  small_square_tool_num,
gcptmpl                  large_ball_tool_num,
gcptmpl                  small_ball_tool_num,
gcptmpl                  large_V_tool_num,
gcptmpl                  small_V_tool_num,
gcptmpl                  DT_tool_num,
gcptmpl                  KH_tool_num,
gcptmpl                  Roundover_tool_num,
gcptmpl                  MISC_tool_num);
gcptmpl
gcptmpl setupstock(stockXwidth, stockYheight, stockZthickness, zeroheight,
        stockzero);
gcptmpl
gcptmpl //echo(gcp);
gcptmpl //gcpversion();
gcptmpl
gcptmpl //c = myfunc(4);
gcptmpl //echo(c);
```

```
111 gcptmpl
112 gcptmpl //echo(getvv());
113 gcptmpl
114 gcptmpl cutline(stockXwidth/2,stockYheight/2,-stockZthickness);
115 gcptmpl
116 gcptmpl rapidZ(retractheight);
117 gcptmpl toolchange(201,10000);
118 gcptmpl rapidXY(0, stockYheight/16);
119 gcptmpl rapidZ(0);
120 gcptmpl cutlinedxfgc(stockXwidth/16*7, stockYheight/2, -stockZthickness);
121 gcptmpl
122 gcptmpl
123 gcptmpl rapidZ(retractheight);
124 gcptmpl toolchange(202,10000);
125 gcptmpl rapidXY(0, stockYheight/8);
126 gcptmpl rapidZ(0);
127 gcptmpl cutlinedxfgc(stockXwidth/16*6, stockYheight/2, -stockZthickness);
128 gcptmpl
129 gcptmpl rapidZ(retractheight);
130 gcptmpl toolchange(101,10000);
131 gcptmpl rapidXY(0, stockYheight/16*3);
132 gcptmpl rapidZ(0);
133 gcptmpl cutlinedxfgc(stockXwidth/16*5, stockYheight/2, -stockZthickness);
134 gcptmpl
135 gcptmpl rapidZ(retractheight);
136 gcptmpl toolchange(390,10000);
137 gcptmpl rapidXY(0, stockYheight/16*4);
138 gcptmpl rapidZ(0);
139 gcptmpl
140 gcptmpl cutlinedxfgc(stockXwidth/16*4, stockYheight/2, -stockZthickness);
141 gcptmpl rapidZ(retractheight);
142 gcptmpl
143 gcptmpl toolchange(301,10000);
144 gcptmpl rapidXY(0, stockYheight/16*6);
145 gcptmpl rapidZ(0);
146 gcptmpl
147 gcptmpl cutlinedxfgc(stockXwidth/16*2, stockYheight/2, -stockZthickness);
148 gcptmpl
149 gcptmpl
150 gcptmpl movetosafeZ();
151 gcptmpl rapid(gcp.xpos(),gcp.ypos(),retractheight);
152 gcptmpl toolchange(102,10000);
153 gcptmpl
154 gcptmpl //rapidXY(stockXwidth/4+stockYheight/8+stockYheight/16, +
              stockYheight/8);
155 gcptmpl rapidXY(-stockXwidth/4+stockXwidth/16, (stockYheight/4));//+
              stockYheight/16
156 gcptmpl rapidZ(0);
157 gcptmpl
158 gcptmpl //cutarcCW(360,270, gcp.xpos()-stockYheight/16, gcp.ypos(),
              stockYheight/16,-stockZthickness);
159 gcptmpl //gcp.cutarcCW(270,180, gcp.xpos(), gcp.ypos()+stockYheight/16,
              stockYheight/16))
160 gcptmpl cutarcCC(0,90, gcp.xpos()-stockYheight/16, gcp.ypos(), stockYheight
              /16, -stockZthickness/4);
161 gcptmpl cutarcCC(90,180, gcp.xpos(), gcp.ypos()-stockYheight/16,
              stockYheight/16, -stockZthickness/4);
162 gcptmpl cutarcCC(180,270, gcp.xpos()+stockYheight/16, gcp.ypos(),
              stockYheight/16, -stockZthickness/4);
163 gcptmpl cutarcCC(270,360, gcp.xpos(), gcp.ypos()+stockYheight/16,
              stockYheight/16, -stockZthickness/4);
164 gcptmpl
165 gcptmpl movetosafeZ();
166 gcptmpl //rapidXY(stockXwidth/4+stockYheight/8-stockYheight/16, -
              stockYheight/8);
167 gcptmpl rapidXY(stockXwidth/4-stockYheight/16, -(stockYheight/4));
168 gcptmpl rapidZ(0);
169 gcptmpl
170 gcptmpl cutarcCW(180,90, gcp.xpos()+stockYheight/16, gcp.ypos(),
              stockYheight/16, -stockZthickness/4);
171 gcptmpl cutarcCW(90,0, gcp.xpos(), gcp.ypos()-stockYheight/16, stockYheight
              /16, -stockZthickness/4);
172 gcptmpl cutarcCW(360,270, gcp.xpos()-stockYheight/16, gcp.ypos(),
              stockYheight/16, -stockZthickness/4);
173 gcptmpl cutarcCW(270,180, gcp.xpos(), gcp.ypos()+stockYheight/16,
              stockYheight/16, -stockZthickness/4);
174 gcptmpl
175 gcptmpl movetosafeZ();
```

```
176 gcptmpl toolchange(201, 10000);
177 gcptmpl rapidXY(stockXwidth /2 -6.34, - stockYheight /2);
178 gcptmpl rapidZ(0);
179 gcptmpl cutarcCW(180, 90, stockXwidth /2 , -stockYheight/2, 6.34, -
            stockZthickness);
180 gcptmpl
181 gcptmpl movetosafeZ();
182 gcptmpl rapidXY(stockXwidth/2, -stockYheight/2);
183 gcptmpl rapidZ(0);
184 gcptmpl
185 gcptmpl gcp.cutlinedxfgc(gcp.xpos(), gcp.ypos(), -stockZthickness);
186 gcptmpl
187 gcptmpl movetosafeZ();
188 gcptmpl toolchange(814, 10000);
189 gcptmpl rapidXY(0, -(stockYheight/2+12.7));
190 gcptmpl rapidZ(0);
191 gcptmpl
192 gcptmpl cutlinedxfgc(xpos(), ypos(), -stockZthickness);
193 gcptmpl cutlinedxfgc(xpos(), -12.7 , -stockZthickness);
194 gcptmpl rapidXY(0, -(stockYheight/2+12.7));
195 gcptmpl
196 gcptmpl //rapidXY(stockXwidth/2-6.34, -stockYheight/2);
197 gcptmpl //rapidZ(0);
198 gcptmpl
199 gcptmpl //movetosafeZ();
200 gcptmpl //toolchange(374, 10000);
201 gcptmpl //rapidXY(-(stockXwidth/4 - stockXwidth /16), -(stockYheight/4 +
            stockYheight/16))
202 gcptmpl
203 gcptmpl //cutline(xpos(), ypos(), (stockZthickness/2) * -1);
204 gcptmpl //cutlinedxfgc(xpos() + stockYheight /9, ypos(), zpos());
205 gcptmpl //cutline(xpos() - stockYheight /9, ypos(), zpos());
206 gcptmpl //cutline(xpos(), ypos(), 0);
207 gcptmpl
208 gcptmpl movetosafeZ();
209 gcptmpl
210 gcptmpl stockandtoolpaths();
211 gcptmpl //stockwotoolpaths();
212 gcptmpl //outputtoolpaths();
213 gcptmpl
214 gcptmpl //makecube(3, 2, 1);
215 gcptmpl
216 gcptmpl //instantiatecube();
217 gcptmpl
218 gcptmpl closegcodefile();
219 gcptmpl closedxffiles();
220 gcptmpl closedxffile();
```

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



Note that there are 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):

- `generatepaths = true` — this has the Python code collect toolpath cuts and rapid movements in variables which are then instantiated by appropriate commands/options (shown in the OpenSCAD template `gcodepreview.scad`)
- `generatepaths = false` — this option affords the user control over how the model elements are handled (shown in the Python template `gcodepreview.py`)

the templates set up these options as noted, and for OpenSCAD, implement code to ensure that `True == true`, and a set of commands are provided to output the stock, toolpaths, or part (toolpaths and rapids differenced from stock).

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 4^{th} additional axis may be worked up as a future option) 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 requires two files:

- A Python file: gcodepreview.py (`gcpy`) — this has variables in the traditional sense which may be 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
- An OpenSCAD file: gcodepreview.scad (`gcpscad`) — which `use`s the Python file and which is `include`d 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.

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 version of a command.

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.

Tool #s where used will always be the first argument — this makes it obvious if they are not used — the negative, that it then doesn't allow for a usage where a DEFAULT tool is used is not an issue since the command `currenttoolnum()` 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 DXF file), filetype)
- Prefixes
  - `generate` (Boolean) — used to identify which types of actions will be done
  - `write` (action) — used to write to files
  - `cut` (action — create 3D object)
  - `rapid` (action — create 3D object so as to show a collision)
  - `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
  - 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, making it straightforward 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 unlikely to be needed, and may not be implemented. 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.

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

Another consideration is that all commands which write files will check to see if a given filetype is enabled or no.

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)
- 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 after v0.6 (see below).

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-argumen

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 after. See `stepsizearc` and `stepsizeroundover`.

stepsizearc
stepsizeroundover

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:

```
gcpy #!/usr/bin/env python
gcpy #icon "C:\Program Files\PythonSCAD\bin\openscad.exe"  --trust-python
gcpy #Currently tested with PythonSCAD_nolibfive-2025.01.02-x86-64-Installer.exe and Python 3.11
gcpy #gcodepreview 0.8, for use with PythonSCAD,
gcpy #if using from PythonSCAD using OpenSCAD code, see gcodepreview.scad
gcpy
gcpy import sys
gcpy
gcpy # getting openscad functions into namespace
gcpy #https://github.com/gsohler/openscad/issues/39
gcpy try:
gcpy     from openscad import *
gcpy except ModuleNotFoundError as e:
gcpy     print("OpenSCAD module not loaded.")
gcpy
gcpy # add math functions (using radians by default, convert to degrees where necessary)
gcpy import math
gcpy
gcpy def pygcpversion():
gcpy     thegcpversion = 0.8
gcpy     return thegcpversion
```

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

```
gcpscad //!OpenSCAD
gcpscad
gcpscad //gcodepreview version 0.8
gcpscad //
gcpscad //used via include <gcodepreview.scad>;
gcpscad //
gcpscad
gcpscad use <gcodepreview.py>
gcpscad
gcpscad module gcpversion(){
gcpscad echo(pygcpversion());
gcpscad }
gcpscad
gcpscad //function myfunc(var) = gcp.myfunc(var);
gcpscad //
gcpscad //function getvv() = gcp.getvv();
gcpscad //
gcpscad //module makecube(xdim, ydim, zdim){
gcpscad //gcp.makecube(xdim, ydim, zdim);
gcpscad //}
gcpscad //
gcpscad //module placecube(){
gcpscad //gcp.placecube();
gcpscad //}
gcpscad //
gcpscad //module instantiatecube(){
gcpscad //gcp.instantiatecube();
gcpscad //}
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 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, generatepaths, generatedxf, generategcode
  - largesquaretool:
    - large_square_tool_num, toolradius, plunge, feed, speed
- 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 init` The class which is defined is `gcodepreview` which includes the `init` method which allows passing in and defining the variables which will be used by the other methods in this class.

```
23 gcpy class gcodepreview:
24 gcpy
25 gcpy     def __init__(self, #basefilename = "export",
26 gcpy                    generatepaths = False,
27 gcpy                    generategcode = False,
28 gcpy                    generatedxf = False,
29 gcpy #                   stockXwidth = 25,
30 gcpy #                   stockYheight = 25,
31 gcpy #                   stockZthickness = 1,
32 gcpy #                   zeroheight = "Top",
33 gcpy #                   stockzero = "Lower-left" ,
34 gcpy #                   retractheight = 6,
35 gcpy #                   currenttoolnum = 102,
36 gcpy #                   toolradius = 3.175,
37 gcpy #                   plunge = 100,
38 gcpy #                   feed = 400,
39 gcpy #                   speed = 10000
40 gcpy                    ):
41 gcpy #         self.basefilename = basefilename
42 gcpy         if (generatepaths == 1):
43 gcpy             self.generatepaths = True
44 gcpy         if (generatepaths == 0):
45 gcpy             self.generatepaths = False
46 gcpy         else:
47 gcpy             self.generatepaths = generatepaths
48 gcpy         if (generategcode == 1):
49 gcpy             self.generategcode = True
50 gcpy         if (generategcode == 0):
51 gcpy             self.generategcode = False
52 gcpy         else:
53 gcpy             self.generategcode = generategcode
54 gcpy         if (generatedxf == 1):
55 gcpy             self.generatedxf = True
56 gcpy         if (generatedxf == 0):
57 gcpy             self.generatedxf = False
58 gcpy         else:
59 gcpy             self.generatedxf = generatedxf
60 gcpy #         self.stockXwidth = stockXwidth
61 gcpy #         self.stockYheight = stockYheight
62 gcpy #         self.stockZthickness = stockZthickness
63 gcpy #         self.zeroheight = zeroheight
64 gcpy #         self.stockzero = stockzero
65 gcpy #         self.retractheight = retractheight
66 gcpy #         self.currenttoolnum = currenttoolnum
67 gcpy #         self.toolradius = toolradius
68 gcpy #         self.plunge = plunge
69 gcpy #         self.feed = feed
70 gcpy #         self.speed = speed
71 gcpy #         global toolpaths
72 gcpy #         if (openscadloaded == True):
73 gcpy #             self.toolpaths = cylinder(0.1, 0.1)
74 gcpy         self.generatedxfs = False
75 gcpy
76 gcpy     def checkgeneratepaths():
77 gcpy         return self.generatepaths
78 gcpy
79 gcpy #     def myfunc(self, var):
80 gcpy #         self.vv = var * var
81 gcpy #         return self.vv
82 gcpy #
83 gcpy #     def getvv(self):
84 gcpy #         return self.vv
85 gcpy #
86 gcpy #     def checkint(self):
87 gcpy #         return self.mc
88 gcpy #
89 gcpy #     def makecube(self, xdim, ydim, zdim):
90 gcpy #         self.c=cube([xdim, ydim, zdim])
91 gcpy #
92 gcpy #     def placecube(self):
93 gcpy #         output(self.c)
94 gcpy #
95 gcpy #     def instantiatecube(self):
96 gcpy #         return self.c
97 gcpy #
```

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, current tool, and the current depth in the current toolpath. This will be done using paired functions (which will set and return the matching variable) and a matching variable.

The first such variables are for XYZ position:

mpx
- `mpx`

mpy
- `mpy`

mpz
- `mpz`

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

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.

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

```
 97 gcpy      def xpos(self):
 98 gcpy  #         global mpx
 99 gcpy          return self.mpx
100 gcpy
101 gcpy      def ypos(self):
102 gcpy  #         global mpy
103 gcpy          return self.mpy
104 gcpy
105 gcpy      def zpos(self):
106 gcpy  #         global mpz
107 gcpy          return self.mpz
108 gcpy
109 gcpy  #     def tpzinc(self):
110 gcpy  #         global tpzinc
111 gcpy  #         return self.tpzinc
```

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

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

setxpos
setypos
setzpos

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

```
113 gcpy      def setxpos(self, newxpos):
114 gcpy  #         global mpx
115 gcpy          self.mpx = newxpos
116 gcpy
117 gcpy      def setypos(self, newypos):
118 gcpy  #         global mpy
119 gcpy          self.mpy = newypos
120 gcpy
121 gcpy      def setzpos(self, newzpos):
122 gcpy  #         global mpz
123 gcpy          self.mpz = newzpos
124 gcpy
125 gcpy  #     def settpzinc(self, newtpzinc):
126 gcpy  #         global tpzinc
127 gcpy  #         self.tpzinc = newtpzinc
```

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

gcodepreview
setupstock

The first such routine, (actually a subroutine, see `gcodepreview`) `setupstock` will be 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, 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.

```
129 gcpy     def setupstock(self, stockXwidth,
130 gcpy                    stockYheight,
131 gcpy                    stockZthickness,
132 gcpy                    zeroheight,
133 gcpy                    stockzero,
134 gcpy                    retracttheight):
135 gcpy         self.stockXwidth = stockXwidth
136 gcpy         self.stockYheight = stockYheight
137 gcpy         self.stockZthickness = stockZthickness
138 gcpy         self.zeroheight = zeroheight
139 gcpy         self.stockzero = stockzero
140 gcpy         self.retracttheight = retracttheight
141 gcpy #        global mpx
142 gcpy         self.mpx = float(0)
143 gcpy #        global mpy
144 gcpy         self.mpy = float(0)
145 gcpy #        global mpz
146 gcpy         self.mpz = float(0)
147 gcpy #        global tpz
148 gcpy #        self.tpzinc = float(0)
149 gcpy #        global currenttoolnum
150 gcpy         self.currenttoolnum = 102
151 gcpy #        global currenttoolshape
152 gcpy         self.currenttoolshape = cylinder(12.7, 1.5875)
153 gcpy         self.rapids = self.currenttoolshape
154 gcpy #        global stock
155 gcpy         self.stock = cube([stockXwidth, stockYheight,
                     stockZthickness])
156 gcpy #%WRITEGC        if self.generategcode == True:
157 gcpy #%WRITEGC            self.writegc("(Design File: " + self.
                 basefilename + ")")
158 gcpy         self.toolpaths = cylinder(0.1, 0.1)
```

Note that since Python in PythonSCAD 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 internal variable `stockzero` is used in an <`if then else`> structure to position the 3D model of the stock and write out the G-code comment which describes it in comments using the terms described for CutViewer.

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

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

```
                                    ",str(self.stockZthickness),", ",str(self.
                                    stockXwidth/2),", str(self.stockYheight
                                    /2),", 0.00)")
209 gcpy          if self.generategcode == True:
210 gcpy              self.writegc("G90");
211 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 OpenSCAD version is simply a descriptor:

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

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

```
setupstock(stockXwidth, stockYheight, stockZthickness, zeroheight, stockzero)

cy = cube([1,2,stockZthickness*2])

diff = stock.difference(cy)
#output(diff)
diff.show()
```

3.3 Tools and Changes

currenttoolnumber
settool

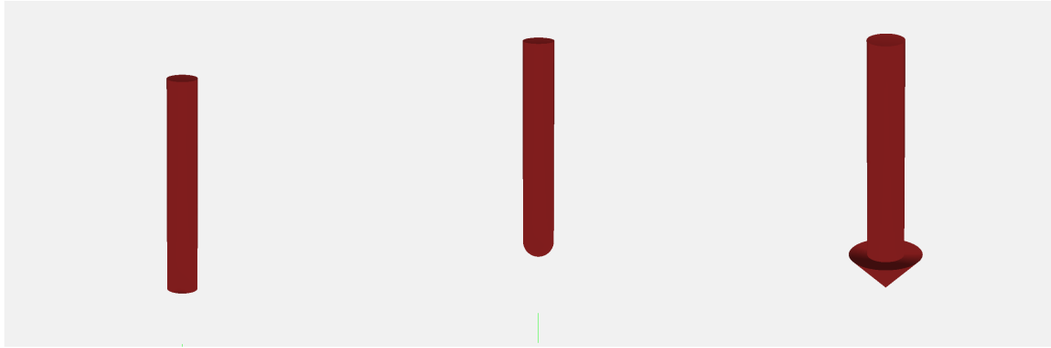
Similarly Python functions and variables will be used in: `currenttoolnumber` (note that it is important to use a different name than the variable `currenttoolnum` and `settool` to track and set and return the current tool:

```
213 gcpy      def settool(self,tn):
214 gcpy #         global currenttoolnum
215 gcpy          self.currenttoolnum = tn
216 gcpy
217 gcpy      def currenttoolnumber(self):
218 gcpy #         global currenttoolnum
219 gcpy          return self.currenttoolnum
220 gcpy
221 gcpy #     def currentroundovertoolnumber(self):
222 gcpy #         global Roundover_tool_num
223 gcpy #         return self.Roundover_tool_num
```

3.3.1 3D Shapes for Tools

Each tool must be modeled in 3D using an OpenSCAD module.

3.3.1.1 Normal Tooling/toolshapes Most tooling has quite standard shapes and are defined by their profile:



- 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 (a radiused form with a flat bottom, often described as a "bowl bit" is not implemented as-of-yet)
- 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 commonly available radiused form is not implemented at this time, *e.g.*, #501 and 502)

Most tools are easily implemented with concise 3D descriptions which may be connected with a simple `hull` operation:

`endmill square` The `endmill square` is a simple cylinder:

```
225 gcpy     def endmill_square(self, es_diameter, es_flute_length):
226 gcpy         return cylinder(r1=(es_diameter / 2), r2=(es_diameter / 2),
                     h=es_flute_length, center = False)
```

`ballnose` The `ballnose` is modeled as a hemisphere joined with a cylinder:

```
228 gcpy     def ballnose(self, es_diameter, es_flute_length):
229 gcpy         b = sphere(r=(es_diameter / 2))
230 gcpy         s = cylinder(r1=(es_diameter / 2), r2=(es_diameter / 2), h=
                    es_flute_length, center=False)
231 gcpy         p = union(b,s)
232 gcpy         return p.translate([0, 0, (es_diameter / 2)])
```

`endmill 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):

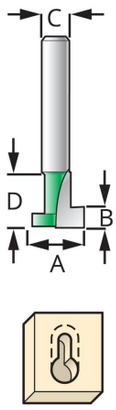
```
234 gcpy     def endmill_v(self, es_v_angle, es_diameter):
235 gcpy         es_v_angle = math.radians(es_v_angle)
236 gcpy         v = cylinder(r1=0, r2=(es_diameter / 2), h=((es_diameter /
                    2) / math.tan((es_v_angle / 2))), center=False)
237 gcpy         s = cylinder(r1=(es_diameter / 2), r2=(es_diameter / 2), h
                    =((es_diameter * 8) ), center=False)
238 gcpy         sh = s.translate([0, 0, ((es_diameter / 2) / math.tan((
                    es_v_angle / 2)))])
239 gcpy         return union(v,sh)
```

3.3.1.2 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 twice, once for the shaft, the second time for the actual keyhole cutting 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.

3.3.1.2.1 Keyhole tools Keyhole toolpaths (see: subsection 3.4.3.2.3 are intended for use with tooling which projects beyond the 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

keyhole The `keyhole` is modeled in two parts, first the cutting base:

```
241 gcpy     def keyhole(self, es_diameter, es_flute_length):
242 gcpy         return cylinder(r1=(es_diameter / 2), r2=(es_diameter / 2),
                     h=es_flute_length, center=False)
```

and a second call for an additional cylinder for the shaft will be necessary:

```
244 gcpy     def keyhole_shaft(self, es_diameter, es_flute_length):
245 gcpy         return cylinder(r1=(es_diameter / 2), r2=(es_diameter / 2),
                     h=es_flute_length, center=False)
```

3.3.1.2.2 Thread mills The implementation of arcs cutting along the Z-axis raises the threadmill possibility of cutting threads using a `threadmill`. See: https://community.carbide3d.com/t/thread-milling-in-metal-on-the-shapeoko-3/5332.

```
247 gcpy     def threadmill(self, minor_diameter, major_diameter, cut_height
                 ):
248 gcpy         btm = cylinder(r1=(minor_diameter / 2), r2=(major_diameter
                     / 2), h=cut_height, center = False)
249 gcpy         top = cylinder(r1=(major_diameter / 2), r2=(minor_diameter
                     / 2), h=cut_height, center = False)
250 gcpy         top = top.translate([0, 0, cut_height/2])
251 gcpy         tm = btm.union(top)
252 gcpy         return tm
253 gcpy
254 gcpy     def threadmill_shaft(self, diameter, cut_height, height):
255 gcpy         shaft = cylinder(r1=(diameter / 2), r2=(diameter / 2), h=
                     height, center = False)
256 gcpy         shaft = shaft.translate([0, 0, cut_height/2])
257 gcpy         return shaft
```

dovetail **3.3.1.2.3 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)

```
259 gcpy     def dovetail(self, dt_bottomdiameter, dt_topdiameter, dt_height
                 , dt_angle):
260 gcpy         return cylinder(r1=(dt_bottomdiameter / 2), r2=(
                     dt_topdiameter / 2), h= dt_height, center=False)
```

3.3.1.3 Concave toolshapes While normal tooling may be represented with a single `hull` operation betwixt two 3D toolshapes (or four 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.

3.3.1.4 Roundover tooling

It is not possible to represent all tools using tool changes as coded above which require using a `hull` operation between 3D representations of the tools at the beginning and end points. Tooling which cannot be so represented will be implemented separately below, see paragraph 3.3.1.3.

```
42 gcpscad module cutroundover(bx, by, bz, ex, ey, ez, radiustn) {
43 gcpscad     if (radiustn == 56125) {
44 gcpscad         cutroundovertool(bx, by, bz, ex, ey, ez, 0.508/2, 1.531);
45 gcpscad     } else if (radiustn == 56142) {
46 gcpscad         cutroundovertool(bx, by, bz, ex, ey, ez, 0.508/2, 2.921);
47 gcpscad //    } else if (radiustn == 312) {
48 gcpscad //        cutroundovertool(bx, by, bz, ex, ey, ez, 1.524/2, 3.175);
49 gcpscad     } else if (radiustn == 1570) {
50 gcpscad         cutroundovertool(bx, by, bz, ex, ey, ez, 0.507/2, 4.509);
51 gcpscad     }
52 gcpscad }
```

which then calls the actual `cutroundovertool` module passing in the tip radius and the radius of the rounding. Note that this module sets its quality relative to the value of `$fn`.

3.3.2 toolchange

toolchange

Then apply 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).

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 implemented, especially in the early iterations of this project.

3.3.2.1 Selecting Tools

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. This approach will not work within Python, so it will be necessary to instead assign and select the tool as part of the cutting command indirectly by first storing

currenttoolshape

it in the variable `currenttoolshape` (if the toolshape will work with the `hull` command) which may be done in this module, or it will be necessary to check for the specific toolnumber in the `cutline` module and handle the tooling in a separate module as is currently done for roundover tooling.

```
262 gcpy     def currenttool(self):
263 gcpy #         global currenttoolshape
264 gcpy         return self.currenttoolshape
```

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:

3.3.2.2 Square and ball nose (including tapered ball nose)

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

3.3.2.3 Roundover (corner rounding)

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

3.3.2.4 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).

3.3.2.5 toolchange routine

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.

```
266 gcpy     def toolchange(self,tool_number,speed = 10000):
267 gcpy #         global currenttoolshape
268 gcpy         self.currenttoolshape = self.endmill_square(0.001, 0.001)
269 gcpy
270 gcpy         self.settool(tool_number)
271 gcpy         if (self.generategcode == True):
```

```
272 gcpy             self.writegc("(Toolpath)")
273 gcpy             self.writegc("M05")
274 gcpy         if (tool_number == 201):
275 gcpy             self.writegc("(TOOL/MILL,6.35,␣0.00,␣0.00,␣0.00)")
276 gcpy             self.currenttoolshape = self.endmill_square(6.35,
                         19.05)
277 gcpy         elif (tool_number == 102):
278 gcpy             self.writegc("(TOOL/MILL,3.175,␣0.00,␣0.00,␣0.00)")
279 gcpy             self.currenttoolshape = self.endmill_square(3.175,
                         12.7)
280 gcpy         elif (tool_number == 112):
281 gcpy             self.writegc("(TOOL/MILL,1.5875,␣0.00,␣0.00,␣0.00)")
282 gcpy             self.currenttoolshape = self.endmill_square(1.5875,
                         6.35)
283 gcpy         elif (tool_number == 122):
284 gcpy             self.writegc("(TOOL/MILL,0.79375,␣0.00,␣0.00,␣0.00)")
285 gcpy             self.currenttoolshape = self.endmill_square(0.79375,
                         1.5875)
286 gcpy         elif (tool_number == 202):
287 gcpy             self.writegc("(TOOL/MILL,6.35,␣3.175,␣0.00,␣0.00)")
288 gcpy             self.currenttoolshape = self.ballnose(6.35, 19.05)
289 gcpy         elif (tool_number == 101):
290 gcpy             self.writegc("(TOOL/MILL,3.175,␣1.5875,␣0.00,␣0.00)")
291 gcpy             self.currenttoolshape = self.ballnose(3.175, 12.7)
292 gcpy         elif (tool_number == 111):
293 gcpy             self.writegc("(TOOL/MILL,1.5875,␣0.79375,␣0.00,␣0.00)")
294 gcpy             self.currenttoolshape = self.ballnose(1.5875, 6.35)
295 gcpy         elif (tool_number == 121):
296 gcpy             self.writegc("(TOOL/MILL,3.175,␣0.79375,␣0.00,␣0.00)")
297 gcpy             self.currenttoolshape = self.ballnose(0.79375, 1.5875)
298 gcpy         elif (tool_number == 327):
299 gcpy             self.writegc("(TOOL/MILL,0.03,␣0.00,␣13.4874,␣30.00)")
300 gcpy             self.currenttoolshape = self.endmill_v(60, 26.9748)
301 gcpy         elif (tool_number == 301):
302 gcpy             self.writegc("(TOOL/MILL,0.03,␣0.00,␣6.35,␣45.00)")
303 gcpy             self.currenttoolshape = self.endmill_v(90, 12.7)
304 gcpy         elif (tool_number == 302):
305 gcpy             self.writegc("(TOOL/MILL,0.03,␣0.00,␣10.998,␣30.00)")
306 gcpy             self.currenttoolshape = self.endmill_v(60, 12.7)
307 gcpy         elif (tool_number == 390):
308 gcpy             self.writegc("(TOOL/MILL,0.03,␣0.00,␣1.5875,␣45.00)")
309 gcpy             self.currenttoolshape = self.endmill_v(90, 3.175)
310 gcpy         elif (tool_number == 374):
311 gcpy             self.writegc("(TOOL/MILL,9.53,␣0.00,␣3.17,␣0.00)")
312 gcpy         elif (tool_number == 375):
313 gcpy             self.writegc("(TOOL/MILL,9.53,␣0.00,␣3.17,␣0.00)")
314 gcpy         elif (tool_number == 376):
315 gcpy             self.writegc("(TOOL/MILL,12.7,␣0.00,␣4.77,␣0.00)")
316 gcpy         elif (tool_number == 378):
317 gcpy             self.writegc("(TOOL/MILL,12.7,␣0.00,␣4.77,␣0.00)")
318 gcpy         elif (tool_number == 814):
319 gcpy             self.writegc("(TOOL/MILL,12.7,␣6.367,␣12.7,␣0.00)")
320 gcpy             #dt_bottomdiameter, dt_topdiameter, dt_height, dt_angle
                         )
321 gcpy             #https://www.leevalley.com/en-us/shop/tools/power-tool-
                         accessories/router-bits/30172-dovetail-bits?item=18
                         J1607
322 gcpy             self.currenttoolshape = self.dovetail(12.7, 6.367,
                         12.7, 14)
323 gcpy         elif (tool_number == 56125):#0.508/2, 1.531
324 gcpy             self.writegc("(TOOL/CRMILL,␣0.508,␣6.35,␣3.175,␣7.9375,
                         ␣3.175)")
325 gcpy         elif (tool_number == 56142):#0.508/2, 2.921
326 gcpy             self.writegc("(TOOL/CRMILL,␣0.508,␣3.571875,␣1.5875,␣
                         5.55625,␣1.5875)")
327 gcpy #         elif (tool_number == 312):#1.524/2, 3.175
328 gcpy #             self.writegc("(TOOL/CRMILL, Diameter1, Diameter2,
        Radius, Height, Length)")
329 gcpy         elif (tool_number == 1570):#0.507/2, 4.509
330 gcpy             self.writegc("(TOOL/CRMILL,␣0.17018,␣9.525,␣4.7625,␣
                         12.7,␣4.7625)")
```

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.

```
331 gcpy         self.writegc("M6T",str(tool_number))
332 gcpy         self.writegc("M03S",str(speed))
```

Note that the `if...else` constructs will need to be extended into the command `cutline` for those toolshapes (keyhole, roundover, &c.) which will not work with a straight-forward `hull...` implementation.

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

```
54 gcpscad module toolchange(tool_number,speed){
55 gcpscad     gcp.toolchange(tool_number,speed);
56 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.)

3.3.3 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 which is what the initial version will implement.

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

```
56 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:

```
334 gcpy     def tool_diameter(self, ptd_tool, ptd_depth):
335 gcpy # Square 122,112,102,201
336 gcpy         if ptd_tool == 122:
337 gcpy             return 0.79375
338 gcpy         if ptd_tool == 112:
339 gcpy             return 1.5875
340 gcpy         if ptd_tool == 102:
341 gcpy             return 3.175
342 gcpy         if ptd_tool == 201:
343 gcpy             return 6.35
344 gcpy # Ball 121,111,101,202
345 gcpy         if ptd_tool == 122:
346 gcpy             if ptd_depth > 0.396875:
347 gcpy                 return 0.79375
348 gcpy             else:
349 gcpy                 return ptd_tool
350 gcpy         if ptd_tool == 112:
351 gcpy             if ptd_depth > 0.79375:
352 gcpy                 return 1.5875
353 gcpy             else:
354 gcpy                 return ptd_tool
355 gcpy         if ptd_tool == 101:
356 gcpy             if ptd_depth > 1.5875:
357 gcpy                 return 3.175
358 gcpy             else:
359 gcpy                 return ptd_tool
360 gcpy         if ptd_tool == 202:
361 gcpy             if ptd_depth > 3.175:
362 gcpy                 return 6.35
363 gcpy             else:
364 gcpy                 return ptd_tool
365 gcpy # V 301, 302, 390
366 gcpy         if ptd_tool == 301:
367 gcpy             return ptd_tool
368 gcpy         if ptd_tool == 302:
369 gcpy             return ptd_tool
370 gcpy         if ptd_tool == 390:
371 gcpy             return ptd_tool
372 gcpy # Keyhole
373 gcpy         if ptd_tool == 374:
374 gcpy             if ptd_depth < 3.175:
```

```
375 gcpy                 return 9.525
376 gcpy             else:
377 gcpy                 return 6.35
378 gcpy         if ptd_tool == 375:
379 gcpy             if ptd_depth < 3.175:
380 gcpy                 return 9.525
381 gcpy             else:
382 gcpy                 return 8
383 gcpy         if ptd_tool == 376:
384 gcpy             if ptd_depth < 4.7625:
385 gcpy                 return 12.7
386 gcpy             else:
387 gcpy                 return 6.35
388 gcpy         if ptd_tool == 378:
389 gcpy             if ptd_depth < 4.7625:
390 gcpy                 return 12.7
391 gcpy             else:
392 gcpy                 return 8
393 gcpy # Dovetail
394 gcpy         if ptd_tool == 814:
395 gcpy             if ptd_depth > 12.7:
396 gcpy                 return 6.35
397 gcpy             else:
398 gcpy                 return 12.7
```

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:

```
400 gcpy     def tool_radius(self, ptd_tool, ptd_depth):
401 gcpy         tr = self.tool_diameter(ptd_tool, ptd_depth)/2
402 gcpy         return tr
```

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

3.3.4 Feeds and Speeds

feed plunge speed There are several possibilities for handling feeds and speeds. Currently, base values for `feed`, `plunge`, and `speed` are used, which may then be adjusted using various `<tooldescriptor>_ratio` 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`

3.4 Movement and Cutting

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

cut... rapid... Note that the variables `self.rapids` and `self.toolpaths` are used to hold the accumulated (`unioned`) 3D models of the rapid motions and cuts so that they may be `differenced` from the stock when the value `generatepaths` is set to `True`.

In order to manage the various options when cutting it will be necessary to have a command where the actual cut is made, passing in the shape used for the cut as a parameter. Since the 3D aspect of `rapid` and `cut` operations are fundamentally the same, the command `rcs` which returns the `hull` of the begin (the current machine position as accessed by the `x/y/zpos()` commands and end positioning (provided as arguments `ex`, `ey`, and `ez`) of the tool shape/cross-section will be defined for the common aspects:

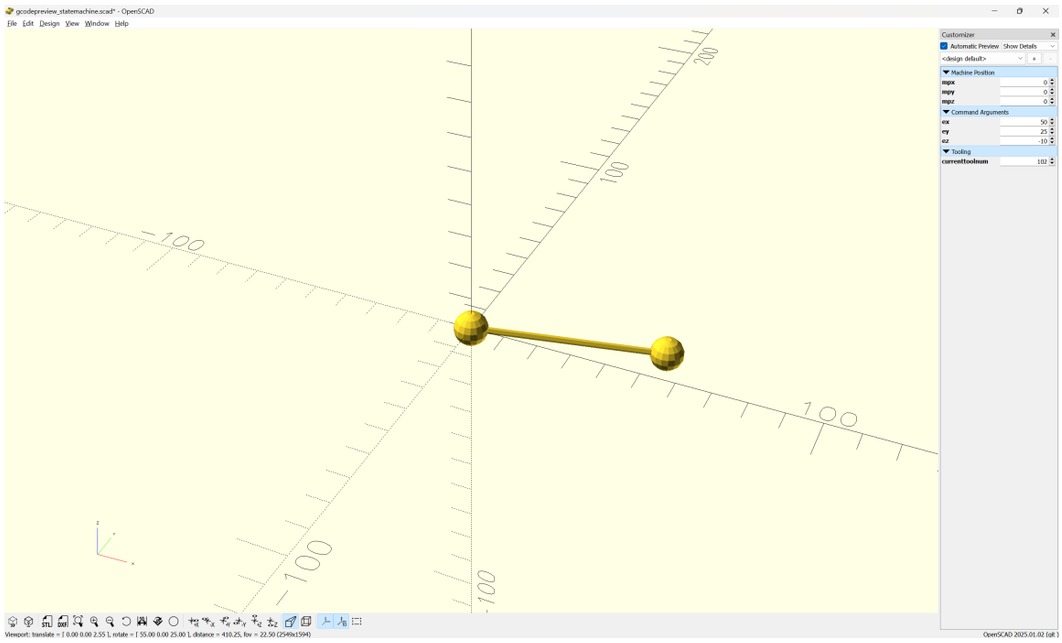
rcs

```
404 gcpy     def rcs(self,ex, ey, ez, shape):
405 gcpy         start = shape
406 gcpy         end = shape
407 gcpy         toolpath = hull(start.translate([self.xpos(), self.ypos(),
                    self.zpos()]),
408 gcpy                         end.translate([ex,ey,ez]))
```

```
409 gcpy          return toolpath
```

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):



Note that this routine does *not* alter the machine position variables since it may be called multiple times for a given toolpath. This command will then be called in the definitions for `rapid` and `cutshape` which only differ in which variable the 3D model is `union`ed with:

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 will appear in both G-code and DXF files.

```
411 gcpy      def rapid(self,ex, ey, ez):
412 gcpy          cts = self.currenttoolshape
413 gcpy          toolpath = self.rcs(ex, ey, ez, cts)
414 gcpy          self.setxpos(ex)
415 gcpy          self.setypos(ey)
416 gcpy          self.setzpos(ez)
417 gcpy          if self.generatepaths == True:
418 gcpy              self.rapids = self.rapids.union(toolpath)
419 gcpy #             return cylinder(0.01, 0, 0.01, center = False, fn = 3)
420 gcpy              return cube([0.001,0.001,0.001])
421 gcpy          else:
422 gcpy              return toolpath
423 gcpy
424 gcpy      def cutshape(self,ex, ey, ez):
425 gcpy          cts = self.currenttoolshape
426 gcpy          toolpath = self.rcs(ex, ey, ez, cts)
427 gcpy          if self.generatepaths == True:
428 gcpy              self.toolpaths = self.toolpaths.union(toolpath)
429 gcpy              return cube([0.001,0.001,0.001])
430 gcpy          else:
431 gcpy              return toolpath
```

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

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:

```
433 gcpy      def movetosafeZ(self):
```

```
434 gcpy          rapid = self.rapid(self.xpos(),self.ypos(),self.
                      retractheight)
435 gcpy #         if self.generatepaths == True:
436 gcpy #             rapid = self.rapid(self.xpos(),self.ypos(),self.
                  retractheight)
437 gcpy #             self.rapids = self.rapids.union(rapid)
438 gcpy #         else:
439 gcpy #  if (generategcode == true) {
440 gcpy #  //    writecomment("PREPOSITION FOR RAPID PLUNGE");Z25.650
441 gcpy #  //G1Z24.663F381.0 ,"F",str(plunge)
442 gcpy          if self.generatepaths == False:
443 gcpy              return rapid
444 gcpy          else:
445 gcpy              return cube([0.001,0.001,0.001])
446 gcpy
447 gcpy      def rapidXY(self, ex, ey):
448 gcpy          rapid = self.rapid(ex,ey,self.zpos())
449 gcpy #         if self.generatepaths == True:
450 gcpy #             self.rapids = self.rapids.union(rapid)
451 gcpy #         else:
452 gcpy          if self.generatepaths == False:
453 gcpy              return rapid
454 gcpy
455 gcpy      def rapidZ(self, ez):
456 gcpy          rapid = self.rapid(self.xpos(),self.ypos(),ez)
457 gcpy #         if self.generatepaths == True:
458 gcpy #             self.rapids = self.rapids.union(rapid)
459 gcpy #         else:
460 gcpy          if self.generatepaths == False:
461 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):

```
58 gcpscad module movetosafeZ(){
59 gcpscad     gcp.rapid(gcp.xpos(),gcp.ypos(),retractheight);
60 gcpscad }
61 gcpscad
62 gcpscad module rapid(ex, ey, ez) {
63 gcpscad     gcp.rapid(ex, ey, ez);
64 gcpscad }
65 gcpscad
66 gcpscad module rapidXY(ex, ey) {
67 gcpscad     gcp.rapid(ex, ey, gcp.zpos());
68 gcpscad }
69 gcpscad
70 gcpscad module rapidZ(ez) {
71 gcpscad     gcp.rapid(gcp.xpos(),gcp.ypos(),ez);
72 gcpscad }
```

3.4.1 Lines

`cut...` `cutline`

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

```
460 gcpy      def cutline(self,ex, ey, ez):\
461 gcpy #below will need to be integrated into if/then structure not yet
          copied
462 gcpy #         cts = self.currenttoolshape
463 gcpy          if (self.currenttoolnumber() == 374):
464 gcpy #             self.writegc("(TOOL/MILL,9.53, 0.00, 3.17, 0.00)")
465 gcpy              self.currenttoolshape = self.keyhole(9.53/2, 3.175)
466 gcpy              toolpath = self.cutshape(ex, ey, ez)
467 gcpy              self.currenttoolshape = self.keyhole_shaft(6.35/2,
                      12.7)
468 gcpy              toolpath = toolpath.union(self.cutshape(ex, ey, ez))
469 gcpy #         elif (self.currenttoolnumber() == 375):
470 gcpy #             self.writegc("(TOOL/MILL,9.53, 0.00, 3.17, 0.00)")
471 gcpy #         elif (self.currenttoolnumber() == 376):
472 gcpy #             self.writegc("(TOOL/MILL,12.7, 0.00, 4.77, 0.00)")
473 gcpy #         elif (self.currenttoolnumber() == 378):
474 gcpy #             self.writegc("(TOOL/MILL,12.7, 0.00, 4.77, 0.00)")
475 gcpy #         elif (self.currenttoolnumber() == 56125):#0.508/2, 1.531
476 gcpy #             self.writegc("(TOOL/CRMILL, 0.508, 6.35, 3.175,
```

```
            7.9375, 3.175)")
477 gcpy          elif (self.currenttoolnumber() == 56142):#0.508/2, 2.921
478 gcpy #             self.writegc("(TOOL/CRMILL, 0.508, 3.571875, 1.5875,
            5.55625, 1.5875)")
479 gcpy              toolpath = self.cutroundovertool(self.xpos(), self.ypos
                        (), self.zpos(), ex, ey, ez, 0.508/2, 1.531)
480 gcpy #         elif (self.currenttoolnumber() == 1570):#0.507/2, 4.509
481 gcpy #             self.writegc("(TOOL/CRMILL, 0.17018, 9.525, 4.7625,
            12.7, 4.7625)")
482 gcpy          else:
483 gcpy              toolpath = self.cutshape(ex, ey, ez)
484 gcpy          self.setxpos(ex)
485 gcpy          self.setypos(ey)
486 gcpy          self.setzpos(ez)
487 gcpy #         if self.generatepaths == True:
488 gcpy #             self.toolpaths = union([self.toolpaths, toolpath])
489 gcpy #         else:
490 gcpy          if self.generatepaths == False:
491 gcpy              return toolpath
492 gcpy          else:
493 gcpy              return cube([0.001,0.001,0.001])
494 gcpy
495 gcpy      def cutlinedxfgc(self,ex, ey, ez):
496 gcpy          self.dxfline(self.currenttoolnumber(), self.xpos(), self.
                    ypos(), ex, ey)
497 gcpy          self.writegc("G01␣X", str(ex), "␣Y", str(ey), "␣Z", str(ez)
                    )
498 gcpy #         if self.generatepaths == False:
499 gcpy          return self.cutline(ex, ey, ez)
500 gcpy
501 gcpy      def cutroundovertool(self, bx, by, bz, ex, ey, ez,
                 tool_radius_tip, tool_radius_width, stepsizeroundover = 1):
502 gcpy #         n = 90 + fn*3
503 gcpy #         print("Tool dimensions", tool_radius_tip,
            tool_radius_width, "begin ",bx, by, bz,"end ", ex, ey, ez)
504 gcpy          step = 4 #360/n
505 gcpy          shaft = cylinder(step,tool_radius_tip,tool_radius_tip)
506 gcpy          toolpath = hull(shaft.translate([bx,by,bz]), shaft.
                    translate([ex,ey,ez]))
507 gcpy          shaft = cylinder(tool_radius_width*2,tool_radius_tip+
                    tool_radius_width,tool_radius_tip+tool_radius_width)
508 gcpy          toolpath = toolpath.union(hull(shaft.translate([bx,by,bz+
                    tool_radius_width]), shaft.translate([ex,ey,ez+
                    tool_radius_width])))
509 gcpy          for i in range(1, 90, stepsizeroundover):
510 gcpy              angle = i
511 gcpy              dx = tool_radius_width*math.cos(math.radians(angle))
512 gcpy              dxx = tool_radius_width*math.cos(math.radians(angle+1))
513 gcpy              dzz = tool_radius_width*math.sin(math.radians(angle))
514 gcpy              dz = tool_radius_width*math.sin(math.radians(angle+1))
515 gcpy              dh = abs(dzz-dz)+0.0001
516 gcpy              slice = cylinder(dh,tool_radius_tip+tool_radius_width-
                        dx,tool_radius_tip+tool_radius_width-dxx)
517 gcpy              toolpath = toolpath.union(hull(slice.translate([bx,by,
                        bz+dz]), slice.translate([ex,ey,ez+dz])))
518 gcpy          if self.generatepaths == True:
519 gcpy              self.toolpaths = self.toolpaths.union(toolpath)
520 gcpy          else:
521 gcpy              return toolpath
```

The matching OpenSCAD command is a descriptor:

```
74 gcpscad module cutline(ex, ey, ez){
75 gcpscad     gcp.cutline(ex, ey, ez);
76 gcpscad }
77 gcpscad
78 gcpscad module cutlinedxfgc(ex, ey, ez){
79 gcpscad     gcp.cutlinedxfgc(ex, ey, ez);
80 gcpscad }
```

3.4.2 Arcs for toolpaths and DXFs

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 wont 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.

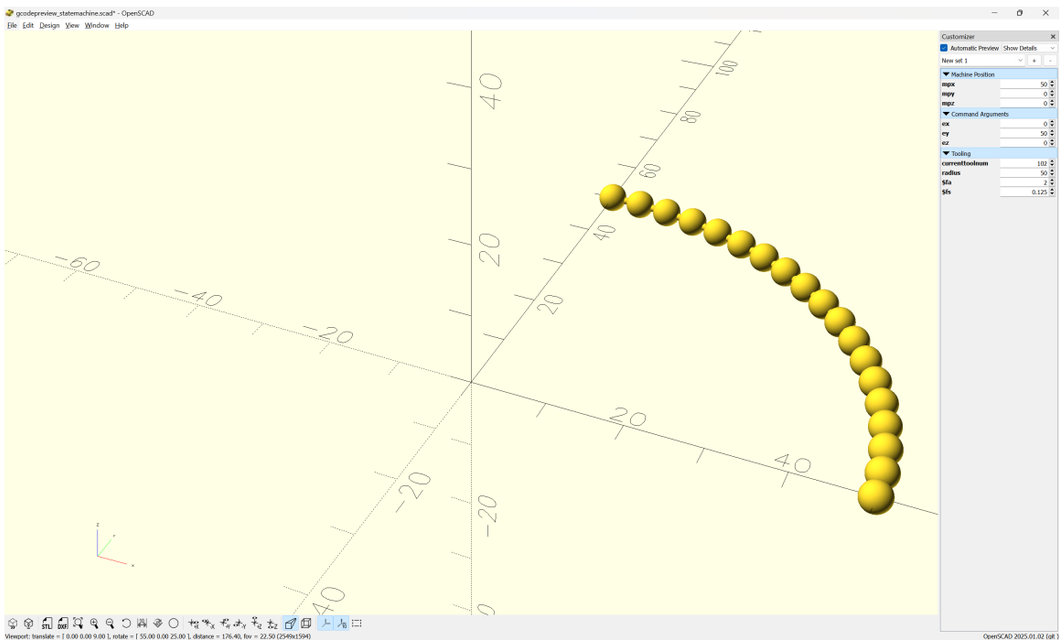
Parameters which will need to be passed in are:

- `ex` — note that the matching origins (`bx`, `by`, `bz`) as well as the (current) toolnumber are accessed using the appropriate commands
- `ey`
- `ez` — allowing a different Z position will make possible threading and similar helical toolpaths
- `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 acts as a check on the other parameters
- `tpzreldim` — the relative depth (or increase in height) of the current cutting motion

`cutarcCW` `cutarcCC`

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

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



The code for which makes this obvious:

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

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

/* [Tooling] */
currenttoolnum = 102;

machine_extents();

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

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

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

```
}
}

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

module plot_cut(bx, by, bz, ex, ey, ez) {
  union(){
    translate([bx, by, bz]){
      sphere(r=5);
    }
    translate([ex, ey, ez]){
      sphere(r=5);
    }
    hull(){
      translate([bx, by, bz]){
        sphere(r=1);
      }
      translate([ex, ey, ez]){
        sphere(r=1);
      }
    }
  }
}
```

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.)

```
562 gcpy      def cutarcCC(self, barc, earc, xcenter, ycenter, radius,
                  tpzreldim, stepsizearc=1):
563 gcpy #         tpzinc = ez - self.zpos() / (earc - barc)
564 gcpy          tpzinc = tpzreldim / (earc - barc)
565 gcpy          cts = self.currenttoolshape
566 gcpy          toolpath = cts
567 gcpy          toolpath = toolpath.translate([self.xpos(),self.ypos(),self
                      .zpos()])
568 gcpy          i = barc
569 gcpy          while i < earc:
570 gcpy              toolpath = toolpath.union(self.cutline(xcenter + radius
                           * math.cos(math.radians(i)), ycenter + radius *
                          math.sin(math.radians(i)), self.zpos()+tpzinc))
571 gcpy              i += stepsizearc
572 gcpy          if self.generatepaths == False:
573 gcpy              return toolpath
574 gcpy          else:
575 gcpy              return cube([0.01,0.01,0.01])
576 gcpy
577 gcpy      def cutarcCW(self, barc,earc, xcenter, ycenter, radius,
                  tpzreldim, stepsizearc=1):
578 gcpy #         print(str(self.zpos()))
579 gcpy #         print(str(ez))
580 gcpy #         print(str(barc - earc))
581 gcpy #         tpzinc = ez - self.zpos() / (barc - earc)
582 gcpy #         print(str(tzinc))
583 gcpy #         global toolpath
584 gcpy #         print("Entering n toolpath")
585 gcpy          tpzinc = tpzreldim / (barc - earc)
586 gcpy          cts = self.currenttoolshape
587 gcpy          toolpath = cts
588 gcpy          toolpath = toolpath.translate([self.xpos(),self.ypos(),self
                      .zpos()])
589 gcpy          i = barc
590 gcpy          while i > earc:
591 gcpy              toolpath = toolpath.union(self.cutline(xcenter + radius
                           * math.cos(math.radians(i)), ycenter + radius *
                          math.sin(math.radians(i)), self.zpos()+tpzinc))
592 gcpy #             self.setxpos(xcenter + radius * math.cos(math.radians(
              i)))
593 gcpy #             self.setypos(ycenter + radius * math.sin(math.radians(
              i)))
```

```
594 gcpy #             print(str(self.xpos()), str(self.ypos(), str(self.zpos
        ())))
595 gcpy #             self.setzpos(self.zpos()+tpzinc)
596 gcpy              i += abs(stepsizearc) * -1
597 gcpy #         self.dxfarc(self.currenttoolnumber(), xcenter, ycenter,
        radius, barc, earc)
598 gcpy #         if self.generatepaths == True:
599 gcpy #             print("Unioning n toolpath")
600 gcpy #             self.toolpaths = self.toolpaths.union(toolpath)
601 gcpy #         else:
602 gcpy          if self.generatepaths == False:
603 gcpy              return toolpath
604 gcpy          else:
605 gcpy              return cube([0.01,0.01,0.01])
```

Matching OpenSCAD modules are easily made:

```
78 gcpscad module cutarcCC(barc, earc, xcenter, ycenter, radius, tpzreldim){
79 gcpscad     gcp.cutarcCC(barc, earc, xcenter, ycenter, radius, tpzreldim);
80 gcpscad }
81 gcpscad
82 gcpscad module cutarcCW(barc, earc, xcenter, ycenter, radius, tpzreldim){
83 gcpscad     gcp.cutarcCW(barc, earc, xcenter, ycenter, radius, tpzreldim);
84 gcpscad }
```

3.4.3 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 3.3.2) 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.

3.4.3.1 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.

3.4.3.2 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.

- 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)
  - astroid (four concave arcs)
  - salinon (four semicircles)
  - three straight lines and one concave arc

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

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

- two straight edges, one concave arc—oddly, an asymmetric form (hyperbolic sector) has a name, but not the symmetrical—while the colloquial/prosaic arrowhead was considered, it was rejected as being better applied to the shape below. (Its 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 (its 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?

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:

[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

- 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 — it should be possible to include support for specialty tooling such as dovetail cutters and to get an accurate 3D model, esp. for tooling which undercuts since they cannot be modeled in Carbide Create.
- Starting and Max Depth — are there CAD programs which will make use of Z-axis information in a DXF? — would it be possible/necessary to further differentiate the DXF geometry? (currently written out separately for each toolpath in addition to one combined file)

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

```
613 gcpy     def dxfcircle(self, tool_num, xcenter, ycenter, radius):
614 gcpy         self.dxfarc(tool_num, xcenter, ycenter, radius,   0,  90)
615 gcpy         self.dxfarc(tool_num, xcenter, ycenter, radius,  90, 180)
616 gcpy         self.dxfarc(tool_num, xcenter, ycenter, radius, 180, 270)
617 gcpy         self.dxfarc(tool_num, xcenter, ycenter, radius, 270, 360)
```

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

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

```
619 gcpy     def dxfrectangle(self, tool_num, xorigin, yorigin, xwidth,
                 yheight, corners = "Square", radius = 6):
620 gcpy         if corners == "Square":
621 gcpy             self.dxfline(tool_num, xorigin, yorigin, xorigin +
                         xwidth, yorigin)
622 gcpy             self.dxfline(tool_num, xorigin + xwidth, yorigin,
                         xorigin + xwidth, yorigin + yheight)
623 gcpy             self.dxfline(tool_num, xorigin + xwidth, yorigin +
                         yheight, xorigin, yorigin + yheight)
624 gcpy             self.dxfline(tool_num, xorigin, yorigin + yheight,
                         xorigin, yorigin)
625 gcpy         elif corners == "Fillet":
626 gcpy             self.dxfrectangleround(tool_num, xorigin, yorigin,
                         xwidth, yheight, radius)
627 gcpy         elif corners == "Chamfer":
628 gcpy             self.dxfrectanglechamfer(tool_num, xorigin, yorigin,
                         xwidth, yheight, radius)
629 gcpy         elif corners == "Flipped␣Fillet":
630 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.

```
625 gcpy     def dxfrectangleround(self, tool_num, xorigin, yorigin, xwidth,
                  yheight, radius):
626 gcpy         self.dxfarc(tool_num, xorigin + xwidth - radius, yorigin +
                     yheight - radius, radius,   0,  90)
627 gcpy         self.dxfarc(tool_num, xorigin + radius, yorigin + yheight -
                      radius, radius,  90, 180)
628 gcpy         self.dxfarc(tool_num, xorigin + radius, yorigin + radius,
                     radius, 180, 270)
629 gcpy         self.dxfarc(tool_num, xorigin + xwidth - radius, yorigin +
                     radius, radius, 270, 360)
630 gcpy
631 gcpy         self.dxfline(tool_num, xorigin + radius, yorigin, xorigin +
                      xwidth - radius, yorigin)
632 gcpy         self.dxfline(tool_num, xorigin + xwidth, yorigin + radius,
                     xorigin + xwidth, yorigin + yheight - radius)
633 gcpy         self.dxfline(tool_num, xorigin + xwidth - radius, yorigin +
                      yheight, xorigin + radius, yorigin + yheight)
634 gcpy         self.dxfline(tool_num, xorigin, yorigin + yheight - radius,
                      xorigin, yorigin + radius)
```

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

```
636 gcpy     def dxfrectanglechamfer(self, tool_num, xorigin, yorigin,
                 xwidth, yheight, radius):
637 gcpy         self.dxfline(tool_num, xorigin + radius, yorigin, xorigin,
                     yorigin + radius)
```

```
638 gcpy          self.dxfline(tool_num, xorigin, yorigin + yheight - radius,
                      xorigin + radius, yorigin + yheight)
639 gcpy          self.dxfline(tool_num, xorigin + xwidth - radius, yorigin +
                      yheight, xorigin + xwidth, yorigin + yheight - radius)
640 gcpy          self.dxfline(tool_num, xorigin + xwidth - radius, yorigin,
                     xorigin + xwidth, yorigin + radius)
641 gcpy
642 gcpy          self.dxfline(tool_num, xorigin + radius, yorigin, xorigin +
                      xwidth - radius, yorigin)
643 gcpy          self.dxfline(tool_num, xorigin + xwidth, yorigin + radius,
                     xorigin + xwidth, yorigin + yheight - radius)
644 gcpy          self.dxfline(tool_num, xorigin + xwidth - radius, yorigin +
                      yheight, xorigin + radius, yorigin + yheight)
645 gcpy          self.dxfline(tool_num, xorigin, yorigin + yheight - radius,
                      xorigin, yorigin + radius)
```

Flipped Fillet:

```
647 gcpy      def dxfrectangleflippedfillet(self, tool_num, xorigin, yorigin,
                   xwidth, yheight, radius):
648 gcpy          self.dxfarc(tool_num, xorigin, yorigin, radius,   0,  90)
649 gcpy          self.dxfarc(tool_num, xorigin + xwidth, yorigin, radius,
                     90, 180)
650 gcpy          self.dxfarc(tool_num, xorigin + xwidth, yorigin + yheight,
                     radius, 180, 270)
651 gcpy          self.dxfarc(tool_num, xorigin, yorigin + yheight, radius,
                     270, 360)
652 gcpy
653 gcpy          self.dxfline(tool_num, xorigin + radius, yorigin, xorigin +
                      xwidth - radius, yorigin)
654 gcpy          self.dxfline(tool_num, xorigin + xwidth, yorigin + radius,
                     xorigin + xwidth, yorigin + yheight - radius)
655 gcpy          self.dxfline(tool_num, xorigin + xwidth - radius, yorigin +
                      yheight, xorigin + radius, yorigin + yheight)
656 gcpy          self.dxfline(tool_num, xorigin, yorigin + yheight - radius,
                      xorigin, yorigin + radius)
```

\paragraph{Rectangles}

Cutting rectangles while writing out their perimeter in the DXF files (so that they may be assigned a ma

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

Th routine \DescribeRoutine{cutrectangledxf} cuts the outline of a rectangle creating sharp corners. No

```
\lstset{firstnumber=\thegcpscad}
\begin{writecode}{a}{gcodepreview.scad}{scad}
module cutrectangledxf(bx, by, bz, rwidth, rheight, rdepth, rtn) {//passes
  movetosafez();
  hull(){
    //  for (i = [0 : abs(1) : passes]) {
    //      rapid(bx+tool_radius(rtn)+i*(rwidth-tool_diameter(current_tool()))/passes,bx+tool_radius(rtn
    //      cutwithfeed(bx+tool_radius(rtn)+i*(rwidth-tool_diameter(current_tool()))/passes,by+tool_radi
rdepth,feed);
    //      cutwithfeed(bx+tool_radius(rtn)+i*(rwidth-tool_diameter(current_tool()))/passes,by+rheight-
tool_radius(rtn),bz-rdepth,feed);

    cutwithfeed(bx+tool_radius(rtn),by+tool_radius(rtn),bz-rdepth,feed);
    cutwithfeed(bx+rwidth-tool_radius(rtn),by+tool_radius(rtn),bz-rdepth,feed);
    cutwithfeed(bx+rwidth-tool_radius(rtn),by+rheight-tool_radius(rtn),bz-rdepth,feed);
    cutwithfeed(bx+tool_radius(rtn),by+rheight-tool_radius(rtn),bz-rdepth,feed);
  }
  //dxfarc(xcenter,ycenter,radius,anglebegin,endangle, tn)
  dxfarc(bx+tool_radius(rtn),by+tool_radius(rtn),tool_radius(rtn),180,270, rtn);
  //dxfline(xbegin,ybegin,xend,yend, tn)
  dxfline(bx,by+tool_radius(rtn),bx,by+rheight-tool_radius(rtn), rtn);
  dxfarc(bx+tool_radius(rtn),by+rheight-tool_radius(rtn),tool_radius(rtn),90,180, rtn);
  dxfline(bx+tool_radius(rtn),by+rheight,bx+rwidth-tool_radius(rtn),by+rheight, rtn);
  dxfarc(bx+rwidth-tool_radius(rtn),by+rheight-tool_radius(rtn),tool_radius(rtn),0,90, rtn);
  dxfline(bx+rwidth,by+rheight-tool_radius(rtn),bx+rwidth,by+tool_radius(rtn), rtn);
  dxfarc(bx+rwidth-tool_radius(rtn),by+tool_radius(rtn),tool_radius(rtn),270,360, rtn);
  dxfline(bx+rwidth-tool_radius(rtn),by,bx+tool_radius(rtn),by, rtn);
}

\end{writecode}
\addtocounter{gcpscad}{25}
```

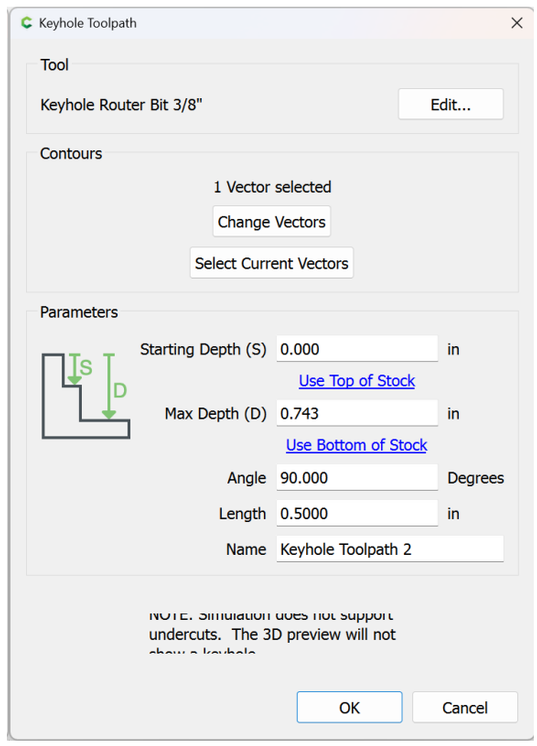
```
A matching command: \DescribeRoutine{cutrectangleoutlinedxf} cuts the outline of a rounded rectangle and

\lstset{firstnumber=\thegcpscad}
\begin{writecode}{a}{gcodepreview.scad}{scad}
module cutrectangleoutlinedxf(bx, by, bz, rwidth, rheight, rdepth, rtn) {//passes
  movetosafez();
  cutwithfeed(bx+tool_radius(rtn),by+tool_radius(rtn),bz-rdepth,feed);
  cutwithfeed(bx+rwidth-tool_radius(rtn),by+tool_radius(rtn),bz-rdepth,feed);
  cutwithfeed(bx+rwidth-tool_radius(rtn),by+rheight-tool_radius(rtn),bz-rdepth,feed);
  cutwithfeed(bx+tool_radius(rtn),by+rheight-tool_radius(rtn),bz-rdepth,feed);
  dxfarc(bx+tool_radius(rtn),by+tool_radius(rtn),tool_radius(rtn),180,270, rtn);
  dxfline(bx,by+tool_radius(rtn),bx,by+rheight-tool_radius(rtn), rtn);
  dxfarc(bx+tool_radius(rtn),by+rheight-tool_radius(rtn),tool_radius(rtn),90,180, rtn);
  dxfline(bx+tool_radius(rtn),by+rheight,bx+rwidth-tool_radius(rtn),by+rheight, rtn);
  dxfarc(bx+rwidth-tool_radius(rtn),by+rheight-tool_radius(rtn),tool_radius(rtn),0,90, rtn);
  dxfline(bx+rwidth,by+rheight-tool_radius(rtn),bx+rwidth,by+tool_radius(rtn), rtn);
  dxfarc(bx+rwidth-tool_radius(rtn),by+tool_radius(rtn),tool_radius(rtn),270,360, rtn);
  dxfline(bx+rwidth-tool_radius(rtn),by,bx+tool_radius(rtn),by, rtn);
}

\end{writecode}
\addtocounter{gcpscad}{16}

Which suggests a further command, \DescribeRoutine{rectangleoutlinedxf} for simply adding a rectangle (a

\lstset{firstnumber=\thegcpscad}
\begin{writecode}{a}{gcodepreview.scad}{scad}
module rectangleoutlinedxf(bx, by, bz, rwidth, rheight, rtn) {
  dxfline(bx,by,bx,by+rheight, rtn);
  dxfline(bx,by+rheight,bx+rwidth,by+rheight, rtn);
  dxfline(bx+rwidth,by+rheight,bx+rwidth,by, rtn);
  dxfline(bx+rwidth,by,bx,by, rtn);
}

\end{writecode}
\addtocounter{gcpscad}{7}

\noindent the initial section performs the cutting operation for the 3D preview while the latter sectio

A variant of the  cutting version of that file, \DescribeRoutine{cutoutrectangledxf} will cut to the ou

\lstset{firstnumber=\thegcpscad}
\begin{writecode}{a}{gcodepreview.scad}{scad}
module cutoutrectangledxf(bx, by, bz, rwidth, rheight, rdepth, rtn) {
  movetosafez();
  cutwithfeed(bx-tool_radius(rtn),by-tool_radius(rtn),bz-rdepth,feed);
  cutwithfeed(bx+rwidth+tool_radius(rtn),by-tool_radius(rtn),bz-rdepth,feed);
  cutwithfeed(bx+rwidth+tool_radius(rtn),by+rheight+tool_radius(rtn),bz-rdepth,feed);
  cutwithfeed(bx-tool_radius(rtn),by+rheight+tool_radius(rtn),bz-rdepth,feed);
  cutwithfeed(bx-tool_radius(rtn),by-tool_radius(rtn),bz-rdepth,feed);
  dxfline(bx,by,bx,by+rheight, rtn);
  dxfline(bx,by+rheight,bx+rwidth,by+rheight, rtn);
  dxfline(bx+rwidth,by+rheight,bx+rwidth,by, rtn);
  dxfline(bx+rwidth,by,bx,by, rtn);
}

\end{writecode}
\addtocounter{gcpscad}{13}
```

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

cutkeyhole toolpath

Tooling for such toolpaths is defined at paragraph 3.3.1.2

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:

```
658 gcpy     def cutkeyholegcdxf(self, kh_tool_num, kh_start_depth,
                 kh_max_depth, kht_direction, kh_distance):
659 gcpy         if (kht_direction == "N"):
660 gcpy             toolpath = self.cutKHgcdxf(kh_tool_num, kh_start_depth,
                         kh_max_depth, 90, kh_distance)
661 gcpy         elif (kht_direction == "S"):
662 gcpy             toolpath = self.cutKHgcdxf(kh_tool_num, kh_start_depth,
                         kh_max_depth, 270, kh_distance)
663 gcpy         elif (kht_direction == "E"):
664 gcpy             toolpath = self.cutKHgcdxf(kh_tool_num, kh_start_depth,
                         kh_max_depth, 0, kh_distance)
665 gcpy         elif (kht_direction == "W"):
666 gcpy             toolpath = self.cutKHgcdxf(kh_tool_num, kh_start_depth,
                         kh_max_depth, 180, kh_distance)
667 gcpy         if self.generatepaths == True:
668 gcpy             self.toolpaths = union([self.toolpaths, toolpath])
669 gcpy             return toolpath
670 gcpy         else:
671 gcpy             return cube([0.01,0.01,0.01])
```

```
672 gcpscad module cutkeyholegcdxf(kh_tool_num, kh_start_depth, kh_max_depth,
        kht_direction, kh_distance){
673 gcpscad     gcp.cutkeyholegcdxf(kh_tool_num, kh_start_depth, kh_max_depth,
            kht_direction, kh_distance);
674 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:

```
672 gcpy     def cutKHgcdxf(self, kh_tool_num, kh_start_depth, kh_max_depth,
             kh_angle, kh_distance):
673 gcpy         oXpos = self.xpos()
674 gcpy         oYpos = self.ypos()
675 gcpy         self.dxfKH(kh_tool_num, self.xpos(), self.ypos(),
                 kh_start_depth, kh_max_depth, kh_angle, kh_distance)
676 gcpy         toolpath = self.cutline(self.xpos(), self.ypos(), -
                 kh_max_depth)
677 gcpy         self.setxpos(oXpos)
678 gcpy         self.setypos(oYpos)
679 gcpy         if self.generatepaths == False:
680 gcpy             return toolpath
681 gcpy         else:
682 gcpy             return cube([0.001,0.001,0.001])
```

```
684 gcpy     def dxfKH(self, kh_tool_num, oXpos, oYpos, kh_start_depth,
             kh_max_depth, kh_angle, kh_distance):
685 gcpy #         oXpos = self.xpos()
686 gcpy #         oYpos = self.ypos()
687 gcpy #Circle at entry hole
688 gcpy         self.dxfarc(kh_tool_num, oXpos,oYpos,self.tool_radius(
                 kh_tool_num, 7),  0, 90)
689 gcpy         self.dxfarc(kh_tool_num, oXpos,oYpos,self.tool_radius(
                 kh_tool_num, 7), 90,180)
690 gcpy         self.dxfarc(kh_tool_num, oXpos,oYpos,self.tool_radius(
                 kh_tool_num, 7),180,270)
691 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:

```
696 gcpy #pre-calculate needed values
697 gcpy         r = self.tool_radius(kh_tool_num, 7)
698 gcpy #         print(r)
699 gcpy         rt = self.tool_radius(kh_tool_num, 1)
700 gcpy #         print(rt)
701 gcpy         ro = math.sqrt((self.tool_radius(kh_tool_num, 1))**2-(self.
                 tool_radius(kh_tool_num, 7))**2)
702 gcpy #         print(ro)
703 gcpy         angle = math.degrees(math.acos(ro/rt))
704 gcpy #Outlines of entry hole and slot
705 gcpy         if (kh_angle == 0):
706 gcpy #Lower left of entry hole
707 gcpy             self.dxfarc(kh_tool_num, self.xpos(),self.ypos(),self.
                     tool_radius(kh_tool_num, 1),180,270)
708 gcpy #Upper left of entry hole
709 gcpy             self.dxfarc(kh_tool_num, self.xpos(),self.ypos(),self.
                     tool_radius(kh_tool_num, 1),90,180)
710 gcpy #Upper right of entry hole
711 gcpy #             self.dxfarc(kh_tool_num, self.xpos(), self.ypos(), rt,
             41.810, 90)
712 gcpy             self.dxfarc(kh_tool_num, self.xpos(), self.ypos(), rt,
                     angle, 90)
713 gcpy #Lower right of entry hole
714 gcpy             self.dxfarc(kh_tool_num, self.xpos(), self.ypos(), rt,
                     270, 360-angle)
715 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))))
716 gcpy #Actual line of cut
717 gcpy #             self.dxfline(kh_tool_num, self.xpos(),self.ypos(),self
            .xpos()+kh_distance,self.ypos())
```

```
718 gcpy #upper right of end of slot (kh_max_depth+4.36))/2
719 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)
720 gcpy #lower right of end of slot
721 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)
722 gcpy #upper right slot
723 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))
724 gcpy #             self.dxfline(kh_tool_num, self.xpos()+(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)
725 gcpy #end position at top of slot
726 gcpy #lower right slot
727 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))
728 gcpy #         dxfline(kh_tool_num, self.xpos()+(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)
729 gcpy #end position at top of slot
730 gcpy #    hull(){
731 gcpy #      translate([xpos(), ypos(), zpos()]){
732 gcpy #        keyhole_shaft(6.35, 9.525);
733 gcpy #      }
734 gcpy #      translate([xpos(), ypos(), zpos()-kh_max_depth]){
735 gcpy #        keyhole_shaft(6.35, 9.525);
736 gcpy #      }
737 gcpy #    }
738 gcpy #    hull(){
739 gcpy #      translate([xpos(), ypos(), zpos()-kh_max_depth]){
740 gcpy #        keyhole_shaft(6.35, 9.525);
741 gcpy #      }
742 gcpy #      translate([xpos()+kh_distance, ypos(), zpos()-kh_max_depth])
          {
743 gcpy #        keyhole_shaft(6.35, 9.525);
744 gcpy #      }
745 gcpy #    }
746 gcpy #    cutwithfeed(getxpos(),getypos(),-kh_max_depth,feed);
747 gcpy #    cutwithfeed(getxpos()+kh_distance,getypos(),-kh_max_depth,feed
          );
748 gcpy #    setxpos(getxpos()-kh_distance);
749 gcpy #  } else if (kh_angle > 0 && kh_angle < 90) {
750 gcpy #//echo(kh_angle);
751 gcpy #  dxfarc(getxpos(),getypos(),tool_diameter(KH_tool_num, (
          kh_max_depth))/2,90+kh_angle,180+kh_angle, KH_tool_num);
752 gcpy #  dxfarc(getxpos(),getypos(),tool_diameter(KH_tool_num, (
          kh_max_depth))/2,180+kh_angle,270+kh_angle, KH_tool_num);
753 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);
754 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);
755 gcpy #dxfarc(getxpos()+(kh_distance*cos(kh_angle)),
756 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);
757 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);
758 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))),
759 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))),
760 gcpy # getxpos()+(kh_distance*cos(kh_angle))-((tool_diameter(KH_tool_num
            , (kh_max_depth+4.36))/2)*sin(kh_angle)),
761 gcpy # getypos()+(kh_distance*sin(kh_angle))+((tool_diameter(KH_tool_num
            , (kh_max_depth+4.36))/2)*cos(kh_angle)), KH_tool_num);
762 gcpy #//echo("a",tool_diameter(KH_tool_num, (kh_max_depth+4.36))/2);
763 gcpy #//echo("c",tool_diameter(KH_tool_num, (kh_max_depth))/2);
764 gcpy #echo("Aangle",asin((tool_diameter(KH_tool_num, (kh_max_depth+4.36)
            )/2)/(tool_diameter(KH_tool_num, (kh_max_depth))/2)));
765 gcpy #//echo(kh_angle);
766 gcpy # cutwithfeed(getxpos()+(kh_distance*cos(kh_angle)),getypos()+(
            kh_distance*sin(kh_angle)),-kh_max_depth,feed);
767 gcpy #             toolpath = toolpath.union(self.cutline(self.xpos()+
            kh_distance, self.ypos(), -kh_max_depth))
768 gcpy          elif (kh_angle == 90):
769 gcpy #Lower left of entry hole
770 gcpy              self.dxfarc(kh_tool_num, oXpos,oYpos,self.tool_radius(
                      kh_tool_num, 1),180,270)
771 gcpy #Lower right of entry hole
772 gcpy              self.dxfarc(kh_tool_num, oXpos,oYpos,self.tool_radius(
                      kh_tool_num, 1),270,360)
773 gcpy #left slot
774 gcpy              self.dxfline(kh_tool_num, oXpos-r, oYpos+ro, oXpos-r,
                      oYpos+kh_distance)
775 gcpy #right slot
776 gcpy              self.dxfline(kh_tool_num, oXpos+r, oYpos+ro, oXpos+r,
                      oYpos+kh_distance)
777 gcpy #upper left of end of slot
778 gcpy              self.dxfarc(kh_tool_num, oXpos,oYpos+kh_distance,r
                      ,90,180)
779 gcpy #upper right of end of slot
780 gcpy              self.dxfarc(kh_tool_num, oXpos,oYpos+kh_distance,r
                      ,0,90)
781 gcpy #Upper right of entry hole
782 gcpy              self.dxfarc(kh_tool_num, oXpos, oYpos, rt, 0, 90-angle)
783 gcpy #Upper left of entry hole
784 gcpy              self.dxfarc(kh_tool_num, oXpos, oYpos, rt, 90+angle,
                      180)
785 gcpy #             toolpath = toolpath.union(self.cutline(oXpos, oYpos+
            kh_distance, -kh_max_depth))
786 gcpy          elif (kh_angle == 180):
787 gcpy #Lower right of entry hole
788 gcpy              self.dxfarc(kh_tool_num, oXpos,oYpos,self.tool_radius(
                      kh_tool_num, 1),270,360)
789 gcpy #Upper right of entry hole
790 gcpy              self.dxfarc(kh_tool_num, oXpos,oYpos,self.tool_radius(
                      kh_tool_num, 1),0,90)
791 gcpy #Upper left of entry hole
792 gcpy              self.dxfarc(kh_tool_num, oXpos, oYpos, rt, 90, 180-
                      angle)
793 gcpy #Lower left of entry hole
794 gcpy              self.dxfarc(kh_tool_num, oXpos, oYpos, rt, 180+angle,
                      270)
795 gcpy #upper slot
796 gcpy              self.dxfline(kh_tool_num, oXpos-ro, oYpos-r, oXpos-
                      kh_distance, oYpos-r)
797 gcpy #lower slot
798 gcpy              self.dxfline(kh_tool_num, oXpos-ro, oYpos+r, oXpos-
                      kh_distance, oYpos+r)
799 gcpy #upper left of end of slot
800 gcpy              self.dxfarc(kh_tool_num, oXpos-kh_distance,oYpos,r
                      ,90,180)
801 gcpy #lower left of end of slot
802 gcpy              self.dxfarc(kh_tool_num, oXpos-kh_distance,oYpos,r
                      ,180,270)
803 gcpy #             toolpath = toolpath.union(self.cutline(oXpos-
            kh_distance, oYpos, -kh_max_depth))
804 gcpy          elif (kh_angle == 270):
805 gcpy #Upper left of entry hole
806 gcpy              self.dxfarc(kh_tool_num, oXpos,oYpos,self.tool_radius(
                      kh_tool_num, 1),90,180)
807 gcpy #Upper right of entry hole
808 gcpy              self.dxfarc(kh_tool_num, oXpos,oYpos,self.tool_radius(
                      kh_tool_num, 1),0,90)
809 gcpy #left slot
810 gcpy              self.dxfline(kh_tool_num, oXpos-r, oYpos-ro, oXpos-r,
                      oYpos-kh_distance)
```

```
811 gcpy #right slot
812 gcpy             self.dxfline(kh_tool_num, oXpos+r, oYpos-ro, oXpos+r,
                         oYpos-kh_distance)
813 gcpy #lower left of end of slot
814 gcpy             self.dxfarc(kh_tool_num, oXpos,oYpos-kh_distance,r
                         ,180,270)
815 gcpy #lower right of end of slot
816 gcpy             self.dxfarc(kh_tool_num, oXpos,oYpos-kh_distance,r
                         ,270,360)
817 gcpy #lower right of entry hole
818 gcpy             self.dxfarc(kh_tool_num, oXpos, oYpos, rt, 180, 270-
                         angle)
819 gcpy #lower left of entry hole
820 gcpy             self.dxfarc(kh_tool_num, oXpos, oYpos, rt, 270+angle,
                         360)
821 gcpy #            toolpath = toolpath.union(self.cutline(oXpos, oYpos-
        kh_distance, -kh_max_depth))
822 gcpy #        print(self.zpos())
823 gcpy #        self.setxpos(oXpos)
824 gcpy #        self.setypos(oYpos)
825 gcpy #        if self.generatepaths == False:
826 gcpy #            return toolpath
827 gcpy
828 gcpy #  } else if (kh_angle == 90) {
829 gcpy #    //Lower left of entry hole
830 gcpy #    dxfarc(getxpos(),getypos(),9.525/2,180,270, KH_tool_num);
831 gcpy #    //Lower right of entry hole
832 gcpy #    dxfarc(getxpos(),getypos(),9.525/2,270,360, KH_tool_num);
833 gcpy #    //Upper right of entry hole
834 gcpy #    dxfarc(getxpos(),getypos(),9.525/2,0,acos(tool_diameter(
        KH_tool_num, 5)/tool_diameter(KH_tool_num, 1)), KH_tool_num);
835 gcpy #    //Upper left of entry hole
836 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)
        ;
837 gcpy #    //Actual line of cut
838 gcpy #    dxfline(getxpos(),getypos(),getxpos(),getypos()+kh_distance);
839 gcpy #    //upper right of slot
840 gcpy #    dxfarc(getxpos(),getypos()+kh_distance,tool_diameter(
        KH_tool_num, (kh_max_depth+4.36))/2,0,90, KH_tool_num);
841 gcpy #    //upper left of slot
842 gcpy #    dxfarc(getxpos(),getypos()+kh_distance,tool_diameter(
        KH_tool_num, (kh_max_depth+6.35))/2,90,180, KH_tool_num);
843 gcpy #    //right of slot
844 gcpy #    dxfline(
845 gcpy #        getxpos()+tool_diameter(KH_tool_num, (kh_max_depth))/2,
846 gcpy #        getypos()+(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,
847 gcpy #        getxpos()+tool_diameter(KH_tool_num, (kh_max_depth))/2,
848 gcpy #    //end position at top of slot
849 gcpy #        getypos()+kh_distance,
850 gcpy #        KH_tool_num);
851 gcpy #    dxfline(getxpos()-tool_diameter(KH_tool_num, (kh_max_depth))
        /2, getypos()+(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);
852 gcpy #    hull(){
853 gcpy #      translate([xpos(), ypos(), zpos()]){
854 gcpy #        keyhole_shaft(6.35, 9.525);
855 gcpy #      }
856 gcpy #      translate([xpos(), ypos(), zpos()-kh_max_depth]){
857 gcpy #        keyhole_shaft(6.35, 9.525);
858 gcpy #      }
859 gcpy #    }
860 gcpy #    hull(){
861 gcpy #      translate([xpos(), ypos(), zpos()-kh_max_depth]){
862 gcpy #        keyhole_shaft(6.35, 9.525);
863 gcpy #      }
864 gcpy #      translate([xpos(), ypos()+kh_distance, zpos()-kh_max_depth])
        {
865 gcpy #        keyhole_shaft(6.35, 9.525);
866 gcpy #      }
867 gcpy #    }
868 gcpy #    cutwithfeed(getxpos(),getypos(),-kh_max_depth,feed);
869 gcpy #    cutwithfeed(getxpos(),getypos()+kh_distance,-kh_max_depth,feed
        );
```

```
870 gcpy #     setypos(getypos()-kh_distance);
871 gcpy #   } else if (kh_angle == 180) {
872 gcpy #     //Lower right of entry hole
873 gcpy #     dxfarc(getxpos(),getypos(),9.525/2,270,360, KH_tool_num);
874 gcpy #     //Upper right of entry hole
875 gcpy #     dxfarc(getxpos(),getypos(),9.525/2,0,90, KH_tool_num);
876 gcpy #     //Upper left of entry hole
877 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);
878 gcpy #     //Lower left of entry hole
879 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
        );
880 gcpy #     //upper left of slot
881 gcpy #     dxfarc(getxpos()-kh_distance,getypos(),tool_diameter(
        KH_tool_num, (kh_max_depth+6.35))/2,90,180, KH_tool_num);
882 gcpy #     //lower left of slot
883 gcpy #     dxfarc(getxpos()-kh_distance,getypos(),tool_diameter(
        KH_tool_num, (kh_max_depth+6.35))/2,180,270, KH_tool_num);
884 gcpy #     //Actual line of cut
885 gcpy #     dxfline(getxpos(),getypos(),getxpos()-kh_distance,getypos());
886 gcpy #     //upper left slot
887 gcpy #     dxfline(
888 gcpy #         getxpos()-(sqrt((tool_diameter(KH_tool_num,1)^2)-(
        tool_diameter(KH_tool_num,5)^2))/2),
889 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,
890 gcpy #         getxpos()-kh_distance,
891 gcpy #     //end position at top of slot
892 gcpy #         getypos()+tool_diameter(KH_tool_num, (kh_max_depth))/2,
893 gcpy #         KH_tool_num);
894 gcpy #     //lower right slot
895 gcpy #     dxfline(
896 gcpy #         getxpos()-(sqrt((tool_diameter(KH_tool_num,1)^2)-(
        tool_diameter(KH_tool_num,5)^2))/2),
897 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,
898 gcpy #         getxpos()-kh_distance,
899 gcpy #     //end position at top of slot
900 gcpy #         getypos()-tool_diameter(KH_tool_num, (kh_max_depth))/2,
901 gcpy #         KH_tool_num);
902 gcpy #     hull(){
903 gcpy #       translate([xpos(), ypos(), zpos()]){
904 gcpy #         keyhole_shaft(6.35, 9.525);
905 gcpy #       }
906 gcpy #       translate([xpos(), ypos(), zpos()-kh_max_depth]){
907 gcpy #         keyhole_shaft(6.35, 9.525);
908 gcpy #       }
909 gcpy #     }
910 gcpy #     hull(){
911 gcpy #       translate([xpos(), ypos(), zpos()-kh_max_depth]){
912 gcpy #         keyhole_shaft(6.35, 9.525);
913 gcpy #       }
914 gcpy #       translate([xpos()-kh_distance, ypos(), zpos()-kh_max_depth])
        {
915 gcpy #         keyhole_shaft(6.35, 9.525);
916 gcpy #       }
917 gcpy #     }
918 gcpy #     cutwithfeed(getxpos(),getypos(),-kh_max_depth,feed);
919 gcpy #     cutwithfeed(getxpos()-kh_distance,getypos(),-kh_max_depth,feed
        );
920 gcpy #     setxpos(getxpos()+kh_distance);
921 gcpy #   } else if (kh_angle == 270) {
922 gcpy #     //Upper right of entry hole
923 gcpy #     dxfarc(getxpos(),getypos(),9.525/2,0,90, KH_tool_num);
924 gcpy #     //Upper left of entry hole
925 gcpy #     dxfarc(getxpos(),getypos(),9.525/2,90,180, KH_tool_num);
926 gcpy #     //lower right of slot
927 gcpy #     dxfarc(getxpos(),getypos()-kh_distance,tool_diameter(
        KH_tool_num, (kh_max_depth+4.36))/2,270,360, KH_tool_num);
928 gcpy #     //lower left of slot
929 gcpy #     dxfarc(getxpos(),getypos()-kh_distance,tool_diameter(
        KH_tool_num, (kh_max_depth+4.36))/2,180,270, KH_tool_num);
930 gcpy #     //Actual line of cut
931 gcpy #     dxfline(getxpos(),getypos(),getxpos(),getypos()-kh_distance);
932 gcpy #     //right of slot
```

```
933 gcpy #    dxfline(
934 gcpy #        getxpos()+tool_diameter(KH_tool_num, (kh_max_depth))/2,
935 gcpy #        getypos()-(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,
936 gcpy #        getxpos()+tool_diameter(KH_tool_num, (kh_max_depth))/2,
937 gcpy #    //end position at top of slot
938 gcpy #        getypos()-kh_distance,
939 gcpy #        KH_tool_num);
940 gcpy #    //left of slot
941 gcpy #    dxfline(
942 gcpy #        getxpos()-tool_diameter(KH_tool_num, (kh_max_depth))/2,
943 gcpy #        getypos()-(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,
944 gcpy #        getxpos()-tool_diameter(KH_tool_num, (kh_max_depth))/2,
945 gcpy #    //end position at top of slot
946 gcpy #        getypos()-kh_distance,
947 gcpy #        KH_tool_num);
948 gcpy #    //Lower right of entry hole
949 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
                );
950 gcpy #    //Lower left of entry hole
951 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);
952 gcpy #    hull(){
953 gcpy #      translate([xpos(), ypos(), zpos()]){
954 gcpy #        keyhole_shaft(6.35, 9.525);
955 gcpy #      }
956 gcpy #      translate([xpos(), ypos(), zpos()-kh_max_depth]){
957 gcpy #        keyhole_shaft(6.35, 9.525);
958 gcpy #      }
959 gcpy #    }
960 gcpy #    hull(){
961 gcpy #      translate([xpos(), ypos(), zpos()-kh_max_depth]){
962 gcpy #        keyhole_shaft(6.35, 9.525);
963 gcpy #      }
964 gcpy #      translate([xpos(), ypos()-kh_distance, zpos()-kh_max_depth])
                {
965 gcpy #        keyhole_shaft(6.35, 9.525);
966 gcpy #      }
967 gcpy #    }
968 gcpy #    cutwithfeed(getxpos(),getypos(),-kh_max_depth,feed);
969 gcpy #    cutwithfeed(getxpos(),getypos()-kh_distance,-kh_max_depth,feed
                );
970 gcpy #    setypos(getypos()+kh_distance);
971 gcpy #  }
972 gcpy #}
```

3.4.4 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. If in OpenSCAD, the 3D model is returned, causing it to be instantiated on the 3D stage unless the Boolean `generatepaths` is True.

```
803 gcpy     def stockandtoolpaths(self, option = "stockandtoolpaths"):
804 gcpy         if option == "stock":
805 gcpy             if self.generatepaths == False:
806 gcpy                 output(self.stock)
807 gcpy #                print("Outputting stock")
808 gcpy             else:
809 gcpy                 return self.stock
810 gcpy         elif option == "toolpaths":
811 gcpy             if self.generatepaths == False:
812 gcpy                 output(self.toolpaths)
813 gcpy             else:
814 gcpy                 return self.toolpaths
815 gcpy         elif option == "rapids":
816 gcpy             if self.generatepaths == False:
817 gcpy                 output(self.rapids)
818 gcpy             else:
819 gcpy                 return self.rapids
820 gcpy         else:
821 gcpy             part = self.stock.difference(self.toolpaths)
822 gcpy             if self.generatepaths == False:
823 gcpy                 output(part)
```

```
824 gcpy             else:
825 gcpy                 return part
```

```
90 gcpscad module stockandtoolpaths(){
91 gcpscad     gcp.stockandtoolpaths();
92 gcpscad }
93 gcpscad
94 gcpscad module stockwotoolpaths(){
95 gcpscad     gcp.stockandtoolpaths("stock");
96 gcpscad }
97 gcpscad
98 gcpscad module outputtoolpaths(){
99 gcpscad     gcp.stockandtoolpaths("toolpaths");
100 gcpscad }
101 gcpscad
102 gcpscad module outputrapids(){
103 gcpscad     gcp.stockandtoolpaths("rapids");
104 gcpscad }
```

3.5 Output files

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

3.5.1 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(...);
 	
cutwithfeed(...);
	(PREPOSITION FOR RAPID PLUNGE)
	G0X0Y0
	Z0.25
	G1Z0F100
	G1 X109.5 Y75 Z-8.35F400
	Z9

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. One possible option would be to make specialized commands for movement which correspond to the various axis combinations (XYZ, XY, XZ, YZ, X, Y, Z).

3.5.2 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
```

3.5.3 Python and OpenSCAD File Handling

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

writeln

```
826 gcpy     def writegc(self, *arguments):
827 gcpy         if self.generategcode == True:
828 gcpy             line_to_write = ""
829 gcpy             for element in arguments:
830 gcpy                 line_to_write += element
831 gcpy             self.gc.write(line_to_write)
832 gcpy             self.gc.write("\n")
833 gcpy
834 gcpy     def writedxf(self, toolnumber, *arguments):
835 gcpy #         global dxfclosed
836 gcpy         line_to_write = ""
837 gcpy         for element in arguments:
838 gcpy             line_to_write += element
839 gcpy         if self.generatedxf == True:
840 gcpy             if self.dxfclosed == False:
841 gcpy                 self.dxf.write(line_to_write)
842 gcpy                 self.dxf.write("\n")
843 gcpy         if self.generatedxfs == True:
844 gcpy             self.writedxfs(toolnumber, line_to_write)
845 gcpy
846 gcpy     def writedxfs(self, toolnumber, line_to_write):
847 gcpy #         print("Processing writing toolnumber", toolnumber)
848 gcpy #         line_to_write = ""
849 gcpy #         for element in arguments:
850 gcpy #             line_to_write += element
851 gcpy         if (toolnumber == 0):
852 gcpy             return
853 gcpy         elif self.generatedxfs == True:
854 gcpy             if (self.large_square_tool_num == toolnumber):
855 gcpy                 self.dxflgsq.write(line_to_write)
856 gcpy                 self.dxflgsq.write("\n")
857 gcpy             if (self.small_square_tool_num == toolnumber):
858 gcpy                 self.dxfsmsq.write(line_to_write)
859 gcpy                 self.dxfsmsq.write("\n")
860 gcpy             if (self.large_ball_tool_num == toolnumber):
861 gcpy                 self.dxflgbl.write(line_to_write)
862 gcpy                 self.dxflgbl.write("\n")
863 gcpy             if (self.small_ball_tool_num == toolnumber):
864 gcpy                 self.dxfsmbl.write(line_to_write)
865 gcpy                 self.dxfsmbl.write("\n")
866 gcpy             if (self.large_V_tool_num == toolnumber):
867 gcpy                 self.dxflgV.write(line_to_write)
868 gcpy                 self.dxflgV.write("\n")
869 gcpy             if (self.small_V_tool_num == toolnumber):
870 gcpy                 self.dxfsmV.write(line_to_write)
871 gcpy                 self.dxfsmV.write("\n")
872 gcpy             if (self.DT_tool_num == toolnumber):
873 gcpy                 self.dxfDT.write(line_to_write)
874 gcpy                 self.dxfDT.write("\n")
875 gcpy             if (self.KH_tool_num == toolnumber):
876 gcpy                 self.dxfKH.write(line_to_write)
877 gcpy                 self.dxfKH.write("\n")
878 gcpy             if (self.Roundover_tool_num == toolnumber):
879 gcpy                 self.dxfRt.write(line_to_write)
880 gcpy                 self.dxfRt.write("\n")
881 gcpy             if (self.MISC_tool_num == toolnumber):
882 gcpy                 self.dxfMt.write(line_to_write)
883 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 opendxffile

For writing to files it will be necessary to have commands for opening the files `opengcodefile` and `opendxffile` and setting 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:

```
140 gcpscad module opendxffile(basefilename){
141 gcpscad     gcp.opendxffile(basefilename);
142 gcpscad }
143 gcpscad
144 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) {
145 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)
                ;
146 gcpscad }
```

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

```
148 gcpscad module opengcodefile(basefilename, currenttoolnum, toolradius,
            plunge, feed, speed) {
149 gcpscad     gcp.opengcodefile(basefilename, currenttoolnum, toolradius,
                plunge, feed, speed);
150 gcpscad }
```

and Python:

```
884 gcpy     def opengcodefile(self, basefilename = "export",
885 gcpy                       currenttoolnum = 102,
886 gcpy                       toolradius = 3.175,
887 gcpy                       plunge = 400,
888 gcpy                       feed = 1600,
889 gcpy                       speed = 10000
890 gcpy                       ):
891 gcpy         self.basefilename = basefilename
892 gcpy         self.currenttoolnum = currenttoolnum
893 gcpy         self.toolradius = toolradius
894 gcpy         self.plunge = plunge
895 gcpy         self.feed = feed
896 gcpy         self.speed = speed
897 gcpy         if self.generategcode == True:
898 gcpy             self.gcodefilename = basefilename + ".nc"
899 gcpy             self.gc = open(self.gcodefilename, "w")
900 gcpy
901 gcpy     def opendxffile(self, basefilename = "export"):
902 gcpy         self.basefilename = basefilename
903 gcpy #         global generatedxfs
904 gcpy #         global dxfclosed
905 gcpy         self.dxfclosed = False
906 gcpy         if self.generatedxf == True:
907 gcpy             self.generatedxfs = False
908 gcpy             self.dxffilename = basefilename + ".dxf"
909 gcpy             self.dxf = open(self.dxffilename, "w")
910 gcpy             self.dxfpreamble(-1)
911 gcpy
912 gcpy     def opendxffiles(self, basefilename = "export",
913 gcpy                       large_square_tool_num = 0,
914 gcpy                       small_square_tool_num = 0,
915 gcpy                       large_ball_tool_num = 0,
916 gcpy                       small_ball_tool_num = 0,
917 gcpy                       large_V_tool_num = 0,
918 gcpy                       small_V_tool_num = 0,
919 gcpy                       DT_tool_num = 0,
920 gcpy                       KH_tool_num = 0,
921 gcpy                       Roundover_tool_num = 0,
922 gcpy                       MISC_tool_num = 0):
923 gcpy #         global generatedxfs
924 gcpy         self.basefilename = basefilename
925 gcpy         self.generatedxfs = True
926 gcpy         self.large_square_tool_num = large_square_tool_num
927 gcpy         self.small_square_tool_num = small_square_tool_num
928 gcpy         self.large_ball_tool_num = large_ball_tool_num
929 gcpy         self.small_ball_tool_num = small_ball_tool_num
930 gcpy         self.large_V_tool_num = large_V_tool_num
931 gcpy         self.small_V_tool_num = small_V_tool_num
```

```
932 gcpy         self.DT_tool_num = DT_tool_num
933 gcpy         self.KH_tool_num = KH_tool_num
934 gcpy         self.Roundover_tool_num = Roundover_tool_num
935 gcpy         self.MISC_tool_num = MISC_tool_num
936 gcpy         if self.generatedxf == True:
937 gcpy             if (large_square_tool_num > 0):
938 gcpy                 self.dxflgsqfilename = basefilename + str(
                             large_square_tool_num) + ".dxf"
939 gcpy #                print("Opening ", str(self.dxflgsqfilename))
940 gcpy                 self.dxflgsq = open(self.dxflgsqfilename, "w")
941 gcpy             if (small_square_tool_num > 0):
942 gcpy #                print("Opening small square")
943 gcpy                 self.dxfsmsqfilename = basefilename + str(
                             small_square_tool_num) + ".dxf"
944 gcpy                 self.dxfsmsq = open(self.dxfsmsqfilename, "w")
945 gcpy             if (large_ball_tool_num > 0):
946 gcpy #                print("Opening large ball")
947 gcpy                 self.dxflgblfilename = basefilename + str(
                             large_ball_tool_num) + ".dxf"
948 gcpy                 self.dxflgbl = open(self.dxflgblfilename, "w")
949 gcpy             if (small_ball_tool_num > 0):
950 gcpy #                print("Opening small ball")
951 gcpy                 self.dxfsmblfilename = basefilename + str(
                             small_ball_tool_num) + ".dxf"
952 gcpy                 self.dxfsmbl = open(self.dxfsmblfilename, "w")
953 gcpy             if (large_V_tool_num > 0):
954 gcpy #                print("Opening large V")
955 gcpy                 self.dxflgVfilename = basefilename + str(
                             large_V_tool_num) + ".dxf"
956 gcpy                 self.dxflgV = open(self.dxflgVfilename, "w")
957 gcpy             if (small_V_tool_num > 0):
958 gcpy #                print("Opening small V")
959 gcpy                 self.dxfsmVfilename = basefilename + str(
                             small_V_tool_num) + ".dxf"
960 gcpy                 self.dxfsmV = open(self.dxfsmVfilename, "w")
961 gcpy             if (DT_tool_num > 0):
962 gcpy #                print("Opening DT")
963 gcpy                 self.dxfDTfilename = basefilename + str(DT_tool_num
                             ) + ".dxf"
964 gcpy                 self.dxfDT = open(self.dxfDTfilename, "w")
965 gcpy             if (KH_tool_num > 0):
966 gcpy #                print("Opening KH")
967 gcpy                 self.dxfKHfilename = basefilename + str(KH_tool_num
                             ) + ".dxf"
968 gcpy                 self.dxfKH = open(self.dxfKHfilename, "w")
969 gcpy             if (Roundover_tool_num > 0):
970 gcpy #                print("Opening Rt")
971 gcpy                 self.dxfRtfilename = basefilename + str(
                             Roundover_tool_num) + ".dxf"
972 gcpy                 self.dxfRt = open(self.dxfRtfilename, "w")
973 gcpy             if (MISC_tool_num > 0):
974 gcpy #                print("Opening Mt")
975 gcpy                 self.dxfMtfilename = basefilename + str(
                             MISC_tool_num) + ".dxf"
976 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:

```
963 gcpy             if (large_square_tool_num > 0):
964 gcpy                 self.dxfpreamble(large_square_tool_num)
965 gcpy             if (small_square_tool_num > 0):
966 gcpy                 self.dxfpreamble(small_square_tool_num)
967 gcpy             if (large_ball_tool_num > 0):
968 gcpy                 self.dxfpreamble(large_ball_tool_num)
969 gcpy             if (small_ball_tool_num > 0):
970 gcpy                 self.dxfpreamble(small_ball_tool_num)
971 gcpy             if (large_V_tool_num > 0):
972 gcpy                 self.dxfpreamble(large_V_tool_num)
973 gcpy             if (small_V_tool_num > 0):
974 gcpy                 self.dxfpreamble(small_V_tool_num)
975 gcpy             if (DT_tool_num > 0):
976 gcpy                 self.dxfpreamble(DT_tool_num)
977 gcpy             if (KH_tool_num > 0):
978 gcpy                 self.dxfpreamble(KH_tool_num)
979 gcpy             if (Roundover_tool_num > 0):
980 gcpy                 self.dxfpreamble(Roundover_tool_num)
981 gcpy             if (MISC_tool_num > 0):
```

```
982 gcpy                    self.dxfpreamble(MISC_tool_num)
```

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

3.5.3.1 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:

- Ball nose, large (lgbl) `writedxflgbl` — writedxflgbl
- Ball nose, small (smbl) `writedxfsmbl` — writedxfsmbl
- Square, large (lgsq) `writedxflgsq` — writedxflgsq
- Square, small (smsq) `writedxfsmsq` — writedxfsmsq
- V, large (lgV) `writedxflgV` — writedxflgV
- V, small (smV) `writedxfsmV` — writedxfsmV
- Keyhole (KH) `writedxfKH` — writedxfKH
- Dovetail (DT) `writedxfDT` — 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.

```
984 gcpy      def dxfpreamble(self, tn):
985 gcpy #          self.writedxf(tn,str(tn))
986 gcpy           self.writedxf(tn,"0")
987 gcpy           self.writedxf(tn,"SECTION")
988 gcpy           self.writedxf(tn,"2")
989 gcpy           self.writedxf(tn,"ENTITIES")
```

DXF Lines and Arcs There are two notable elements which may be written to a DXF:

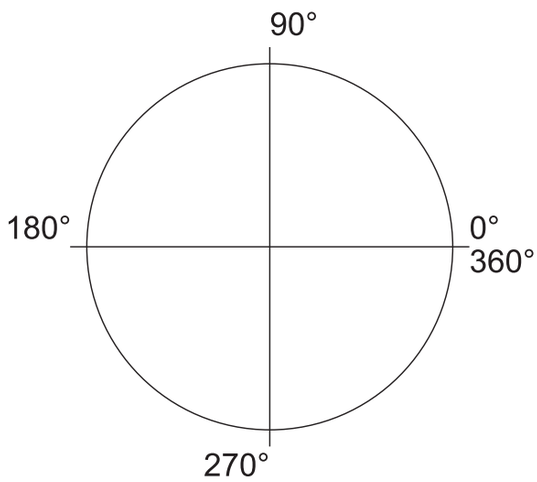
dxfline

- a line `dxfline`

dxfarc

- ARC — a notable option would be for the arc to close on itself, creating a circle: `dxfarc`

DXF orders arcs counter-clockwise:



Note that arcs of greater than 90 degrees are not rendered accurately, 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);
```

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 Keyhhole

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.

```
991 gcpy      def dxfline(self, tn, xbegin,ybegin,xend,yend):
992 gcpy          self.writedxf(tn,"0")
993 gcpy          self.writedxf(tn,"LWPOLYLINE")
994 gcpy          self.writedxf(tn,"90")
995 gcpy          self.writedxf(tn,"2")
996 gcpy          self.writedxf(tn,"70")
997 gcpy          self.writedxf(tn,"0")
998 gcpy          self.writedxf(tn,"43")
999 gcpy          self.writedxf(tn,"0")
1000 gcpy         self.writedxf(tn,"10")
1001 gcpy         self.writedxf(tn,str(xbegin))
1002 gcpy         self.writedxf(tn,"20")
1003 gcpy         self.writedxf(tn,str(ybegin))
1004 gcpy         self.writedxf(tn,"10")
1005 gcpy         self.writedxf(tn,str(xend))
1006 gcpy         self.writedxf(tn,"20")
1007 gcpy         self.writedxf(tn,str(yend))
```

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).

```
1009 gcpy     def dxfarc(self, tn, xcenter, ycenter, radius, anglebegin,
                  endangle):
1010 gcpy         if (self.generatedxf == True):
1011 gcpy             self.writedxf(tn, "0")
1012 gcpy             self.writedxf(tn, "ARC")
1013 gcpy             self.writedxf(tn, "10")
1014 gcpy             self.writedxf(tn, str(xcenter))
1015 gcpy             self.writedxf(tn, "20")
1016 gcpy             self.writedxf(tn, str(ycenter))
1017 gcpy             self.writedxf(tn, "40")
1018 gcpy             self.writedxf(tn, str(radius))
1019 gcpy             self.writedxf(tn, "50")
1020 gcpy             self.writedxf(tn, str(anglebegin))
1021 gcpy             self.writedxf(tn, "51")
1022 gcpy             self.writedxf(tn, str(endangle))
1023 gcpy
1024 gcpy     def gcodearc(self, tn, xcenter, ycenter, radius, anglebegin,
                  endangle):
1025 gcpy         if (self.generategcode == True):
1026 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.

```
1028 gcpy     def cutarcNECCdxf(self, ex, ey, ez, xcenter, ycenter, radius):
1029 gcpy #         global toolpath
1030 gcpy #         toolpath = self.currenttool()
1031 gcpy #         toolpath = toolpath.translate([self.xpos(),self.ypos(),
          self.zpos()])
1032 gcpy         self.dxfarc(self.currenttoolnumber(), xcenter,ycenter,
                     radius,0,90)
1033 gcpy         if (self.zpos == ez):
1034 gcpy             self.settzpos(0)
1035 gcpy         else:
1036 gcpy             self.settzpos((self.zpos()-ez)/90)
1037 gcpy #         self.setxpos(ex)
1038 gcpy #         self.setypos(ey)
1039 gcpy #         self.setzpos(ez)
1040 gcpy         if self.generatepaths == True:
```

```
1041 gcpy             print("Unioning␣cutarcNECCdxf␣toolpath")
1042 gcpy             self.arcloop(1,90, xcenter, ycenter, radius)
1043 gcpy #             self.toolpaths = self.toolpaths.union(toolpath)
1044 gcpy         else:
1045 gcpy             toolpath = self.arcloop(1,90, xcenter, ycenter, radius)
1046 gcpy #             print("Returning cutarcNECCdxf toolpath")
1047 gcpy             return toolpath
1048 gcpy
1049 gcpy     def cutarcNWCCdxf(self, ex, ey, ez, xcenter, ycenter, radius):
1050 gcpy #         global toolpath
1051 gcpy #         toolpath = self.currenttool()
1052 gcpy #         toolpath = toolpath.translate([self.xpos(),self.ypos(),
        self.zpos()])
1053 gcpy         self.dxfarc(self.currenttoolnumber(), xcenter,ycenter,
                    radius,90,180)
1054 gcpy         if (self.zpos == ez):
1055 gcpy             self.settzpos(0)
1056 gcpy         else:
1057 gcpy             self.settzpos((self.zpos()-ez)/90)
1058 gcpy #         self.setxpos(ex)
1059 gcpy #         self.setypos(ey)
1060 gcpy #         self.setzpos(ez)
1061 gcpy         if self.generatepaths == True:
1062 gcpy             self.arcloop(91,180, xcenter, ycenter, radius)
1063 gcpy #             self.toolpaths = self.toolpaths.union(toolpath)
1064 gcpy         else:
1065 gcpy             toolpath = self.arcloop(91,180, xcenter, ycenter,
                        radius)
1066 gcpy             return toolpath
1067 gcpy
1068 gcpy     def cutarcSWCCdxf(self, ex, ey, ez, xcenter, ycenter, radius):
1069 gcpy #         global toolpath
1070 gcpy #         toolpath = self.currenttool()
1071 gcpy #         toolpath = toolpath.translate([self.xpos(),self.ypos(),
        self.zpos()])
1072 gcpy         self.dxfarc(self.currenttoolnumber(), xcenter,ycenter,
                    radius,180,270)
1073 gcpy         if (self.zpos == ez):
1074 gcpy             self.settzpos(0)
1075 gcpy         else:
1076 gcpy             self.settzpos((self.zpos()-ez)/90)
1077 gcpy #         self.setxpos(ex)
1078 gcpy #         self.setypos(ey)
1079 gcpy #         self.setzpos(ez)
1080 gcpy         if self.generatepaths == True:
1081 gcpy             self.arcloop(181,270, xcenter, ycenter, radius)
1082 gcpy #             self.toolpaths = self.toolpaths.union(toolpath)
1083 gcpy         else:
1084 gcpy             toolpath = self.arcloop(181,270, xcenter, ycenter,
                        radius)
1085 gcpy             return toolpath
1086 gcpy
1087 gcpy     def cutarcSECCdxf(self, ex, ey, ez, xcenter, ycenter, radius):
1088 gcpy #         global toolpath
1089 gcpy #         toolpath = self.currenttool()
1090 gcpy #         toolpath = toolpath.translate([self.xpos(),self.ypos(),
        self.zpos()])
1091 gcpy         self.dxfarc(self.currenttoolnumber(), xcenter,ycenter,
                    radius,270,360)
1092 gcpy         if (self.zpos == ez):
1093 gcpy             self.settzpos(0)
1094 gcpy         else:
1095 gcpy             self.settzpos((self.zpos()-ez)/90)
1096 gcpy #         self.setxpos(ex)
1097 gcpy #         self.setypos(ey)
1098 gcpy #         self.setzpos(ez)
1099 gcpy         if self.generatepaths == True:
1100 gcpy             self.arcloop(271,360, xcenter, ycenter, radius)
1101 gcpy #             self.toolpaths = self.toolpaths.union(toolpath)
1102 gcpy         else:
1103 gcpy             toolpath = self.arcloop(271,360, xcenter, ycenter,
                        radius)
1104 gcpy             return toolpath
1105 gcpy
1106 gcpy     def cutarcNECWdxf(self, ex, ey, ez, xcenter, ycenter, radius):
1107 gcpy #         global toolpath
1108 gcpy #         toolpath = self.currenttool()
1109 gcpy #         toolpath = toolpath.translate([self.xpos(),self.ypos(),
```

```
            self.zpos()])
1110 gcpy           self.dxfarc(self.currenttoolnumber(), xcenter,ycenter,
                        radius,0,90)
1111 gcpy           if (self.zpos == ez):
1112 gcpy               self.settzpos(0)
1113 gcpy           else:
1114 gcpy               self.settzpos((self.zpos()-ez)/90)
1115 gcpy #          self.setxpos(ex)
1116 gcpy #          self.setypos(ey)
1117 gcpy #          self.setzpos(ez)
1118 gcpy           if self.generatepaths == True:
1119 gcpy               self.narcloop(89,0, xcenter, ycenter, radius)
1120 gcpy #              self.toolpaths = self.toolpaths.union(toolpath)
1121 gcpy           else:
1122 gcpy               toolpath = self.narcloop(89,0, xcenter, ycenter, radius
                           )
1123 gcpy               return toolpath
1124 gcpy
1125 gcpy       def cutarcSECWdxf(self, ex, ey, ez, xcenter, ycenter, radius):
1126 gcpy #          global toolpath
1127 gcpy #          toolpath = self.currenttool()
1128 gcpy #          toolpath = toolpath.translate([self.xpos(),self.ypos(),
            self.zpos()])
1129 gcpy           self.dxfarc(self.currenttoolnumber(), xcenter,ycenter,
                        radius,270,360)
1130 gcpy           if (self.zpos == ez):
1131 gcpy               self.settzpos(0)
1132 gcpy           else:
1133 gcpy               self.settzpos((self.zpos()-ez)/90)
1134 gcpy #          self.setxpos(ex)
1135 gcpy #          self.setypos(ey)
1136 gcpy #          self.setzpos(ez)
1137 gcpy           if self.generatepaths == True:
1138 gcpy               self.narcloop(359,270, xcenter, ycenter, radius)
1139 gcpy #              self.toolpaths = self.toolpaths.union(toolpath)
1140 gcpy           else:
1141 gcpy               toolpath = self.narcloop(359,270, xcenter, ycenter,
                           radius)
1142 gcpy               return toolpath
1143 gcpy
1144 gcpy       def cutarcSWCWdxf(self, ex, ey, ez, xcenter, ycenter, radius):
1145 gcpy #          global toolpath
1146 gcpy #          toolpath = self.currenttool()
1147 gcpy #          toolpath = toolpath.translate([self.xpos(),self.ypos(),
            self.zpos()])
1148 gcpy           self.dxfarc(self.currenttoolnumber(), xcenter,ycenter,
                        radius,180,270)
1149 gcpy           if (self.zpos == ez):
1150 gcpy               self.settzpos(0)
1151 gcpy           else:
1152 gcpy               self.settzpos((self.zpos()-ez)/90)
1153 gcpy #          self.setxpos(ex)
1154 gcpy #          self.setypos(ey)
1155 gcpy #          self.setzpos(ez)
1156 gcpy           if self.generatepaths == True:
1157 gcpy               self.narcloop(269,180, xcenter, ycenter, radius)
1158 gcpy #              self.toolpaths = self.toolpaths.union(toolpath)
1159 gcpy           else:
1160 gcpy               toolpath = self.narcloop(269,180, xcenter, ycenter,
                           radius)
1161 gcpy               return toolpath
1162 gcpy
1163 gcpy       def cutarcNWCWdxf(self, ex, ey, ez, xcenter, ycenter, radius):
1164 gcpy #          global toolpath
1165 gcpy #          toolpath = self.currenttool()
1166 gcpy #          toolpath = toolpath.translate([self.xpos(),self.ypos(),
            self.zpos()])
1167 gcpy           self.dxfarc(self.currenttoolnumber(), xcenter,ycenter,
                        radius,90,180)
1168 gcpy           if (self.zpos == ez):
1169 gcpy               self.settzpos(0)
1170 gcpy           else:
1171 gcpy               self.settzpos((self.zpos()-ez)/90)
1172 gcpy #          self.setxpos(ex)
1173 gcpy #          self.setypos(ey)
1174 gcpy #          self.setzpos(ez)
1175 gcpy           if self.generatepaths == True:
1176 gcpy               self.narcloop(179,90, xcenter, ycenter, radius)
```

```
1177 gcpy #             self.toolpaths = self.toolpaths.union(toolpath)
1178 gcpy         else:
1179 gcpy             toolpath = self.narcloop(179,90, xcenter, ycenter,
                         radius)
1180 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
```

```
1148 gcpy     def arcCCgc(self, ex, ey, ez, xcenter, ycenter, radius):
1149 gcpy         self.writegc("G03␣X", str(ex), "␣Y", str(ey), "␣Z", str(ez)
                     , "␣R", str(radius))
1150 gcpy
1151 gcpy     def arcCWgc(self, ex, ey, ez, xcenter, ycenter, radius):
1152 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:

```
1154 gcpy     def cutarcNECCdxfgc(self, ex, ey, ez, xcenter, ycenter, radius)
                 :
1155 gcpy         self.arcCCgc(ex, ey, ez, xcenter, ycenter, radius)
1156 gcpy         if self.generatepaths == True:
1157 gcpy             self.cutarcNECCdxf(ex, ey, ez, xcenter, ycenter, radius
                         )
1158 gcpy         else:
1159 gcpy             return self.cutarcNECCdxf(ex, ey, ez, xcenter, ycenter,
                         radius)
1160 gcpy
1161 gcpy     def cutarcNWCCdxfgc(self, ex, ey, ez, xcenter, ycenter, radius)
                 :
1162 gcpy         self.arcCCgc(ex, ey, ez, xcenter, ycenter, radius)
1163 gcpy         if self.generatepaths == False:
1164 gcpy             return self.cutarcNWCCdxf(ex, ey, ez, xcenter, ycenter,
                         radius)
1165 gcpy
1166 gcpy     def cutarcSWCCdxfgc(self, ex, ey, ez, xcenter, ycenter, radius)
                 :
1167 gcpy         self.arcCCgc(ex, ey, ez, xcenter, ycenter, radius)
1168 gcpy         if self.generatepaths == False:
1169 gcpy             return self.cutarcSWCCdxf(ex, ey, ez, xcenter, ycenter,
                         radius)
1170 gcpy
1171 gcpy     def cutarcSECCdxfgc(self, ex, ey, ez, xcenter, ycenter, radius)
                 :
1172 gcpy         self.arcCCgc(ex, ey, ez, xcenter, ycenter, radius)
1173 gcpy         if self.generatepaths == False:
1174 gcpy             return self.cutarcSECCdxf(ex, ey, ez, xcenter, ycenter,
                         radius)
1175 gcpy
1176 gcpy     def cutarcNECWdxfgc(self, ex, ey, ez, xcenter, ycenter, radius)
                 :
1177 gcpy         self.arcCWgc(ex, ey, ez, xcenter, ycenter, radius)
1178 gcpy         if self.generatepaths == False:
1179 gcpy             return self.cutarcNECWdxf(ex, ey, ez, xcenter, ycenter,
                         radius)
1180 gcpy
1181 gcpy     def cutarcSECWdxfgc(self, ex, ey, ez, xcenter, ycenter, radius)
                 :
1182 gcpy         self.arcCWgc(ex, ey, ez, xcenter, ycenter, radius)
1183 gcpy         if self.generatepaths == False:
1184 gcpy             return self.cutarcSECWdxf(ex, ey, ez, xcenter, ycenter,
                         radius)
1185 gcpy
1186 gcpy     def cutarcSWCWdxfgc(self, ex, ey, ez, xcenter, ycenter, radius)
                 :
1187 gcpy         self.arcCWgc(ex, ey, ez, xcenter, ycenter, radius)
1188 gcpy         if self.generatepaths == False:
1189 gcpy             return self.cutarcSWCWdxf(ex, ey, ez, xcenter, ycenter,
                         radius)
1190 gcpy
1191 gcpy     def cutarcNWCWdxfgc(self, ex, ey, ez, xcenter, ycenter, radius)
                 :
```

```
1192 gcpy           self.arcCWgc(ex, ey, ez, xcenter, ycenter, radius)
1193 gcpy           if self.generatepaths == False:
1194 gcpy               return self.cutarcNWCWdxf(ex, ey, ez, xcenter, ycenter,
                         radius)
```

```
152 gcpscad module cutarcNECCdxfgc(ex, ey, ez, xcenter, ycenter, radius){
153 gcpscad     gcp.cutarcNECCdxfgc(ex, ey, ez, xcenter, ycenter, radius);
154 gcpscad }
155 gcpscad
156 gcpscad module cutarcNWCCdxfgc(ex, ey, ez, xcenter, ycenter, radius){
157 gcpscad     gcp.cutarcNWCCdxfgc(ex, ey, ez, xcenter, ycenter, radius);
158 gcpscad }
159 gcpscad
160 gcpscad module cutarcSWCCdxfgc(ex, ey, ez, xcenter, ycenter, radius){
161 gcpscad     gcp.cutarcSWCCdxfgc(ex, ey, ez, xcenter, ycenter, radius);
162 gcpscad }
163 gcpscad
164 gcpscad module cutarcSECCdxfgc(ex, ey, ez, xcenter, ycenter, radius){
165 gcpscad     gcp.cutarcSECCdxfgc(ex, ey, ez, xcenter, ycenter, radius);
166 gcpscad }
```

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

closegcodefile
closedxffile

```
1186 gcpy       def dxfpostamble(self,tn):
1187 gcpy  #         self.writedxf(tn,str(tn))
1188 gcpy           self.writedxf(tn,"0")
1189 gcpy           self.writedxf(tn,"ENDSEC")
1190 gcpy           self.writedxf(tn,"0")
1191 gcpy           self.writedxf(tn,"EOF")
```

```
1193 gcpy       def gcodepostamble(self):
1194 gcpy           self.writegc("Z12.700")
1195 gcpy           self.writegc("M05")
1196 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.

```
1198 gcpy       def closegcodefile(self):
1199 gcpy           self.gcodepostamble()
1200 gcpy           self.gc.close()
1201 gcpy
1202 gcpy       def closedxffile(self):
1203 gcpy           if self.generatedxf == True:
1204 gcpy  #             global dxfclosed
1205 gcpy               self.dxfpostamble(-1)
1206 gcpy  #             self.dxfclosed = True
1207 gcpy               self.dxf.close()
1208 gcpy
1209 gcpy       def closedxffiles(self):
1210 gcpy           if self.generatedxfs == True:
1211 gcpy               if (self.large_square_tool_num > 0):
1212 gcpy                   self.dxfpostamble(self.large_square_tool_num)
1213 gcpy               if (self.small_square_tool_num > 0):
1214 gcpy                   self.dxfpostamble(self.small_square_tool_num)
1215 gcpy               if (self.large_ball_tool_num > 0):
1216 gcpy                   self.dxfpostamble(self.large_ball_tool_num)
1217 gcpy               if (self.small_ball_tool_num > 0):
1218 gcpy                   self.dxfpostamble(self.small_ball_tool_num)
1219 gcpy               if (self.large_V_tool_num > 0):
1220 gcpy                   self.dxfpostamble(self.large_V_tool_num)
1221 gcpy               if (self.small_V_tool_num > 0):
1222 gcpy                   self.dxfpostamble(self.small_V_tool_num)
1223 gcpy               if (self.DT_tool_num > 0):
1224 gcpy                   self.dxfpostamble(self.DT_tool_num)
1225 gcpy               if (self.KH_tool_num > 0):
1226 gcpy                   self.dxfpostamble(self.KH_tool_num)
1227 gcpy               if (self.Roundover_tool_num > 0):
1228 gcpy                   self.dxfpostamble(self.Roundover_tool_num)
1229 gcpy               if (self.MISC_tool_num > 0):
```

```
1230 gcpy                 self.dxfpostamble(self.MISC_tool_num)
1231 gcpy
1232 gcpy             if (self.large_square_tool_num > 0):
1233 gcpy                 self.dxflgsq.close()
1234 gcpy             if (self.small_square_tool_num > 0):
1235 gcpy                 self.dxfsmsq.close()
1236 gcpy             if (self.large_ball_tool_num > 0):
1237 gcpy                 self.dxflgbl.close()
1238 gcpy             if (self.small_ball_tool_num > 0):
1239 gcpy                 self.dxfsmbl.close()
1240 gcpy             if (self.large_V_tool_num > 0):
1241 gcpy                 self.dxflgV.close()
1242 gcpy             if (self.small_V_tool_num > 0):
1243 gcpy                 self.dxfsmV.close()
1244 gcpy             if (self.DT_tool_num > 0):
1245 gcpy                 self.dxfDT.close()
1246 gcpy             if (self.KH_tool_num > 0):
1247 gcpy                 self.dxfKH.close()
1248 gcpy             if (self.Roundover_tool_num > 0):
1249 gcpy                 self.dxfRt.close()
1250 gcpy             if (self.MISC_tool_num > 0):
1251 gcpy                 self.dxfMt.close()
```

closegcodefile
closedxffile
dxfpostamble

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

```
202 gcpscad
203 gcpscad module closegcodefile(){
204 gcpscad     gcp.closegcodefile();
205 gcpscad }
206 gcpscad
207 gcpscad module closedxffiles(){
208 gcpscad     gcp.closedxffiles();
209 gcpscad }
210 gcpscad
211 gcpscad module closedxffile(){
212 gcpscad     gcp.closedxffile();
213 gcpscad }
```

4 Notes

Other Resources

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)

Adding a Command Glossary may be a useful addition or alternative to the Index.

Holidays

Holidays are from https://nationaltoday.com/

DXFs

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

Future

Images

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

Import G-code

Use a tool to read in a G-code file, then create a 3D model which would serve as a preview of the cut?

- https://stackoverflow.com/questions/34638372/simple-python-program-to-read-gcode-file
- https://pypi.org/project/gcodeparser/
- https://github.com/fragmuffin/pygcode/wiki

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

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

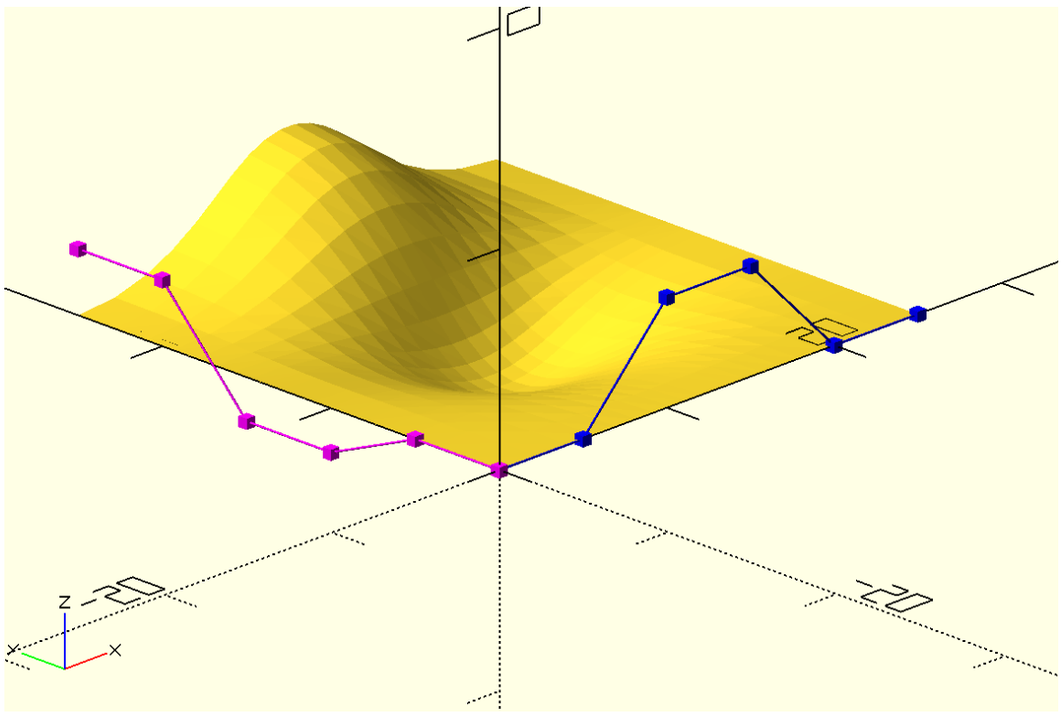
it should be possible to have three Bézier curves (left-most/right-most or front-back or top/bottom for two, and a mid-line for the third), so a region which can be so represented would be definable by:

```
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/

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

Index

Routines

Variables