

The gcodepreview OpenSCAD library*

Author: William F. Adams
willadams at aol dot com

2024/09/08

Abstract

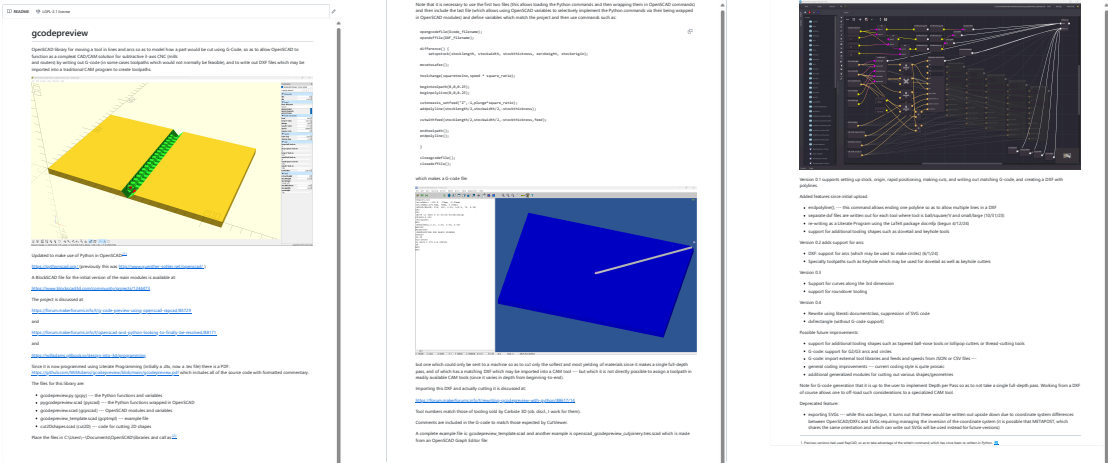
The gcodepreview library allows using OpenPythonSCAD to move a tool in lines and arcs and output dxf and G-code files so as to work as a CAD/CAM program for CNC.

Contents

1	readme.md	2
2	gcodepreview	5
2.1	Position and Variables	5
2.2	Modules	6
2.2.1	Initial Modules	7
2.3	Tools and Changes	11
2.3.1	toolchange	11
2.3.1.1	Normal Tooling	11
2.3.1.2	Tooling for Keyhole Toolpaths	12
2.3.1.3	Thread mills	13
2.3.1.4	Roundover tooling	13
2.3.1.5	Selecting Tools	13
2.3.2	3D Shapes for Tools	14
2.3.2.1	Normal toolshapes	14
2.3.2.2	Concave toolshapes	15
2.3.3	tooldiameter	16
2.3.4	Feeds and Speeds	17
2.4	File Handling	18
2.4.1	Writing to files	20
2.4.1.1	Beginning Writing to DXFs	22
2.4.1.2	DXF Lines and Arcs	23
2.5	Movement and Cutting	27
3	Cutting shapes, cut2Dshapes, and expansion	29
3.1	Arcs for toolpaths and DXFs	31
3.2	Keyhole toolpath and undercut tooling	33
3.3	Shapes and tool movement	37
3.3.1	Generalized commands and cuts	37
3.3.1.1	begincutdxf	37
3.3.1.2	Rectangles	37
3.4	Expansion	39
4	gcodepreviewtemplate.scad	39
4.1	G-code and modules and commands	40
4.2	DXF	41
5	Future	42
5.1	Images	42
5.2	Generalized DXF creation	42
5.3	Import G-code	42
5.4	Bézier curves in 2 dimensions	42
5.5	Bézier curves in 3 dimensions	42
6	Other Resources	43
	Index	45
	Routines	45
	Variables	46

*This file (gcodepreview) has version number vo.61, last revised 2024/09/08.

1 **readme.md**



```
1 rdme # gcodepreview
2 rdme
3 rdme OpenSCAD library for moving a tool in lines and arcs
4 rdme so as to model how a part would be cut using G-Code,
5 rdme so as to allow OpenSCAD to function as a compleat
6 rdme CAD/CAM solution for subtractive 3-axis CNC (mills
7 rdme and routers) by writing out G-code (in some cases
8 rdme toolpaths which would not normally be feasible),
9 rdme and to write out DXF files which may be imported
10 rdme into a traditional CAM program to create toolpaths.
11 rdme
12 rdme ![OpenSCAD Cut Joinery Module](https://raw.githubusercontent.com/
13 rdme WillAdams/gcodepreview/main/openscad_cutjoinery.png?raw=true)
14 rdme Updated to make use of Python in OpenSCAD:[^rapcad]
15 rdme
16 rdme [^rapcad]: Previous versions had used RapCAD, so as to take
17 rdme advantage of the writeln command, which has since been re-
18 rdme written in Python.
19 rdme
20 rdme https://pythonscad.org/ (previously this was http://www.guenther-
21 rdme sohler.net/openscad/ )
22 rdme
23 rdme A BlockSCAD file for the initial version of the
24 rdme main modules is available at:
25 rdme https://www.blockscad3d.com/community/projects/1244473
26 rdme
27 rdme The project is discussed at:
28 rdme https://forum.makerforums.info/t/g-code-preview-using-openscad-
29 rdme rapcad/85729
30 rdme
31 rdme https://forum.makerforums.info/t/openscad-and-python-looking-to-
32 rdme finally-be-resolved/88171
33 rdme
34 rdme
35 rdme https://willadams.gitbook.io/design-into-3d/programming
36 rdme
37 rdme Since it is now programmed using Literate Programming
38 rdme (initially a .dtx, now a .tex file) there is a PDF:
39 rdme https://github.com/WillAdams/gcodepreview/blob/main/gcodepreview.
40 rdme pdf
41 rdme which includes all of the source code with formatted
42 rdme commentary.
43 rdme
44 rdme The files for this library are:
45 rdme - gcodepreview.py (gcpy) --- the Python functions and variables
46 rdme - pygcodepreview.scad (pyscad) --- the Python functions wrapped in
47 rdme OpenSCAD
48 rdme - gcodepreview.scad (gcpscad) --- OpenSCAD modules and variables
49 rdme - gcodepreview_template.scad (gcptmpl) --- example file
50 rdme - cut2Dshapes.scad (cut2D) --- code for cutting 2D shapes
```

```

51 rdme Place the files in C:\Users\\~\Documents\OpenSCAD\libraries and
    call as:[~libraries]
52 rdme
53 rdme [~libraries]: C:\Users\\~\Documents\RapCAD\libraries is deprecated
    since RapCAD is no longer needed since Python is now used for
    writing out files)
54 rdme
55 rdme     use <gcodepreview.py>;
56 rdme     use <pygcodepreview.scad>;
57 rdme     include <gcodepreview.scad>;
58 rdme
59 rdme Note that it is necessary to use the first two files
60 rdme (this allows loading the Python commands and then
61 rdme wrapping them in OpenSCAD commands) and then include
62 rdme the last file (which allows using OpenSCAD variables
63 rdme to selectively implement the Python commands via their
64 rdme being wrapped in OpenSCAD modules) and define
65 rdme variables which match the project and then use
66 rdme commands such as:
67 rdme
68 rdme    .opengcodefile(Gcode_filename);
69 rdme    .opendxffile(DXF_filename);
70 rdme
71 rdme     difference() {
72 rdme         setupstock(stocklength, stockwidth, stockthickness,
            zeroheight, stockorigin);
73 rdme
74 rdme     movetosafez();
75 rdme
76 rdme     toolchange(squaretoolno,speed * square_ratio);
77 rdme
78 rdme     begintoolpath(0,0,0.25);
79 rdme     beginpolyline(0,0,0.25);
80 rdme
81 rdme     cutoneaxis_setfeed("Z",-1,plunge*square_ratio);
82 rdme     addpolyline(stocklength/2,stockwidth/2,-stockthickness);
83 rdme
84 rdme     cutwithfeed(stocklength/2,stockwidth/2,-stockthickness,feed);
85 rdme
86 rdme     endtoolpath();
87 rdme     endpolyline();
88 rdme
89 rdme     }
90 rdme
91 rdme     closegcodefile();
92 rdme     closedxffile();
93 rdme
94 rdme which makes a G-code file:
95 rdme
96 rdme ![OpenSCAD template G-code file](https://raw.githubusercontent.com/
    WillAdams/gcodepreview/main/gcodepreview_template.png?raw=true)
97 rdme
98 rdme but one which could only be sent to a machine so as to
99 rdme cut only the softest and most yielding of materials
100 rdme since it makes a single full-depth pass, and of which
101 rdme has a matching DXF which may be imported into a
102 rdme CAM tool --- but which it is not directly possible
103 rdme to assign a toolpath in readily available CAM tools
104 rdme (since it varies in depth from beginning-to-end).
105 rdme
106 rdme Importing this DXF and actually cutting it
107 rdme is discussed at:
108 rdme
109 rdme https://forum.makerforums.info/t/rewriting-gcodepreview-with-python
    /88617/14
110 rdme
111 rdme Tool numbers match those of tooling sold by Carbide 3D
112 rdme (ob. discl., I work for them).
113 rdme
114 rdme Comments are included in the G-code to match those
115 rdme expected by CutViewer.
116 rdme
117 rdme A complete example file is: gcodepreview_template.scad
118 rdme and another example is openscad_gcodepreview_cutjoinery.tres.scad
119 rdme which is made from an OpenSCAD Graph Editor file:
120 rdme
121 rdme ![OpenSCAD Graph Editor Cut Joinery File](https://raw.
    githubusercontent.com/WillAdams/gcodepreview/main/

```

```

OSGE_cutjoinery.png?raw=true)
122 rdme
123 rdme Version 0.1 supports setting up stock, origin, rapid
124 rdme positioning, making cuts, and writing out matching
125 rdme G-code, and creating a DXF with polylines.
126 rdme
127 rdme Added features since initial upload:
128 rdme
129 rdme - endpolyline(); --- this command allows ending one polyline so as
      to allow multiple lines in a DXF
130 rdme - separate dxf files are written out for each tool where tool is
      ball/square/V and small/large (10/31/23)
131 rdme - re-writing as a Literate Program using the LaTeX package docmfp
      (begun 4/12/24)
132 rdme - support for additional tooling shapes such as dovetail and
      keyhole tools
133 rdme
134 rdme Version 0.2 adds support for arcs
135 rdme
136 rdme - DXF: support for arcs (which may be used to make circles)
      (6/1/24)
137 rdme - Specialty toolpaths such as Keyhole which may be used for
      dovetail as well as keyhole cutters
138 rdme
139 rdme Version 0.3
140 rdme
141 rdme - Support for curves along the 3rd dimension
142 rdme - support for roundover tooling
143 rdme
144 rdme Version 0.4
145 rdme
146 rdme - Rewrite using literati documentclass, suppression of SVG code
147 rdme - dxfrectangle (without G-code support)
148 rdme
149 rdme Version 0.5
150 rdme
151 rdme - more shapes
152 rdme - consolidate rectangles, arcs, and circles in gcodepreview.scad
153 rdme
154 rdme Version 0.6
155 rdme
156 rdme - notes on modules
157 rdme - change file for setupstock
158 rdme
159 rdme Version 0.7
160 rdme
161 rdme - reduce usage of tool numbers
162 rdme - validate all code so that it runs without errors from sample
      file
163 rdme
164 rdme Possible future improvements:
165 rdme
166 rdme - support for additional tooling shapes such as tapered ball-nose
      tools or lollipop cutters or thread-cutting tools
167 rdme - G-code: support for G2/G3 arcs and circles
168 rdme - G-code: import external tool libraries and feeds and speeds from
      JSON or CSV files ---
169 rdme - general coding improvements --- current coding style is quite
      prosaic
170 rdme - additional generalized modules for cutting out various shapes/
      geometries
171 rdme
172 rdme Note for G-code generation that it is up to the user
173 rdme to implement Depth per Pass so as to not take a
174 rdme single full-depth pass. Working from a DXF of course
175 rdme allows one to off-load such considerations to a
176 rdme specialized CAM tool.
177 rdme
178 rdme Deprecated feature:
179 rdme
180 rdme - exporting SVGs --- while this was begun, it turns out that these
      would be written out upside down due to coordinate system
      differences between OpenSCAD/DXFs and SVGs requiring managing
      the inversion of the coordinate system (it is possible that
      METAPOST, which shares the same orientation and which can write
      out SVGs will be used instead for future versions)

```

2 gcodepreview

This library for OpenPythonSCAD 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 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). Doing so requires a total of three files:

- A Python file: gcodepreview.py (gcpy) — this will have variables in the traditional sense which may be used for tracking machine position and so forth
- An OpenSCAD file: pygcodepreview.scad (pyscad) — which wraps the Python code in OpenSCAD
- An OpenSCAD file: gcodepreview.scad (gcpscad) — which uses the other two files and which is included allowing it to access OpenSCAD variables for branching

Each file will begin with a suitable comment indicating the file type and suitable notes:

```
1 gcpy #!/usr/bin/env python
2 gcpy #icon "C:\Program Files\PythonSCAD\bin\openscad.exe" --trust-python
3 gcpy #Currently tested with 2024.09.03 and Python 3.11
4 gcpy #gcodepreview 0.61, see gcodepreview.scad

1 pyscad //!OpenSCAD
2 pyscad
3 pyscad //gcodepreview 0.61, see gcodepreview.scad

1 gcpscad //!OpenSCAD
2 gcpscad
3 gcpscad //gcodepreview 0.61
4 gcpscad //
5 gcpscad //used via use <gcodepreview.py>;
6 gcpscad // use <pygcodepreview.scad>;
7 gcpscad // include <gcodepreview.scad>;
8 gcpscad //
```

writeln The original implementation in RapSCAD used a command writeln — fortunately, this command is easily re-created in Python:

```
6 gcpy def writeln(*arguments):
7 gcpy     line_to_write = ""
8 gcpy     for element in arguments:
9 gcpy         line_to_write += element
10 gcpy     f.write(line_to_write)
11 gcpy     f.write("\n")
```

which command will accept a series of arguments and then write them out to a file object.

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, depth in toolpath, &c. This will be done using paired functions (which will set and return the matching variable) and a matching (global) variable, as well as additional functions for setting the matching variable(s).

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:

- tpz
- tpz

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

- currenttool
- currenttool

Note that the currenttool variable should always be used for any specification of a tool, being read in whenever a tool needs to be specified.

For each intended command it will be necessary to implement an appropriate aspect in each file. The Python file will manage the Python variables and handle things which can only be done in Python, while there will be two OpenSCAD files as noted above, one which calls the Python code (this will be used), while the other will be able to access and use OpenSCAD variables, as well as implement Customizer options (this will be included).

2.2 Modules

Note that as a convention, where it is necessary for a module to coordinate between Python and OpenSCAD, in certain cases it will be necessary for there to be three separate versions: a `p<foo>` Python definition for the manipulation of Python variables and any file routines, an `o<foo>` OpenSCAD module which will wrap up the Python function call, and lastly a `<foo>` OpenSCAD module which will be `<include>d` so as to be able to make use of OpenSCAD variables.

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, action, function, parameter, filetype

- Both prefix and suffix
 - `dx` (action (write out dxf file), filetype)
- Prefixes
 - `begin` (sequence)
 - `continue` (sequence)
 - `end` (sequence)
 - `cut` (action)
 - `move` (action)
 - `rapid` (action)
 - `open` (action)
 - `close` (action)
 - `set` (action/function)
- Suffixes
 - `feed` (parameter)
 - `gcode` (filetype)
 - `polyline` (file (element))

For the sake of convenience, all user-facing modules will be listed here with their interface requirements/variables. Where appropriate, modules which interact will be listed together.

```
begincutdxf(rh, ex, ey, ez, fr); and continuecutdxf(ex, ey, ez, fr);

beginpolyline(bx,by,bz); and addpolyline(bx,by,bz); and closepolyline();

begintoolpath(bx,by,bz); and endtoolpath();

current_tool(); [function]

cut(ex, ey, ez);
cutoneaxis_setfeed(axis,depth,feed);
cutwithfeed(ex, ey, ez, feed);

cutarcNECCdxf(ex, ey, ez, xcenter, ycenter, radius);
cutarcNWCCdxf(ex, ey, ez, xcenter, ycenter, radius);
cutarcSWCCdxf(ex, ey, ez, xcenter, ycenter, radius);
cutarcSECCdxf(ex, ey, ez, xcenter, ycenter, radius);
cutarcNECWdxf(ex, ey, ez, xcenter, ycenter, radius);
cutarcSECWdxf(ex, ey, ez, xcenter, ycenter, radius);
cutarcSWCWdxf(ex, ey, ez, xcenter, ycenter, radius);

cutkeyhole_toolpath(kh_start_depth, kh_max_depth, kht_direction, kh_distance);

cutoutrectangledxf(bx, by, bz, rwidth, rheight, rdepth);
cutrectangledxf(bx, by, bz, rwidth, rheight, rdepth);
cutrectangleoutlinedxf(bx, by, bz, rwidth, rheight, rdepth);

cutroundover(bx, by, bz, ex, ey, ez);

dxfarc(xcenter, ycenter, radius, anglebegin, endangle);
dxfpolyline(xbegin,ybegin,xend,yend);
```

```

movetosafeheight();
movetosafez();

opendxfile(fn); and closedxfile();
opengcodefile(fn); and closegcodefile();

rapidbx(bx, by, bz, ex, ey, ez);
rapid(ex, ey, ez);

rectangleoutlinedxf(bx, by, bz, rwidth, rheight);

setupstock(stocklength, stockwidth, stockthickness, zeroheight, stockorigin);

setxpos(newxpos);
setypos(newypos);
setzpos(newzpos);

settzpos(newtzpos);

toolchange(tool_number,speed);
tool_diameter(td_tool, td_depth); [function]
tool_radius(td_tool, td_depth); [function]

writecomment(comment);

```

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 management, etc.) and esp. note the use of o(penscad) and p(ython) as prefixes

Structurally, this will typically look like:

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 definition \DescribeSubroutine{FOOBAR}{pFOOBAR}

```

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

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

```

Further note that this definition will not be necessary for some later modules since they are in turn calling internal modules which already use this structure.

2.2.1 Initial Modules

The first such routine, (actually a subroutine, see `setupstock`) `psetupstock` will be appropriately enough, to set up the stock, and perform other initializations — in Python all that needs to be done is to set the value of the persistent (Python) variables:

```

13 gcpy def psetupstock(stocklength, stockwidth, stockthickness, zeroheight
, stockorigin):

```

```
14 gcpy      global mpx
15 gcpy      mpx = float(0)
16 gcpy      global mpy
17 gcpy      mpy = float(0)
18 gcpy      global mpz
19 gcpy      mpz = float(0)
20 gcpy      global tpz
21 gcpy      tpz = float(0)
22 gcpy      global currenttool
23 gcpy      currenttool = 102
```

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 a but which interfered with persistently setting a variable directly was fixed.

osetupstock The intermediary OpenSCAD code, osetupstock simply calls the Python version. Note that while the parameters are passed all the way down (for consistency) they are not used.

```
5 pyscad module osetupstock(stocklength, stockwidth, stockthickness,
                             zeroheight, stockorigin) {
6 pyscad     psetupstock(stocklength, stockwidth, stockthickness,
                             zeroheight, stockorigin);
7 pyscad }
```

setupstock The OpenSCAD code, setupstock requires that the user set parameters for stock dimensions and so forth, and will create comments in the G-code which incorporate the stock dimensions and its position relative to the zero as set relative to the stock.

The internal variable stockorigin is used in an <if then else> structure to position the 3D model of the stock.

```
10 gcpscad module setupstock(stocklength, stockwidth, stockthickness,
                             zeroheight, stockorigin) {
11 gcpscad     osetupstock(stocklength, stockwidth, stockthickness, zeroheight,
                             stockorigin);
12 gcpscad //initialize default tool and XYZ origin
13 gcpscad     osettool(102);
14 gcpscad     oset(0,0,0);
15 gcpscad     if (zeroheight == "Top") {
16 gcpscad         if (stockorigin == "Lower-Left") {
17 gcpscad             translate([0, 0, (-stockthickness)]){
18 gcpscad                 cube([stocklength, stockwidth, stockthickness], center=false);
19 gcpscad                 if (generategcode == true) {
20 gcpscad                     owritethree("(stockMin:0.00mm, 0.00mm, -",str(stockthickness)
                                     , "mm)");
21 gcpscad                     owritefive("(stockMax:",str(stocklength),"mm, ",str(
                                     stockwidth),"mm, 0.00mm)");
22 gcpscad                     owritenine("(STOCK/BLOCK, ",str(stocklength)," ",str(
                                     stockwidth)," ",str(stockthickness)," 0.00, 0.00, ",str(
                                     stockthickness),")");
23 gcpscad             }
24 gcpscad         }
25 gcpscad     }
26 gcpscad     else if (stockorigin == "Center-Left") {
27 gcpscad         translate([0, (-stockwidth / 2), -stockthickness]){
28 gcpscad             cube([stocklength, stockwidth, stockthickness], center=false)
                ;
29 gcpscad         if (generategcode == true) {
30 gcpscad             owritefive("(stockMin:0.00mm, -",str(stockwidth/2),"mm, -",str(
                                     stockthickness),"mm)");
31 gcpscad             owritefive("(stockMax:",str(stocklength),"mm, ",str(stockwidth/2),"
                                     mm, 0.00mm)");
32 gcpscad             owriteeleven("(STOCK/BLOCK, ",str(stocklength)," ",str(
                                     stockwidth)," ",str(stockthickness)," 0.00, ",str(
                                     stockwidth/2)," ",str(stockthickness),")");
33 gcpscad         }
34 gcpscad     }
35 gcpscad     } else if (stockorigin == "Top-Left") {
36 gcpscad         translate([0, (-stockwidth), -stockthickness]){
37 gcpscad             cube([stocklength, stockwidth, stockthickness], center=false)
                ;
38 gcpscad         if (generategcode == true) {
39 gcpscad             owritefive("(stockMin:0.00mm, -",str(stockwidth),"mm, -",str(
                                     stockthickness),"mm)");
40 gcpscad             owritethree("(stockMax:",str(stocklength),"mm, 0.00mm, 0.00mm)");
41 gcpscad             owriteeleven("(STOCK/BLOCK, ",str(stocklength)," ",str(stockwidth)
                                     , " ",str(stockthickness)," 0.00, ",str(stockwidth)," ",str(
                                     stockthickness),")");
42 gcpscad         }
```



```

43 gpcscad }
44 gpcscad }
45 gpcscad     else if (stockorigin == "Center") {
46 gpcscad //owritecomment("Center");
47 gpcscad     translate([(-stocklength / 2), (-stockwidth / 2), -
                        stockthickness]){
48 gpcscad         cube([stocklength, stockwidth, stockthickness], center=false)
                        ;
49 gpcscad if (generategcode == true) {
50 gpcscad owriteseven("(stockMin: -",str(stocklength/2)," -",str(stockwidth
                        /2),"mm, -",str(stockthickness),"mm)");
51 gpcscad owritefive("(stockMax:",str(stocklength/2),"mm, ",str(stockwidth/2)
                        ,"mm, 0.00mm)");
52 gpcscad owritethirteen("(STOCK/BLOCK, ",str(stocklength)," ",str(
                        stockwidth)," ",str(stockthickness)," ",str(stocklength/2),"
                        ", str(stockwidth/2)," ",str(stockthickness),"");
53 gpcscad }
54 gpcscad }
55 gpcscad }
56 gpcscad } else if (zeroheight == "Bottom") {
57 gpcscad //owritecomment("Bottom");
58 gpcscad     if (stockorigin == "Lower-Left") {
59 gpcscad         cube([stocklength, stockwidth, stockthickness], center=false);
60 gpcscad if (generategcode == true) {
61 gpcscad owriteone("(stockMin:0.00mm, 0.00mm, 0.00mm)");
62 gpcscad owriteseven("(stockMax:",str(stocklength),"mm, ",str(stockwidth),"
                        mm, ",str(stockthickness),"mm)");
63 gpcscad owriteseven("(STOCK/BLOCK, ",str(stocklength)," ",str(stockwidth)
                        ," ",str(stockthickness),"0.00, 0.00, 0.00)");
64 gpcscad }
65 gpcscad } else if (stockorigin == "Center-Left") {
66 gpcscad     translate([0, (-stockwidth / 2), 0]){
67 gpcscad         cube([stocklength, stockwidth, stockthickness], center=false)
                        ;
68 gpcscad if (generategcode == true) {
69 gpcscad owritethree("(stockMin:0.00mm, -",str(stockwidth/2),"mm, 0.00mm)");
70 gpcscad owriteseven("(stockMax:",str(stocklength),"mm, ",str(stockwidth/2)
                        ,"mm, ",str(stockthickness),"mm)");
71 gpcscad owritenine("(STOCK/BLOCK, ",str(stocklength)," ",str(stockwidth)
                        ," ",str(stockthickness),"0.00, ",str(stockwidth/2)," 0.00")
                        ;
72 gpcscad }
73 gpcscad }
74 gpcscad } else if (stockorigin == "Top-Left") {
75 gpcscad     translate([0, (-stockwidth), 0]){
76 gpcscad         cube([stocklength, stockwidth, stockthickness], center=false)
                        ;
77 gpcscad }
78 gpcscad if (generategcode == true) {
79 gpcscad owritethree("(stockMin:0.00mm, -",str(stockwidth),"mm, 0.00mm)");
80 gpcscad owritefive("(stockMax:",str(stocklength),"mm, 0.00mm, ",str(
                        stockthickness),"mm)");
81 gpcscad owritenine("(STOCK/BLOCK, ",str(stocklength)," ",str(stockwidth)
                        ," ",str(stockthickness)," 0.00, ", str(stockwidth)," 0.00")
                        ;
82 gpcscad }
83 gpcscad } else if (stockorigin == "Center") {
84 gpcscad     translate([(-stocklength / 2), (-stockwidth / 2), 0]){
85 gpcscad         cube([stocklength, stockwidth, stockthickness], center=false)
                        ;
86 gpcscad }
87 gpcscad if (generategcode == true) {
88 gpcscad owritefive("(stockMin:-",str(stocklength/2)," -",str(stockwidth/2)
                        ,"mm, 0.00mm)");
89 gpcscad owriteseven("(stockMax:",str(stocklength/2),"mm, ",str(stockwidth
                        /2),"mm, ",str(stockthickness),"mm)");
90 gpcscad owriteeleven("(STOCK/BLOCK, ",str(stocklength)," ",str(stockwidth)
                        ," ",str(stockthickness)," ",str(stocklength/2)," ", str(
                        stockwidth/2)," 0.00)");
91 gpcscad }
92 gpcscad }
93 gpcscad }
94 gpcscad if (generategcode == true) {
95 gpcscad     owriteone("G90");
96 gpcscad     owriteone("G21");
97 gpcscad //     owriteone("Move to safe Z to avoid workholding");
98 gpcscad //     owriteone("G53G0Z-5.000");
99 gpcscad }

```

```
100 gpcscad //owritecomment("ENDSETUP");
101 gpcscad }
```

An example usage would be:

```
difference() {
  setupstock(stocklength, stockwidth, stockthickness, zeroheight, stockorigin);
  ... // Cutting commands go here
}
```

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

zpos

```
25 gcpy def xpos():
26 gcpy     global mpx
27 gcpy     return mpx
28 gcpy
29 gcpy def ypos():
30 gcpy     global mpy
31 gcpy     return mpy
32 gcpy
33 gcpy def zpos():
34 gcpy     global mpz
35 gcpy     return mpz
36 gcpy
37 gcpy def tzpos():
38 gcpy     global tpz
39 gcpy     return tpz
```

psetxpos and in turn, functions which set the positions: psetxpos, psetypos, psetzpos, and psettzpos

psetypos

psetzpos

psettzpos

```
41 gcpy def psetxpos(newxpos):
42 gcpy     global mpx
43 gcpy     mpx = newxpos
44 gcpy
45 gcpy def psetypos(newypos):
46 gcpy     global mpy
47 gcpy     mpy = newypos
48 gcpy
49 gcpy def psetzpos(newzpos):
50 gcpy     global mpz
51 gcpy     mpz = newzpos
52 gcpy
53 gcpy def psettzpos(newtzpos):
54 gcpy     global tpz
55 gcpy     tpz = newtzpos
```

setxpos and as noted above, there will need to be matching OpenSCAD versions which will set: setxpos,
setypos setypos, setzpos, and settzpos; as well as return the value: getxpos, getypos, getzpos, and
setzpos gettzpos Note that for routines where the variable is directly passed from OpenSCAD to Python
settzpos it is possible to have OpenSCAD directly call the matching Python module with no need to use
getxpos an intermediary OpenSCAD module.

getypos

getzpos

gettzpos

```
9 pyscad function getxpos() = xpos();
10 pyscad function getypos() = ypos();
11 pyscad function getzpos() = zpos();
12 pyscad function gettzpos() = tzpos();
13 pyscad
14 pyscad module setxpos(newxpos) {
15 pyscad     psetxpos(newxpos);
16 pyscad }
17 pyscad
18 pyscad module setypos(newypos) {
19 pyscad     psetypos(newypos);
20 pyscad }
21 pyscad
22 pyscad module setzpos(newzpos) {
23 pyscad     psetzpos(newzpos);
24 pyscad }
25 pyscad
26 pyscad module settzpos(newtzpos) {
27 pyscad     psettzpos(newtzpos);
28 pyscad }
```

oset oset while for setting all three of the variables, there is an internal OpenSCAD module:

```
103 gpcscad module oset(ex, ey, ez) {
104 gpcscad     setxpos(ex);
105 gpcscad     setypos(ey);
106 gpcscad     setzpos(ez);
107 gpcscad }
```

osettz and some toolpaths will require the storing and usage of an intermediate value via osettz for the Z-axis position during calculation:

```
109 gpcscad module osettz(tz) {
110 gpcscad     settzpos(tz);
111 gpcscad }
```

2.3 Tools and Changes

pcurrenttool Similarly Python functions and variables will be used in: pcurrenttool and psettool to track
psettool and set and return the current tool

```
57 gcpy def psettool(tn):
58 gcpy     global currenttool
59 gcpy     currenttool = tn
60 gcpy
61 gcpy def pcurrent_tool():
62 gcpy     global currenttool
63 gcpy     return currenttool
```

osettool and matching OpenSCAD modules: osettool and current tool set and return the current tool:
current tool

```
30 pyscad module osettool(tn){
31 pyscad     psettool(tn);
32 pyscad }
33 pyscad
34 pyscad function current_tool() = pcurrent_tool();
```

2.3.1 toolchange

toolchange and 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 below.

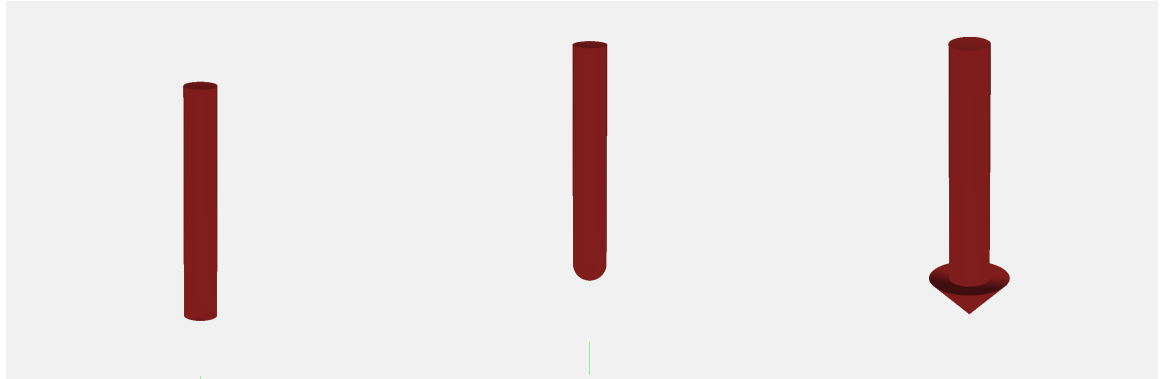
Note that the comments written out in G-code correspond to that used by the G-code previewing tool CutViewer (which is unfortunately, no longer readily available).

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 currenttool().

It is possible that rather than hard-coding the tool definitions, a future update will instead read them in from an external file — the .csv format used for tool libraries in Carbide Create seems a likely candidate and worth exploring.

Note that there are many varieties of tooling and not all will be implemented, especially in the early versions of this project

2.3.1.1 Normal Tooling 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)

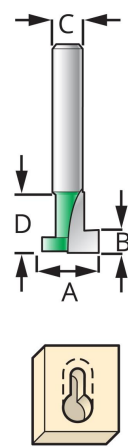
```

113 gpcscad module toolchange(tool_number,speed) {
114 gpcscad   osettool(tool_number);
115 gpcscad if (generategcode == true) {
116 gpcscad   writecomment("Toolpath");
117 gpcscad   owriteone("M05");
118 gpcscad //   writecomment("Move to safe Z to avoid workholding");
119 gpcscad //   owriteone("G53G0Z-5.000");
120 gpcscad //   writecomment("Begin toolpath");
121 gpcscad   if (tool_number == 201) {
122 gpcscad     writecomment("TOOL/MILL,6.35, 0.00, 0.00, 0.00");
123 gpcscad   } else if (tool_number == 202) {
124 gpcscad     writecomment("TOOL/MILL,6.35, 3.17, 0.00, 0.00");
125 gpcscad   } else if (tool_number == 102) {
126 gpcscad     writecomment("TOOL/MILL,3.17, 0.00, 0.00, 0.00");
127 gpcscad   } else if (tool_number == 101) {
128 gpcscad     writecomment("TOOL/MILL,3.17, 1.58, 0.00, 0.00");
129 gpcscad   } else if (tool_number == 301) {
130 gpcscad     writecomment("TOOL/MILL,0.03, 0.00, 6.35, 45.00");
131 gpcscad   } else if (tool_number == 302) {
132 gpcscad     writecomment("TOOL/MILL,0.03, 0.00, 10.998, 30.00");
133 gpcscad   } else if (tool_number == 390) {
134 gpcscad     writecomment("TOOL/MILL,0.03, 0.00, 1.5875, 45.00");

```

2.3.1.2 Tooling for Keyhole Toolpaths Keyhole toolpaths (see: subsection 3.2 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.

There are several notable candidates for such tooling:



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 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 completeness’ sake and are not (at this time) implemented

```
135 gcpscad } else if (tool_number == 374) {
136 gcpscad   writecomment("TOOL/MILL,9.53, 0.00, 3.17, 0.00");
137 gcpscad } else if (tool_number == 375) {
138 gcpscad   writecomment("TOOL/MILL,9.53, 0.00, 3.17, 0.00");
139 gcpscad } else if (tool_number == 376) {
140 gcpscad   writecomment("TOOL/MILL,12.7, 0.00, 4.77, 0.00");
141 gcpscad } else if (tool_number == 378) {
142 gcpscad   writecomment("TOOL/MILL,12.7, 0.00, 4.77, 0.00");

143 gcpscad } else if (tool_number == 814) {
144 gcpscad   writecomment("TOOL/MILL,12.7, 6.367, 12.7, 0.00");
```

2.3.1.3 Thread mills The implementation of arcs cutting along the Z-axis raises the possibility of cutting threads using “thread mills”. See: <https://community.carbide3d.com/t/thread-milling-in-metal-on-the-shapeoko-3/5332>

Note that it will be necessary to to define modules (see below) for each tool shape.

With the tools delineated, the module is closed out and the tooling information written into the G-code.

```
145 gcpscad }
146 gcpscad   select_tool(tool_number);
147 gcpscad   owritetwo("M6T",str(tool_number));
148 gcpscad   owritetwo("M03S",str(speed));
149 gcpscad }
150 gcpscad }
```

For example:

```
toolchange(small_square_tool_no,speed * square_ratio);
```

2.3.1.4 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 2.3.2.2.

2.3.1.5 Selecting Tools There must also be a module for selecting tools: selecttool which will select the matching module for 3D modeling based on the currenttool (which is fed in to the module as tool_number, and pass the appropriate parameters to that module:

```
152 gcpscad module select_tool(tool_number) {
153 gcpscad //echo(tool_number);
154 gcpscad   if (tool_number == 201) {
155 gcpscad     gcp_endmill_square(6.35, 19.05);
156 gcpscad   } else if (tool_number == 202) {
157 gcpscad     gcp_endmill_ball(6.35, 19.05);
158 gcpscad   } else if (tool_number == 102) {
159 gcpscad     gcp_endmill_square(3.175, 19.05);
160 gcpscad   } else if (tool_number == 101) {
161 gcpscad     gcp_endmill_ball(3.175, 19.05);
162 gcpscad   } else if (tool_number == 301) {
163 gcpscad     gcp_endmill_v(90, 12.7);
164 gcpscad   } else if (tool_number == 302) {
165 gcpscad     gcp_endmill_v(60, 12.7);
166 gcpscad   } else if (tool_number == 390) {
167 gcpscad     gcp_endmill_v(90, 3.175);
```

For a keyhole tool:

```
168 gcpscad   } else if (tool_number == 374) {
169 gcpscad     gcp_keyhole(9.525, 3.175);
170 gcpscad   } else if (tool_number == 375) {
171 gcpscad     gcp_keyhole(9.525, 3.175);
172 gcpscad   } else if (tool_number == 376) {
173 gcpscad     gcp_keyhole(12.7, 4.7625);
174 gcpscad   } else if (tool_number == 378) {
175 gcpscad     gcp_keyhole(12.7, 4.7625);
```

and dovetail tool:

```
176 gcpscad   } else if (tool_number == 814) {
177 gcpscad     gcp_dovetail(12.7, 6.367, 12.7, 14);
```

Once all tools have been defined the if statement and module may be closed:

```
178 gcpscad   }
179 gcpscad }
```

2.3.2 3D Shapes for Tools

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

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

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

```
181 gcpscad module gcp_endmill_square(es_diameter, es_flute_length) {
182 gcpscad   cylinder(r1=(es_diameter / 2), r2=(es_diameter / 2), h=
                  es_flute_length, center=false);
183 gcpscad }
```

gcp keyhole The gcp keyhole is modeled by the cutting base:

```
185 gcpscad module gcp_keyhole(es_diameter, es_flute_length) {
186 gcpscad   cylinder(r1=(es_diameter / 2), r2=(es_diameter / 2), h=
                  es_flute_length, center=false);
187 gcpscad }
```

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

```
189 gcpscad module gcp_keyhole_shaft(es_diameter, es_flute_length) {
190 gcpscad   cylinder(r1=(es_diameter / 2), r2=(es_diameter / 2), h=
                  es_flute_length, center=false);
191 gcpscad }
```

gcp dovetail The gcp 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)

```
193 gcpscad module gcp_dovetail(dt_bottomdiameter, dt_topdiameter, dt_height,
                  dt_angle) {
194 gcpscad   cylinder(r1=(dt_bottomdiameter / 2), r2=(dt_topdiameter / 2), h=
                  dt_height, center=false);
```

```
195 gpcscad }
```

gcp endmill ball The gcp endmill ball is modeled as a hemisphere joined with a cylinder:

```
197 gpcscad module gcp_endmill_ball(es_diameter, es_flute_length) {
198 gpcscad   translate([0, 0, (es_diameter / 2)]){
199 gpcscad     union(){
200 gpcscad       sphere(r=(es_diameter / 2));
201 gpcscad       cylinder(r1=(es_diameter / 2), r2=(es_diameter / 2), h=
                es_flute_length, center=false);
202 gpcscad     }
203 gpcscad   }
204 gpcscad }
```

gcp endmill v The gcp endmill v is modeled as a cylinder with a zero width base and a second cylinder for the shaft:

```
206 gpcscad module gcp_endmill_v(es_v_angle, es_diameter) {
207 gpcscad   union(){
208 gpcscad     cylinder(r1=0, r2=(es_diameter / 2), h=((es_diameter / 2) / tan
                ((es_v_angle / 2))), center=false);
209 gpcscad     translate([0, 0, ((es_diameter / 2) / tan((es_v_angle / 2)))]{
210 gpcscad       cylinder(r1=(es_diameter / 2), r2=(es_diameter / 2), h=((
                es_diameter * 8) ), center=false);/// tan((es_v_angle / 2)
                )
211 gpcscad     }
212 gpcscad   }
213 gpcscad }
```

2.3.2.2 Concave toolshapes While normal tooling may be represented with a single hull operation betwixt two 3D toolshapes, concave tooling such as roundover/radius tooling require multiple slices of the tool shape which are then hulled together. Something of this can be seen in the manual work-around for previewing them: <https://community.carbide3d.com/t/using-unsupported-tooling-in-carbide-create-roundover-cove-radius-bits/43723>.

Ideally, it would be possible to simply identify such tooling using the tool # in the code used for normal toolshapes as above, but the most expedient option is to simply use a specific command for this. Since such tooling is quite limited in its use and normally only used at the surface of the part along an edge, this separation is easily justified.

Because it is necessary to divide the tooling into vertical slices and call the hull operation for each slice the tool definitions are tightly coupled with the module. Note that there are two different modules, the public-facing version which includes the tool number: cutroundover

```
215 gpcscad module cutroundover(bx, by, bz, ex, ey, ez, radiustn) {
216 gpcscad   if (radiustn == 56125) {
217 gpcscad     cutroundovertool(bx, by, bz, ex, ey, ez, 0.508/2, 1.531);
218 gpcscad   } else if (radiustn == 56142) {
219 gpcscad     cutroundovertool(bx, by, bz, ex, ey, ez, 0.508/2, 2.921);
220 gpcscad   } else if (radiustn == 312) {
221 gpcscad     cutroundovertool(bx, by, bz, ex, ey, ez, 1.524/2, 3.175);
222 gpcscad   } else if (radiustn == 1570) {
223 gpcscad     cutroundovertool(bx, by, bz, ex, ey, ez, 0.507/2, 4.509);
224 gpcscad   }
225 gpcscad }
```

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.

```
227 gpcscad module cutroundovertool(bx, by, bz, ex, ey, ez, tool_radius_tip,
                tool_radius_width) {
228 gpcscad   n = 90 + $fn*3;
229 gpcscad   step = 360/n;
230 gpcscad
231 gpcscad   hull(){
232 gpcscad     translate([bx,by,bz])
233 gpcscad     cylinder(step,tool_radius_tip,tool_radius_tip);
234 gpcscad     translate([ex,ey,ez])
235 gpcscad     cylinder(step,tool_radius_tip,tool_radius_tip);
236 gpcscad   }
237 gpcscad
238 gpcscad   hull(){
239 gpcscad     translate([bx,by,bz+tool_radius_width])
240 gpcscad     cylinder(tool_radius_width*2,tool_radius_tip+tool_radius_width,
                tool_radius_tip+tool_radius_width);
241 gpcscad }
```

```
242 gcpscad translate([ex,ey,ez+tool_radius_width])
243 gcpscad cylinder(tool_radius_width*2,tool_radius_tip+tool_radius_width,
    tool_radius_tip+tool_radius_width);
244 gcpscad }
245 gcpscad
246 gcpscad for (i=[0:step:90]) {
247 gcpscad     angle = i;
248 gcpscad     dx = tool_radius_width*cos(angle);
249 gcpscad     dxx = tool_radius_width*cos(angle+step);
250 gcpscad     dzz = tool_radius_width*sin(angle);
251 gcpscad     dz = tool_radius_width*sin(angle+step);
252 gcpscad     dh = dz-dzz;
253 gcpscad     hull(){
254 gcpscad         translate([bx,by,bz+dz])
255 gcpscad             cylinder(dh,tool_radius_tip+tool_radius_width-dx,
                tool_radius_tip+tool_radius_width-dxx);
256 gcpscad         translate([ex,ey,ez+dz])
257 gcpscad             cylinder(dh,tool_radius_tip+tool_radius_width-dx,
                tool_radius_tip+tool_radius_width-dxx);
258 gcpscad     }
259 gcpscad }
260 gcpscad }
```

2.3.3 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:

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

otool diameter the matching OpenSCAD function, otool diameter calls the Python function:

```
36 pyscad function otool_diameter(td_tool, td_depth) = ptool_diameter(td_tool
    , td_depth);
```

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

```
65 gcpy def ptool_diameter(ptd_tool, ptd_depth):
66 gcpy # Square 122,112,102,201
67 gcpy     if ptd_tool == 122:
68 gcpy         return 0.79375
69 gcpy     if ptd_tool == 112:
70 gcpy         return 1.5875
71 gcpy     if ptd_tool == 102:
72 gcpy         return 3.175
73 gcpy     if ptd_tool == 201:
74 gcpy         return 6.35
75 gcpy # Ball 121,111,101,202
76 gcpy     if ptd_tool == 122:
77 gcpy         return
78 gcpy         if ptd_depth > 0.396875:
79 gcpy             return 0.79375
80 gcpy         else:
81 gcpy             return 0
82 gcpy     if ptd_tool == 112:
83 gcpy         if ptd_depth > 0.79375:
84 gcpy             return 1.5875
85 gcpy         else:
86 gcpy             return 0
87 gcpy     if ptd_tool == 101:
88 gcpy         if ptd_depth > 1.5875:
89 gcpy             return 3.175
90 gcpy         else:
91 gcpy             return 0
92 gcpy     if ptd_tool == 202:
93 gcpy         if ptd_depth > 3.175:
```



```
94 gcpy          return 6.35
95 gcpy          else:
96 gcpy          return 0
97 gcpy # V 301, 302, 390
98 gcpy      if ptd_tool == 301:
99 gcpy          return 0
100 gcpy      if ptd_tool == 302:
101 gcpy          return 0
102 gcpy      if ptd_tool == 390:
103 gcpy          return 0
104 gcpy # Keyhole
105 gcpy      if ptd_tool == 374:
106 gcpy          if ptd_depth < 3.175:
107 gcpy              return 9.525
108 gcpy          else:
109 gcpy              return 6.35
110 gcpy      if ptd_tool == 375:
111 gcpy          if ptd_depth < 3.175:
112 gcpy              return 9.525
113 gcpy          else:
114 gcpy              return 8
115 gcpy      if ptd_tool == 376:
116 gcpy          if ptd_depth < 4.7625:
117 gcpy              return 12.7
118 gcpy          else:
119 gcpy              return 6.35
120 gcpy      if ptd_tool == 378:
121 gcpy          if ptd_depth < 4.7625:
122 gcpy              return 12.7
123 gcpy          else:
124 gcpy              return 8
125 gcpy # Dovetail
126 gcpy      if ptd_tool == 814:
127 gcpy          if ptd_depth > 12.7:
128 gcpy              return 6.35
129 gcpy          else:
130 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:

```
264 gcpscad function tool_radius(td_tool, td_depth) = otool_diameter(td_tool,
                                td_depth)/2;
```

(Note that zero (o) and other not fully calculated values will need to be replaced with code which calculates the appropriate values.)

2.3.4 Feeds and Speeds

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

One notable possibility for the future would be to load it from the .csv files used for User tool libraries in Carbide Create. Ideally, any use of such values in modules would be such that some other scheme could replace that usage with minimal editing and updating.

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

2.4 File Handling

popengcodefile popendxfile
popendxfile popendxflgsqfile, popendxfsmsqfile, popendxflgblfile, popendxfsmblfile, popendxflgVfile,
popendxflgsqfile and popendxfsmVfile. There is a separate function for each type of file, and for DXFs, there are
popendxfsmsqfile multiple file instances, one for each combination of different type and size of tool which it is
popendxflgblfile expected a project will work with. Each such file will be suffixed with the tool number.
popendxfsmblfile Integrating G-code and DXF generation with everything else would be ideal, but will require
popendxflgVfile ensuring that each command which moves the tool creates a matching command for both files.
popendxfsmVfile

```
97 gcpy def popengcodefile(fn):
98 gcpy     global f
99 gcpy     f = open(fn, "w")
100 gcpy
101 gcpy def popendxfile(fn):
102 gcpy     global dxf
103 gcpy     dxf = open(fn, "w")
104 gcpy
105 gcpy def popendxflgblfile(fn):
106 gcpy     global dxflgbl
107 gcpy     dxflgbl = open(fn, "w")
108 gcpy
109 gcpy def popendxflgsqfile(fn):
110 gcpy     global dxflgsq
111 gcpy     dxflgsq = open(fn, "w")
112 gcpy
113 gcpy def popendxflgVfile(fn):
114 gcpy     global dxflgV
115 gcpy     dxflgV = open(fn, "w")
116 gcpy
117 gcpy def popendxfsmblfile(fn):
118 gcpy     global dxfsmb1
119 gcpy     dxfsmb1 = open(fn, "w")
120 gcpy
121 gcpy def popendxfsmsqfile(fn):
122 gcpy     global dxfsmsq
123 gcpy     dxfsmsq = open(fn, "w")
124 gcpy
125 gcpy def popendxfsmVfile(fn):
126 gcpy     global dx fsmV
127 gcpy     dx fsmV = open(fn, "w")
128 gcpy
129 gcpy def popendxfKHfile(fn):
130 gcpy     global dx fKH
131 gcpy     dx fKH = open(fn, "w")
132 gcpy
133 gcpy def popendxfDTfile(fn):
134 gcpy     global dx fDT
135 gcpy     dx fDT = open(fn, "w")
```

oopengcodefile There will need to be matching OpenSCAD modules oopengcodefile, and oopendxfile, for
oopendxfile the Python functions.

```
38 pycad module oopengcodefile(fn) {
39 pycad     popengcodefile(fn);
40 pycad }
41 pycad
42 pycad module oopendxfile(fn) {
43 pycad //     echo(fn);
44 pycad     popendxfile(fn);
45 pycad }
46 pycad
47 pycad module oopendxflgblfile(fn) {
48 pycad     popendxflgblfile(fn);
49 pycad }
50 pycad
51 pycad module oopendxflgsqfile(fn) {
52 pycad     popendxflgsqfile(fn);
53 pycad }
54 pycad
55 pycad module oopendxflgVfile(fn) {
56 pycad     popendxflgVfile(fn);
57 pycad }
58 pycad
59 pycad module oopendxfsmblfile(fn) {
60 pycad     popendxfsmblfile(fn);
61 pycad }
```

```

62 pycscad
63 pycscad module oopendxfmsqfile(fn) {
64 pycscad //    echo(fn);
65 pycscad    popendxfmsqfile(fn);
66 pycscad }
67 pycscad
68 pycscad module oopendxfsmVfile(fn) {
69 pycscad    popendxfsmVfile(fn);
70 pycscad }
71 pycscad
72 pycscad module oopendxfKHfile(fn) {
73 pycscad    popendxfKHfile(fn);
74 pycscad }
75 pycscad
76 pycscad module oopendxfDTfile(fn) {
77 pycscad    popendxfDTfile(fn);
78 pycscad }

```

opengcodefile With matching OpenSCAD commands: opengcodefile

```

266 gcpcscad module opengcodefile(fn) {
267 gcpcscad if (generategcode == true) {
268 gcpcscad    oopengcodefile(fn);
269 gcpcscad //    echo(fn);
270 gcpcscad    owriterecomment(fn);
271 gcpcscad }
272 gcpcscad }

```

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

```

274 gcpcscad module opendxffile(fn) {
275 gcpcscad    if (generatedxif == true) {
276 gcpcscad        oopendxffile(str(fn, ".dxf"));
277 gcpcscad //    echo(fn);
278 gcpcscad        dxfwriteone("0");
279 gcpcscad        dxfwriteone("SECTION");
280 gcpcscad        dxfwriteone("2");
281 gcpcscad        dxfwriteone("ENTITIES");
282 gcpcscad        if (large_ball_tool_no > 0) {    oopendxfllgblfile(str(fn, ".",
        large_ball_tool_no, ".dxf"));
283 gcpcscad        dxfpreamble(large_ball_tool_no);
284 gcpcscad    }
285 gcpcscad        if (large_square_tool_no > 0) {    oopendxfllgsqfile(str(fn
        , ".", large_square_tool_no, ".dxf"));
286 gcpcscad        dxfpreamble(large_square_tool_no);
287 gcpcscad    }
288 gcpcscad        if (large_V_tool_no > 0) {    oopendxfllgVfile(str(fn, ".",
        large_V_tool_no, ".dxf"));
289 gcpcscad        dxfpreamble(large_V_tool_no);
290 gcpcscad    }
291 gcpcscad        if (small_ball_tool_no > 0) { oopendxfsmblfile(str(fn, ".",
        small_ball_tool_no, ".dxf"));
292 gcpcscad        dxfpreamble(small_ball_tool_no);
293 gcpcscad    }
294 gcpcscad        if (small_square_tool_no > 0) {    oopendxfmsqfile(str(fn
        , ".", small_square_tool_no, ".dxf"));
295 gcpcscad //    echo(str("tool no", small_square_tool_no));
296 gcpcscad        dxfpreamble(small_square_tool_no);
297 gcpcscad    }
298 gcpcscad        if (small_V_tool_no > 0) {    oopendxfsmVfile(str(fn, ".",
        small_V_tool_no, ".dxf"));
299 gcpcscad        dxfpreamble(small_V_tool_no);
300 gcpcscad    }
301 gcpcscad        if (KH_tool_no > 0) {    oopendxfKHfile(str(fn, ".", KH_tool_no
        , ".dxf"));
302 gcpcscad        dxfpreamble(KH_tool_no);
303 gcpcscad    }
304 gcpcscad        if (DT_tool_no > 0) {    oopendxfDTfile(str(fn, ".", DT_tool_no
        , ".dxf"));
305 gcpcscad        dxfpreamble(DT_tool_no);
306 gcpcscad    }
307 gcpcscad }
308 gcpcscad }

```

2.4.1 Writing to files

writedxf Once files have been opened they may be written to. The base command: writedxf

```
137 gcpy def writedxf(*arguments):
138 gcpy     line_to_write = ""
139 gcpy     for element in arguments:
140 gcpy         line_to_write += element
141 gcpy         dxf.write(line_to_write)
142 gcpy         dxf.write("\n")
```

has a matching command each tool/size combination:

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

```
144 gcpy def writedxflgbl(*arguments):
145 gcpy     line_to_write = ""
146 gcpy     for element in arguments:
147 gcpy         line_to_write += element
148 gcpy     dxflgbl.write(line_to_write)
149 gcpy     print(line_to_write)
150 gcpy     dxflgbl.write("\n")
151 gcpy
152 gcpy def writedxflgsq(*arguments):
153 gcpy     line_to_write = ""
154 gcpy     for element in arguments:
155 gcpy         line_to_write += element
156 gcpy     dxflgsq.write(line_to_write)
157 gcpy     print(line_to_write)
158 gcpy     dxflgsq.write("\n")
159 gcpy
160 gcpy def writedxflgV(*arguments):
161 gcpy     line_to_write = ""
162 gcpy     for element in arguments:
163 gcpy         line_to_write += element
164 gcpy     dxflgV.write(line_to_write)
165 gcpy     print(line_to_write)
166 gcpy     dxflgV.write("\n")
167 gcpy
168 gcpy def writedxfsmbl(*arguments):
169 gcpy     line_to_write = ""
170 gcpy     for element in arguments:
171 gcpy         line_to_write += element
172 gcpy     dxfsmbbl.write(line_to_write)
173 gcpy     print(line_to_write)
174 gcpy     dxfsmbbl.write("\n")
175 gcpy
176 gcpy def writedxfsmsq(*arguments):
177 gcpy     line_to_write = ""
178 gcpy     for element in arguments:
179 gcpy         line_to_write += element
180 gcpy     dxfsmsq.write(line_to_write)
181 gcpy     print(line_to_write)
182 gcpy     dxfsmsq.write("\n")
183 gcpy
184 gcpy def writedxfsmV(*arguments):
185 gcpy     line_to_write = ""
186 gcpy     for element in arguments:
187 gcpy         line_to_write += element
188 gcpy     dx fsmV.write(line_to_write)
189 gcpy     print(line_to_write)
190 gcpy     dx fsmV.write("\n")
191 gcpy
192 gcpy def writedxfKH(*arguments):
193 gcpy     line_to_write = ""
```

```
194 gcpy      for element in arguments:
195 gcpy          line_to_write += element
196 gcpy      dxfKH.write(line_to_write)
197 gcpy      print(line_to_write)
198 gcpy      dxfKH.write("\n")
199 gcpy
200 gcpy def writedxDT(*arguments):
201 gcpy     line_to_write = ""
202 gcpy     for element in arguments:
203 gcpy         line_to_write += element
204 gcpy     dxfDT.write(line_to_write)
205 gcpy     print(line_to_write)
206 gcpy     dxfDT.write("\n")
```

owritecomment Separate OpenSCAD modules, owritecomment, dxfwriteone, dxfwritelgbl, dxfwritelgsq, dxfwriteone dxfwritelgV, dxfwritesmbl, dxfwritesmsq, and dxfwritesmV will be used for either writing out dxfwritelgbl comments in G-code (.nc) files or adding to a DXF file — for each different tool in a file there will dxfwritelgsq be a matching module to write to it.

```
dxfwritelgV
dxfwritesmbl 80 pycad module owritecomment(comment) {
dxfwritesmsq 81 pycad     writeln("(",comment,")");
dxfwritesmV 82 pycad }
83 pycad
84 pycad module dxfwriteone(first) {
85 pycad     writedxf(first);
86 pycad //     writeln(first);
87 pycad //     echo(first);
88 pycad }
89 pycad
90 pycad module dxfwritelgbl(first) {
91 pycad     writedxflgbl(first);
92 pycad }
93 pycad
94 pycad module dxfwritelgsq(first) {
95 pycad     writedxflgsq(first);
96 pycad }
97 pycad
98 pycad module dxfwritelgV(first) {
99 pycad     writedxflgV(first);
100 pycad }
101 pycad
102 pycad module dxfwritesmbl(first) {
103 pycad     writedxfsmbl(first);
104 pycad }
105 pycad
106 pycad module dxfwritesmsq(first) {
107 pycad     writedxfsmsq(first);
108 pycad }
109 pycad
110 pycad module dxfwritesmV(first) {
111 pycad     writedxfsmV(first);
112 pycad }
113 pycad
114 pycad module dxfwriteKH(first) {
115 pycad     writedxfKH(first);
116 pycad }
117 pycad
118 pycad module dxfwriteDT(first) {
119 pycad     writedxDT(first);
120 pycad }
```

Since it is not convenient to stitch together and then write out multiple elements, the most expedient thing to do is to have discrete commands for each possible number of arguments, one through thirteen, owrite...

```
122 pycad module owriteone(first) {
123 pycad     writeln(first);
124 pycad }
125 pycad
126 pycad module owritetwo(first, second) {
127 pycad     writeln(first, second);
128 pycad }
129 pycad
130 pycad module owritethree(first, second, third) {
131 pycad     writeln(first, second, third);
132 pycad }
133 pycad
```

```

134 pycscad module owritefour(first, second, third, fourth) {
135 pycscad     writeln(first, second, third, fourth);
136 pycscad }
137 pycscad
138 pycscad module owritefive(first, second, third, fourth, fifth) {
139 pycscad     writeln(first, second, third, fourth, fifth);
140 pycscad }
141 pycscad
142 pycscad module owritesix(first, second, third, fourth, fifth, sixth) {
143 pycscad     writeln(first, second, third, fourth, fifth, sixth);
144 pycscad }
145 pycscad
146 pycscad module owriteseven(first, second, third, fourth, fifth, sixth,
    seventh) {
147 pycscad     writeln(first, second, third, fourth, fifth, sixth, seventh);
148 pycscad }
149 pycscad
150 pycscad module owriteeight(first, second, third, fourth, fifth, sixth,
    seventh, eighth) {
151 pycscad     writeln(first, second, third, fourth, fifth, sixth, seventh,
    eighth);
152 pycscad }
153 pycscad
154 pycscad module owritenine(first, second, third, fourth, fifth, sixth,
    seventh, eighth, ninth) {
155 pycscad     writeln(first, second, third, fourth, fifth, sixth, seventh,
    eighth, ninth);
156 pycscad }
157 pycscad
158 pycscad module owriteten(first, second, third, fourth, fifth, sixth,
    seventh, eighth, ninth, tenth) {
159 pycscad     writeln(first, second, third, fourth, fifth, sixth, seventh,
    eighth, ninth, tenth);
160 pycscad }
161 pycscad
162 pycscad module owriteeleven(first, second, third, fourth, fifth, sixth,
    seventh, eighth, ninth, tenth, eleventh) {
163 pycscad     writeln(first, second, third, fourth, fifth, sixth, seventh,
    eighth, ninth, tenth, eleventh);
164 pycscad }
165 pycscad
166 pycscad module owritetwelve(first, second, third, fourth, fifth, sixth,
    seventh, eighth, ninth, tenth, eleventh, twelfth) {
167 pycscad     writeln(first, second, third, fourth, fifth, sixth, seventh,
    eighth, ninth, tenth, eleventh, twelfth);
168 pycscad }
169 pycscad
170 pycscad module owritethirteen(first, second, third, fourth, fifth, sixth,
    seventh, eighth, ninth, tenth, eleventh, twelfth, thirteenth) {
171 pycscad     writeln(first, second, third, fourth, fifth, sixth, seventh,
    eighth, ninth, tenth, eleventh, twelfth, thirteenth);
172 pycscad }

```

dxfwrite 2.4.1.1 Beginning Writing to DXFs The `dxfwrite` 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.

```

310 gcpcscad module dxfwrite(tn,arg) {
311 gcpcscad if (tn == large_ball_tool_no) {
312 gcpcscad     dxfwritelgbl(arg);}
313 gcpcscad if (tn == large_square_tool_no) {
314 gcpcscad     dxfwritelgsq(arg);}
315 gcpcscad if (tn == large_V_tool_no) {
316 gcpcscad     dxfwritelgV(arg);}
317 gcpcscad if (tn == small_ball_tool_no) {
318 gcpcscad     dxfwritesmbl(arg);}
319 gcpcscad if (tn == small_square_tool_no) {
320 gcpcscad     dxfwritesmsq(arg);}
321 gcpcscad if (tn == small_V_tool_no) {
322 gcpcscad     dxfwritesmV(arg);}
323 gcpcscad if (tn == DT_tool_no) {
324 gcpcscad     dxfwriteDT(arg);}
325 gcpcscad if (tn == KH_tool_no) {
326 gcpcscad     dxfwriteKH(arg);}
327 gcpcscad }
328 gcpcscad

```

```

329 gcpscad module dxfpreamble(tn) {
330 gcpscad //     echo(str("dxfpreamble",small_square_tool_no));
331 gcpscad     dxfwrite(tn,"0");
332 gcpscad     dxfwrite(tn,"SECTION");
333 gcpscad     dxfwrite(tn,"2");
334 gcpscad     dxfwrite(tn,"ENTITIES");
335 gcpscad }

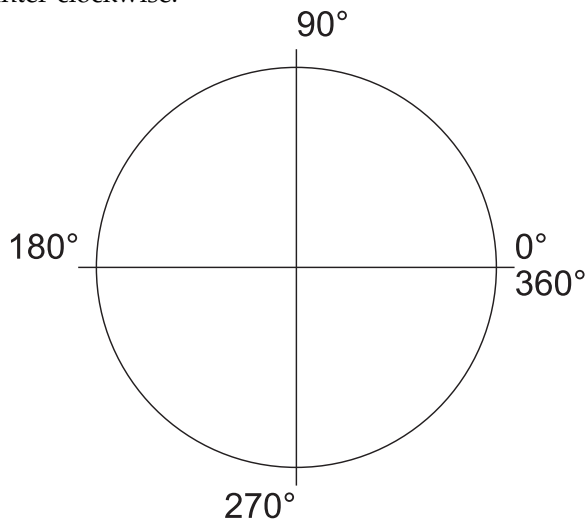
```

2.4.1.2 DXF Lines and Arcs Similarly, 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.

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

- dxfbpl • a line: LWPOLYLINE is one possible implementation: dxfbpl
- dxfarf • ARC — a notable option would be for the arc to close on itself, creating a circle: dxfarf

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:

```

dxfarf(10, 10, 5, 0, 90, small_square_tool_no);
dxfarf(10, 10, 5, 90, 180, small_square_tool_no);
dxfarf(10, 10, 5, 180, 270, small_square_tool_no);
dxfarf(10, 10, 5, 270, 360, small_square_tool_no);

```

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

There are three possible interactions for DXF elements and toolpaths:

- describe the motion of the tool
- define a perimeter of an area which will be cut by a tool
- define a centerpoint for a specialty toolpath such as Drill or Keyhole

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

```

337 gcpscad module dxfp1(tn,xbegin,ybegin,xend,yend) {
338 gcpscad     dxfwrite(tn,"0");
339 gcpscad     dxfwrite(tn,"LWPOLYLINE");
340 gcpscad     dxfwrite(tn,"90");
341 gcpscad     dxfwrite(tn,"2");
342 gcpscad     dxfwrite(tn,"70");
343 gcpscad     dxfwrite(tn,"0");
344 gcpscad     dxfwrite(tn,"43");
345 gcpscad     dxfwrite(tn,"0");
346 gcpscad     dxfwrite(tn,"10");
347 gcpscad     dxfwrite(tn,str(xbegin));
348 gcpscad     dxfwrite(tn,"20");
349 gcpscad     dxfwrite(tn,str(ybegin));
350 gcpscad     dxfwrite(tn,"10");
351 gcpscad     dxfwrite(tn,str(xend));
352 gcpscad     dxfwrite(tn,"20");
353 gcpscad     dxfwrite(tn,str(yend));
354 gcpscad }
355 gcpscad
356 gcpscad module dxfpolyline(xbegin,ybegin,xend,yend, tn) {

```

```

357 gcpscad if (generatedxf == true) {
358 gcpscad     dxfwriteone("0");
359 gcpscad     dxfwriteone("LWPOLYLINE");
360 gcpscad     dxfwriteone("90");
361 gcpscad     dxfwriteone("2");
362 gcpscad     dxfwriteone("70");
363 gcpscad     dxfwriteone("0");
364 gcpscad     dxfwriteone("43");
365 gcpscad     dxfwriteone("0");
366 gcpscad     dxfwriteone("10");
367 gcpscad     dxfwriteone(str(xbegin));
368 gcpscad     dxfwriteone("20");
369 gcpscad     dxfwriteone(str(ybegin));
370 gcpscad     dxfwriteone("10");
371 gcpscad     dxfwriteone(str(xend));
372 gcpscad     dxfwriteone("20");
373 gcpscad     dxfwriteone(str(yend));
374 gcpscad     dxflpl(tn,xbegin,ybegin,xend,yend);
375 gcpscad     }
376 gcpscad }

```

dxfa As for other files, we have two versions, **dxfa** and **dxfarf**, one which accepts a **tn** (tool number), writing only to it, while a publicly facing version writes to the main DXF file *and* writes to the specific DXF file for the specified tool.

```

378 gcpscad module dxfa(tn,xcenter,ycenter,radius,anglebegin,endangle) {
379 gcpscad     dxfwrite(tn,"0");
380 gcpscad     dxfwrite(tn,"ARC");
381 gcpscad     dxfwrite(tn,"10");
382 gcpscad     dxfwrite(tn,str(xcenter));
383 gcpscad     dxfwrite(tn,"20");
384 gcpscad     dxfwrite(tn,str(ycenter));
385 gcpscad     dxfwrite(tn,"40");
386 gcpscad     dxfwrite(tn,str(radius));
387 gcpscad     dxfwrite(tn,"50");
388 gcpscad     dxfwrite(tn,str(anglebegin));
389 gcpscad     dxfwrite(tn,"51");
390 gcpscad     dxfwrite(tn,str(endangle));
391 gcpscad }
392 gcpscad
393 gcpscad module dxfarf(xcenter,ycenter,radius,anglebegin,endangle,tn) {
394 gcpscad if (generatedxf == true) {
395 gcpscad     dxfwriteone("0");
396 gcpscad     dxfwriteone("ARC");
397 gcpscad     dxfwriteone("10");
398 gcpscad     dxfwriteone(str(xcenter));
399 gcpscad     dxfwriteone("20");
400 gcpscad     dxfwriteone(str(ycenter));
401 gcpscad     dxfwriteone("40");
402 gcpscad     dxfwriteone(str(radius));
403 gcpscad     dxfwriteone("50");
404 gcpscad     dxfwriteone(str(anglebegin));
405 gcpscad     dxfwriteone("51");
406 gcpscad     dxfwriteone(str(endangle));
407 gcpscad     dxfa(tn,xcenter,ycenter,radius,anglebegin,endangle);
408 gcpscad     }
409 gcpscad }

```

The original implementation of polylines worked, but may be removed.

```

411 gcpscad module dxfbpl(tn,bx,by) {
412 gcpscad     dxfwrite(tn,"0");
413 gcpscad     dxfwrite(tn,"POLYLINE");
414 gcpscad     dxfwrite(tn,"8");
415 gcpscad     dxfwrite(tn,"default");
416 gcpscad     dxfwrite(tn,"66");
417 gcpscad     dxfwrite(tn,"1");
418 gcpscad     dxfwrite(tn,"70");
419 gcpscad     dxfwrite(tn,"0");
420 gcpscad     dxfwrite(tn,"0");
421 gcpscad     dxfwrite(tn,"VERTEX");
422 gcpscad     dxfwrite(tn,"8");
423 gcpscad     dxfwrite(tn,"default");
424 gcpscad     dxfwrite(tn,"70");
425 gcpscad     dxfwrite(tn,"32");
426 gcpscad     dxfwrite(tn,"10");
427 gcpscad     dxfwrite(tn,str(bx));

```



```

428 gcpscad      dxfwrite(tn,"20");
429 gcpscad      dxfwrite(tn,str(by));
430 gcpscad }
431 gcpscad
432 gcpscad module beginpolyline(bx,by,bz) {
433 gcpscad if (generatedxf == true) {
434 gcpscad     dxfwriteone("0");
435 gcpscad     dxfwriteone("POLYLINE");
436 gcpscad     dxfwriteone("8");
437 gcpscad     dxfwriteone("default");
438 gcpscad     dxfwriteone("66");
439 gcpscad     dxfwriteone("1");
440 gcpscad     dxfwriteone("70");
441 gcpscad     dxfwriteone("0");
442 gcpscad     dxfwriteone("0");
443 gcpscad     dxfwriteone("VERTEX");
444 gcpscad     dxfwriteone("8");
445 gcpscad     dxfwriteone("default");
446 gcpscad     dxfwriteone("70");
447 gcpscad     dxfwriteone("32");
448 gcpscad     dxfwriteone("10");
449 gcpscad     dxfwriteone(str(bx));
450 gcpscad     dxfwriteone("20");
451 gcpscad     dxfwriteone(str(by));
452 gcpscad     dxfbpl(current_tool(),bx,by);}
453 gcpscad }
454 gcpscad
455 gcpscad module dxfcpl(tn,bx,by) {
456 gcpscad     dxfwrite(tn,"0");
457 gcpscad     dxfwrite(tn,"VERTEX");
458 gcpscad     dxfwrite(tn,"8");
459 gcpscad     dxfwrite(tn,"default");
460 gcpscad     dxfwrite(tn,"70");
461 gcpscad     dxfwrite(tn,"32");
462 gcpscad     dxfwrite(tn,"10");
463 gcpscad     dxfwrite(tn,str(bx));
464 gcpscad     dxfwrite(tn,"20");
465 gcpscad     dxfwrite(tn,str(by));
466 gcpscad }
467 gcpscad
468 gcpscad module addpolyline(bx,by,bz) {
469 gcpscad if (generatedxf == true) {
470 gcpscad     dxfwriteone("0");
471 gcpscad     dxfwriteone("VERTEX");
472 gcpscad     dxfwriteone("8");
473 gcpscad     dxfwriteone("default");
474 gcpscad     dxfwriteone("70");
475 gcpscad     dxfwriteone("32");
476 gcpscad     dxfwriteone("10");
477 gcpscad     dxfwriteone(str(bx));
478 gcpscad     dxfwriteone("20");
479 gcpscad     dxfwriteone(str(by));
480 gcpscad     dxfcpl(current_tool(),bx,by);
481 gcpscad     }
482 gcpscad }
483 gcpscad
484 gcpscad module dxfcpl(tn) {
485 gcpscad     dxfwrite(tn,"0");
486 gcpscad     dxfwrite(tn,"SEQEND");
487 gcpscad }
488 gcpscad
489 gcpscad module closepolyline() {
490 gcpscad     if (generatedxf == true) {
491 gcpscad         dxfwriteone("0");
492 gcpscad         dxfwriteone("SEQEND");
493 gcpscad         dxfcpl(current_tool());
494 gcpscad     }
495 gcpscad }
496 gcpscad
497 gcpscad module writecomment(comment) {
498 gcpscad     if (generategcode == true) {
499 gcpscad         owritecomment(comment);
500 gcpscad     }
501 gcpscad }

```

At the end of the project it will be necessary to close each file using the commands: `pclosegcodefile`, `pclosegcodefile`, and `closedxf`. In some instances it will be necessary to write additional `closedxf` information, depending on the file format.

```
208 gcpy def pclosegcodefile():
209 gcpy     f.close()
210 gcpy
211 gcpy def pclosedxffile():
212 gcpy     dxf.close()
213 gcpy
214 gcpy def pclosedxflgblfile():
215 gcpy     dxflgbl.close()
216 gcpy
217 gcpy def pclosedxflgsqfile():
218 gcpy     dxflgsq.close()
219 gcpy
220 gcpy def pclosedxflgVfile():
221 gcpy     dxflgV.close()
222 gcpy
223 gcpy def pclosedxfsmblfile():
224 gcpy     dxfsmb1.close()
225 gcpy
226 gcpy def pclosedxfsmsqfile():
227 gcpy     dxfsmsq.close()
228 gcpy
229 gcpy def pclosedxfsmVfile():
230 gcpy     dxfsmV.close()
231 gcpy
232 gcpy def pclosedxfDTfile():
233 gcpy     dxfdT.close()
234 gcpy
235 gcpy def pclosedxfKHfile():
236 gcpy     dxfKH.close()
```

oclosegcodefile In addition to the Python forms, there will need to be matching OpenSCAD commands to call them: oclosegcodefile, and oclosedxffile.

oclosedxffile

```
174 pycad module oclosegcodefile() {
175 pycad     pclosegcodefile();
176 pycad }
177 pycad
178 pycad module oclosedxffile() {
179 pycad     pclosedxffile();
180 pycad }
181 pycad
182 pycad module oclosedxflgblfile() {
183 pycad     pclosedxflgblfile();
184 pycad }
185 pycad
186 pycad module oclosedxflgsqfile() {
187 pycad     pclosedxflgsqfile();
188 pycad }
189 pycad
190 pycad module oclosedxflgVfile() {
191 pycad     pclosedxflgVfile();
192 pycad }
193 pycad
194 pycad module oclosedxfsmblfile() {
195 pycad     pclosedxfsmblfile();
196 pycad }
197 pycad
198 pycad module oclosedxfsmsqfile() {
199 pycad     pclosedxfsmsqfile();
200 pycad }
201 pycad
202 pycad module oclosedxfsmVfile() {
203 pycad     pclosedxfsmVfile();
204 pycad }
205 pycad
206 pycad module oclosedxfDTfile() {
207 pycad     pclosedxfDTfile();
208 pycad }
209 pycad
210 pycad module oclosedxfKHfile() {
211 pycad     pclosedxfKHfile();
212 pycad }
```

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

```

503 gcpscad module closegcodefile() {
504 gcpscad   if (generategcode == true) {
505 gcpscad     owriteone("M05");
506 gcpscad     owriteone("M02");
507 gcpscad     oclosegcodefile();
508 gcpscad   }
509 gcpscad }
510 gcpscad
511 gcpscad module dxfpostamble(arg) {
512 gcpscad   dxfwrite(arg,"0");
513 gcpscad   dxfwrite(arg,"ENDSEC");
514 gcpscad   dxfwrite(arg,"0");
515 gcpscad   dxfwrite(arg,"EOF");
516 gcpscad }
517 gcpscad
518 gcpscad module closedxfile() {
519 gcpscad   if (generatedxf == true) {
520 gcpscad     dxfwriteone("0");
521 gcpscad     dxfwriteone("ENDSEC");
522 gcpscad     dxfwriteone("0");
523 gcpscad     dxfwriteone("EOF");
524 gcpscad     oclosedxfile();
525 gcpscad //   echo("CLOSING");
526 gcpscad     if (large_ball_tool_no > 0) {      dxfpostamble(
                    large_ball_tool_no);
527 gcpscad         oclosedxflgblfile();
528 gcpscad     }
529 gcpscad     if (large_square_tool_no > 0) {      dxfpostamble(
                    large_square_tool_no);
530 gcpscad         oclosedxflgsqfile();
531 gcpscad     }
532 gcpscad     if (large_V_tool_no > 0) {      dxfpostamble(large_V_tool_no);
533 gcpscad         oclosedxflgVfile();
534 gcpscad     }
535 gcpscad     if (small_ball_tool_no > 0) {      dxfpostamble(
                    small_ball_tool_no);
536 gcpscad         oclosedxfsmbfile();
537 gcpscad     }
538 gcpscad     if (small_square_tool_no > 0) {      dxfpostamble(
                    small_square_tool_no);
539 gcpscad         oclosedxfsmsqfile();
540 gcpscad     }
541 gcpscad     if (small_V_tool_no > 0) {      dxfpostamble(small_V_tool_no);
542 gcpscad         oclosedxfsmVfile();
543 gcpscad     }
544 gcpscad     if (DT_tool_no > 0) {      dxfpostamble(DT_tool_no);
545 gcpscad         oclosedxfDTfile();
546 gcpscad     }
547 gcpscad     if (KH_tool_no > 0) {      dxfpostamble(KH_tool_no);
548 gcpscad         oclosedxfKHfile();
549 gcpscad     }
550 gcpscad   }
551 gcpscad }

```

2.5 Movement and Cutting

otm With all the scaffolding in place, it is possible to model the tool: otm, (colors the tool model so as
ocut to differentiate cut areas) and cutting: ocut, as well as Rapid movements to position the tool to
orapid begin a cut: orapid, rapid, and rapidbx which will also need to write out files which represent
rapid the desired machine motions.
rapidbx

```

553 gcpscad module otm(ex, ey, ez, r,g,b) {
554 gcpscad   color([r,g,b]) hull(){
555 gcpscad     translate([xpos(), ypos(), zpos()]){
556 gcpscad       select_tool(current_tool());
557 gcpscad     }
558 gcpscad     translate([ex, ey, ez]){
559 gcpscad       select_tool(current_tool());
560 gcpscad     }
561 gcpscad   }
562 gcpscad   oset(ex, ey, ez);
563 gcpscad }
564 gcpscad
565 gcpscad module ocut(ex, ey, ez) {
566 gcpscad   //color([0.2,1,0.2]) hull(){
567 gcpscad     otm(ex, ey, ez, 0.2,1,0.2);

```

```

568 gcpscad }
569 gcpscad
570 gcpscad module orapid(ex, ey, ez) {
571 gcpscad //color([0.93,0,0]) hull(){
572 gcpscad   otm(ex, ey, ez, 0.93,0,0);
573 gcpscad }
574 gcpscad
575 gcpscad module rapidbx(bx, by, bz, ex, ey, ez) {
576 gcpscad //   writeln("G0 X",bx," Y", by, "Z", bz);
577 gcpscad   if (generategcode == true) {
578 gcpscad     writecomment("rapid");
579 gcpscad     owritesix("G0 X",str(ex)," Y", str(ey), " Z", str(ez));
580 gcpscad   }
581 gcpscad   orapid(ex, ey, ez);
582 gcpscad }
583 gcpscad
584 gcpscad module rapid(ex, ey, ez) {
585 gcpscad //   writeln("G0 X",bx," Y", by, "Z", bz);
586 gcpscad   if (generategcode == true) {
587 gcpscad     writecomment("rapid");
588 gcpscad     owritesix("G0 X",str(ex)," Y", str(ey), " Z", str(ez));
589 gcpscad   }
590 gcpscad   orapid(ex, ey, ez);
591 gcpscad }
592 gcpscad
593 gcpscad module movetosafez() {
594 gcpscad //this should be move to retract height
595 gcpscad   if (generategcode == true) {
596 gcpscad     writecomment("Move to safe Z to avoid workholding");
597 gcpscad     owriteone("G53G0Z-5.000");
598 gcpscad   }
599 gcpscad   orapid(getxpos(), getypos(), retractheight+55);
600 gcpscad }
601 gcpscad
602 gcpscad module begintoolpath(bx,by,bz) {
603 gcpscad   if (generategcode == true) {
604 gcpscad     writecomment("PREPOSITION FOR RAPID PLUNGE");
605 gcpscad     owritefour("G0X", str(bx), "Y",str(by));
606 gcpscad     owritetwo("Z", str(bz));
607 gcpscad   }
608 gcpscad   orapid(bx,by,bz);
609 gcpscad }
610 gcpscad
611 gcpscad module movetosafeheight() {
612 gcpscad //this should be move to machine position
613 gcpscad   if (generategcode == true) {
614 gcpscad     //   writecomment("PREPOSITION FOR RAPID PLUNGE");Z25.650
615 gcpscad     //G1Z24.663F381.0 ,"F",str(plunge)
616 gcpscad     if (zeroheight == "Top") {
617 gcpscad       owritetwo("Z",str(retractheight));
618 gcpscad     }
619 gcpscad   }
620 gcpscad   orapid(getxpos(), getypos(), retractheight+55);
621 gcpscad }
622 gcpscad
623 gcpscad module cutoneaxis_setfeed(axis,depth,feed) {
624 gcpscad   if (generategcode == true) {
625 gcpscad     //   writecomment("PREPOSITION FOR RAPID PLUNGE");Z25.650
626 gcpscad     //G1Z24.663F381.0 ,"F",str(plunge) G1Z7.612F381.0
627 gcpscad     if (zeroheight == "Top") {
628 gcpscad       owritefive("G1",axis,str(depth),"F",str(feed));
629 gcpscad     }
630 gcpscad   }
631 gcpscad   if (axis == "X") {setxpos(depth);
632 gcpscad     ocut(depth, getypos(), getzpos());}
633 gcpscad   if (axis == "Y") {setypos(depth);
634 gcpscad     ocut(getxpos(), depth, getzpos());
635 gcpscad   }
636 gcpscad   if (axis == "Z") {setzpos(depth);
637 gcpscad     ocut(getxpos(), getypos(), depth);
638 gcpscad   }
639 gcpscad }
640 gcpscad
641 gcpscad module cut(ex, ey, ez) {
642 gcpscad //   writeln("G0 X",bx," Y", by, "Z", bz);
643 gcpscad   if (generategcode == true) {
644 gcpscad     owritesix("G1 X",str(ex)," Y", str(ey), " Z", str(ez));
645 gcpscad   }

```

```

646 gcpscad //if (generatesvg == true) {
647 gcpscad //    owritesix("G1 X",str(ex)," Y", str(ey), " Z", str(ez));
648 gcpscad //    orapid(getxpos(), getypos(), retractheight+5);
649 gcpscad //    writesvgline(getxpos(),getypos(),ex,ey);
650 gcpscad //}
651 gcpscad ocut(ex, ey, ez);
652 gcpscad }
653 gcpscad
654 gcpscad module cutwithfeed(ex, ey, ez, feed) {
655 gcpscad //    writeln("G0 X",bx," Y", by, "Z", bz);
656 gcpscad if (generategcode == true) {
657 gcpscad //    writecomment("rapid");
658 gcpscad    owriteeight("G1 X",str(ex)," Y", str(ey), " Z", str(ez),"F",str
        (feed));
659 gcpscad }
660 gcpscad ocut(ex, ey, ez);
661 gcpscad }
662 gcpscad
663 gcpscad module endtoolpath() {
664 gcpscad if (generategcode == true) {
665 gcpscad //Z31.750
666 gcpscad //    owriteone("G53G0Z-5.000");
667 gcpscad    writetwo("Z",str(retractheight));
668 gcpscad }
669 gcpscad orapid(getxpos(),getypos(),retractheight);
670 gcpscad }

```

3 Cutting shapes, cut2Dshapes, and expansion

Certain basic shapes (arcs, circles, rectangles), will be incorporated in the main code. Other shapes will be added to the additional/optional file, `cut2Dshapes.scad` as they are developed, and of course the user is free to develop their own systems.

It is most expedient to test out new features in a new/separate file insofar as the file structures will allow (tool definitions for example will need to be consolidated in 2.3.1) which will need to be included in the projects which will make use of said features until such time as they are added into the main `gcodepreview.scad` file.

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

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.

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., `cutrectangledxf`
- 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, a nice feature for this would be to include/model the areas which should be cleared for the sake of reducing wear on the tool and ensuring chip clearance

Some further considerations:

- relationship of geometry to toolpath — arguably there should be an option for each toolpath (we will use Carbide Create as a reference implementation) which is to be supported. Note that there are several possibilities: modeling the tool movement, describing the outline which the tool will cut, modeling a reference shape for the toolpath
- tool geometry — 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.
- feeds and speeds — if outputting G-code it would be nice to be able to import feeds and speeds from external files such as the .csv files used for user tool libraries in Carbide Create

- 0
 - circle
 - ellipse (oval) (requires some sort of non-arc curve)
 - * egg-shaped
 - annulus (one circle within another, forming a ring)
 - 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 sector¹
 - two convex, one concave arc
- 4
 - rectangle (including square) — `cutrectanglexf`, `cutoutrectanglexf`, `rectangleoutlinedxf`
 - 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

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

- two straight edges, one concave arcoddly, 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 arcwith 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?

- 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)

3.1 Arcs for toolpaths and DXFs

A further consideration here is that G-code supports arcs in addition to the lines and polylines already implemented.

Implementing arcs wants at least the following options for quadrant and direction:

- cutarcNWCW — cut the upper-left quadrant of a circle moving clockwise
- cutarcNWCC — upper-left quadrant counter-clockwise
- cutarcNECW
- cutarcNECC
- cutarcSECW
- cutarcSECC
- cutarcNECW
- cutarcNECC
- cutcircleCW — while it wont matter for generating a DXF, when G-code is implemented direction of cut will be a consideration for that
- cutcircleCCdxf

It will be necessary to have two separate representations of arcs — the 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. At this time, the current version of Carbide Create only imports circles in DXF as curves, any other example is converted into polylines — unfortunately, the implementation of this is not such as would allow directly matching that representation. A work-around to import a DXF as curves is to convert the arc into a reasonable number of line segments so as to approximate the arc.

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) — note that this may also be used in DXF so as to sidestep the question of how many line segments there would be for a given arc representation

Cutting the quadrant arcs will greatly simplify the calculation and interface for the modules. A full set of 8 will be necessary, then circles may either be stitched together manually or a pair of modules made for them.

At this time, despite what the module names imply (`cutarcNWCwdxf, &c.`), only cutting and DXF generation is supported. Adding support for G-code will be done at a later time. Since these modules will ultimately support G-code, the interface will assume the stored `xpos` and `ypos` as the origin. Parameters which will need to be passed in are:

- `tn`
- `ex`
- `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

Since OpenSCAD does not have an arc movement command it is necessary to iterate through `arcloop` a loop: `arcloop` (clockwise), `narcloop` (counterclockwise) to handle the drawing and processing `narcloop` of the `cut()` 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 (two when the need to have a version which steps down):

```

672 gpcscad module arclloop(barcl,earcl, xcenter, ycenter, radius) {
673 gpcscad   for (i = [barcl : abs(1) : earcl]) {
674 gpcscad       cut(xcenter + radius * cos(i),
675 gpcscad         ycenter + radius * sin(i),
676 gpcscad         getzpos()-(gettzpos()))
677 gpcscad       );
678 gpcscad       setxpos(xcenter + radius * cos(i));
679 gpcscad       setypos(ycenter + radius * sin(i));
680 gpcscad   }
681 gpcscad }
682 gpcscad
683 gpcscad module narcloop(barcl,earcl, xcenter, ycenter, radius) {
684 gpcscad   for (i = [barcl : -1 : earcl]) {
685 gpcscad       cut(xcenter + radius * cos(i),
686 gpcscad         ycenter + radius * sin(i),
687 gpcscad         getzpos()-(gettzpos()))
688 gpcscad       );
689 gpcscad       setxpos(xcenter + radius * cos(i));
690 gpcscad       setypos(ycenter + radius * sin(i));
691 gpcscad   }
692 gpcscad }

```

The various textual versions are quite obvious:

```

694 gpcscad module cutarcNECCdxfl(ex, ey, ez, xcenter, ycenter, radius, tn) {
695 gpcscad   dxfarcl(xcenter,ycenter,radius,0,90, tn);
696 gpcscad   settzpos((getzpos()-ez)/90);
697 gpcscad   arclloop(1,90, xcenter, ycenter, radius);
698 gpcscad }
699 gpcscad
700 gpcscad module cutarcNWCCdxfl(ex, ey, ez, xcenter, ycenter, radius, tn) {
701 gpcscad   dxfarcl(xcenter,ycenter,radius,90,180, tn);
702 gpcscad   settzpos((getzpos()-ez)/90);
703 gpcscad   arclloop(91,180, xcenter, ycenter, radius);
704 gpcscad }
705 gpcscad
706 gpcscad module cutarcSWCCdxfl(ex, ey, ez, xcenter, ycenter, radius, tn) {
707 gpcscad   dxfarcl(xcenter,ycenter,radius,180,270, tn);
708 gpcscad   settzpos((getzpos()-ez)/90);
709 gpcscad   arclloop(181,270, xcenter, ycenter, radius);
710 gpcscad }
711 gpcscad
712 gpcscad module cutarcSECCdxfl(ex, ey, ez, xcenter, ycenter, radius, tn) {
713 gpcscad   dxfarcl(xcenter,ycenter,radius,270,360, tn);
714 gpcscad   settzpos((getzpos()-ez)/90);
715 gpcscad   arclloop(271,360, xcenter, ycenter, radius);
716 gpcscad }
717 gpcscad
718 gpcscad module cutarcNECWdxfl(ex, ey, ez, xcenter, ycenter, radius, tn) {
719 gpcscad   dxfarcl(xcenter,ycenter,radius,0,90, tn);
720 gpcscad   settzpos((getzpos()-ez)/90);
721 gpcscad   narcloop(89,0, xcenter, ycenter, radius);
722 gpcscad }
723 gpcscad
724 gpcscad module cutarcSECWdxfl(ex, ey, ez, xcenter, ycenter, radius, tn) {
725 gpcscad   dxfarcl(xcenter,ycenter,radius,270,360, tn);
726 gpcscad   settzpos((getzpos()-ez)/90);
727 gpcscad   narcloop(359,270, xcenter, ycenter, radius);
728 gpcscad }
729 gpcscad
730 gpcscad module cutarcSWCWdxfl(ex, ey, ez, xcenter, ycenter, radius, tn) {
731 gpcscad   dxfarcl(xcenter,ycenter,radius,180,270, tn);
732 gpcscad   settzpos((getzpos()-ez)/90);
733 gpcscad   narcloop(269,180, xcenter, ycenter, radius);
734 gpcscad }
735 gpcscad
736 gpcscad module cutarcNWCWdxfl(ex, ey, ez, xcenter, ycenter, radius, tn) {
737 gpcscad   dxfarcl(xcenter,ycenter,radius,90,180, tn);
738 gpcscad   settzpos((getzpos()-ez)/90);
739 gpcscad   narcloop(179,90, xcenter, ycenter, radius);
740 gpcscad }

```

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

```

cutarcNECCdxfl(-stocklength/4, stockwidth/4+stockwidth/16, -stockthickness, -stocklength/4, stockwidth/4
cutarcNWCCdxfl(-(stocklength/4+stockwidth/16), stockwidth/4, -stockthickness, -stocklength/4, stockwidth
cutarcSWCCdxfl(-stocklength/4, stockwidth/4-stockwidth/16, -stockthickness, -stocklength/4, stockwidth/4

```



```
cutarcSECCdx(-(stocklength/4-stockwidth/16), stockwidth/4, -stockthickness, -stocklength/4, stockwidth,
```

3.2 Keyhole toolpath and undercut tooling

cutkeyhole toolpath 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.

Tooling for such toolpaths is defined at paragraph 2.3.1.2

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:

```
742 gpcscad module cutkeyhole_toolpath(kh_start_depth, kh_max_depth,
    kht_direction, kh_distance, kh_tool_no) {
743 gpcscad if (kht_direction == "N") {
744 gpcscad cutKH_toolpath_degrees(kh_start_depth, kh_max_depth, 90,
    kh_distance, kh_tool_no);
745 gpcscad } else if (kht_direction == "S") {
746 gpcscad cutKH_toolpath_degrees(kh_start_depth, kh_max_depth, 270,
    kh_distance, kh_tool_no);
747 gpcscad } else if (kht_direction == "E") {
748 gpcscad cutKH_toolpath_degrees(kh_start_depth, kh_max_depth, 0,
    kh_distance, kh_tool_no);
749 gpcscad } else if (kht_direction == "W") {
750 gpcscad cutKH_toolpath_degrees(kh_start_depth, kh_max_depth, 180,
    kh_distance, kh_tool_no);
751 gpcscad }
752 gpcscad }
```

cutKH toolpath degrees The original version of the command, cutKH toolpath degrees 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:

```
754 gpcscad module cutKH_toolpath_degrees(kh_start_depth, kh_max_depth,
    kh_angle, kh_distance, kh_tool_no) {
755 gpcscad //Circle at entry hole
756 gpcscad dxfarc(getxpos(),getypos(),tool_diameter(KH_tool_no, (7))/2,0,90,
    KH_tool_no);
757 gpcscad dxfarc(getxpos(),getypos(),tool_diameter(KH_tool_no, (7))/2,90,180,
    KH_tool_no);
758 gpcscad dxfarc(getxpos(),getypos(),tool_diameter(KH_tool_no, (7))
    /2,180,270, KH_tool_no);
759 gpcscad dxfarc(getxpos(),getypos(),tool_diameter(KH_tool_no, (7))
    /2,270,360, KH_tool_no);
```

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

```
1 gpcscad //Outlines of entry hole and slot
2 gpcscad if (kh_angle == 0) {
3 gpcscad //Lower left of entry hole
4 gpcscad dxfarc(getxpos(),getypos(),9.525/2,180,270, KH_tool_no);
5 gpcscad //Upper left of entry hole
6 gpcscad dxfarc(getxpos(),getypos(),9.525/2,90,180, KH_tool_no);
7 gpcscad //Upper right of entry hole
8 gpcscad dxfarc(getxpos(),getypos(),9.525/2,90-acos(tool_diameter(
    KH_tool_no, 5)/tool_diameter(KH_tool_no, 1)), 90, KH_tool_no
    );
9 gpcscad //Lower right of entry hole
10 gpcscad dxfarc(getxpos(),getypos(),9.525/2,270, 270+acos(tool_diameter(
    KH_tool_no, 5)/tool_diameter(KH_tool_no, 1)), KH_tool_no);
```

```

11 gcpcscad //Actual line of cut
12 gcpcscad dxfpolyline(getxpos(),getypos(),getxpos()+kh_distance,getypos()
);
13 gcpcscad //upper right of slot
14 gcpcscad dxfarc(getxpos()+kh_distance,getypos(),tool_diameter(KH_tool_no
, (kh_max_depth+4.36))/2,0,90, KH_tool_no);
15 gcpcscad //lower right of slot
16 gcpcscad dxfarc(getxpos()+kh_distance,getypos(),tool_diameter(KH_tool_no
, (kh_max_depth+4.36))/2,270,360, KH_tool_no);
17 gcpcscad //upper right slot
18 gcpcscad dxfpolyline(
19 gcpcscad getxpos()+(sqrt((tool_diameter(KH_tool_no,1)^2)-(
tool_diameter(KH_tool_no,5)^2))/2),
20 gcpcscad getypos()+tool_diameter(KH_tool_no, (kh_max_depth))/2,/( (
kh_max_depth-6.34))/2)^2-(tool_diameter(KH_tool_no, (
kh_max_depth-6.34))/2)^2,
21 gcpcscad getxpos()+kh_distance,
22 gcpcscad //end position at top of slot
23 gcpcscad getypos()+tool_diameter(KH_tool_no, (kh_max_depth))/2,
24 gcpcscad KH_tool_no);
25 gcpcscad //lower right slot
26 gcpcscad dxfpolyline(
27 gcpcscad getxpos()+(sqrt((tool_diameter(KH_tool_no,1)^2)-(
tool_diameter(KH_tool_no,5)^2))/2),
28 gcpcscad getypos()-tool_diameter(KH_tool_no, (kh_max_depth))/2,/( (
kh_max_depth-6.34))/2)^2-(tool_diameter(KH_tool_no, (
kh_max_depth-6.34))/2)^2,
29 gcpcscad getxpos()+kh_distance,
30 gcpcscad //end position at top of slot
31 gcpcscad getypos()-tool_diameter(KH_tool_no, (kh_max_depth))/2,
32 gcpcscad KH_tool_no);
33 gcpcscad hull(){
34 gcpcscad translate([xpos(), ypos(), zpos()]){
35 gcpcscad gcp_keyhole_shaft(6.35, 9.525);
36 gcpcscad }
37 gcpcscad translate([xpos(), ypos(), zpos()-kh_max_depth]){
38 gcpcscad gcp_keyhole_shaft(6.35, 9.525);
39 gcpcscad }
40 gcpcscad }
41 gcpcscad hull(){
42 gcpcscad translate([xpos(), ypos(), zpos()-kh_max_depth]){
43 gcpcscad gcp_keyhole_shaft(6.35, 9.525);
44 gcpcscad }
45 gcpcscad translate([xpos()+kh_distance, ypos(), zpos()-kh_max_depth]){
46 gcpcscad gcp_keyhole_shaft(6.35, 9.525);
47 gcpcscad }
48 gcpcscad }
49 gcpcscad cutwithfeed(getxpos(),getypos(),-kh_max_depth,feed);
50 gcpcscad cutwithfeed(getxpos()+kh_distance,getypos(),-kh_max_depth,feed)
;
51 gcpcscad setxpos(getxpos()-kh_distance);
52 gcpcscad } else if (kh_angle > 0 && kh_angle < 90) {
53 gcpcscad //echo(kh_angle);
54 gcpcscad dxfarc(getxpos(),getypos(),tool_diameter(KH_tool_no, (
kh_max_depth))/2,90+kh_angle,180+kh_angle, KH_tool_no);
55 gcpcscad dxfarc(getxpos(),getypos(),tool_diameter(KH_tool_no, (
kh_max_depth))/2,180+kh_angle,270+kh_angle, KH_tool_no);
56 gcpcscad dxfarc(getxpos(),getypos(),tool_diameter(KH_tool_no, (kh_max_depth)
)/2,kh_angle+asin((tool_diameter(KH_tool_no, (kh_max_depth+4.36)
)/2)/(tool_diameter(KH_tool_no, (kh_max_depth))/2)),90+kh_angle,
KH_tool_no);
57 gcpcscad dxfarc(getxpos(),getypos(),tool_diameter(KH_tool_no, (kh_max_depth)
)/2,270+kh_angle,360+kh_angle-asin((tool_diameter(KH_tool_no, (
kh_max_depth+4.36))/2)/(tool_diameter(KH_tool_no, (kh_max_depth)
)/2)), KH_tool_no);
58 gcpcscad dxfarc(getxpos()+(kh_distance*cos(kh_angle)),
59 gcpcscad getypos()+(kh_distance*sin(kh_angle)),tool_diameter(KH_tool_no, (
kh_max_depth+4.36))/2,0+kh_angle,90+kh_angle, KH_tool_no);
60 gcpcscad dxfarc(getxpos()+(kh_distance*cos(kh_angle)),getypos()+(kh_distance
*sin(kh_angle)),tool_diameter(KH_tool_no, (kh_max_depth+4.36)
)/2,270+kh_angle,360+kh_angle, KH_tool_no);
61 gcpcscad dxfpolyline( getxpos()+tool_diameter(KH_tool_no, (kh_max_depth))/2*
cos(kh_angle+asin((tool_diameter(KH_tool_no, (kh_max_depth+4.36)
)/2)/(tool_diameter(KH_tool_no, (kh_max_depth))/2))),
62 gcpcscad getypos()+tool_diameter(KH_tool_no, (kh_max_depth))/2*sin(kh_angle
+asin((tool_diameter(KH_tool_no, (kh_max_depth+4.36))/2)/(
tool_diameter(KH_tool_no, (kh_max_depth))/2))),
63 gcpcscad getxpos()+(kh_distance*cos(kh_angle))-((tool_diameter(KH_tool_no,

```

```

        (kh_max_depth+4.36))/2)*sin(kh_angle)),
64 gcpcscad  getypos()+(kh_distance*sin(kh_angle))+((tool_diameter(KH_tool_no,
        (kh_max_depth+4.36))/2)*cos(kh_angle)), KH_tool_no);
65 gcpcscad //echo("a",tool_diameter(KH_tool_no, (kh_max_depth+4.36))/2);
66 gcpcscad //echo("c",tool_diameter(KH_tool_no, (kh_max_depth))/2);
67 gcpcscad echo("Aangle",asin((tool_diameter(KH_tool_no, (kh_max_depth+4.36))
        /2)/(tool_diameter(KH_tool_no, (kh_max_depth))/2)));
68 gcpcscad //echo(kh_angle);
69 gcpcscad  cutwithfeed(getxpos()+(kh_distance*cos(kh_angle)),getypos()+(
        kh_distance*sin(kh_angle)),-kh_max_depth,feed);
70 gcpcscad  setxpos(getxpos()-(kh_distance*cos(kh_angle)));
71 gcpcscad  setypos(getypos()-(kh_distance*sin(kh_angle)));
72 gcpcscad  } else if (kh_angle == 90) {
73 gcpcscad    //Lower left of entry hole
74 gcpcscad    dxfarc(getxpos(),getypos(),9.525/2,180,270, KH_tool_no);
75 gcpcscad    //Lower right of entry hole
76 gcpcscad    dxfarc(getxpos(),getypos(),9.525/2,270,360, KH_tool_no);
77 gcpcscad    //Upper right of entry hole
78 gcpcscad    dxfarc(getxpos(),getypos(),9.525/2,0,acos(tool_diameter(
        KH_tool_no, 5)/tool_diameter(KH_tool_no, 1)), KH_tool_no);
79 gcpcscad    //Upper left of entry hole
80 gcpcscad    dxfarc(getxpos(),getypos(),9.525/2,180-acos(tool_diameter(
        KH_tool_no, 5)/tool_diameter(KH_tool_no, 1)), 180,KH_tool_no
        );
81 gcpcscad    //Actual line of cut
82 gcpcscad    dxfpolyline(getxpos(),getypos(),getxpos(),getypos()+kh_distance
        );
83 gcpcscad    //upper right of slot
84 gcpcscad    dxfarc(getxpos(),getypos()+kh_distance,tool_diameter(KH_tool_no
        , (kh_max_depth+4.36))/2,0,90, KH_tool_no);
85 gcpcscad    //upper left of slot
86 gcpcscad    dxfarc(getxpos(),getypos()+kh_distance,tool_diameter(KH_tool_no
        , (kh_max_depth+6.35))/2,90,180, KH_tool_no);
87 gcpcscad    //right of slot
88 gcpcscad    dxfpolyline(
89 gcpcscad      getxpos()+tool_diameter(KH_tool_no, (kh_max_depth))/2,
90 gcpcscad      getypos()+(sqrt((tool_diameter(KH_tool_no,1)^2)-(
        tool_diameter(KH_tool_no,5)^2))/2),//( (kh_max_depth
        -6.34))/2)^2-(tool_diameter(KH_tool_no, (kh_max_depth
        -6.34))/2)^2,
91 gcpcscad      getxpos()+tool_diameter(KH_tool_no, (kh_max_depth))/2,
92 gcpcscad    //end position at top of slot
93 gcpcscad      getypos()+kh_distance,
94 gcpcscad      KH_tool_no);
95 gcpcscad    dxfpolyline(getxpos()-tool_diameter(KH_tool_no, (kh_max_depth))
        /2, getypos()+(sqrt((tool_diameter(KH_tool_no,1)^2)-(
        tool_diameter(KH_tool_no,5)^2))/2), getxpos()-tool_diameter(
        KH_tool_no, (kh_max_depth+6.35))/2,getypos()+kh_distance,
        KH_tool_no);
96 gcpcscad    hull(){
97 gcpcscad      translate([xpos(), ypos(), zpos()]){
98 gcpcscad        gcp_keyhole_shaft(6.35, 9.525);
99 gcpcscad      }
100 gcpcscad      translate([xpos(), ypos(), zpos()-kh_max_depth]){
101 gcpcscad        gcp_keyhole_shaft(6.35, 9.525);
102 gcpcscad      }
103 gcpcscad    }
104 gcpcscad    hull(){
105 gcpcscad      translate([xpos(), ypos(), zpos()-kh_max_depth]){
106 gcpcscad        gcp_keyhole_shaft(6.35, 9.525);
107 gcpcscad      }
108 gcpcscad      translate([xpos(), ypos()+kh_distance, zpos()-kh_max_depth]){
109 gcpcscad        gcp_keyhole_shaft(6.35, 9.525);
110 gcpcscad      }
111 gcpcscad    }
112 gcpcscad    cutwithfeed(getxpos(),getypos(),-kh_max_depth,feed);
113 gcpcscad    cutwithfeed(getxpos(),getypos()+kh_distance,-kh_max_depth,feed)
        ;
114 gcpcscad    setypos(getypos()-kh_distance);
115 gcpcscad  } else if (kh_angle == 180) {
116 gcpcscad    //Lower right of entry hole
117 gcpcscad    dxfarc(getxpos(),getypos(),9.525/2,270,360, KH_tool_no);
118 gcpcscad    //Upper right of entry hole
119 gcpcscad    dxfarc(getxpos(),getypos(),9.525/2,0,90, KH_tool_no);
120 gcpcscad    //Upper left of entry hole
121 gcpcscad    dxfarc(getxpos(),getypos(),9.525/2,90, 90+acos(tool_diameter(
        KH_tool_no, 5)/tool_diameter(KH_tool_no, 1)), KH_tool_no);
122 gcpcscad    //Lower left of entry hole

```

```

123 gcpscad      dxfarcc(getxpos(),getypos(),9.525/2, 270-acos(tool_diameter(
                KH_tool_no, 5)/tool_diameter(KH_tool_no, 1)), 270,
                KH_tool_no);
124 gcpscad      //upper left of slot
125 gcpscad      dxfarcc(getxpos()-kh_distance,getypos(),tool_diameter(KH_tool_no
                , (kh_max_depth+6.35))/2,90,180, KH_tool_no);
126 gcpscad      //lower left of slot
127 gcpscad      dxfarcc(getxpos()-kh_distance,getypos(),tool_diameter(KH_tool_no
                , (kh_max_depth+6.35))/2,180,270, KH_tool_no);
128 gcpscad      //Actual line of cut
129 gcpscad      dxfpolyline(getxpos(),getypos(),getxpos()-kh_distance,getypos()
                );
130 gcpscad      //upper left slot
131 gcpscad      dxfpolyline(
132 gcpscad          getxpos()-(sqrt((tool_diameter(KH_tool_no,1)^2)-(
                tool_diameter(KH_tool_no,5)^2))/2),
133 gcpscad          getypos()+tool_diameter(KH_tool_no, (kh_max_depth))/2,/( (
                kh_max_depth-6.34))/2)^2-(tool_diameter(KH_tool_no, (
                kh_max_depth-6.34))/2)^2,
134 gcpscad          getxpos()-kh_distance,
135 gcpscad      //end position at top of slot
136 gcpscad          getypos()+tool_diameter(KH_tool_no, (kh_max_depth))/2,
137 gcpscad          KH_tool_no);
138 gcpscad      //lower right slot
139 gcpscad      dxfpolyline(
140 gcpscad          getxpos()-(sqrt((tool_diameter(KH_tool_no,1)^2)-(
                tool_diameter(KH_tool_no,5)^2))/2),
141 gcpscad          getypos()-tool_diameter(KH_tool_no, (kh_max_depth))/2,/( (
                kh_max_depth-6.34))/2)^2-(tool_diameter(KH_tool_no, (
                kh_max_depth-6.34))/2)^2,
142 gcpscad          getxpos()-kh_distance,
143 gcpscad      //end position at top of slot
144 gcpscad          getypos()-tool_diameter(KH_tool_no, (kh_max_depth))/2,
145 gcpscad          KH_tool_no);
146 gcpscad      hull(){
147 gcpscad          translate([xpos(), ypos(), zpos()]){
148 gcpscad              gcp_keyhole_shaft(6.35, 9.525);
149 gcpscad          }
150 gcpscad          translate([xpos(), ypos(), zpos()-kh_max_depth]){
151 gcpscad              gcp_keyhole_shaft(6.35, 9.525);
152 gcpscad          }
153 gcpscad      }
154 gcpscad      hull(){
155 gcpscad          translate([xpos(), ypos(), zpos()-kh_max_depth]){
156 gcpscad              gcp_keyhole_shaft(6.35, 9.525);
157 gcpscad          }
158 gcpscad          translate([xpos()-kh_distance, ypos(), zpos()-kh_max_depth]){
159 gcpscad              gcp_keyhole_shaft(6.35, 9.525);
160 gcpscad          }
161 gcpscad      }
162 gcpscad      cutwithfeed(getxpos(),getypos(),-kh_max_depth,feed);
163 gcpscad      cutwithfeed(getxpos()-kh_distance,getypos(),-kh_max_depth,feed)
                ;
164 gcpscad      setxpos(getxpos()+kh_distance);
165 gcpscad } else if (kh_angle == 270) {
166 gcpscad     //Upper right of entry hole
167 gcpscad     dxfarcc(getxpos(),getypos(),9.525/2,0,90, KH_tool_no);
168 gcpscad     //Upper left of entry hole
169 gcpscad     dxfarcc(getxpos(),getypos(),9.525/2,90,180, KH_tool_no);
170 gcpscad     //lower right of slot
171 gcpscad     dxfarcc(getxpos(),getypos()-kh_distance,tool_diameter(KH_tool_no
                , (kh_max_depth+4.36))/2,270,360, KH_tool_no);
172 gcpscad     //lower left of slot
173 gcpscad     dxfarcc(getxpos(),getypos()-kh_distance,tool_diameter(KH_tool_no
                , (kh_max_depth+4.36))/2,180,270, KH_tool_no);
174 gcpscad     //Actual line of cut
175 gcpscad     dxfpolyline(getxpos(),getypos(),getxpos(),getypos()-kh_distance
                );
176 gcpscad     //right of slot
177 gcpscad     dxfpolyline(
178 gcpscad         getxpos()+tool_diameter(KH_tool_no, (kh_max_depth))/2,
179 gcpscad         getypos()-(sqrt((tool_diameter(KH_tool_no,1)^2)-(
                tool_diameter(KH_tool_no,5)^2))/2),/( (kh_max_depth
                -6.34))/2)^2-(tool_diameter(KH_tool_no, (kh_max_depth
                -6.34))/2)^2,
180 gcpscad         getxpos()+tool_diameter(KH_tool_no, (kh_max_depth))/2,
181 gcpscad     //end position at top of slot
182 gcpscad         getypos()-kh_distance,

```

```

183 gcpscad          KH_tool_no);
184 gcpscad          //left of slot
185 gcpscad          dxfpolyline(
186 gcpscad              getxpos()-tool_diameter(KH_tool_no, (kh_max_depth))/2,
187 gcpscad              getypos()-(sqrt((tool_diameter(KH_tool_no,1)^2)-(
                    tool_diameter(KH_tool_no,5)^2))/2),//( (kh_max_depth
                    -6.34))/2)^2-(tool_diameter(KH_tool_no, (kh_max_depth
                    -6.34))/2)^2,
188 gcpscad              getxpos()-tool_diameter(KH_tool_no, (kh_max_depth))/2,
189 gcpscad          //end position at top of slot
190 gcpscad              getypos()-kh_distance,
191 gcpscad              KH_tool_no);
192 gcpscad          //Lower right of entry hole
193 gcpscad          dxfarc(getxpos(),getypos(),9.525/2,360-acos(tool_diameter(
                    KH_tool_no, 5)/tool_diameter(KH_tool_no, 1)), 360,
                    KH_tool_no);
194 gcpscad          //Lower left of entry hole
195 gcpscad          dxfarc(getxpos(),getypos(),9.525/2,180, 180+acos(tool_diameter(
                    KH_tool_no, 5)/tool_diameter(KH_tool_no, 1)), KH_tool_no);
196 gcpscad          hull(){
197 gcpscad              translate([xpos(), ypos(), zpos()]){
198 gcpscad                  gcp_keyhole_shaft(6.35, 9.525);
199 gcpscad              }
200 gcpscad              translate([xpos(), ypos(), zpos()-kh_max_depth]){
201 gcpscad                  gcp_keyhole_shaft(6.35, 9.525);
202 gcpscad              }
203 gcpscad          }
204 gcpscad          hull(){
205 gcpscad              translate([xpos(), ypos(), zpos()-kh_max_depth]){
206 gcpscad                  gcp_keyhole_shaft(6.35, 9.525);
207 gcpscad              }
208 gcpscad              translate([xpos(), ypos()-kh_distance, zpos()-kh_max_depth]){
209 gcpscad                  gcp_keyhole_shaft(6.35, 9.525);
210 gcpscad              }
211 gcpscad          }
212 gcpscad          cutwithfeed(getxpos(),getypos(),-kh_max_depth,feed);
213 gcpscad          cutwithfeed(getxpos(),getypos()-kh_distance,-kh_max_depth,feed)
214 gcpscad          ;
215 gcpscad          setypos(getypos()+kh_distance);
216 gcpscad      }

```

3.3 Shapes and tool movement

The majority of commands will be more general, focusing on tooling which is generally supported by this library, moving in lines and arcs so as to describe shapes which lend themselves to representation with those tool and which match up with both toolpaths and supported geometry in Carbide Create, and the usage requirements of the typical user.

3.3.1 Generalized commands and cuts

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

3.3.1.1 begincutdxf The first command, `begincutdxf` will need to allow the machine to rapid to the beginning point of the cut and then rapid down to the surface of the stock, and then plunge down to the depth of the cut. The implementation will need to allow for a hook where the Depth per Pass is applied to the plunge operation so that multiple passes are made.

The first module will ensure that the tool is safely up above the stock and will rapid to the position specified at the retract height (moving to that position as an initial step, then will `cutwithfeed` to the specified position at the specified feed rate. Despite `dxf` being included in the filename no change is made to the `dxf` file at this time, this simply indicates that this file is preparatory to the use of `continuecutdxf`.

```

867 gcpscad module begincutdxf(rh, ex, ey, ez, fr) {
868 gcpscad     rapid(getxpos(),getypos(),rh);
869 gcpscad     cutwithfeed(ex,ey,ez,fr);
870 gcpscad }

```

```

872 gcpscad module continuecutdxf(ex, ey, ez, fr) {
873 gcpscad     cutwithfeed(ex,ey,ez,fr);
874 gcpscad }

```

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

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

cutrectangledxf Th routine cutrectangledxfcuts the outline of a rectangle creating sharp corners. Note that the initial version would work as a beginning point for vertical cutting if the hull() operation was removed and the loop was uncommented:

```
876 gpcscad module cutrectangledxf(bx, by, bz, rwidth, rheight, rdepth, rtn)
      { //passes
877 gpcscad   movetosafez();
878 gpcscad   hull(){
879 gpcscad     // for (i = [0 : abs(1) : passes]) {
880 gpcscad     //   rapid(bx+tool_radius(rtn)+i*(rwidth-tool_diameter(
      current_tool()))/passes,bx+tool_radius(rtn),1);
881 gpcscad     //   cutwithfeed(bx+tool_radius(rtn)+i*(rwidth-tool_diameter
      (current_tool()))/passes,by+tool_radius(rtn),bz-rdepth,feed)
      ;
882 gpcscad     //   cutwithfeed(bx+tool_radius(rtn)+i*(rwidth-tool_diameter
      (current_tool()))/passes,by+rheight-tool_radius(rtn),bz-
      rdepth,feed);
883 gpcscad
884 gpcscad     cutwithfeed(bx+tool_radius(rtn),by+tool_radius(rtn),bz-rdepth,
      feed);
885 gpcscad     cutwithfeed(bx+rwidth-tool_radius(rtn),by+tool_radius(rtn),bz-
      rdepth,feed);
886 gpcscad     cutwithfeed(bx+rwidth-tool_radius(rtn),by+rheight-tool_radius(
      rtn),bz-rdepth,feed);
887 gpcscad     cutwithfeed(bx+tool_radius(rtn),by+rheight-tool_radius(rtn),bz-
      rdepth,feed);
888 gpcscad   }
889 gpcscad   //dxfarc(xcenter,ycenter,radius,anglebegin,endangle, tn)
890 gpcscad   dxfarc(bx+tool_radius(rtn),by+tool_radius(rtn),tool_radius(rtn)
      ,180,270, rtn);
891 gpcscad   //dxfpolyline(xbegin,ybegin,xend,yend, tn)
892 gpcscad   dxfpolyline(bx,by+tool_radius(rtn),bx,by+rheight-tool_radius(rtn)
      , rtn);
893 gpcscad   dxfarc(bx+tool_radius(rtn),by+rheight-tool_radius(rtn),
      tool_radius(rtn),90,180, rtn);
894 gpcscad   dxfpolyline(bx+tool_radius(rtn),by+rheight,bx+rwidth-tool_radius(
      rtn),by+rheight, rtn);
895 gpcscad   dxfarc(bx+rwidth-tool_radius(rtn),by+rheight-tool_radius(rtn),
      tool_radius(rtn),0,90, rtn);
896 gpcscad   dxfpolyline(bx+rwidth,by+rheight-tool_radius(rtn),bx+rwidth,by+
      tool_radius(rtn), rtn);
897 gpcscad   dxfarc(bx+rwidth-tool_radius(rtn),by+tool_radius(rtn),tool_radius
      (rtn),270,360, rtn);
898 gpcscad   dxfpolyline(bx+rwidth-tool_radius(rtn),by,bx+tool_radius(rtn),by,
      rtn);
899 gpcscad }
```

cutrectangleoutlinedxf A matching command: cutrectangleoutlinedxf cuts the outline of a rounded rectangle and is a simplification of the above:

```
901 gpcscad module cutrectangleoutlinedxf(bx, by, bz, rwidth, rheight, rdepth,
      rtn) { //passes
902 gpcscad   movetosafez();
903 gpcscad   cutwithfeed(bx+tool_radius(rtn),by+tool_radius(rtn),bz-rdepth,
      feed);
904 gpcscad   cutwithfeed(bx+rwidth-tool_radius(rtn),by+tool_radius(rtn),bz-
      rdepth,feed);
905 gpcscad   cutwithfeed(bx+rwidth-tool_radius(rtn),by+rheight-tool_radius(rtn)
      ,bz-rdepth,feed);
906 gpcscad   cutwithfeed(bx+tool_radius(rtn),by+rheight-tool_radius(rtn),bz-
      rdepth,feed);
907 gpcscad   dxfarc(bx+tool_radius(rtn),by+tool_radius(rtn),tool_radius(rtn)
      ,180,270, rtn);
908 gpcscad   dxfpolyline(bx,by+tool_radius(rtn),bx,by+rheight-tool_radius(rtn)
      , rtn);
909 gpcscad   dxfarc(bx+tool_radius(rtn),by+rheight-tool_radius(rtn),
```

```

    tool_radius(rtn),90,180, rtn);
910 gcpscad  dxfpolyline(bx+tool_radius(rtn),by+rheight,bx+rwidth-tool_radius(
    rtn),by+rheight, rtn);
911 gcpscad  dxfarc(bx+rwidth-tool_radius(rtn),by+rheight-tool_radius(rtn),
    tool_radius(rtn),0,90, rtn);
912 gcpscad  dxfpolyline(bx+rwidth,by+rheight-tool_radius(rtn),bx+rwidth,by+
    tool_radius(rtn), rtn);
913 gcpscad  dxfarc(bx+rwidth-tool_radius(rtn),by+tool_radius(rtn),tool_radius
    (rtn),270,360, rtn);
914 gcpscad  dxfpolyline(bx+rwidth-tool_radius(rtn),by,bx+tool_radius(rtn),by,
    rtn);
915 gcpscad }
```

rectangleoutlinedxf Which suggests a further command, rectangleoutlinedxf for simply adding a rectangle (a potential use of which would be in Job Setup to add the stock outline to DXFs to assist in registration of jobs with multiple tools):

```

917 gcpscad module rectangleoutlinedxf(bx, by, bz, rwidth, rheight, rtn) {
918 gcpscad  dxfpolyline(bx,by,bx,by+rheight, rtn);
919 gcpscad  dxfpolyline(bx,by+rheight,bx+rwidth,by+rheight, rtn);
920 gcpscad  dxfpolyline(bx+rwidth,by+rheight,bx+rwidth,by, rtn);
921 gcpscad  dxfpolyline(bx+rwidth,by,bx,by, rtn);
922 gcpscad }
```

the initial section performs the cutting operation for the 3D preview while the latter section writes out the outline to the DXF files.

cutoutrectangledxf A variant of the cutting version of that file, cutoutrectangledxf will cut to the outside:

```

924 gcpscad module cutoutrectangledxf(bx, by, bz, rwidth, rheight, rdepth, rtn)
    {
925 gcpscad  movetosafez();
926 gcpscad  cutwithfeed(bx-tool_radius(rtn),by-tool_radius(rtn),bz-rdepth,
    feed);
927 gcpscad  cutwithfeed(bx+rwidth+tool_radius(rtn),by-tool_radius(rtn),bz-
    rdepth,feed);
928 gcpscad  cutwithfeed(bx+rwidth+tool_radius(rtn),by+rheight+tool_radius(rtn
    ),bz-rdepth,feed);
929 gcpscad  cutwithfeed(bx-tool_radius(rtn),by+rheight+tool_radius(rtn),bz-
    rdepth,feed);
930 gcpscad  cutwithfeed(bx-tool_radius(rtn),by-tool_radius(rtn),bz-rdepth,
    feed);
931 gcpscad  dxfpolyline(bx,by,bx,by+rheight, rtn);
932 gcpscad  dxfpolyline(bx,by+rheight,bx+rwidth,by+rheight, rtn);
933 gcpscad  dxfpolyline(bx+rwidth,by+rheight,bx+rwidth,by, rtn);
934 gcpscad  dxfpolyline(bx+rwidth,by,bx,by, rtn);
935 gcpscad }
```

3.4 Expansion

The balance of shapes will go into cut2Dshapes.scad and of course it will be possible to create additional files for specific purposes.

```

1 cut2D //! OpenSCAD
```

4 gcodepreviewtemplate.scad

The commands may then be put together using a template which will ensure that the various files are used/included as necessary, that files are opened before being written to, and that they are closed at the end.

```

1 gcptmpl //! OpenSCAD
2 gcptmpl
3 gcptmpl use <gcodepreview.py>;
4 gcptmpl use <pygcodepreview.scad>;
5 gcptmpl include <gcodepreview.scad>;
6 gcptmpl
7 gcptmpl $fa = 2;
8 gcptmpl $fs = 0.125;
9 gcptmpl
10 gcptmpl /* [Export] */
11 gcptmpl Base_filename = "export";
12 gcptmpl /* [Export] */
```

```

13 gcptmpl generatedxf = true;
14 gcptmpl /* [Export] */
15 gcptmpl generategcode = true;
16 gcptmpl /*** [Export] */
17 gcptmpl //generatesvg = false;
18 gcptmpl
19 gcptmpl /* [CAM] */
20 gcptmpl toolradius = 1.5875;
21 gcptmpl /* [CAM] */
22 gcptmpl large_ball_tool_no = 0; // [0:0,111:111,101:101,202:202]
23 gcptmpl /* [CAM] */
24 gcptmpl large_square_tool_no = 0; // [0:0,112:112,102:102,201:201]
25 gcptmpl /* [CAM] */
26 gcptmpl large_V_tool_no = 0; // [0:0,301:301,690:690]
27 gcptmpl /* [CAM] */
28 gcptmpl small_ball_tool_no = 0; // [0:0,121:121,111:111,101:101]
29 gcptmpl /* [CAM] */
30 gcptmpl small_square_tool_no = 102; // [0:0,122:122,112:112,102:102]
31 gcptmpl /* [CAM] */
32 gcptmpl small_V_tool_no = 0; // [0:0,390:390,301:301]
33 gcptmpl /* [CAM] */
34 gcptmpl KH_tool_no = 0; // [0:0,374:374,375:375,376:376,378]
35 gcptmpl /* [CAM] */
36 gcptmpl DT_tool_no = 0; // [0:0,814:814]
37 gcptmpl
38 gcptmpl /* [Feeds and Speeds] */
39 gcptmpl plunge = 100;
40 gcptmpl /* [Feeds and Speeds] */
41 gcptmpl feed = 400;
42 gcptmpl /* [Feeds and Speeds] */
43 gcptmpl speed = 16000;
44 gcptmpl /* [Feeds and Speeds] */
45 gcptmpl small_square_ratio = 0.75; // [0.25:2]
46 gcptmpl /* [Feeds and Speeds] */
47 gcptmpl small_ball_ratio = 0.75; // [0.25:2]
48 gcptmpl /* [Feeds and Speeds] */
49 gcptmpl large_ball_ratio = 1.0; // [0.25:2]
50 gcptmpl /* [Feeds and Speeds] */
51 gcptmpl small_V_ratio = 0.625; // [0.25:2]
52 gcptmpl /* [Feeds and Speeds] */
53 gcptmpl large_V_ratio = 0.875; // [0.25:2]
54 gcptmpl /* [Feeds and Speeds] */
55 gcptmpl KH_ratio = 0.75; // [0.25:2]
56 gcptmpl /* [Feeds and Speeds] */
57 gcptmpl DT_ratio = 0.75; // [0.25:2]
58 gcptmpl
59 gcptmpl /* [Stock] */
60 gcptmpl stocklength = 219;
61 gcptmpl /* [Stock] */
62 gcptmpl stockwidth = 150;
63 gcptmpl /* [Stock] */
64 gcptmpl stockthickness = 8.35;
65 gcptmpl /* [Stock] */
66 gcptmpl zeroheight = "Top"; // [Top, Bottom]
67 gcptmpl /* [Stock] */
68 gcptmpl stockorigin = "Center"; // [Lower-Left, Center-Left, Top-Left,
    Center]
69 gcptmpl /* [Stock] */
70 gcptmpl retractheight = 9;
71 gcptmpl
72 gcptmpl filename_gcode = str(Base_filename, ".nc");
73 gcptmpl filename_dxf = str(Base_filename);
74 gcptmpl //filename_svg = str(Base_filename, ".svg");
75 gcptmpl
76 gcptmpl.opengcodefile(filename_gcode);
77 gcptmpl.opendxf(file(filename_dxf));
78 gcptmpl
79 gcptmpl difference() {
80 gcptmpl   setupstock(stocklength, stockwidth, stockthickness, zeroheight,
    stockorigin);
81 gcptmpl
82 gcptmpl movetosafez();
83 gcptmpl
84 gcptmpl toolchange(small_square_tool_no,speed * small_square_ratio);
85 gcptmpl
86 gcptmpl begintoolpath(0,0,0.25);
87 gcptmpl beginpolyline(0,0,0.25);
88 gcptmpl

```



```
89 gcptmpl cutoneaxis_setfeed("Z",0,plunge*small_square_ratio);
90 gcptmpl
91 gcptmpl cutwithfeed(stocklength/2,stockwidth/2,-stockthickness,feed);
92 gcptmpl addpolyline(stocklength/2,stockwidth/2,-stockthickness);
93 gcptmpl
94 gcptmpl endtoolpath();
95 gcptmpl closepolyline();
96 gcptmpl }
97 gcptmpl
98 gcptmpl closegcodefile();
99 gcptmpl closedxfiler();
```

Note that the line:
toolchange(small_square_tool_no,speed * square_ratio);
may be commented out — whether or no it is actually necessary will need to be decided upon.

4.1 G-code and modules and commands

Each module/command will write out certain G-code commands:

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

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

G-code	Command/Module
(Design File:) (stockMin:0.00mm, -152.40mm, -34.92mm) (stockMax:109.50mm, -77.40mm, 0.00mm) (STOCK/BLOCK,109.50, 75.00, 34.92,0.00, 152.40, 34.92) G90 G21	opengcodefile(...); 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) G0X0.000Y-152.400 Z0.250	writecomment(...) rapid(...) rapid(...)
G1Z-1.000F203.2 X109.500Y-77.400F508.0 Z12.700	cutwithfeed(...); cutwithfeed(...); rapid(...)
M05 M02	closegcodefile();

4.2 DXF

```
0
SECTION
2
ENTITIES
0
POLYLINE
8
default
66
1
70
0
0
0
VERTEX
8
default
70
32
10
0
20
0
0
0
VERTEX
8
default
70
32
10
109.5
20
75
0
SEQEND
0
ENDSEC
0
EOF
```

5 Future

5.1 Images

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

5.2 Generalized DXF creation

Generalize the creation of DXFs based on the `projection()` of a toolpath?

5.3 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>

5.4 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/>
 c.f., <https://linuxcnc.org/docs/html/gcode/g-code.html#gcode:g5>

5.5 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:

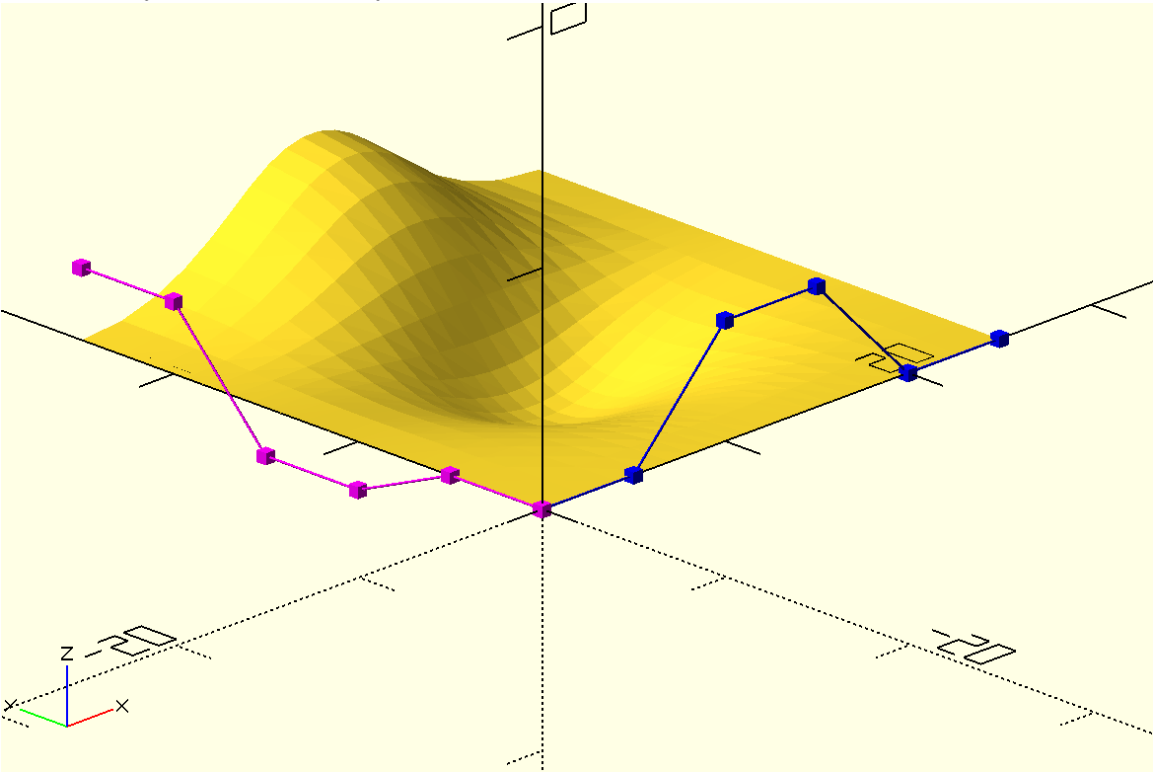
- XY
- XZ
- ZY

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:

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

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>

6 Other Resources

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

References

[ConstGeom]	Walmsley, Brian. <i>Construction Geometry</i> . 2d ed., Centennial College Press, 1981.
[MkCalc]	Horvath, Joan, and Rich Cameron. <i>Make: Calculus: Build models to learn, visualize, and explore</i> . First edition., Make: Community LLC, 2022.
[MkGeom]	Horvath, Joan, and Rich Cameron. <i>Make: Geometry: Learn by 3D Printing, Coding and Exploring</i> . First edition., Make: Community LLC, 2021.
[MkTrig]	Horvath, Joan, and Rich Cameron. <i>Make: Trigonometry: Build your way from triangles to analytic geometry</i> . First edition., Make: Community LLC, 2023.
[PractShopMath]	Begnal, Tom. <i>Practical Shop Math: Simple Solutions to Workshop Fractions, Formulas + Geometric Shapes</i> . 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. <i>A Philosophy of Software Design</i> . First Edition., Yaknyam Press, Palo Alto, Ca., 2018

Index

- arcloop, 31
- begincutdxf, 37
- closedxfile, 25, 26
 - oclosedxfile, 26
- closegcodefile, 26
 - oclosegcodefile, 26
 - pclosegcodefile, 25
- continuecutdxf, 37
- current tool, 11
 - pcurrenttool, 11
- currenttool, 5, 13
- cut
 - ocut, 27
- cutkeyhole toolpath, 33
- cutKH toolpath degrees, 33
- cutoutrectangledxf, 39
- cutrectangledxf, 38
- cutrectangleoutlinedxf, 38
- cutroundover, 15
- dxfa, 24
- dxfarc, 23, 24
- dxfbpl, 23
- dxfpreamble, 26
- dxfpreamble, 22
- dxfwrite, 22
- dxfwritelgbl, 21
- dxfwritelgsq, 21
- dxfwritelgV, 21
- dxfwritetone, 21
- dxfwritesmbl, 21
- dxfwritesmsq, 21
- dxfwritesmV, 21
- feed, 17
- gcp dovetail, 14
- gcp endmill ball, 15
- gcp endmill square, 14
- gcp endmill v, 15
- gcp keyhole, 14
- gettzpos, 10
- getxpos, 10
- getypos, 10
- getzpos, 10
- mpx, 5
- mpy, 5
- mpz, 5
- narcloop, 31
- opendxfile, 19
 - oopendxfile, 18
 - popendxfile, 18
- opengcodefile, 19
 - oopengcodefile, 18
 - popengcodefile, 18
- osettool, 11
- otm, 27
- owrite..., 21
- owritecomment, 21
- plunge, 17
- popendxflgblfile, 18
- popendxflgsqfile, 18
- popendxflgVfile, 18
- popendxfsmblfile, 18
- popendxfsmsqfile, 18
- popendxfsmVfile, 18
- psettool, 11
- rapid, 27
 - orapid, 27
- rapidbx, 27
- rectangleoutlinedxf, 39
- selecttool, 13
- set...
 - oset, 10
 - osettz, 11
- settzpos, 10
 - psettzpos, 10
- setupstock, 7, 8
 - osetupstock, 8
 - psetupstock, 7
- setxpos, 10
 - psetxpos, 10
- setypos, 10
 - psetypos, 10
- setzpos, 10
 - psetzpos, 10
- speed, 17
- subroutine
 - oclosedxfile, 26
 - oclosegcodefile, 26
 - ocut, 27
 - oopendxfile, 18
 - oopengcodefile, 18
 - orapid, 27
 - oset, 10
 - osettz, 11
 - osetupstock, 8
 - otool diameter, 16
 - pclosegcodefile, 25
 - pcurrenttool, 11
 - popendxfile, 18
 - popengcodefile, 18
 - psettzpos, 10
 - psetupstock, 7
 - psetxpos, 10
 - psetypos, 10
 - psetzpos, 10
 - ptool diameter, 16
- tool diameter, 16
 - otool diameter, 16
 - ptool diameter, 16
- tool radius, 17
- toolchange, 11
- tpz, 5
- writedxf, 20
- writedxfDT, 20
- writedxfKH, 20
- writedxfkgbl, 20
- writedxfkgV, 20
- writedxfsmbl, 20
- writedxfsmsq, 20
- writedxfsmV, 20
- writeln, 5
- xpos, 10
- ypos, 10
- zpos, 10

Routines

- arcloop, 31
- begincutdx, 37
- closedxfile, 25, 26
- closegcodefile, 26
- continuecutdx, 37
- current tool, 11
- cutkeyhole toolpath, 33
- cutKH toolpath degrees, 33
- cutoutrectangledx, 39
- cutrectangledx, 38
- cutrectangleoutlinedx, 38
- cutroundover, 15
- dxfa, 24
- dxfar, 23, 24
- dxfbpl, 23
- dxfpreamble, 26
- dxfpreamble, 22
- dxfwrite, 22
- dxfwritelgbl, 21
- dxfwritelgsq, 21
- dxfwritelgV, 21
- dxfwriteone, 21
- dxfwritesmbl, 21
- dxfwritesmsq, 21
- dxfwritesmV, 21
- gcp dovetail, 14
- gcp endmill ball, 15
- gcp endmill square, 14
- gcp endmill v, 15
- gcp keyhole, 14
- gettzpos, 10
- getxpos, 10
- getypos, 10
- getzpos, 10
- narcloop, 31
- oclosedxfile, 26
- oclosegcodefile, 26
- ocut, 27
- oopenxfile, 18
- oopengcodefile, 18
- opendxfile, 19
- opengcodefile, 19
- orapid, 27
- oset, 10
- osettool, 11
- osettz, 11
- osetupstock, 8
- otm, 27
- otool diameter, 16
- owrite..., 21
- owritecomment, 21
- pclosegcodefile, 25
- pcurrenttool, 11
- popendxfile, 18
- popendxflgblfile, 18
- popendxflgsqfile, 18
- popendxflgVfile, 18
- popendxfsmbfile, 18
- popendxfsmsqfile, 18
- popendxfsmVfile, 18
- popengcodefile, 18
- psettool, 11
- psettzpos, 10
- psetupstock, 7
- psetxpos, 10
- psetypos, 10
- psetzpos, 10
- ptool diameter, 16
- rapid, 27
- rapidbx, 27
- rectangleoutlinedx, 39
- selecttool, 13
- settzpos, 10
- setupstock, 7, 8
- setxpos, 10
- setypos, 10
- setzpos, 10
- tool diameter, 16
- tool radius, 17
- toolchange, 11
- writedx, 20
- writedxDT, 20
- writedxKH, 20
- writedxflgbl, 20
- writedxflgsq, 20
- writedxflgV, 20
- writedxfsmb, 20
- writedxfsmsq, 20
- writedxsmV, 20
- writeln, 5
- xpos, 10
- ypos, 10
- zpos, 10

Variables

currenttool, 5, 13	plunge, 17
feed, 17	speed, 17
mpx, 5	tpz, 5
mpy, 5	
mpz, 5	