The gcodepreview OpenSCAD library*

Author: William F. Adams
willadams at aol dot com

2024/08/10

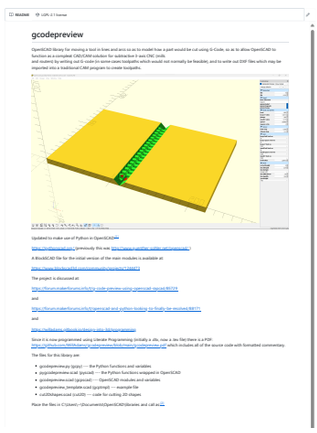
Abstract

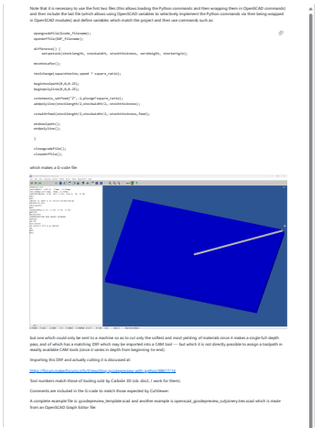
The gcodepreview library allows using OpenPythonSCAD to move a tool in lines and arcs and output dxf and G-code files so as to work as a CAD/CAM program for CNC.

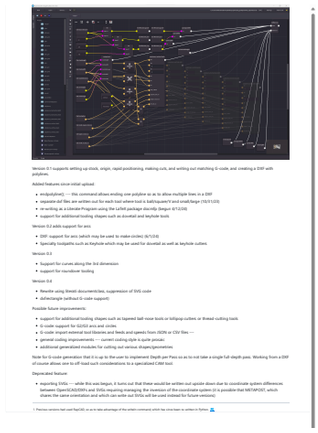
Contents

*This file (gcodepreview) has version number v0.5, last revised 2024/08/10.

1 readme.md







```
rdme # gcodepreview
rdme
rdme OpenSCAD library for moving a tool in lines and arcs
rdme so as to model how a part would be cut using G-Code,
rdme so as to allow OpenSCAD to function as a compleat
rdme CAD/CAM solution for subtractive 3-axis CNC (mills
rdme and routers) by writing out G-code (in some cases
rdme toolpaths which would not normally be feasible),
rdme and to write out DXF files which may be imported
rdme into a traditional CAM program to create toolpaths.
rdme
rdme ![OpenSCAD Cut Joinery Module](https://raw.githubusercontent.com/
         WillAdams/gcodepreview/main/openscad_cutjoinery.png?raw=true)
rdme
rdme Updated to make use of Python in OpenSCAD:[^rapcad]
rdme
rdme [^rapcad]: Previous versions had used RapCAD, so as to take
         advantage of the writeln command, which has since been re-
         written in Python.
rdme
rdme https://pythonscad.org/ (previously this was http://www.guenther-
         sohler.net/openscad/ )
rdme
rdme A BlockSCAD file for the initial version of the
rdme main modules is available at:
rdme
rdme https://www.blockscad3d.com/community/projects/1244473
rdme
rdme The project is discussed at:
rdme
rdme https://forum.makerforums.info/t/g-code-preview-using-openscad-
         rapcad/85729
rdme
rdme and
rdme
rdme https://forum.makerforums.info/t/openscad-and-python-looking-to-
         finally-be-resolved/88171
rdme
rdme and
rdme
rdme https://willadams.gitbook.io/design-into-3d/programming
rdme
rdme Since it is now programmed using Literate Programming
rdme (initially a .dtx, now a .tex file) there is a PDF:
rdme https://github.com/WillAdams/gcodepreview/blob/main/gcodepreview.
         pdf
rdme which includes all of the source code with formatted
rdme commentary.
rdme
rdme The files for this library are:
rdme
rdme  - gcodepreview.py (gcpy) --- the Python functions and variables
rdme  - pygcodepreview.scad (pyscad) --- the Python functions wrapped in
          OpenSCAD
rdme  - gcodepreview.scad (gcpscad) --- OpenSCAD modules and variables
rdme  - gcodepreview_template.scad (gcptmpl) --- example file
rdme  - cut2Dshapes.scad (cut2D) --- code for cutting 2D shapes
rdme
```

```
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 Possible future improvements:
155 rdme
156 rdme  - support for additional tooling shapes such as tapered ball-nose
             tools or lollipop cutters or thread-cutting tools
157 rdme  - G-code: support for G2/G3 arcs and circles
158 rdme  - G-code: import external tool libraries and feeds and speeds from
              JSON or CSV files ---
159 rdme  - general coding improvements --- current coding style is quite
             prosaic
160 rdme  - additional generalized modules for cutting out various shapes/
             geometries
161 rdme
162 rdme Note for G-code generation that it is up to the user
163 rdme to implement Depth per Pass so as to not take a
164 rdme single full-depth pass. Working from a DXF of course
165 rdme allows one to off-load such considerations to a
166 rdme specialized CAM tool.
167 rdme
168 rdme Deprecated feature:
169 rdme
170 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 `include`d allowing it to access OpenSCAD variables for branching

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

```
gcpy #!/usr/bin/env python
gcpy #icon "C:\Program Files\OpenSCAD\bin\openscad.exe" --trust-python
gcpy #Currently tested with 2023.11.30 and Python 3.11
gcpy #gcodepreview 0.5, see gcodepreview.scad
```

```
pyscad //!OpenSCAD
pyscad
pyscad //gcodepreview 0.5, see gcodepreview.scad
```

```
gcpscad //!OpenSCAD
gcpscad
gcpscad //gcodepreview 0.5
gcpscad //
gcpscad //used via use <gcodepreview.py>;
gcpscad //          use <pygcodepreview.scad>;
gcpscad //          include <gcodepreview.scad>;
gcpscad //
```

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

```
gcpy def writeln(*arguments):
gcpy     line_to_write = ""
gcpy     for element in arguments:
gcpy         line_to_write += element
gcpy     f.write(line_to_write)
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`

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).

Note that as a convention, where it is necessary for a module to coordinate between Python and OpenSCAD, 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.

setupstock psetupstock 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
```

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.

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

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

```
35 pyscad      translate([0, (-stockwidth), -stockthickness]){
36 pyscad        cube([stocklength, stockwidth, stockthickness], center=false)
                     ;
37 pyscad if (generategcode == true) {
38 pyscad owritefive("(stockMin:0.00mm, -",str(stockwidth),"mm, -",str(
            stockthickness),"mm)");
39 pyscad owritethree("(stockMax:",str(stocklength),"mm, 0.00mm, 0.00mm)");
40 pyscad owriteeleven("(STOCK/BLOCK, ",str(stocklength),", ",str(stockwidth)
            ,", ",str(stockthickness),", 0.00, ",str(stockwidth),", ",str(
            stockthickness),")");
41 pyscad      }
42 pyscad    }
43 pyscad      }
44 pyscad      else if (stockorigin == "Center") {
45 pyscad //owritecomment("Center");
46 pyscad      translate([(-stocklength / 2), (-stockwidth / 2), -
                 stockthickness]){
47 pyscad        cube([stocklength, stockwidth, stockthickness], center=false)
                     ;
48 pyscad if (generategcode == true) {
49 pyscad owriteseven("(stockMin: -",str(stocklength/2),", -",str(stockwidth
            /2),"mm, -",str(stockthickness),"mm)");
50 pyscad owritefive("(stockMax:",str(stocklength/2),"mm, ",str(stockwidth/2)
            ,"mm, 0.00mm)");
51 pyscad owritethirteen("(STOCK/BLOCK, ",str(stocklength),", ",str(
            stockwidth),", ",str(stockthickness),", ",str(stocklength/2),",
            ", str(stockwidth/2),", ",str(stockthickness),")");
52 pyscad        }
53 pyscad      }
54 pyscad    }
55 pyscad } else if (zeroheight == "Bottom") {
56 pyscad //owritecomment("Bottom");
57 pyscad      if (stockorigin == "Lower-Left") {
58 pyscad      cube([stocklength, stockwidth, stockthickness], center=false);
59 pyscad if (generategcode == true) {
60 pyscad owriteone("(stockMin:0.00mm, 0.00mm, 0.00mm)");
61 pyscad owriteseven("(stockMax:",str(stocklength),"mm, ",str(stockwidth),"
            mm, ",str(stockthickness),"mm)");
62 pyscad owriteseven("(STOCK/BLOCK, ",str(stocklength),", ",str(stockwidth)
            ,", ",str(stockthickness),",0.00, 0.00, 0.00)");
63 pyscad      }
64 pyscad }     else if (stockorigin == "Center-Left") {
65 pyscad      translate([0, (-stockwidth / 2), 0]){
66 pyscad        cube([stocklength, stockwidth, stockthickness], center=false)
                     ;
67 pyscad if (generategcode == true) {
68 pyscad owritethree("(stockMin:0.00mm, -",str(stockwidth/2),"mm, 0.00mm)");
69 pyscad owriteseven("(stockMax:",str(stocklength),"mm, ",str(stockwidth/2)
            ,"mm, ",str(stockthickness),"mm)");
70 pyscad owritenine("(STOCK/BLOCK, ",str(stocklength),", ",str(stockwidth)
            ,", ",str(stockthickness),",0.00, ",str(stockwidth/2),", 0.00)")
            ;
71 pyscad      }
72 pyscad    }
73 pyscad      } else if (stockorigin == "Top-Left") {
74 pyscad      translate([0, (-stockwidth), 0]){
75 pyscad        cube([stocklength, stockwidth, stockthickness], center=false)
                     ;
76 pyscad      }
77 pyscad if (generategcode == true) {
78 pyscad owritethree("(stockMin:0.00mm, -",str(stockwidth),"mm, 0.00mm)");
79 pyscad owritefive("(stockMax:",str(stocklength),"mm, 0.00mm, ",str(
            stockthickness),"mm)");
80 pyscad owritenine("(STOCK/BLOCK, ",str(stocklength),", ",str(stockwidth)
            ,", ",str(stockthickness),", 0.00, ", str(stockwidth),", 0.00)")
            ;
81 pyscad    }
82 pyscad }     else if (stockorigin == "Center") {
83 pyscad      translate([(-stocklength / 2), (-stockwidth / 2), 0]){
84 pyscad        cube([stocklength, stockwidth, stockthickness], center=false)
                     ;
85 pyscad      }
86 pyscad if (generategcode == true) {
87 pyscad owritefive("(stockMin:-",str(stocklength/2),", -",str(stockwidth/2)
            ,"mm, 0.00mm)");
88 pyscad owriteseven("(stockMax:",str(stocklength/2),"mm, ",str(stockwidth
            /2),"mm, ",str(stockthickness),"mm)");
89 pyscad owriteeleven("(STOCK/BLOCK, ",str(stocklength),", ",str(stockwidth)
```

```
            ,", ",str(stockthickness),", ",str(stocklength/2),", ", str(
            stockwidth/2),", 0.00)");
 90 pyscad     }
 91 pyscad   }
 92 pyscad }
 93 pyscad if (generategcode == true) {
 94 pyscad     owriteone("G90");
 95 pyscad     owriteone("G21");
 96 pyscad //    owriteone("(Move to safe Z to avoid workholding)");
 97 pyscad //    owriteone("G53G0Z-5.000");
 98 pyscad   }
 99 pyscad //owritecomment("ENDSETUP");
100 pyscad }
```

xpos ypos zpos

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

```
 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 psetypos psetzpos psettzpos

and in turn, functions which set the positions: psetxpos, psetypos, psetzpos, and 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 setypos setzpos settzpos getxpos getypos getzpos gettzpos

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

```
102 pyscad function getxpos() = xpos();
103 pyscad function getypos() = ypos();
104 pyscad function getzpos() = zpos();
105 pyscad function gettzpos() = tzpos();
106 pyscad
107 pyscad module setxpos(newxpos) {
108 pyscad     psetxpos(newxpos);
109 pyscad }
110 pyscad
111 pyscad module setypos(newypos) {
112 pyscad     psetypos(newypos);
113 pyscad }
114 pyscad
115 pyscad module setzpos(newzpos) {
116 pyscad     psetzpos(newzpos);
117 pyscad }
118 pyscad
119 pyscad module settzpos(newtzpos) {
120 pyscad     psettzpos(newtzpos);
```

```
121 pyscad }
```

oset `oset` while for setting the variables, it is necessary to have an OpenSCAD module:

```
10 gcpscad module oset(ex, ey, ez) {
11 gcpscad     setxpos(ex);
12 gcpscad     setypos(ey);
13 gcpscad     setzpos(ez);
14 gcpscad }
```

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

```
16 gcpscad module osettz(tz) {
17 gcpscad     settzpos(tz);
18 gcpscad }
```

2.2 Tools and Changes

pcurrenttool psettool Similarly Python functions and variables will be used in: `pcurrenttool` and `psettool` to track 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 currenttool and matching OpenSCAD modules: `osettool` and `currenttool` set and return the current tool:

```
123 pyscad module osettool(tn){
124 pyscad     psettool(tn);
125 pyscad }
126 pyscad
127 pyscad function current_tool() = pcurrent_tool();
```

2.2.1 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.

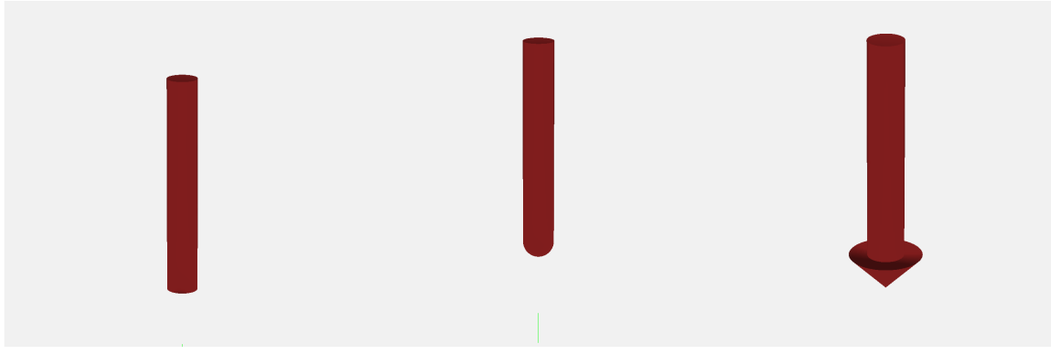
Note that the comments written out in G-code correspond to that used by the G-code previewing tool CutViewer (which is unfortunately, no longer readily available).

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

2.2.1.1 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)

```
20 gcpscad module toolchange(tool_number,speed) {
21 gcpscad    osettool(tool_number);
22 gcpscad if (generategcode == true) {
23 gcpscad     writecomment("Toolpath");
24 gcpscad     owriteone("M05");
25 gcpscad //     writecomment("Move to safe Z to avoid workholding");
26 gcpscad //     owriteone("G53G0Z-5.000");
27 gcpscad //     writecomment("Begin toolpath");
28 gcpscad     if (tool_number == 201) {
29 gcpscad       writecomment("TOOL/MILL,6.35, 0.00, 0.00, 0.00");
30 gcpscad     } else if (tool_number == 202) {
31 gcpscad       writecomment("TOOL/MILL,6.35, 3.17, 0.00, 0.00");
32 gcpscad     } else if (tool_number == 102) {
33 gcpscad       writecomment("TOOL/MILL,3.17, 0.00, 0.00, 0.00");
34 gcpscad     } else if (tool_number == 101) {
35 gcpscad       writecomment("TOOL/MILL,3.17, 1.58, 0.00, 0.00");
36 gcpscad     } else if (tool_number == 301) {
37 gcpscad       writecomment("TOOL/MILL,0.03, 0.00, 6.35, 45.00");
38 gcpscad     } else if (tool_number == 302) {
39 gcpscad       writecommment("TOOL/MILL,0.03, 0.00, 10.998, 30.00");
40 gcpscad     } else if (tool_number == 390) {
41 gcpscad       writecomment("TOOL/MILL,0.03, 0.00, 1.5875, 45.00");
```

2.2.1.2 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 tools — intended to cut slots for retaining hardware used for picture hanging, they may be used to create slots for other purposes
- Dovetail cutters — used for the joinery of the same name, they cut a large area at the bottom which slants up to a narrower region at a defined angle
- Lollipop cutters — normally used for 3D work, as their name suggests they are essentially a (cutting) ball on a narrow stick (the tool shaft), they are mentioned here only for compleatness' sake and are not (at this time) implemented

```
42 gcpscad     } else if (tool_number == 375) {
43 gcpscad       writecomment("TOOL/MILL,9.53, 0.00, 3.17, 0.00");
```

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

2.2.1.3 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.

```
46 gcpscad     }
47 gcpscad       select_tool(tool_number);
48 gcpscad       owritetwo("M6T",str(tool_number));
49 gcpscad       owritetwo("M03S",str(speed));
50 gcpscad   }
51 gcpscad }
```

2.2.1.4 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 2.2.2.2.

`selecttool tool number`

2.2.1.5 Selecting Tools There must also be a module for selecting tools: `selecttool` which will select the matching module for 3D modeling based on the `tool number`, and pass the appropriate parameters to that module:

```
53 gcpscad module select_tool(tool_number) {
54 gcpscad //echo(tool_number);
55 gcpscad   if (tool_number == 201) {
56 gcpscad     gcp_endmill_square(6.35, 19.05);
57 gcpscad   } else if (tool_number == 202) {
58 gcpscad     gcp_endmill_ball(6.35, 19.05);
59 gcpscad   } else if (tool_number == 102) {
60 gcpscad     gcp_endmill_square(3.175, 19.05);
61 gcpscad   } else if (tool_number == 101) {
62 gcpscad     gcp_endmill_ball(3.175, 19.05);
63 gcpscad   } else if (tool_number == 301) {
64 gcpscad     gcp_endmill_v(90, 12.7);
65 gcpscad   } else if (tool_number == 302) {
66 gcpscad     gcp_endmill_v(60, 12.7);
67 gcpscad   } else if (tool_number == 390) {
68 gcpscad     gcp_endmill_v(90, 3.175);
```

For a keyhole tool:

```
69 gcpscad   } else if (tool_number == 375) {
70 gcpscad     gcp_keyhole(9.525, 3.175);
```

and dovetail tool:

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

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

```
73 gcpscad   }
74 gcpscad }
```

2.2.2 3D Shapes for Tools

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

2.2.2.1 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:

```
76 gcpscad module gcp_endmill_square(es_diameter, es_flute_length) {
77 gcpscad   cylinder(r1=(es_diameter / 2), r2=(es_diameter / 2), h=
               es_flute_length, center=false);
78 gcpscad }
```

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

```
80 gcpscad module gcp_keyhole(es_diameter, es_flute_length) {
81 gcpscad   cylinder(r1=(es_diameter / 2), r2=(es_diameter / 2), h=
               es_flute_length, center=false);
82 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)

`dt angle`

```
84 gcpscad module gcp_dovetail(dt_bottomdiameter, dt_topdiameter, dt_height,
               dt_angle) {
85 gcpscad   cylinder(r1=(dt_bottomdiameter / 2), r2=(dt_topdiameter / 2), h=
               dt_height, center=false);
86 gcpscad }
```

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

```
88 gcpscad module gcp_endmill_ball(es_diameter, es_flute_length) {
89 gcpscad   translate([0, 0, (es_diameter / 2)]){
90 gcpscad     union(){
91 gcpscad       sphere(r=(es_diameter / 2));
92 gcpscad       cylinder(r1=(es_diameter / 2), r2=(es_diameter / 2), h=
                   es_flute_length, center=false);
93 gcpscad     }
94 gcpscad   }
95 gcpscad }
```

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

```
97 gcpscad module gcp_endmill_v(es_v_angle, es_diameter) {
98 gcpscad   union(){
99 gcpscad     cylinder(r1=0, r2=(es_diameter / 2), h=((es_diameter / 2) / tan
                 ((es_v_angle / 2))), center=false);
100 gcpscad    translate([0, 0, ((es_diameter / 2) / tan((es_v_angle / 2)))]){
101 gcpscad      cylinder(r1=(es_diameter / 2), r2=(es_diameter / 2), h=((
                   es_diameter * 8) ), center=false);/// tan((es_v_angle / 2)
                   )
102 gcpscad    }
103 gcpscad  }
104 gcpscad }
```

2.2.2.2 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 `hull`ed together. Something of this can be seen in the manual work-around for previewing them: https://community.carbide3d.com/t/using-unsupported-tooling-in-carbide-create-roundover-cove-radius-bits/43723.

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: `radiuscut`

`radiuscut`

```
106 gcpscad module radiuscut(bx, by, bz, ex, ey, ez, radiustn) {
107 gcpscad     if (radiustn == 56125) {
108 gcpscad         radiuscuttool(bx, by, bz, ex, ey, ez, 0.508/2, 1.531);
109 gcpscad     } else if (radiustn == 56142) {
110 gcpscad         radiuscuttool(bx, by, bz, ex, ey, ez, 0.508/2, 2.921);
111 gcpscad     } else if (radiustn == 312) {
112 gcpscad         radiuscuttool(bx, by, bz, ex, ey, ez, 1.524/2, 3.175);
113 gcpscad     } else if (radiustn == 1570) {
114 gcpscad         radiuscuttool(bx, by, bz, ex, ey, ez, 0.507/2, 4.509);
115 gcpscad     }
116 gcpscad }
```

which then calls the actual `radiuscuttool` 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`.

```
118 gcpscad module radiuscuttool(bx, by, bz, ex, ey, ez, tool_radius_tip,
               tool_radius_width) {
119 gcpscad n = 90 + $fn*3;
```

```
120 gcpscad step = 360/n;
121 gcpscad
122 gcpscad hull(){
123 gcpscad     translate([bx,by,bz])
124 gcpscad     cylinder(step,tool_radius_tip,tool_radius_tip);
125 gcpscad     translate([ex,ey,ez])
126 gcpscad     cylinder(step,tool_radius_tip,tool_radius_tip);
127 gcpscad }
128 gcpscad
129 gcpscad hull(){
130 gcpscad translate([bx,by,bz+tool_radius_width])
131 gcpscad cylinder(tool_radius_width*2,tool_radius_tip+tool_radius_width,
            tool_radius_tip+tool_radius_width);
132 gcpscad
133 gcpscad translate([ex,ey,ez+tool_radius_width])
134 gcpscad   cylinder(tool_radius_width*2,tool_radius_tip+tool_radius_width,
              tool_radius_tip+tool_radius_width);
135 gcpscad }
136 gcpscad
137 gcpscad for (i=[0:step:90]) {
138 gcpscad     angle = i;
139 gcpscad     dx = tool_radius_width*cos(angle);
140 gcpscad     dxx = tool_radius_width*cos(angle+step);
141 gcpscad     dzz = tool_radius_width*sin(angle);
142 gcpscad     dz = tool_radius_width*sin(angle+step);
143 gcpscad     dh = dz-dzz;
144 gcpscad     hull(){
145 gcpscad         translate([bx,by,bz+dz])
146 gcpscad             cylinder(dh,tool_radius_tip+tool_radius_width-dx,
                        tool_radius_tip+tool_radius_width-dxx);
147 gcpscad         translate([ex,ey,ez+dz])
148 gcpscad             cylinder(dh,tool_radius_tip+tool_radius_width-dx,
                        tool_radius_tip+tool_radius_width-dxx);
149 gcpscad         }
150 gcpscad     }
151 gcpscad }
```

2.2.3 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:

```
153 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:

```
129 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     if ptd_tool == 201:
 67 gcpy         return 6.35
 68 gcpy     if ptd_tool == 202:
 69 gcpy         if ptd_depth > 3.175:
 70 gcpy             return 6.35
 71 gcpy         else:
 72 gcpy             return 0
 73 gcpy     if ptd_tool == 102:
 74 gcpy         return 3.175
 75 gcpy     if ptd_tool == 101:
 76 gcpy         if ptd_depth > 1.5875:
 77 gcpy             return 3.175
 78 gcpy         else:
 79 gcpy             return 0
```

```
80 gcpy     if ptd_tool == 301:
81 gcpy         return 0
82 gcpy     if ptd_tool == 302:
83 gcpy         return 0
84 gcpy     if ptd_tool == 390:
85 gcpy         return 0
86 gcpy     if ptd_tool == 375:
87 gcpy         if ptd_depth < 6.35:
88 gcpy             return 9.525
89 gcpy         else:
90 gcpy             return 6.35
91 gcpy     if ptd_tool == 814:
92 gcpy         if ptd_depth > 12.7:
93 gcpy             return 6.35
94 gcpy         else:
95 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:

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

(Note that zero (0) values will need to be replaced with appropriate code.)

2.3 File Handling

`popengcodefile` `popendxffile` `popendxflgsqfile` `popendxfsmsqfile` `popendxlgblffile` `popendxfsmblfile` `popendxflgVfile` `popendxfsmVfile` For writing to files it will be necessary to have commands: `popengcodefile`, `popendxffile`, `popendxflgsqfile`, `popendxfsmsqfile`, `popendxlgblffile`, `popendxfsmblfile`, `popendxflgVfile`, and `popendxfsmVfile`. There is a separate function for each type of file, and for DXFs, there are multiple file instances, one for each combination of different type and size of tool which it is expected a project will work with. Each such file will be suffixed with the tool number.

```
 97 gcpy def popengcodefile(fn):
 98 gcpy     global f
 99 gcpy     f = open(fn, "w")
100 gcpy
101 gcpy def popendxffile(fn):
102 gcpy     global dxf
103 gcpy     dxf = open(fn, "w")
104 gcpy
105 gcpy def popendxlgblffile(fn):
106 gcpy     global dxflgbl
107 gcpy     dxflgbl = open(fn, "w")
108 gcpy
109 gcpy def popendxflgsqfile(fn):
110 gcpy     global dxfldsq
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 dxfsmbl
119 gcpy     dxfsmbl = 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 dxfsmV
127 gcpy     dxfsmV = open(fn, "w")
128 gcpy
129 gcpy def popendxfKHfile(fn):
130 gcpy     global dxfKH
131 gcpy     dxfKH = open(fn, "w")
132 gcpy
133 gcpy def popendxDTfile(fn):
134 gcpy     global dxfDT
135 gcpy     dxfDT = open(fn, "w")
```

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

```
131 pyscad module oopengcodefile(fn) {
132 pyscad     popengcodefile(fn);
133 pyscad }
134 pyscad
135 pyscad module oopendxffile(fn) {
136 pyscad     echo(fn);
137 pyscad     popendxffile(fn);
138 pyscad }
139 pyscad
140 pyscad module oopendxflgblfile(fn) {
141 pyscad     popendxflgblfile(fn);
142 pyscad }
143 pyscad
144 pyscad module oopendxflgsqfile(fn) {
145 pyscad     popendxflgsqfile(fn);
146 pyscad }
147 pyscad
148 pyscad module oopendxflgVfile(fn) {
149 pyscad     popendxflgVfile(fn);
150 pyscad }
151 pyscad
152 pyscad module oopendxfsmblfile(fn) {
153 pyscad     popendxfsmblfile(fn);
154 pyscad }
155 pyscad
156 pyscad module oopendxfsmsqfile(fn) {
157 pyscad     echo(fn);
158 pyscad     popendxfsmsqfile(fn);
159 pyscad }
160 pyscad
161 pyscad module oopendxfsmVfile(fn) {
162 pyscad     popendxfsmVfile(fn);
163 pyscad }
164 pyscad
165 pyscad module oopendxfKHfile(fn) {
166 pyscad     popendxfKHfile(fn);
167 pyscad }
168 pyscad
169 pyscad module oopendxfDTfile(fn) {
170 pyscad     popendxfDTfile(fn);
171 pyscad }
```

opengcodefile With matching OpenSCAD commands: `opengcodefile`

```
157 gcpscad module opengcodefile(fn) {
158 gcpscad if (generategcode == true) {
159 gcpscad     oopengcodefile(fn);
160 gcpscad     echo(fn);
161 gcpscad     owritecomment(fn);
162 gcpscad     }
163 gcpscad }
```

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:

```
165 gcpscad module opendxffile(fn) {
166 gcpscad   if (generatedxf == true) {
167 gcpscad       oopendxffile(str(fn,".dxf"));
168 gcpscad //     echo(fn);
169 gcpscad       dxfwriteone("0");
170 gcpscad       dxfwriteone("SECTION");
171 gcpscad       dxfwriteone("2");
172 gcpscad       dxfwriteone("ENTITIES");
173 gcpscad     if (large_ball_tool_no >  0) {    oopendxflgblfile(str(fn,".",
                    large_ball_tool_no,".dxf"));
174 gcpscad       dxfpreamble(large_ball_tool_no);
175 gcpscad     }
176 gcpscad     if (large_square_tool_no >  0) {    oopendxflgsqfile(str(fn
                    ,".",large_square_tool_no,".dxf"));
177 gcpscad       dxfpreamble(large_square_tool_no);
178 gcpscad     }
179 gcpscad     if (large_V_tool_no >  0) {    oopendxflgVfile(str(fn,".",
                    large_V_tool_no,".dxf"));
180 gcpscad       dxfpreamble(large_V_tool_no);
181 gcpscad     }
182 gcpscad     if (small_ball_tool_no >  0) { oopendxfsmblfile(str(fn,".",
```

```
                small_ball_tool_no,".dxf"));
183 gcpscad       dxfpreamble(small_ball_tool_no);
184 gcpscad     }
185 gcpscad     if (small_square_tool_no >  0) {    oopendxfsmsqfile(str(fn
                    ,".",small_square_tool_no,".dxf"));
186 gcpscad //    echo(str("tool no",small_square_tool_no));
187 gcpscad       dxfpreamble(small_square_tool_no);
188 gcpscad     }
189 gcpscad     if (small_V_tool_no >  0) {    oopendxfsmVfile(str(fn,".",
                    small_V_tool_no,".dxf"));
190 gcpscad       dxfpreamble(small_V_tool_no);
191 gcpscad     }
192 gcpscad     if (KH_tool_no >  0) {    oopendxfKHfile(str(fn,".",KH_tool_no
                    ,".dxf"));
193 gcpscad       dxfpreamble(KH_tool_no);
194 gcpscad     }
195 gcpscad     if (DT_tool_no >  0) {    oopendxfDTfile(str(fn,".",DT_tool_no
                    ,".dxf"));
196 gcpscad       dxfpreamble(DT_tool_no);
197 gcpscad     }
198 gcpscad   }
199 gcpscad }
```

2.3.1 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):
```

```
gcpy      line_to_write = ""
gcpy      for element in arguments:
gcpy          line_to_write += element
gcpy      dxfsmbl.write(line_to_write)
gcpy      print(line_to_write)
gcpy      dxfsmbl.write("\n")
gcpy
gcpy def writedxfsmsq(*arguments):
gcpy      line_to_write = ""
gcpy      for element in arguments:
gcpy          line_to_write += element
gcpy      dxfsmsq.write(line_to_write)
gcpy      print(line_to_write)
gcpy      dxfsmsq.write("\n")
gcpy
gcpy def writedxfsmV(*arguments):
gcpy      line_to_write = ""
gcpy      for element in arguments:
gcpy          line_to_write += element
gcpy      dxfsmV.write(line_to_write)
gcpy      print(line_to_write)
gcpy      dxfsmV.write("\n")
gcpy
gcpy def writedxfKH(*arguments):
gcpy      line_to_write = ""
gcpy      for element in arguments:
gcpy          line_to_write += element
gcpy      dxfKH.write(line_to_write)
gcpy      print(line_to_write)
gcpy      dxfKH.write("\n")
gcpy
gcpy def writedxfDT(*arguments):
gcpy      line_to_write = ""
gcpy      for element in arguments:
gcpy          line_to_write += element
gcpy      dxfDT.write(line_to_write)
gcpy      print(line_to_write)
gcpy      dxfDT.write("\n")
```

owritecomment
dxfwriteone
dxfwritelgbl
dxfwritelgsq
dxfwritelgV
dxfwritesmbl
dxfwritesmsq
dxfwritesmV

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

```
pyscad module owritecomment(comment) {
pyscad     writeln("(",comment,")");
pyscad }
pyscad
pyscad module dxfwriteone(first) {
pyscad     writedxf(first);
pyscad //    writeln(first);
pyscad //    echo(first);
pyscad }
pyscad
pyscad module dxfwritelgbl(first) {
pyscad     writedxflgbl(first);
pyscad }
pyscad
pyscad module dxfwritelgsq(first) {
pyscad     writedxflgsq(first);
pyscad }
pyscad
pyscad module dxfwritelgV(first) {
pyscad     writedxflgV(first);
pyscad }
pyscad
pyscad module dxfwritesmbl(first) {
pyscad     writedxfsmbl(first);
pyscad }
pyscad
pyscad module dxfwritesmsq(first) {
pyscad     writedxfsmsq(first);
pyscad }
pyscad
pyscad module dxfwritesmV(first) {
pyscad     writedxfsmV(first);
pyscad }
pyscad
```

```
207 pyscad module dxfwriteKH(first) {
208 pyscad     writedxfKH(first);
209 pyscad }
210 pyscad
211 pyscad module dxfwriteDT(first) {
212 pyscad     writedxfDT(first);
213 pyscad }
```

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...`

owrite...

```
215 pyscad module owriteone(first) {
216 pyscad     writeln(first);
217 pyscad }
218 pyscad
219 pyscad module owritetwo(first, second) {
220 pyscad     writeln(first, second);
221 pyscad }
222 pyscad
223 pyscad module owritethree(first, second, third) {
224 pyscad     writeln(first, second, third);
225 pyscad }
226 pyscad
227 pyscad module owritefour(first, second, third, fourth) {
228 pyscad     writeln(first, second, third, fourth);
229 pyscad }
230 pyscad
231 pyscad module owritefive(first, second, third, fourth, fifth) {
232 pyscad     writeln(first, second, third, fourth, fifth);
233 pyscad }
234 pyscad
235 pyscad module owritesix(first, second, third, fourth, fifth, sixth) {
236 pyscad     writeln(first, second, third, fourth, fifth, sixth);
237 pyscad }
238 pyscad
239 pyscad module owriteseven(first, second, third, fourth, fifth, sixth,
        seventh) {
240 pyscad     writeln(first, second, third, fourth, fifth, sixth, seventh);
241 pyscad }
242 pyscad
243 pyscad module owriteeight(first, second, third, fourth, fifth, sixth,
        seventh,eighth) {
244 pyscad     writeln(first, second, third, fourth, fifth, sixth, seventh,
            eighth);
245 pyscad }
246 pyscad
247 pyscad module owritenine(first, second, third, fourth, fifth, sixth,
        seventh, eighth, ninth) {
248 pyscad     writeln(first, second, third, fourth, fifth, sixth, seventh,
            eighth, ninth);
249 pyscad }
250 pyscad
251 pyscad module owriteten(first, second, third, fourth, fifth, sixth,
        seventh, eighth, ninth, tenth) {
252 pyscad     writeln(first, second, third, fourth, fifth, sixth, seventh,
            eighth, ninth, tenth);
253 pyscad }
254 pyscad
255 pyscad module owriteeleven(first, second, third, fourth, fifth, sixth,
        seventh, eighth, ninth, tenth, eleventh) {
256 pyscad     writeln(first, second, third, fourth, fifth, sixth, seventh,
            eighth, ninth, tenth, eleventh);
257 pyscad }
258 pyscad
259 pyscad module owritetwelve(first, second, third, fourth, fifth, sixth,
        seventh, eighth, ninth, tenth, eleventh, twelfth) {
260 pyscad     writeln(first, second, third, fourth, fifth, sixth, seventh,
            eighth, ninth, tenth, eleventh, twelfth);
261 pyscad }
262 pyscad
263 pyscad module owritethirteen(first, second, third, fourth, fifth, sixth,
        seventh, eighth, ninth, tenth, eleventh, twelfth, thirteenth) {
264 pyscad     writeln(first, second, third, fourth, fifth, sixth, seventh,
            eighth, ninth, tenth, eleventh, twelfth, thirteenth);
265 pyscad }
```

dxfwrite dxfpreamble **2.3.1.1 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.

```
201 gcpscad module dxfwrite(tn,arg) {
202 gcpscad if (tn == large_ball_tool_no) {
203 gcpscad     dxfwritelgbl(arg);}
204 gcpscad if (tn == large_square_tool_no) {
205 gcpscad     dxfwritelgsq(arg);}
206 gcpscad if (tn == large_V_tool_no) {
207 gcpscad     dxfwritelgV(arg);}
208 gcpscad if (tn == small_ball_tool_no) {
209 gcpscad     dxfwritesmbl(arg);}
210 gcpscad if (tn == small_square_tool_no) {
211 gcpscad     dxfwritesmsq(arg);}
212 gcpscad if (tn == small_V_tool_no) {
213 gcpscad     dxfwritesmV(arg);}
214 gcpscad if (tn == DT_tool_no) {
215 gcpscad     dxfwriteDT(arg);}
216 gcpscad if (tn == KH_tool_no) {
217 gcpscad     dxfwriteKH(arg);}
218 gcpscad }
219 gcpscad
220 gcpscad module dxfpreamble(tn) {
221 gcpscad //     echo(str("dxfpreamble",small_square_tool_no));
222 gcpscad     dxfwrite(tn,"0");
223 gcpscad     dxfwrite(tn,"SECTION");
224 gcpscad     dxfwrite(tn,"2");
225 gcpscad     dxfwrite(tn,"ENTITIES");
226 gcpscad }
```

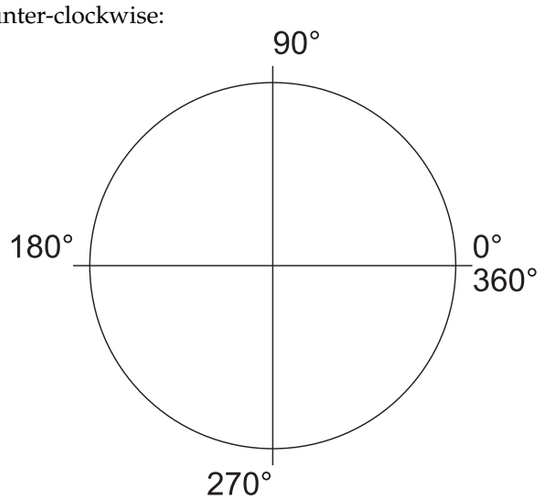
2.3.1.2 DXF Lines and Arcs Similarly, each each element which may be written to a DXF file will have a user module as well as an internal module which will be called by it so as to write to the file for the current tool.

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

dxfbpl
- a line: LWPOLYLINE is one possible implementation: dxfbpl

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

DXF orders arcs counter-clockwise:



Note that arcs of greater than 90 degrees are not rendered accurately, so, for the sake of precision, they should be limited to a swing of 90 degrees or less. Further note that 4 arcs may be stitched together to make a circle:

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

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

There are three possible interactions for DXF elements and toolpaths:

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

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

```
228 gcpscad module dxfpl(tn,xbegin,ybegin,xend,yend) {
229 gcpscad     dxfwrite(tn,"0");
230 gcpscad     dxfwrite(tn,"LWPOLYLINE");
231 gcpscad     dxfwrite(tn,"90");
232 gcpscad     dxfwrite(tn,"2");
233 gcpscad     dxfwrite(tn,"70");
234 gcpscad     dxfwrite(tn,"0");
235 gcpscad     dxfwrite(tn,"43");
236 gcpscad     dxfwrite(tn,"0");
237 gcpscad     dxfwrite(tn,"10");
238 gcpscad     dxfwrite(tn,str(xbegin));
239 gcpscad     dxfwrite(tn,"20");
240 gcpscad     dxfwrite(tn,str(ybegin));
241 gcpscad     dxfwrite(tn,"10");
242 gcpscad     dxfwrite(tn,str(xend));
243 gcpscad     dxfwrite(tn,"20");
244 gcpscad     dxfwrite(tn,str(yend));
245 gcpscad }
246 gcpscad
247 gcpscad module dxfpolyline(tn,xbegin,ybegin,xend,yend) {
248 gcpscad if (generatedxf == true) {
249 gcpscad     dxfwriteone("0");
250 gcpscad     dxfwriteone("LWPOLYLINE");
251 gcpscad     dxfwriteone("90");
252 gcpscad     dxfwriteone("2");
253 gcpscad     dxfwriteone("70");
254 gcpscad     dxfwriteone("0");
255 gcpscad     dxfwriteone("43");
256 gcpscad     dxfwriteone("0");
257 gcpscad     dxfwriteone("10");
258 gcpscad     dxfwriteone(str(xbegin));
259 gcpscad     dxfwriteone("20");
260 gcpscad     dxfwriteone(str(ybegin));
261 gcpscad     dxfwriteone("10");
262 gcpscad     dxfwriteone(str(xend));
263 gcpscad     dxfwriteone("20");
264 gcpscad     dxfwriteone(str(yend));
265 gcpscad     dxfpl(tn,xbegin,ybegin,xend,yend);
266 gcpscad     }
267 gcpscad }
```

`dxfa` `dxfarc` As for other files, we have two versions, `dxfa` and `dxfarc`, 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.

```
269 gcpscad module dxfa(tn,xcenter,ycenter,radius,anglebegin,endangle) {
270 gcpscad     dxfwrite(tn,"0");
271 gcpscad     dxfwrite(tn,"ARC");
272 gcpscad     dxfwrite(tn,"10");
273 gcpscad     dxfwrite(tn,str(xcenter));
274 gcpscad     dxfwrite(tn,"20");
275 gcpscad     dxfwrite(tn,str(ycenter));
276 gcpscad     dxfwrite(tn,"40");
277 gcpscad     dxfwrite(tn,str(radius));
278 gcpscad     dxfwrite(tn,"50");
279 gcpscad     dxfwrite(tn,str(anglebegin));
280 gcpscad     dxfwrite(tn,"51");
281 gcpscad     dxfwrite(tn,str(endangle));
282 gcpscad }
283 gcpscad
284 gcpscad module dxfarc(tn,xcenter,ycenter,radius,anglebegin,endangle) {
285 gcpscad if (generatedxf == true) {
286 gcpscad     dxfwriteone("0");
287 gcpscad     dxfwriteone("ARC");
288 gcpscad     dxfwriteone("10");
289 gcpscad     dxfwriteone(str(xcenter));
290 gcpscad     dxfwriteone("20");
291 gcpscad     dxfwriteone(str(ycenter));
292 gcpscad     dxfwriteone("40");
293 gcpscad     dxfwriteone(str(radius));
294 gcpscad     dxfwriteone("50");
295 gcpscad     dxfwriteone(str(anglebegin));
296 gcpscad     dxfwriteone("51");
297 gcpscad     dxfwriteone(str(endangle));
298 gcpscad     dxfa(tn,xcenter,ycenter,radius,anglebegin,endangle);
```

```
299 gcpscad     }
300 gcpscad }
```

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

```
302 gcpscad module dxfbpl(tn,bx,by) {
303 gcpscad     dxfwrite(tn,"0");
304 gcpscad     dxfwrite(tn,"POLYLINE");
305 gcpscad     dxfwrite(tn,"8");
306 gcpscad     dxfwrite(tn,"default");
307 gcpscad     dxfwrite(tn,"66");
308 gcpscad     dxfwrite(tn,"1");
309 gcpscad     dxfwrite(tn,"70");
310 gcpscad     dxfwrite(tn,"0");
311 gcpscad     dxfwrite(tn,"0");
312 gcpscad     dxfwrite(tn,"VERTEX");
313 gcpscad     dxfwrite(tn,"8");
314 gcpscad     dxfwrite(tn,"default");
315 gcpscad     dxfwrite(tn,"70");
316 gcpscad     dxfwrite(tn,"32");
317 gcpscad     dxfwrite(tn,"10");
318 gcpscad     dxfwrite(tn,str(bx));
319 gcpscad     dxfwrite(tn,"20");
320 gcpscad     dxfwrite(tn,str(by));
321 gcpscad }
322 gcpscad
323 gcpscad module beginpolyline(bx,by,bz) {
324 gcpscad if (generatedxf == true) {
325 gcpscad     dxfwriteone("0");
326 gcpscad     dxfwriteone("POLYLINE");
327 gcpscad     dxfwriteone("8");
328 gcpscad     dxfwriteone("default");
329 gcpscad     dxfwriteone("66");
330 gcpscad     dxfwriteone("1");
331 gcpscad     dxfwriteone("70");
332 gcpscad     dxfwriteone("0");
333 gcpscad     dxfwriteone("0");
334 gcpscad     dxfwriteone("VERTEX");
335 gcpscad     dxfwriteone("8");
336 gcpscad     dxfwriteone("default");
337 gcpscad     dxfwriteone("70");
338 gcpscad     dxfwriteone("32");
339 gcpscad     dxfwriteone("10");
340 gcpscad     dxfwriteone(str(bx));
341 gcpscad     dxfwriteone("20");
342 gcpscad     dxfwriteone(str(by));
343 gcpscad     dxfbpl(current_tool(),bx,by);}
344 gcpscad }
345 gcpscad
346 gcpscad module dxfapl(tn,bx,by) {
347 gcpscad     dxfwriteone("0");
348 gcpscad     dxfwrite(tn,"VERTEX");
349 gcpscad     dxfwrite(tn,"8");
350 gcpscad     dxfwrite(tn,"default");
351 gcpscad     dxfwrite(tn,"70");
352 gcpscad     dxfwrite(tn,"32");
353 gcpscad     dxfwrite(tn,"10");
354 gcpscad     dxfwrite(tn,str(bx));
355 gcpscad     dxfwrite(tn,"20");
356 gcpscad     dxfwrite(tn,str(by));
357 gcpscad }
358 gcpscad
359 gcpscad module addpolyline(bx,by,bz) {
360 gcpscad if (generatedxf == true) {
361 gcpscad     dxfwrite(tn,"0");
362 gcpscad     dxfwriteone("VERTEX");
363 gcpscad     dxfwriteone("8");
364 gcpscad     dxfwriteone("default");
365 gcpscad     dxfwriteone("70");
366 gcpscad     dxfwriteone("32");
367 gcpscad     dxfwriteone("10");
368 gcpscad     dxfwriteone(str(bx));
369 gcpscad     dxfwriteone("20");
370 gcpscad     dxfwriteone(str(by));
371 gcpscad     dxfapl(current_tool(),bx,by);
372 gcpscad     }
373 gcpscad }
374 gcpscad
```

```
375 gcpscad module dxfcpl(tn) {
376 gcpscad     dxfwrite(tn,"0");
377 gcpscad     dxfwrite(tn,"SEQEND");
378 gcpscad }
379 gcpscad
380 gcpscad module closepolyline() {
381 gcpscad   if (generatedxf == true) {
382 gcpscad     dxfwriteone("0");
383 gcpscad     dxfwriteone("SEQEND");
384 gcpscad     dxfcpl(current_tool());
385 gcpscad   }
386 gcpscad }
387 gcpscad
388 gcpscad module writecomment(comment) {
389 gcpscad   if (generategcode == true) {
390 gcpscad     owritecomment(comment);
391 gcpscad   }
392 gcpscad }
```

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

`closedxffile`

```
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     dxfsmbl.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()
```

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

`oclosegcodefile` `oclosedxffile`

```
267 pyscad module oclosegcodefile() {
268 pyscad     pclosegcodefile();
269 pyscad }
270 pyscad
271 pyscad module oclosedxffile() {
272 pyscad     pclosedxffile();
273 pyscad }
274 pyscad
275 pyscad module oclosedxflgblfile() {
276 pyscad     pclosedxflgblfile();
277 pyscad }
278 pyscad
279 pyscad module oclosedxflgsqfile() {
280 pyscad     pclosedxflgsqfile();
281 pyscad }
282 pyscad
283 pyscad module oclosedxflgVfile() {
284 pyscad     pclosedxflgVfile();
285 pyscad }
286 pyscad
287 pyscad module oclosedxfsmblfile() {
```

```
288 pyscad      pclosedxfsmblfile();
289 pyscad }
290 pyscad
291 pyscad module oclosedxfsmsqfile() {
292 pyscad      pclosedxfsmsqfile();
293 pyscad }
294 pyscad
295 pyscad module oclosedxfsmVfile() {
296 pyscad      pclosedxfsmVfile();
297 pyscad }
298 pyscad
299 pyscad module oclosedxfDTfile() {
300 pyscad      pclosedxfDTfile();
301 pyscad }
302 pyscad
303 pyscad module oclosedxfKHfile() {
304 pyscad      pclosedxfKHfile();
305 pyscad }
```

closegcodefile
closedxffile
dxfpostamble

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

```
394 gcpscad module closegcodefile() {
395 gcpscad   if (generategcode == true) {
396 gcpscad     owriteone("M05");
397 gcpscad     owriteone("M02");
398 gcpscad     oclosegcodefile();
399 gcpscad   }
400 gcpscad }
401 gcpscad
402 gcpscad module dxfpostamble(arg) {
403 gcpscad     dxfwrite(arg,"0");
404 gcpscad     dxfwrite(arg,"ENDSEC");
405 gcpscad     dxfwrite(arg,"0");
406 gcpscad     dxfwrite(arg,"EOF");
407 gcpscad }
408 gcpscad
409 gcpscad module closedxffile() {
410 gcpscad   if (generatedxf == true) {
411 gcpscad     dxfwriteone("0");
412 gcpscad     dxfwriteone("ENDSEC");
413 gcpscad     dxfwriteone("0");
414 gcpscad     dxfwriteone("EOF");
415 gcpscad     oclosedxffile();
416 gcpscad     echo("CLOSING");
417 gcpscad     if (large_ball_tool_no >  0) {    dxfpostamble(
                    large_ball_tool_no);
418 gcpscad       oclosedxflgblfile();
419 gcpscad     }
420 gcpscad     if (large_square_tool_no >  0) {    dxfpostamble(
                    large_square_tool_no);
421 gcpscad       oclosedxflgsqfile();
422 gcpscad     }
423 gcpscad     if (large_V_tool_no >  0) {    dxfpostamble(large_V_tool_no);
424 gcpscad       oclosedxflgVfile();
425 gcpscad     }
426 gcpscad     if (small_ball_tool_no >  0) {    dxfpostamble(
                    small_ball_tool_no);
427 gcpscad       oclosedxfsmblfile();
428 gcpscad     }
429 gcpscad     if (small_square_tool_no >  0) {    dxfpostamble(
                    small_square_tool_no);
430 gcpscad       oclosedxfsmsqfile();
431 gcpscad     }
432 gcpscad     if (small_V_tool_no >  0) {    dxfpostamble(small_V_tool_no);
433 gcpscad       oclosedxfsmVfile();
434 gcpscad     }
435 gcpscad     if (DT_tool_no >  0) {    dxfpostamble(DT_tool_no);
436 gcpscad       oclosedxfDTfile();
437 gcpscad     }
438 gcpscad     if (KH_tool_no >  0) {    dxfpostamble(KH_tool_no);
439 gcpscad       oclosedxfKHfile();
440 gcpscad     }
441 gcpscad   }
442 gcpscad }
```

2.4 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: orapidwhich will also need to write out files which represent the desired machine motions.

```
444 gcpscad module otm(ex, ey, ez, r,g,b) {
445 gcpscad color([r,g,b]) hull(){
446 gcpscad     translate([xpos(), ypos(), zpos()]){
447 gcpscad       select_tool(current_tool());
448 gcpscad     }
449 gcpscad     translate([ex, ey, ez]){
450 gcpscad       select_tool(current_tool());
451 gcpscad     }
452 gcpscad   }
453 gcpscad oset(ex, ey, ez);
454 gcpscad }
455 gcpscad
456 gcpscad module ocut(ex, ey, ez) {
457 gcpscad   //color([0.2,1,0.2]) hull(){
458 gcpscad   otm(ex, ey, ez, 0.2,1,0.2);
459 gcpscad }
460 gcpscad
461 gcpscad module orapid(ex, ey, ez) {
462 gcpscad   //color([0.93,0,0]) hull(){
463 gcpscad   otm(ex, ey, ez, 0.93,0,0);
464 gcpscad }
465 gcpscad
466 gcpscad module rapidbx(bx, by, bz, ex, ey, ez) {
467 gcpscad   //    writeln("G0 X",bx," Y", by, "Z", bz);
468 gcpscad   if (generategcode == true) {
469 gcpscad     writecomment("rapid");
470 gcpscad     owritesix("G0 X",str(ex)," Y", str(ey), " Z", str(ez));
471 gcpscad   }
472 gcpscad     orapid(ex, ey, ez);
473 gcpscad }
474 gcpscad
475 gcpscad module rapid(ex, ey, ez) {
476 gcpscad   //    writeln("G0 X",bx," Y", by, "Z", bz);
477 gcpscad   if (generategcode == true) {
478 gcpscad       writecomment("rapid");
479 gcpscad       owritesix("G0 X",str(ex)," Y", str(ey), " Z", str(ez));
480 gcpscad   }
481 gcpscad   orapid(ex, ey, ez);
482 gcpscad }
483 gcpscad
484 gcpscad module movetosafez() {
485 gcpscad   //this should be move to retract height
486 gcpscad   if (generategcode == true) {
487 gcpscad       writecomment("Move to safe Z to avoid workholding");
488 gcpscad       owriteone("G53G0Z-5.000");
489 gcpscad   }
490 gcpscad   orapid(getxpos(), getypos(), retractheight+55);
491 gcpscad }
492 gcpscad
493 gcpscad module begintoolpath(bx,by,bz) {
494 gcpscad   if (generategcode == true) {
495 gcpscad     writecomment("PREPOSITION FOR RAPID PLUNGE");
496 gcpscad     owritefour("G0X", str(bx), "Y",str(by));
497 gcpscad     owritetwo("Z", str(bz));
498 gcpscad   }
499 gcpscad   orapid(bx,by,bz);
500 gcpscad }
501 gcpscad
502 gcpscad module movetosafeheight() {
503 gcpscad   //this should be move to machine position
504 gcpscad   if (generategcode == true) {
505 gcpscad   //    writecomment("PREPOSITION FOR RAPID PLUNGE");Z25.650
506 gcpscad   //G1Z24.663F381.0 ,"F",str(plunge)
507 gcpscad     if (zeroheight == "Top") {
508 gcpscad       owritetwo("Z",str(retractheight));
509 gcpscad     }
510 gcpscad   }
511 gcpscad     orapid(getxpos(), getypos(), retractheight+55);
512 gcpscad }
513 gcpscad
514 gcpscad module cutoneaxis_setfeed(axis,depth,feed) {
```

```
515 gcpscad   if (generategcode == true) {
516 gcpscad   //    writecomment("PREPOSITION FOR RAPID PLUNGE");Z25.650
517 gcpscad   //G1Z24.663F381.0 ,"F",str(plunge) G1Z7.612F381.0
518 gcpscad     if (zeroheight == "Top") {
519 gcpscad       owritefive("G1",axis,str(depth),"F",str(feed));
520 gcpscad     }
521 gcpscad   }
522 gcpscad   if (axis == "X") {setxpos(depth);
523 gcpscad     ocut(depth, getypos(), getzpos());}
524 gcpscad     if (axis == "Y") {setypos(depth);
525 gcpscad       ocut(getxpos(), depth, getzpos());
526 gcpscad     }
527 gcpscad       if (axis == "Z") {setzpos(depth);
528 gcpscad         ocut(getxpos(), getypos(), depth);
529 gcpscad       }
530 gcpscad }
531 gcpscad
532 gcpscad module cut(ex, ey, ez) {
533 gcpscad   //    writeln("G0 X",bx," Y", by, "Z", bz);
534 gcpscad   if (generategcode == true) {
535 gcpscad      owritesix("G1 X",str(ex)," Y", str(ey), " Z", str(ez));
536 gcpscad   }
537 gcpscad   //if (generatesvg == true) {
538 gcpscad   //    owritesix("G1 X",str(ex)," Y", str(ey), " Z", str(ez));
539 gcpscad   //    orapid(getxpos(), getypos(), retractheight+5);
540 gcpscad   //    writesvgline(getxpos(),getypos(),ex,ey);
541 gcpscad   //}
542 gcpscad   ocut(ex, ey, ez);
543 gcpscad }
544 gcpscad
545 gcpscad module cutwithfeed(ex, ey, ez, feed) {
546 gcpscad   //    writeln("G0 X",bx," Y", by, "Z", bz);
547 gcpscad   if (generategcode == true) {
548 gcpscad   //    writecomment("rapid");
549 gcpscad     owriteeight("G1 X",str(ex)," Y", str(ey), " Z", str(ez),"F",str
                  (feed));
550 gcpscad   }
551 gcpscad   ocut(ex, ey, ez);
552 gcpscad }
553 gcpscad
554 gcpscad module endtoolpath() {
555 gcpscad   if (generategcode == true) {
556 gcpscad   //Z31.750
557 gcpscad   //    owriteone("G53G0Z-5.000");
558 gcpscad     owritetwo("Z",str(retractheight));
559 gcpscad   }
560 gcpscad   orapid(getxpos(),getypos(),retractheight);
561 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 consolidated in 2.2.1) 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
- (Rectangular) Pocket — such toolpaths/geometry should include the rounding of the tool at the corners

- 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 sector[1]
  - two convex, one concave arc
- 4
  - rectangle (including square)
  - 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?

- 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 dove-tail 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
- 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(tn,xcenter,ycenter,radius,anglebegin,endangle)`
- 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 a loop: arcloop (clockwise), narcloop (counterclockwise) to handle the drawing and processing 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):

arcloop
narcloop

```
563 gcpscad //!OpenSCAD
564 gcpscad
565 gcpscad module arcloop(barc,earc, xcenter, ycenter, radius) {
566 gcpscad   for (i = [barc : abs(1) : earc]) {
567 gcpscad         cut(xcenter + radius * cos(i),
568 gcpscad         ycenter + radius * sin(i),
569 gcpscad         getzpos()-(gettzpos())
570 gcpscad         );
571 gcpscad     setxpos(xcenter + radius * cos(i));
572 gcpscad     setypos(ycenter + radius * sin(i));
573 gcpscad   }
574 gcpscad }
575 gcpscad
576 gcpscad module narcloop(barc,earc, xcenter, ycenter, radius) {
577 gcpscad   for (i = [barc : -1 : earc]) {
578 gcpscad         cut(xcenter + radius * cos(i),
579 gcpscad         ycenter + radius * sin(i),
580 gcpscad         getzpos()-(gettzpos())
581 gcpscad         );
582 gcpscad     setxpos(xcenter + radius * cos(i));
583 gcpscad     setypos(ycenter + radius * sin(i));
584 gcpscad   }
585 gcpscad }
```

The various textual versions are quite obvious:

```
588 gcpscad module cutarcNECCdxf(tn, ex, ey, ez, xcenter, ycenter, radius) {
589 gcpscad   dxfarc(tn,xcenter,ycenter,radius,0,90);
590 gcpscad   settzpos((getzpos()-ez)/90);
591 gcpscad     arcloop(1,90, xcenter, ycenter, radius);
592 gcpscad }
593 gcpscad
594 gcpscad module cutarcNWCCdxf(tn, ex, ey, ez, xcenter, ycenter, radius) {
595 gcpscad   dxfarc(tn,xcenter,ycenter,radius,90,180);
596 gcpscad   settzpos((getzpos()-ez)/90);
597 gcpscad     arcloop(91,180, xcenter, ycenter, radius);
598 gcpscad }
599 gcpscad
600 gcpscad module cutarcSWCCdxf(tn, ex, ey, ez, xcenter, ycenter, radius) {
601 gcpscad   dxfarc(tn,xcenter,ycenter,radius,180,270);
602 gcpscad   settzpos((getzpos()-ez)/90);
603 gcpscad     arcloop(181,270, xcenter, ycenter, radius);
604 gcpscad }
605 gcpscad
606 gcpscad module cutarcSECCdxf(tn, ex, ey, ez, xcenter, ycenter, radius) {
607 gcpscad   dxfarc(tn,xcenter,ycenter,radius,270,360);
608 gcpscad   settzpos((getzpos()-ez)/90);
609 gcpscad     arcloop(271,360, xcenter, ycenter, radius);
610 gcpscad }
611 gcpscad
612 gcpscad module cutarcNECWdxf(tn, ex, ey, ez, xcenter, ycenter, radius) {
613 gcpscad   dxfarc(tn,xcenter,ycenter,radius,0,90);
614 gcpscad   settzpos((getzpos()-ez)/90);
615 gcpscad     narcloop(89,0, xcenter, ycenter, radius);
616 gcpscad }
```

```
617 gcpscad
618 gcpscad module cutarcSECWdxf(tn, ex, ey, ez, xcenter, ycenter, radius) {
619 gcpscad   dxfarc(tn,xcenter,ycenter,radius,270,360);
620 gcpscad   settzpos((getzpos()-ez)/90);
621 gcpscad     narcloop(359,270, xcenter, ycenter, radius);
622 gcpscad }
623 gcpscad
624 gcpscad module cutarcSWCWdxf(tn, ex, ey, ez, xcenter, ycenter, radius) {
625 gcpscad   dxfarc(tn,xcenter,ycenter,radius,180,270);
626 gcpscad   settzpos((getzpos()-ez)/90);
627 gcpscad     narcloop(269,180, xcenter, ycenter, radius);
628 gcpscad }
629 gcpscad
630 gcpscad module cutarcNWCWdxf(tn, ex, ey, ez, xcenter, ycenter, radius) {
631 gcpscad   dxfarc(tn,xcenter,ycenter,radius,90,180);
632 gcpscad   settzpos((getzpos()-ez)/90);
633 gcpscad     narcloop(179,90, xcenter, ycenter, radius);
634 gcpscad }
```

3.2 Keyhole toolpath and undercut tooling

keyhole toolpath The first topologically unusual toolpath is `keyhole 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 2.2.1.2

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:

```
628 gcpscad module keyhole_toolpath(kh_tool_no, kh_start_depth, kh_max_depth,
            kht_angle, kh_length) {
629 gcpscad if (kht_angle == "N") {
630 gcpscad   keyhole_toolpath_degrees(kh_tool_no, kh_start_depth, kh_max_depth
              , 90, kh_length);
631 gcpscad     } else if (kht_angle == "S") {
632 gcpscad   keyhole_toolpath_degrees(kh_tool_no, kh_start_depth, kh_max_depth
              , 270, kh_length);
633 gcpscad     } else if (kht_angle == "E") {
634 gcpscad   keyhole_toolpath_degrees(kh_tool_no, kh_start_depth, kh_max_depth
              , 0, kh_length);
635 gcpscad     } else if (kht_angle == "W") {
636 gcpscad   keyhole_toolpath_degrees(kh_tool_no, kh_start_depth, kh_max_depth
              , 180, kh_length);
637 gcpscad     }
638 gcpscad }
```

eyhole toolpath degrees The original version of the command, `keyhole 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:

```
640 gcpscad module keyhole_toolpath_degrees(kh_tool_no, kh_start_depth,
            kh_max_depth, kh_angle, kh_length) {
641 gcpscad dxfarc(KH_tool_no,getxpos(),getypos(),tool_diameter(KH_tool_no, (
            kh_max_depth+4.36))/2,0,90);
642 gcpscad dxfarc(KH_tool_no,getxpos(),getypos(),tool_diameter(KH_tool_no, (
            kh_max_depth+4.36))/2,90,180);
643 gcpscad dxfarc(KH_tool_no,getxpos(),getypos(),tool_diameter(KH_tool_no, (
            kh_max_depth+4.36))/2,180,270);
644 gcpscad dxfarc(KH_tool_no,getxpos(),getypos(),tool_diameter(KH_tool_no, (
            kh_max_depth+4.36))/2,270,360);
```

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

```
646 gcpscad   if (kh_angle == 0) {
647 gcpscad dxfarc(KH_tool_no,getxpos(),getypos(),tool_diameter(KH_tool_no, (
            kh_max_depth))/2,180,270);
648 gcpscad dxfarc(KH_tool_no,getxpos(),getypos(),tool_diameter(KH_tool_no, (
            kh_max_depth))/2,90,180);
649 gcpscad dxfarc(KH_tool_no,getxpos(),getypos(),tool_diameter(KH_tool_no, (
            kh_max_depth))/2,asin((tool_diameter(KH_tool_no, (kh_max_depth
            +4.36))/2)/(tool_diameter(KH_tool_no, (kh_max_depth))/2)),90);
650 gcpscad dxfarc(KH_tool_no,getxpos(),getypos(),tool_diameter(KH_tool_no, (
            kh_max_depth))/2,270,360-asin((tool_diameter(KH_tool_no, (
            kh_max_depth+4.36))/2)/(tool_diameter(KH_tool_no, (kh_max_depth)
            )/2)));
651 gcpscad dxfarc(KH_tool_no,getxpos()+kh_length,getypos(),tool_diameter(
            KH_tool_no, (kh_max_depth+4.36))/2,0,90);
652 gcpscad dxfarc(KH_tool_no,getxpos()+kh_length,getypos(),tool_diameter(
            KH_tool_no, (kh_max_depth+4.36))/2,270,360);
653 gcpscad dxfpolyline(KH_tool_no, getxpos()+sqrt((tool_diameter(KH_tool_no, (
            kh_max_depth))/2)^2-(tool_diameter(KH_tool_no, (kh_max_depth
            +4.36))/2)^2), getypos()+tool_diameter(KH_tool_no, (kh_max_depth
            +4.36))/2, getxpos()+kh_length, getypos()+tool_diameter(
            KH_tool_no, (kh_max_depth+4.36))/2);
654 gcpscad dxfpolyline(KH_tool_no, getxpos()+sqrt((tool_diameter(KH_tool_no, (
            kh_max_depth))/2)^2-(tool_diameter(KH_tool_no, (kh_max_depth
            +4.36))/2)^2), getypos()-tool_diameter(KH_tool_no, (kh_max_depth
            +4.36))/2, getxpos()+kh_length, getypos()-tool_diameter(
            KH_tool_no, (kh_max_depth+4.36))/2);
655 gcpscad dxfpolyline(KH_tool_no,getxpos(),getypos(),getxpos()+kh_length,
            getypos());
656 gcpscad cutwithfeed(getxpos()+kh_length,getypos(),-kh_max_depth,feed);
657 gcpscad setxpos(getxpos()-kh_length);
658 gcpscad   } else if (kh_angle > 0 && kh_angle < 90) {
659 gcpscad echo(kh_angle);
660 gcpscad   dxfarc(KH_tool_no,getxpos(),getypos(),tool_diameter(KH_tool_no, (
              kh_max_depth))/2,90+kh_angle,180+kh_angle);
661 gcpscad   dxfarc(KH_tool_no,getxpos(),getypos(),tool_diameter(KH_tool_no, (
              kh_max_depth))/2,180+kh_angle,270+kh_angle);
662 gcpscad dxfarc(KH_tool_no,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);
663 gcpscad dxfarc(KH_tool_no,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)));
664 gcpscad dxfarc(KH_tool_no,
665 gcpscad   getxpos()+(kh_length*cos(kh_angle)),
666 gcpscad   getypos()+(kh_length*sin(kh_angle)),tool_diameter(KH_tool_no, (
              kh_max_depth+4.36))/2,0+kh_angle,90+kh_angle);
667 gcpscad dxfarc(KH_tool_no,getxpos()+(kh_length*cos(kh_angle)),getypos()+(
            kh_length*sin(kh_angle)),tool_diameter(KH_tool_no, (kh_max_depth
            +4.36))/2,270+kh_angle,360+kh_angle);
668 gcpscad dxfpolyline(KH_tool_no,
669 gcpscad  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))),
670 gcpscad  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))),
671 gcpscad  getxpos()+(kh_length*cos(kh_angle))-((tool_diameter(KH_tool_no, (
              kh_max_depth+4.36))/2)*sin(kh_angle)),
672 gcpscad  getypos()+(kh_length*sin(kh_angle))+((tool_diameter(KH_tool_no, (
              kh_max_depth+4.36))/2)*cos(kh_angle)));
673 gcpscad echo("a",tool_diameter(KH_tool_no, (kh_max_depth+4.36))/2);
674 gcpscad echo("c",tool_diameter(KH_tool_no, (kh_max_depth))/2);
675 gcpscad echo("Aangle",asin((tool_diameter(KH_tool_no, (kh_max_depth+4.36))
            /2)/(tool_diameter(KH_tool_no, (kh_max_depth))/2)));
676 gcpscad echo(kh_angle);
677 gcpscad  cutwithfeed(getxpos()+(kh_length*cos(kh_angle)),getypos()+(
             kh_length*sin(kh_angle)),-kh_max_depth,feed);
678 gcpscad  setxpos(getxpos()-(kh_length*cos(kh_angle)));
679 gcpscad  setypos(getypos()-(kh_length*sin(kh_angle)));
680 gcpscad   } else if (kh_angle == 90) {
681 gcpscad dxfarc(KH_tool_no,getxpos(),getypos(),tool_diameter(KH_tool_no, (
            kh_max_depth))/2,180,270);
682 gcpscad dxfarc(KH_tool_no,getxpos(),getypos(),tool_diameter(KH_tool_no, (
            kh_max_depth))/2,270,360);
```

```
683 gcpscad dxfarc(KH_tool_no,getxpos(),getypos(),tool_diameter(KH_tool_no, (
            kh_max_depth))/2,0,90-asin(
684 gcpscad     (tool_diameter(KH_tool_no, (kh_max_depth+4.36))/2)/(
                tool_diameter(KH_tool_no, (kh_max_depth))/2)));
685 gcpscad dxfarc(KH_tool_no,getxpos(),getypos(),tool_diameter(KH_tool_no, (
            kh_max_depth))/2,90+asin(
686 gcpscad     (tool_diameter(KH_tool_no, (kh_max_depth+4.36))/2)/(
                tool_diameter(KH_tool_no, (kh_max_depth))/2)),180);
687 gcpscad  dxfpolyline(KH_tool_no,getxpos(),getypos(),getxpos(),getypos()+
             kh_length);
688 gcpscad dxfarc(KH_tool_no,getxpos(),getypos()+kh_length,tool_diameter(
            KH_tool_no, (kh_max_depth+4.36))/2,0,90);
689 gcpscad dxfarc(KH_tool_no,getxpos(),getypos()+kh_length,tool_diameter(
            KH_tool_no, (kh_max_depth+4.36))/2,90,180);
690 gcpscad  dxfpolyline(KH_tool_no,getxpos()+tool_diameter(KH_tool_no, (
             kh_max_depth+4.36))/2,getypos()+sqrt((tool_diameter(KH_tool_no,
              (kh_max_depth))/2)^2-(tool_diameter(KH_tool_no, (kh_max_depth
             +4.36))/2)^2),getxpos()+tool_diameter(KH_tool_no, (kh_max_depth
             +4.36))/2,getypos()+kh_length);
691 gcpscad  dxfpolyline(KH_tool_no,getxpos()-tool_diameter(KH_tool_no, (
             kh_max_depth+4.36))/2,getypos()+sqrt((tool_diameter(KH_tool_no,
              (kh_max_depth))/2)^2-(tool_diameter(KH_tool_no, (kh_max_depth
             +4.36))/2)^2),getxpos()-tool_diameter(KH_tool_no, (kh_max_depth
             +4.36))/2,getypos()+kh_length);
692 gcpscad  cutwithfeed(getxpos(),getypos()+kh_length,-kh_max_depth,feed);
693 gcpscad  setypos(getypos()-kh_length);
694 gcpscad   } else if (kh_angle == 180) {
695 gcpscad dxfarc(KH_tool_no,getxpos(),getypos(),tool_diameter(KH_tool_no, (
            kh_max_depth))/2,0,90);
696 gcpscad dxfarc(KH_tool_no,getxpos(),getypos(),tool_diameter(KH_tool_no, (
            kh_max_depth))/2,270,360);
697 gcpscad dxfarc(KH_tool_no,getxpos(),getypos(),tool_diameter(KH_tool_no, (
            kh_max_depth))/2,90,180-asin((tool_diameter(KH_tool_no, (
            kh_max_depth+4.36))/2)/(tool_diameter(KH_tool_no, (kh_max_depth)
            )/2)));
698 gcpscad dxfarc(KH_tool_no,getxpos(),getypos(),tool_diameter(KH_tool_no, (
            kh_max_depth))/2,180+asin((tool_diameter(KH_tool_no, (
            kh_max_depth+4.36))/2)/(tool_diameter(KH_tool_no, (kh_max_depth)
            )/2)),270);
699 gcpscad dxfarc(KH_tool_no,getxpos()-kh_length,getypos(),tool_diameter(
            KH_tool_no, (kh_max_depth+4.36))/2,90,180);
700 gcpscad dxfarc(KH_tool_no,getxpos()-kh_length,getypos(),tool_diameter(
            KH_tool_no, (kh_max_depth+4.36))/2,180,270);
701 gcpscad dxfpolyline(KH_tool_no,
702 gcpscad  getxpos()-sqrt((tool_diameter(KH_tool_no, (kh_max_depth))/2)^2-(
             tool_diameter(KH_tool_no, (kh_max_depth+4.36))/2)^2),
703 gcpscad  getypos()+tool_diameter(KH_tool_no, (kh_max_depth+4.36))/2,
704 gcpscad  getxpos()-kh_length,
705 gcpscad  getypos()+tool_diameter(KH_tool_no, (kh_max_depth+4.36))/2);
706 gcpscad dxfpolyline(KH_tool_no,
707 gcpscad  getxpos()-sqrt((tool_diameter(KH_tool_no, (kh_max_depth))/2)^2-(
             tool_diameter(KH_tool_no, (kh_max_depth+4.36))/2)^2),
708 gcpscad  getypos()-tool_diameter(KH_tool_no, (kh_max_depth+4.36))/2,
709 gcpscad  getxpos()-kh_length,
710 gcpscad  getypos()-tool_diameter(KH_tool_no, (kh_max_depth+4.36))/2);
711 gcpscad  dxfpolyline(KH_tool_no,getxpos(),getypos(),getxpos()-kh_length,
             getypos());
712 gcpscad  cutwithfeed(getxpos()-kh_length,getypos(),-kh_max_depth,feed);
713 gcpscad  setxpos(getxpos()+kh_length);
714 gcpscad   } else if (kh_angle == 270) {
715 gcpscad dxfarc(KH_tool_no,getxpos(),getypos(),tool_diameter(KH_tool_no, (
            kh_max_depth))/2,0,90);
716 gcpscad dxfarc(KH_tool_no,getxpos(),getypos(),tool_diameter(KH_tool_no, (
            kh_max_depth))/2,90,180);
717 gcpscad dxfarc(KH_tool_no,getxpos(),getypos(),tool_diameter(KH_tool_no, (
            kh_max_depth))/2,270+asin((tool_diameter(KH_tool_no, (
            kh_max_depth+4.36))/2)/(tool_diameter(KH_tool_no, (kh_max_depth)
            )/2)),360);
718 gcpscad dxfarc(KH_tool_no,getxpos(),getypos(),tool_diameter(KH_tool_no, (
            kh_max_depth))/2,180, 270-asin((tool_diameter(KH_tool_no, (
            kh_max_depth+4.36))/2)/(tool_diameter(KH_tool_no, (kh_max_depth)
            )/2)));
719 gcpscad dxfarc(KH_tool_no,getxpos(),getypos()-kh_length,tool_diameter(
            KH_tool_no, (kh_max_depth+4.36))/2,180,270);
720 gcpscad dxfarc(KH_tool_no,getxpos(),getypos()-kh_length,tool_diameter(
            KH_tool_no, (kh_max_depth+4.36))/2,270,360);
721 gcpscad  dxfpolyline(KH_tool_no,getxpos()+tool_diameter(KH_tool_no, (
             kh_max_depth+4.36))/2,getypos()-sqrt((tool_diameter(KH_tool_no,
```

```
                (kh_max_depth))/2)^2-(tool_diameter(KH_tool_no, (kh_max_depth
                +4.36))/2)^2),getxpos()+tool_diameter(KH_tool_no, (kh_max_depth
                +4.36))/2,getypos()-kh_length);
722 gcpscad  dxfpolyline(KH_tool_no,getxpos()-tool_diameter(KH_tool_no, (
                kh_max_depth+4.36))/2,getypos()-sqrt((tool_diameter(KH_tool_no,
                 (kh_max_depth))/2)^2-(tool_diameter(KH_tool_no, (kh_max_depth
                +4.36))/2)^2),getxpos()-tool_diameter(KH_tool_no, (kh_max_depth
                +4.36))/2,getypos()-kh_length);
723 gcpscad  dxfpolyline(KH_tool_no,getxpos(),getypos(),getxpos(),getypos()-
                kh_length);
724 gcpscad  cutwithfeed(getxpos(),getypos()-kh_length,-kh_max_depth,feed);
725 gcpscad  setypos(getypos()+kh_length);
726 gcpscad   }
727 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.

begincutdxf **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
continuecutdxf use of `continuecutdxf`.

```
737 gcpscad module begincutdxf(rh, ex, ey, ez, fr) {
738 gcpscad   rapid(getxpos(),getypos(),rh);
739 gcpscad   cutwithfeed(ex,ey,ez,fr);
740 gcpscad }
```

```
742 gcpscad module continuecutdxf(ex, ey, ez, fr) {
743 gcpscad   cutwithfeed(ex,ey,ez,fr);
744 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 `cutrectangledxf`cuts 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:

```
746 gcpscad module cutrectangledxf(bx, by, bz, rwidth, rheight, rdepth, rtn)
                {//passes
747 gcpscad   movetosafez();
748 gcpscad   hull(){
749 gcpscad     //  for (i = [0 : abs(1) : passes]) {
750 gcpscad     //      rapid(bx+tool_radius(rtn)+i*(rwidth-tool_diameter(
                    current_tool()))/passes,bx+tool_radius(rtn),1);
```

```
751 gcpscad     //      cutwithfeed(bx+tool_radius(rtn)+i*(rwidth-tool_diameter
                    (current_tool()))/passes,by+tool_radius(rtn),bz-rdepth,feed)
                    ;
752 gcpscad     //      cutwithfeed(bx+tool_radius(rtn)+i*(rwidth-tool_diameter
                    (current_tool()))/passes,by+rheight-tool_radius(rtn),bz-
                    rdepth,feed);
753 gcpscad
754 gcpscad     cutwithfeed(bx+tool_radius(rtn),by+tool_radius(rtn),bz-rdepth,
                    feed);
755 gcpscad     cutwithfeed(bx+rwidth-tool_radius(rtn),by+tool_radius(rtn),bz-
                    rdepth,feed);
756 gcpscad     cutwithfeed(bx+rwidth-tool_radius(rtn),by+rheight-tool_radius(
                    rtn),bz-rdepth,feed);
757 gcpscad     cutwithfeed(bx+tool_radius(rtn),by+rheight-tool_radius(rtn),bz-
                    rdepth,feed);
758 gcpscad   }
759 gcpscad   //dxfarc(tn,xcenter,ycenter,radius,anglebegin,endangle)
760 gcpscad   dxfarc(rtn,bx+tool_radius(rtn),by+tool_radius(rtn),tool_radius(
                    rtn),180,270);
761 gcpscad   //dxfpolyline(tn,xbegin,ybegin,xend,yend)
762 gcpscad   dxfpolyline(rtn,bx,by+tool_radius(rtn),bx,by+rheight-tool_radius(
                    rtn));
763 gcpscad   dxfarc(rtn,bx+tool_radius(rtn),by+rheight-tool_radius(rtn),
                    tool_radius(rtn),90,180);
764 gcpscad   dxfpolyline(rtn,bx+tool_radius(rtn),by+rheight,bx+rwidth-
                    tool_radius(rtn),by+rheight);
765 gcpscad   dxfarc(rtn,bx+rwidth-tool_radius(rtn),by+rheight-tool_radius(rtn)
                    ,tool_radius(rtn),0,90);
766 gcpscad   dxfpolyline(rtn,bx+rwidth,by+rheight-tool_radius(rtn),bx+rwidth,
                    by+tool_radius(rtn));
767 gcpscad   dxfarc(rtn,bx+rwidth-tool_radius(rtn),by+tool_radius(rtn),
                    tool_radius(rtn),270,360);
768 gcpscad   dxfpolyline(rtn,bx+rwidth-tool_radius(rtn),by,bx+tool_radius(rtn)
                    ,by);
769 gcpscad }
```

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

```
771 gcpscad module cutrectangleoutlinedxf(bx, by, bz, rwidth, rheight, rdepth,
                rtn) {//passes
772 gcpscad   movetosafez();
773 gcpscad   cutwithfeed(bx+tool_radius(rtn),by+tool_radius(rtn),bz-rdepth,
                    feed);
774 gcpscad   cutwithfeed(bx+rwidth-tool_radius(rtn),by+tool_radius(rtn),bz-
                    rdepth,feed);
775 gcpscad   cutwithfeed(bx+rwidth-tool_radius(rtn),by+rheight-tool_radius(rtn
                    ),bz-rdepth,feed);
776 gcpscad   cutwithfeed(bx+tool_radius(rtn),by+rheight-tool_radius(rtn),bz-
                    rdepth,feed);
777 gcpscad   dxfarc(rtn,bx+tool_radius(rtn),by+tool_radius(rtn),tool_radius(
                    rtn),180,270);
778 gcpscad   dxfpolyline(rtn,bx,by+tool_radius(rtn),bx,by+rheight-tool_radius(
                    rtn));
779 gcpscad   dxfarc(rtn,bx+tool_radius(rtn),by+rheight-tool_radius(rtn),
                    tool_radius(rtn),90,180);
780 gcpscad   dxfpolyline(rtn,bx+tool_radius(rtn),by+rheight,bx+rwidth-
                    tool_radius(rtn),by+rheight);
781 gcpscad   dxfarc(rtn,bx+rwidth-tool_radius(rtn),by+rheight-tool_radius(rtn)
                    ,tool_radius(rtn),0,90);
782 gcpscad   dxfpolyline(rtn,bx+rwidth,by+rheight-tool_radius(rtn),bx+rwidth,
                    by+tool_radius(rtn));
783 gcpscad   dxfarc(rtn,bx+rwidth-tool_radius(rtn),by+tool_radius(rtn),
                    tool_radius(rtn),270,360);
784 gcpscad   dxfpolyline(rtn,bx+rwidth-tool_radius(rtn),by,bx+tool_radius(rtn)
                    ,by);
785 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):

```
787 gcpscad module rectangleoutlinedxf(bx, by, bz, rwidth, rheight, rtn) {
788 gcpscad   dxfpolyline(rtn,bx,by,bx,by+rheight);
789 gcpscad   dxfpolyline(rtn,bx,by+rheight,bx+rwidth,by+rheight);
790 gcpscad   dxfpolyline(rtn,bx+rwidth,by+rheight,bx+rwidth,by);
791 gcpscad   dxfpolyline(rtn,bx+rwidth,by,bx,by);
```

```
792 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:

```
794 gcpscad module cutoutrectangledxf(bx, by, bz, rwidth, rheight, rdepth, rtn)
                {
795 gcpscad   movetosafez();
796 gcpscad   cutwithfeed(bx-tool_radius(rtn),by-tool_radius(rtn),bz-rdepth,
                feed);
797 gcpscad   cutwithfeed(bx+rwidth+tool_radius(rtn),by-tool_radius(rtn),bz-
                rdepth,feed);
798 gcpscad   cutwithfeed(bx+rwidth+tool_radius(rtn),by+rheight+tool_radius(rtn
                ),bz-rdepth,feed);
799 gcpscad   cutwithfeed(bx-tool_radius(rtn),by+rheight+tool_radius(rtn),bz-
                rdepth,feed);
800 gcpscad   cutwithfeed(bx-tool_radius(rtn),by-tool_radius(rtn),bz-rdepth,
                feed);
801 gcpscad   dxfpolyline(rtn,bx,by,bx,by+rheight);
802 gcpscad   dxfpolyline(rtn,bx,by+rheight,bx+rwidth,by+rheight);
803 gcpscad   dxfpolyline(rtn,bx+rwidth,by+rheight,bx+rwidth,by);
804 gcpscad   dxfpolyline(rtn,bx+rwidth,by,bx,by);
805 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
13 gcptmpl /* [Export] */
14 gcptmpl generatedxf = true;
15 gcptmpl
16 gcptmpl /* [Export] */
17 gcptmpl generategcode = true;
18 gcptmpl
19 gcptmpl ///* [Export] */
20 gcptmpl //generatesvg = false;
21 gcptmpl
22 gcptmpl /* [CAM] */
23 gcptmpl toolradius = 1.5875;
24 gcptmpl
25 gcptmpl /* [CAM] */
26 gcptmpl large_ball_tool_no = 0; // [0:0,111:111,101:101,202:202]
27 gcptmpl
28 gcptmpl /* [CAM] */
29 gcptmpl large_square_tool_no = 0; // [0:0,112:112,102:102,201:201]
30 gcptmpl
31 gcptmpl /* [CAM] */
32 gcptmpl large_V_tool_no = 0; // [0:0,301:301,690:690]
33 gcptmpl
34 gcptmpl /* [CAM] */
35 gcptmpl small_ball_tool_no = 0; // [0:0,121:121,111:111,101:101]
```

```
gcptmpl
gcptmpl /* [CAM] */
gcptmpl small_square_tool_no = 102; // [0:0,122:122,112:112,102:102]
gcptmpl
gcptmpl /* [CAM] */
gcptmpl small_V_tool_no = 0; // [0:0,390:390,301:301]
gcptmpl
gcptmpl /* [CAM] */
gcptmpl KH_tool_no = 0; // [0:0,375:375]
gcptmpl
gcptmpl /* [CAM] */
gcptmpl DT_tool_no = 0; // [0:0,814:814]
gcptmpl
gcptmpl /* [Feeds and Speeds] */
gcptmpl plunge = 100;
gcptmpl
gcptmpl /* [Feeds and Speeds] */
gcptmpl feed = 400;
gcptmpl
gcptmpl /* [Feeds and Speeds] */
gcptmpl speed = 16000;
gcptmpl
gcptmpl /* [Feeds and Speeds] */
gcptmpl square_ratio = 1.0; // [0.25:2]
gcptmpl
gcptmpl /* [Feeds and Speeds] */
gcptmpl small_V_ratio = 0.75; // [0.25:2]
gcptmpl
gcptmpl /* [Feeds and Speeds] */
gcptmpl large_V_ratio = 0.875; // [0.25:2]
gcptmpl
gcptmpl /* [Stock] */
gcptmpl stocklength = 219;
gcptmpl
gcptmpl /* [Stock] */
gcptmpl stockwidth = 150;
gcptmpl
gcptmpl /* [Stock] */
gcptmpl stockthickness = 8.35;
gcptmpl
gcptmpl /* [Stock] */
gcptmpl zeroheight = "Top"; // [Top, Bottom]
gcptmpl
gcptmpl /* [Stock] */
gcptmpl stockorigin = "Center"; // [Lower-Left, Center-Left, Top-Left,
        Center]
gcptmpl
gcptmpl /* [Stock] */
gcptmpl retractheight = 9;
gcptmpl
gcptmpl filename_gcode = str(Base_filename, ".nc");
gcptmpl filename_dxf = str(Base_filename);
gcptmpl //filename_svg = str(Base_filename, ".svg");
gcptmpl
gcptmpl opengcodefile(filename_gcode);
gcptmpl opendxffile(filename_dxf);
gcptmpl
gcptmpl difference() {
gcptmpl setupstock(stocklength, stockwidth, stockthickness, zeroheight,
        stockorigin);
gcptmpl
gcptmpl movetosafez();
gcptmpl
gcptmpl toolchange(small_square_tool_no,speed * square_ratio);
gcptmpl
gcptmpl begintoolpath(0,0,0.25);
gcptmpl beginpolyline(0,0,0.25);
gcptmpl
gcptmpl cutoneaxis_setfeed("Z",0,plunge*square_ratio);
gcptmpl
gcptmpl cutwithfeed(stocklength/2,stockwidth/2,-stockthickness,feed);
gcptmpl addpolyline(stocklength/2,stockwidth/2,-stockthickness);
gcptmpl
gcptmpl endtoolpath();
gcptmpl closepolyline();
gcptmpl }
gcptmpl
gcptmpl closegcodefile();
```

```
112 gcptmpl closedxffile();
```

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:

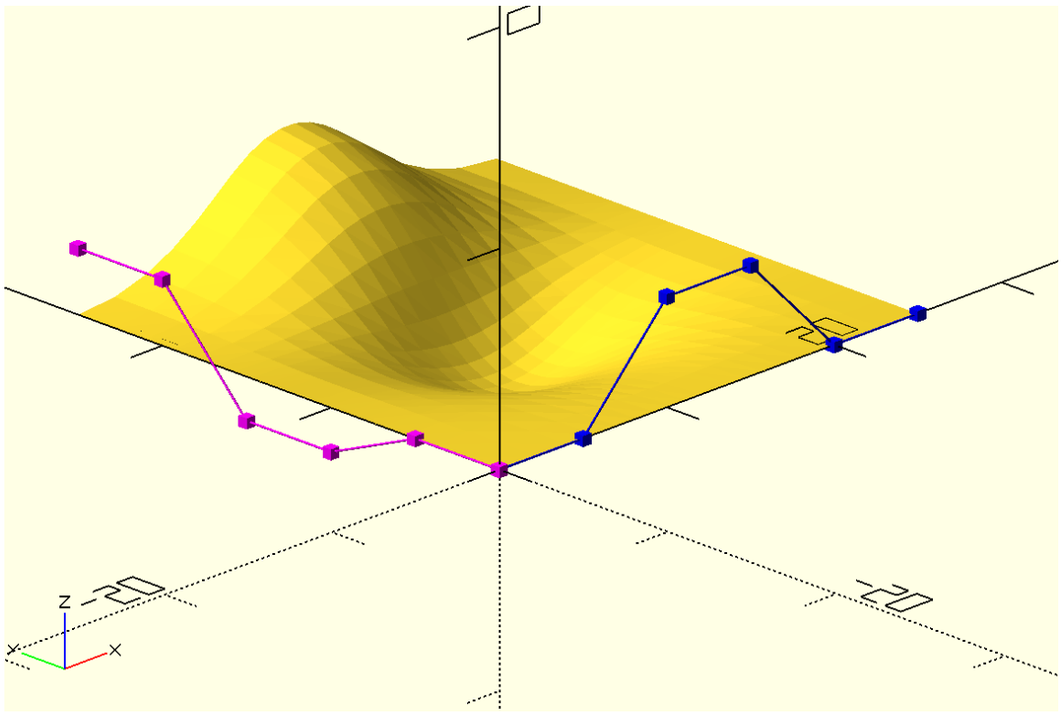
- XY
- XZ
- ZY

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

which is a marked contrast to representations such as:
https://github.com/DavidPhillipOster/Teapot
and regions which could not be so represented could be sub-divided until the representation is workable.

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



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

6 Other Resources

Holidays are from https://nationaltoday.com/

References

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

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

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

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

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

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

Index

Routines

Variables