The gcodepreview OpenSCAD library*

Author: William F. Adams
willadams at aol dot com

2024/09/08

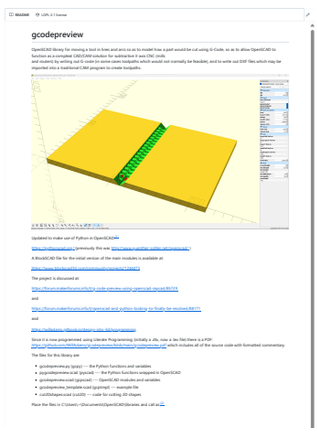
Abstract

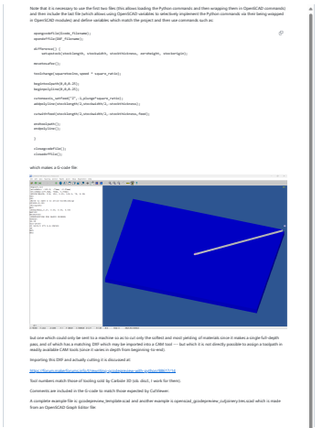
The gcodepreview library allows using OpenPythonSCAD to move a tool in lines and arcs and output dxf and G-code files so as to work as a CAD/CAM program for CNC.

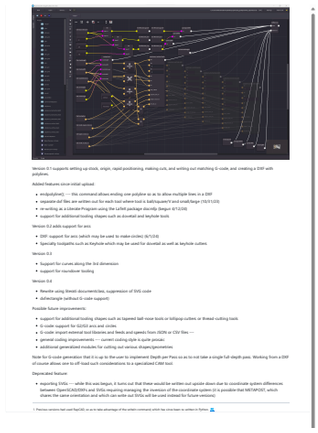
Contents

*This file (gcodepreview) has version number v0.61, last revised 2024/09/08.

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)
rdme
rdme Version 0.1 supports setting up stock, origin, rapid
rdme positioning, making cuts, and writing out matching
rdme G-code, and creating a DXF with polylines.
rdme
rdme Added features since initial upload:
rdme
rdme  - endpolyline(); --- this command allows ending one polyline so as
            to allow multiple lines in a DXF
rdme  - separate dxf files are written out for each tool where tool is
            ball/square/V and small/large (10/31/23)
rdme  - re-writing as a Literate Program using the LaTeX package docmfp
            (begun 4/12/24)
rdme  - support for additional tooling shapes such as dovetail and
            keyhole tools
rdme
rdme Version 0.2 adds support for arcs
rdme
rdme  - DXF: support for arcs (which may be used to make circles)
            (6/1/24)
rdme  - Specialty toolpaths such as Keyhole which may be used for
            dovetail as well as keyhole cutters
rdme
rdme Version 0.3
rdme
rdme  - Support for curves along the 3rd dimension
rdme  - support for roundover tooling
rdme
rdme Version 0.4
rdme
rdme  - Rewrite using literati documentclass, suppression of SVG code
rdme  - dxfrectangle (without G-code support)
rdme
rdme Version 0.5
rdme
rdme  - more shapes
rdme  - consolidate rectangles, arcs, and circles in gcodepreview.scad
rdme
rdme Version 0.6
rdme
rdme  - notes on modules
rdme  - change file for setupstock
rdme
rdme Version 0.7
rdme
rdme  - reduce usage of tool numbers
rdme  - validate all code so that it runs without errors from sample
            file
rdme
rdme Possible future improvements:
rdme
rdme  - support for additional tooling shapes such as tapered ball-nose
            tools or lollipop cutters or thread-cutting tools
rdme  - G-code: support for G2/G3 arcs and circles
rdme  - G-code: import external tool libraries and feeds and speeds from
            JSON or CSV files ---
rdme  - general coding improvements --- current coding style is quite
            prosaic
rdme  - additional generalized modules for cutting out various shapes/
            geometries
rdme
rdme Note for G-code generation that it is up to the user
rdme to implement Depth per Pass so as to not take a
rdme single full-depth pass. Working from a DXF of course
rdme allows one to off-load such considerations to a
rdme specialized CAM tool.
rdme
rdme Deprecated feature:
rdme
rdme  - exporting SVGs --- while this was begun, it turns out that these
            would be written out upside down due to coordinate system
            differences between OpenSCAD/DXFs and SVGs requiring managing
            the inversion of the coordinate system (it is possible that
            METAPOST, which shares the same orientation and which can write
            out SVGs will be used instead for future versions)
```

2 gcodepreview

This library for OpenPythonSCAD works by using Python code as a back-end so as to persistently store and access variables, and to write out files while both modeling the motion of a 3-axis CNC machine and if desired, writing out DXF and/or G-code files (as opposed to the normal technique of rendering to a 3D model and writing out an STL). Doing so requires a total of three files:

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

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

```
1 gcpy #!/usr/bin/env python
2 gcpy #icon "C:\Program Files\PythonSCAD\bin\openscad.exe"  --trust-
      python
3 gcpy #Currently tested with 2024.09.01 and Python 3.11
4 gcpy #gcodepreview 0.7, see gcodepreview.scad
```

```
1 pyscad //!OpenSCAD
2 pyscad
3 pyscad //gcodepreview 0.7, see gcodepreview.scad
```

```
1 gcpscad //!OpenSCAD
2 gcpscad
3 gcpscad //gcodepreview 0.7
4 gcpscad //
5 gcpscad //used via use <gcodepreview.py>;
6 gcpscad //          use <pygcodepreview.scad>;
7 gcpscad //          include <gcodepreview.scad>;
8 gcpscad //
```

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

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

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

2.1 Position and Variables

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

The first such variables are for XYZ position:

mpx
- `mpx`

mpy
- `mpy`

mpz
- `mpz`

Similarly, for some toolpaths it will be necessary to track the depth along the Z-axis as the toolpath is cut out:

tpz
- `tpz`

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

currenttool
- `currenttool`

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

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

2.2 Modules

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

In natural languages such as English, there is an order to various parts of speech such as adjectives — since various prefixes and suffixes will be used for module names, having a consistent ordering/usage will help in consistency and make expression clearer. The ordering should be: sequence, action, function, parameter, filetype

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

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

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

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

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

current_tool(); [function]

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

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

cutkeyhole_toolpath(kh_start_depth, kh_max_depth, kht_direction, kh_distance);

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

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

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

```
movetosafeheight();
movetosafez();

opendxffile(fn); and closedxffile();
opengcodefile(fn); and closegcodefile();

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

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

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

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

settzpos(newtzpos);

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

writecomment(comment);
```

Principles for naming modules (and variables):

- minimize use of underscores (for convenience sake, underscores are not used for index entries)
- identify which aspect of the project structure is being worked with (`cut`(ting), `dxf`, `gcode`, `tool` management, etc.) and esp. note the use of `o`(penscad) and `p`(ython) as prefixes

Structurally, this will typically look like:

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

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

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


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

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

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

which in turn calls the internal Python definitioon \DescribeSubroutine{FOOBAR}{pFOOBAR}

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

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

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

2.2.1 Initial Modules

`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
```

Note that while the #102 is declared as a default tool, while it was originally necessary to call a tool change after invoking `setupstock` in the 2024.09.03 version of PythonSCAD this requirement went away when a but which interfered with persistently setting a variable directly was fixed.

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

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

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

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

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

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

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

An example usage would be:

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

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.

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

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

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

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

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

2.3 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:

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

2.3.1 toolchange

toolchange and apply the appropriate commands for a `toolchange`. Note that it is expected that this code will be updated as needed when new tooling is introduced as additional modules which require specific tooling are added below.

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

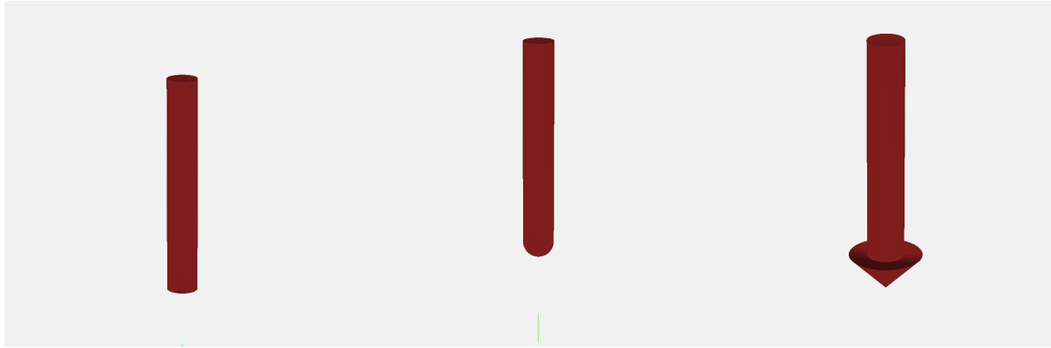
A further concern is that early versions often passed the tool into a module using a parameter. That ceased to be necessary in the 2024.09.03 version of PythonSCAD, and all modules should read the tool # from `currenttool()`.

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

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

2.3.1.1 Normal Tooling

Most tooling has quite standard shapes and are defined by their profile:



- Square (#201 and 102) — able to cut a flat bottom, perpendicular side and right angle their simple and easily understood geometry makes them a standard choice (a radiused form with a flat bottom, often described as a "bowl bit" is not implemented as-of-yet)
- Ballnose (#202 and 101) — rounded, they are the standard choice for concave and organic shapes
- V tooling (#301, 302 and 390) — pointed at the tip, they are available in a variety of angles and diameters and may be used for decorative V carving, or for chamfering or cutting specific angles (note that the commonly available radiused form is not implemented at this time, *e.g.*, #501 and 502)

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

2.3.1.2 Tooling for Keyhole Toolpaths Keyhole toolpaths (see: subsection 3.2 are intended for use with tooling which projects beyond the the narrower shaft and so will cut usefully underneath the visible surface. Also described as "undercut" tooling, but see below.

There are several notable candidates for such tooling:

- Keyhole 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

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

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

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

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

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

```
139 gcpscad     }
140 gcpscad       select_tool(tool_number);
141 gcpscad       owritetwo("M6T",str(tool_number));
142 gcpscad       owritetwo("M03S",str(speed));
143 gcpscad   }
144 gcpscad }
```

For example:

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

2.3.1.4 Roundover tooling It is not possible to represent all tools using tool changes as coded above which require using a `hull` operation between 3D representations of the tools at the beginning and end points. Tooling which cannot be so represented will be implemented separately below, see paragraph 2.3.2.2.

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

```
146 gcpscad module select_tool(tool_number) {
147 gcpscad echo(tool_number);
148 gcpscad   if (tool_number == 201) {
149 gcpscad     gcp_endmill_square(6.35, 19.05);
150 gcpscad   } else if (tool_number == 202) {
151 gcpscad     gcp_endmill_ball(6.35, 19.05);
152 gcpscad   } else if (tool_number == 102) {
153 gcpscad     gcp_endmill_square(3.175, 19.05);
154 gcpscad   } else if (tool_number == 101) {
155 gcpscad     gcp_endmill_ball(3.175, 19.05);
156 gcpscad   } else if (tool_number == 301) {
157 gcpscad     gcp_endmill_v(90, 12.7);
158 gcpscad   } else if (tool_number == 302) {
159 gcpscad     gcp_endmill_v(60, 12.7);
160 gcpscad   } else if (tool_number == 390) {
161 gcpscad     gcp_endmill_v(90, 3.175);
```

For a keyhole tool:

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

and dovetail tool:

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

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

```
166 gcpscad   }
167 gcpscad }
```

2.3.2 3D Shapes for Tools

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

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

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

```
169 gcpscad module gcp_endmill_square(es_diameter, es_flute_length) {
170 gcpscad   cylinder(r1=(es_diameter / 2), r2=(es_diameter / 2), h=
                es_flute_length, center=false);
171 gcpscad }
```

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

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

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

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

```
gcpscad module gcp_endmill_ball(es_diameter, es_flute_length) {
gcpscad   translate([0, 0, (es_diameter / 2)]){
gcpscad     union(){
gcpscad       sphere(r=(es_diameter / 2));
gcpscad       cylinder(r1=(es_diameter / 2), r2=(es_diameter / 2), h=
                  es_flute_length, center=false);
gcpscad     }
gcpscad   }
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:

```
gcpscad module gcp_endmill_v(es_v_angle, es_diameter) {
gcpscad   union(){
gcpscad     cylinder(r1=0, r2=(es_diameter / 2), h=((es_diameter / 2) / tan
                ((es_v_angle / 2))), center=false);
gcpscad     translate([0, 0, ((es_diameter / 2) / tan((es_v_angle / 2)))]){
gcpscad       cylinder(r1=(es_diameter / 2), r2=(es_diameter / 2), h=((
                  es_diameter * 8) ), center=false);/// tan((es_v_angle / 2)
                  )
gcpscad     }
gcpscad   }
gcpscad }
```

2.3.2.2 Concave toolshapes

While normal tooling may be represented with a single `hull` operation betwixt two 3D toolshapes, concave tooling such as roundover/radius tooling require multiple slices of the tool shape which are then `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: `cutroundover`

```
gcpscad module cutroundover(bx, by, bz, ex, ey, ez, radiustn) {
gcpscad     if (radiustn == 56125) {
gcpscad         cutroundovertool(bx, by, bz, ex, ey, ez, 0.508/2, 1.531);
gcpscad     } else if (radiustn == 56142) {
gcpscad         cutroundovertool(bx, by, bz, ex, ey, ez, 0.508/2, 2.921);
gcpscad     } else if (radiustn == 312) {
gcpscad         cutroundovertool(bx, by, bz, ex, ey, ez, 1.524/2, 3.175);
gcpscad     } else if (radiustn == 1570) {
gcpscad         cutroundovertool(bx, by, bz, ex, ey, ez, 0.507/2, 4.509);
gcpscad     }
gcpscad }
```

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

```
gcpscad module cutroundovertool(bx, by, bz, ex, ey, ez, tool_radius_tip,
            tool_radius_width) {
```

```
212 gcpscad n = 90 + $fn*3;
213 gcpscad step = 360/n;
214 gcpscad
215 gcpscad hull(){
216 gcpscad     translate([bx,by,bz])
217 gcpscad     cylinder(step,tool_radius_tip,tool_radius_tip);
218 gcpscad     translate([ex,ey,ez])
219 gcpscad     cylinder(step,tool_radius_tip,tool_radius_tip);
220 gcpscad }
221 gcpscad
222 gcpscad hull(){
223 gcpscad translate([bx,by,bz+tool_radius_width])
224 gcpscad cylinder(tool_radius_width*2,tool_radius_tip+tool_radius_width,
              tool_radius_tip+tool_radius_width);
225 gcpscad
226 gcpscad translate([ex,ey,ez+tool_radius_width])
227 gcpscad   cylinder(tool_radius_width*2,tool_radius_tip+tool_radius_width,
                tool_radius_tip+tool_radius_width);
228 gcpscad }
229 gcpscad
230 gcpscad for (i=[0:step:90]) {
231 gcpscad     angle = i;
232 gcpscad     dx = tool_radius_width*cos(angle);
233 gcpscad     dxx = tool_radius_width*cos(angle+step);
234 gcpscad     dzz = tool_radius_width*sin(angle);
235 gcpscad     dz = tool_radius_width*sin(angle+step);
236 gcpscad     dh = dz-dzz;
237 gcpscad     hull(){
238 gcpscad         translate([bx,by,bz+dz])
239 gcpscad             cylinder(dh,tool_radius_tip+tool_radius_width-dx,
                        tool_radius_tip+tool_radius_width-dxx);
240 gcpscad         translate([ex,ey,ez+dz])
241 gcpscad             cylinder(dh,tool_radius_tip+tool_radius_width-dx,
                        tool_radius_tip+tool_radius_width-dxx);
242 gcpscad         }
243 gcpscad     }
244 gcpscad }
```

2.3.3 tooldiameter

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

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

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

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

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

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

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

```
 65 gcpy def ptool_diameter(ptd_tool, ptd_depth):
 66 gcpy # Square 122,112,102,201
 67 gcpy     if ptd_tool == 122:
 68 gcpy         return 0.79375
 69 gcpy     if ptd_tool == 112:
 70 gcpy         return 1.5875
 71 gcpy     if ptd_tool == 102:
 72 gcpy         return 3.175
 73 gcpy     if ptd_tool == 201:
 74 gcpy         return 6.35
 75 gcpy # Ball 121,111,101,202
 76 gcpy     if ptd_tool == 122:
 77 gcpy         return
 78 gcpy         if ptd_depth > 0.396875:
```

```
 79 gcpy             return 0.79375
 80 gcpy         else:
 81 gcpy             return 0
 82 gcpy     if ptd_tool == 112:
 83 gcpy         if ptd_depth > 0.79375:
 84 gcpy             return 1.5875
 85 gcpy         else:
 86 gcpy             return 0
 87 gcpy     if ptd_tool == 101:
 88 gcpy         if ptd_depth > 1.5875:
 89 gcpy             return 3.175
 90 gcpy         else:
 91 gcpy             return 0
 92 gcpy     if ptd_tool == 202:
 93 gcpy         if ptd_depth > 3.175:
 94 gcpy             return 6.35
 95 gcpy         else:
 96 gcpy             return 0
 97 gcpy # V 301, 302, 390
 98 gcpy     if ptd_tool == 301:
 99 gcpy         return 0
100 gcpy     if ptd_tool == 302:
101 gcpy         return 0
102 gcpy     if ptd_tool == 390:
103 gcpy         return 0
104 gcpy # Keyhole
105 gcpy     if ptd_tool == 375:
106 gcpy         if ptd_depth < 6.35:
107 gcpy             return 9.525
108 gcpy         else:
109 gcpy             return 6.35
110 gcpy # Dovetail
111 gcpy     if ptd_tool == 814:
112 gcpy         if ptd_depth > 12.7:
113 gcpy             return 6.35
114 gcpy         else:
115 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:

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

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

2.4 File Handling

popengcodefile
popendxffile
popendxflgsqfile
popendxfsmsqfile
popendxflgblfile
popendxfsmblfile
popendxflgVfile
popendxfsmVfile

For writing to files it will be necessary to have commands: `popengcodefile`, `popendxffile`, `popendxflgsqfile`, `popendxfsmsqfile`, `popendxflgblfile`, `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 popendxflgblfile(fn):
106 gcpy     global dxflgbl
107 gcpy     dxflgbl = open(fn, "w")
108 gcpy
109 gcpy def popendxflgsqfile(fn):
110 gcpy     global dxflgsq
111 gcpy     dxflgsq = open(fn, "w")
112 gcpy
113 gcpy def popendxflgVfile(fn):
114 gcpy     global dxflgV
115 gcpy     dxflgV = open(fn, "w")
116 gcpy
117 gcpy def popendxfsmblfile(fn):
```

```
118 gcpy      global 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 popendxfDTfile(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.

```
38 pyscad module oopengcodefile(fn) {
39 pyscad      popengcodefile(fn);
40 pyscad }
41 pyscad
42 pyscad module oopendxffile(fn) {
43 pyscad      echo(fn);
44 pyscad      popendxffile(fn);
45 pyscad }
46 pyscad
47 pyscad module oopendxflgblfile(fn) {
48 pyscad      popendxflgblfile(fn);
49 pyscad }
50 pyscad
51 pyscad module oopendxflgsqfile(fn) {
52 pyscad      popendxflgsqfile(fn);
53 pyscad }
54 pyscad
55 pyscad module oopendxflgVfile(fn) {
56 pyscad      popendxflgVfile(fn);
57 pyscad }
58 pyscad
59 pyscad module oopendxfsmblfile(fn) {
60 pyscad      popendxfsmblfile(fn);
61 pyscad }
62 pyscad
63 pyscad module oopendxfsmsqfile(fn) {
64 pyscad //     echo(fn);
65 pyscad      popendxfsmsqfile(fn);
66 pyscad }
67 pyscad
68 pyscad module oopendxfsmVfile(fn) {
69 pyscad      popendxfsmVfile(fn);
70 pyscad }
71 pyscad
72 pyscad module oopendxfKHfile(fn) {
73 pyscad      popendxfKHfile(fn);
74 pyscad }
75 pyscad
76 pyscad module oopendxfDTfile(fn) {
77 pyscad      popendxfDTfile(fn);
78 pyscad }
```

opengcodefile

With matching OpenSCAD commands: `opengcodefile`

```
250 gcpscad module opengcodefile(fn) {
251 gcpscad if (generategcode == true) {
252 gcpscad      oopengcodefile(fn);
253 gcpscad      echo(fn);
254 gcpscad      owritecomment(fn);
255 gcpscad      }
256 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:

```
258 gcpscad module opendxffile(fn) {
259 gcpscad   if (generatedxf == true) {
260 gcpscad       oopendxffile(str(fn,".dxf"));
261 gcpscad //     echo(fn);
262 gcpscad       dxfwriteone("0");
263 gcpscad       dxfwriteone("SECTION");
264 gcpscad       dxfwriteone("2");
265 gcpscad       dxfwriteone("ENTITIES");
266 gcpscad     if (large_ball_tool_no >  0) {    oopendxflgblfile(str(fn,".",
                    large_ball_tool_no,".dxf"));
267 gcpscad       dxfpreamble(large_ball_tool_no);
268 gcpscad     }
269 gcpscad     if (large_square_tool_no >  0) {    oopendxflgsqfile(str(fn
                    ,".",large_square_tool_no,".dxf"));
270 gcpscad       dxfpreamble(large_square_tool_no);
271 gcpscad     }
272 gcpscad     if (large_V_tool_no >  0) {    oopendxflgVfile(str(fn,".",
                    large_V_tool_no,".dxf"));
273 gcpscad       dxfpreamble(large_V_tool_no);
274 gcpscad     }
275 gcpscad     if (small_ball_tool_no >  0) { oopendxfsmblfile(str(fn,".",
                    small_ball_tool_no,".dxf"));
276 gcpscad       dxfpreamble(small_ball_tool_no);
277 gcpscad     }
278 gcpscad     if (small_square_tool_no >  0) {    oopendxfsmsqfile(str(fn
                    ,".",small_square_tool_no,".dxf"));
279 gcpscad //     echo(str("tool no",small_square_tool_no));
280 gcpscad       dxfpreamble(small_square_tool_no);
281 gcpscad     }
282 gcpscad     if (small_V_tool_no >  0) {    oopendxfsmVfile(str(fn,".",
                    small_V_tool_no,".dxf"));
283 gcpscad       dxfpreamble(small_V_tool_no);
284 gcpscad     }
285 gcpscad     if (KH_tool_no >  0) {    oopendxfKHfile(str(fn,".",KH_tool_no
                    ,".dxf"));
286 gcpscad       dxfpreamble(KH_tool_no);
287 gcpscad     }
288 gcpscad     if (DT_tool_no >  0) {    oopendxfDTfile(str(fn,".",DT_tool_no
                    ,".dxf"));
289 gcpscad       dxfpreamble(DT_tool_no);
290 gcpscad     }
291 gcpscad   }
292 gcpscad }
```

2.4.1 Writing to files

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

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

has a matching command each tool/size combination:

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

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

writedxflgsq • Square, large (lgsq) `writedxflgsq`

writedxfsmsq • Square, small (smsq) `writedxfsmsq`

writedxflgV • V, large (lgV) `writedxflgV`

writedxfsmV • V, small (smV) `writedxfsmV`

writedxfKH • Keyhole (KH) `writedxfKH`

writedxfDT • Dovetail (DT) `writedxfDT`

```
144 gcpy def writedxflgbl(*arguments):
145 gcpy     line_to_write = ""
146 gcpy     for element in arguments:
```

```
147 gcpy             line_to_write += element
148 gcpy         dxflgbl.write(line_to_write)
149 gcpy         print(line_to_write)
150 gcpy         dxflgbl.write("\n")
151 gcpy
152 gcpy def writedxflgsq(*arguments):
153 gcpy         line_to_write = ""
154 gcpy         for element in arguments:
155 gcpy             line_to_write += element
156 gcpy         dxflgsq.write(line_to_write)
157 gcpy         print(line_to_write)
158 gcpy         dxflgsq.write("\n")
159 gcpy
160 gcpy def writedxflgV(*arguments):
161 gcpy         line_to_write = ""
162 gcpy         for element in arguments:
163 gcpy             line_to_write += element
164 gcpy         dxflgV.write(line_to_write)
165 gcpy         print(line_to_write)
166 gcpy         dxflgV.write("\n")
167 gcpy
168 gcpy def writedxfsmbl(*arguments):
169 gcpy         line_to_write = ""
170 gcpy         for element in arguments:
171 gcpy             line_to_write += element
172 gcpy         dxfsmbl.write(line_to_write)
173 gcpy         print(line_to_write)
174 gcpy         dxfsmbl.write("\n")
175 gcpy
176 gcpy def writedxfsmsq(*arguments):
177 gcpy         line_to_write = ""
178 gcpy         for element in arguments:
179 gcpy             line_to_write += element
180 gcpy         dxfsmsq.write(line_to_write)
181 gcpy         print(line_to_write)
182 gcpy         dxfsmsq.write("\n")
183 gcpy
184 gcpy def writedxfsmV(*arguments):
185 gcpy         line_to_write = ""
186 gcpy         for element in arguments:
187 gcpy             line_to_write += element
188 gcpy         dxfsmV.write(line_to_write)
189 gcpy         print(line_to_write)
190 gcpy         dxfsmV.write("\n")
191 gcpy
192 gcpy def writedxfKH(*arguments):
193 gcpy         line_to_write = ""
194 gcpy         for element in arguments:
195 gcpy             line_to_write += element
196 gcpy         dxfKH.write(line_to_write)
197 gcpy         print(line_to_write)
198 gcpy         dxfKH.write("\n")
199 gcpy
200 gcpy def writedxfDT(*arguments):
201 gcpy         line_to_write = ""
202 gcpy         for element in arguments:
203 gcpy             line_to_write += element
204 gcpy         dxfDT.write(line_to_write)
205 gcpy         print(line_to_write)
206 gcpy         dxfDT.write("\n")
```

owritecomment
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.

```
80 pyscad module owritecomment(comment) {
81 pyscad     writeln("(",comment,")");
82 pyscad }
83 pyscad
84 pyscad module dxfwriteone(first) {
85 pyscad     writedxf(first);
86 pyscad //     writeln(first);
87 pyscad //     echo(first);
88 pyscad }
89 pyscad
90 pyscad module dxfwritelgbl(first) {
91 pyscad     writedxflgbl(first);
```

```
 92 pyscad }
 93 pyscad
 94 pyscad module dxfwritelgsq(first) {
 95 pyscad     writedxflgsq(first);
 96 pyscad }
 97 pyscad
 98 pyscad module dxfwritelgV(first) {
 99 pyscad     writedxflgV(first);
100 pyscad }
101 pyscad
102 pyscad module dxfwritesmbl(first) {
103 pyscad     writedxfsmbl(first);
104 pyscad }
105 pyscad
106 pyscad module dxfwritesmsq(first) {
107 pyscad     writedxfsmsq(first);
108 pyscad }
109 pyscad
110 pyscad module dxfwritesmV(first) {
111 pyscad     writedxfsmV(first);
112 pyscad }
113 pyscad
114 pyscad module dxfwriteKH(first) {
115 pyscad     writedxfKH(first);
116 pyscad }
117 pyscad
118 pyscad module dxfwriteDT(first) {
119 pyscad     writedxfDT(first);
120 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...

```
122 pyscad module owriteone(first) {
123 pyscad     writeln(first);
124 pyscad }
125 pyscad
126 pyscad module owritetwo(first, second) {
127 pyscad     writeln(first, second);
128 pyscad }
129 pyscad
130 pyscad module owritethree(first, second, third) {
131 pyscad     writeln(first, second, third);
132 pyscad }
133 pyscad
134 pyscad module owritefour(first, second, third, fourth) {
135 pyscad     writeln(first, second, third, fourth);
136 pyscad }
137 pyscad
138 pyscad module owritefive(first, second, third, fourth, fifth) {
139 pyscad     writeln(first, second, third, fourth, fifth);
140 pyscad }
141 pyscad
142 pyscad module owritesix(first, second, third, fourth, fifth, sixth) {
143 pyscad     writeln(first, second, third, fourth, fifth, sixth);
144 pyscad }
145 pyscad
146 pyscad module owriteseven(first, second, third, fourth, fifth, sixth,
        seventh) {
147 pyscad     writeln(first, second, third, fourth, fifth, sixth, seventh);
148 pyscad }
149 pyscad
150 pyscad module owriteeight(first, second, third, fourth, fifth, sixth,
        seventh,eighth) {
151 pyscad     writeln(first, second, third, fourth, fifth, sixth, seventh,
            eighth);
152 pyscad }
153 pyscad
154 pyscad module owritenine(first, second, third, fourth, fifth, sixth,
        seventh, eighth, ninth) {
155 pyscad     writeln(first, second, third, fourth, fifth, sixth, seventh,
            eighth, ninth);
156 pyscad }
157 pyscad
158 pyscad module owriteten(first, second, third, fourth, fifth, sixth,
        seventh, eighth, ninth, tenth) {
159 pyscad     writeln(first, second, third, fourth, fifth, sixth, seventh,
```

```
                eighth, ninth, tenth);
160 pyscad }
161 pyscad
162 pyscad module owriteeleven(first, second, third, fourth, fifth, sixth,
        seventh, eighth, ninth, tenth, eleventh) {
163 pyscad     writeln(first, second, third, fourth, fifth, sixth, seventh,
                eighth, ninth, tenth, eleventh);
164 pyscad }
165 pyscad
166 pyscad module owritetwelve(first, second, third, fourth, fifth, sixth,
        seventh, eighth, ninth, tenth, eleventh, twelfth) {
167 pyscad     writeln(first, second, third, fourth, fifth, sixth, seventh,
                eighth, ninth, tenth, eleventh, twelfth);
168 pyscad }
169 pyscad
170 pyscad module owritethirteen(first, second, third, fourth, fifth, sixth,
        seventh, eighth, ninth, tenth, eleventh, twelfth, thirteenth) {
171 pyscad     writeln(first, second, third, fourth, fifth, sixth, seventh,
                eighth, ninth, tenth, eleventh, twelfth, thirteenth);
172 pyscad }
```

dxfwrite
dxfpreamble

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

```
294 gcpscad module dxfwrite(tn,arg) {
295 gcpscad if (tn == large_ball_tool_no) {
296 gcpscad     dxfwritelgbl(arg);}
297 gcpscad if (tn == large_square_tool_no) {
298 gcpscad     dxfwritelgsq(arg);}
299 gcpscad if (tn == large_V_tool_no) {
300 gcpscad     dxfwritelgV(arg);}
301 gcpscad if (tn == small_ball_tool_no) {
302 gcpscad     dxfwritesmbl(arg);}
303 gcpscad if (tn == small_square_tool_no) {
304 gcpscad     dxfwritesmsq(arg);}
305 gcpscad if (tn == small_V_tool_no) {
306 gcpscad     dxfwritesmV(arg);}
307 gcpscad if (tn == DT_tool_no) {
308 gcpscad     dxfwriteDT(arg);}
309 gcpscad if (tn == KH_tool_no) {
310 gcpscad     dxfwriteKH(arg);}
311 gcpscad }
312 gcpscad
313 gcpscad module dxfpreamble(tn) {
314 gcpscad //    echo(str("dxfpreamble",small_square_tool_no));
315 gcpscad     dxfwrite(tn,"0");
316 gcpscad     dxfwrite(tn,"SECTION");
317 gcpscad     dxfwrite(tn,"2");
318 gcpscad     dxfwrite(tn,"ENTITIES");
319 gcpscad }
```

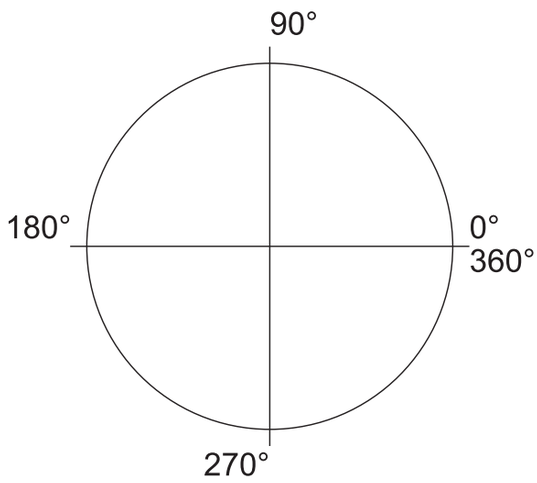
2.4.1.2 DXF Lines and Arcs Similarly, each each element which may be written to a DXF file will have a user module as well as an internal module which will be called by it so as to write to the file for the current tool.

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

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

There are three possible interactions for DXF elements and toolpaths:

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

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

```
321 gcpscad module dxfpl(tn,xbegin,ybegin,xend,yend) {
322 gcpscad     dxfwrite(tn,"0");
323 gcpscad     dxfwrite(tn,"LWPOLYLINE");
324 gcpscad     dxfwrite(tn,"90");
325 gcpscad     dxfwrite(tn,"2");
326 gcpscad     dxfwrite(tn,"70");
327 gcpscad     dxfwrite(tn,"0");
328 gcpscad     dxfwrite(tn,"43");
329 gcpscad     dxfwrite(tn,"0");
330 gcpscad     dxfwrite(tn,"10");
331 gcpscad     dxfwrite(tn,str(xbegin));
332 gcpscad     dxfwrite(tn,"20");
333 gcpscad     dxfwrite(tn,str(ybegin));
334 gcpscad     dxfwrite(tn,"10");
335 gcpscad     dxfwrite(tn,str(xend));
336 gcpscad     dxfwrite(tn,"20");
337 gcpscad     dxfwrite(tn,str(yend));
338 gcpscad }
339 gcpscad
340 gcpscad module dxfpolyline(xbegin,ybegin,xend,yend, tn) {
341 gcpscad if (generatedxf == true) {
342 gcpscad     dxfwriteone("0");
343 gcpscad     dxfwriteone("LWPOLYLINE");
344 gcpscad     dxfwriteone("90");
345 gcpscad     dxfwriteone("2");
346 gcpscad     dxfwriteone("70");
347 gcpscad     dxfwriteone("0");
348 gcpscad     dxfwriteone("43");
349 gcpscad     dxfwriteone("0");
350 gcpscad     dxfwriteone("10");
351 gcpscad     dxfwriteone(str(xbegin));
352 gcpscad     dxfwriteone("20");
353 gcpscad     dxfwriteone(str(ybegin));
354 gcpscad     dxfwriteone("10");
355 gcpscad     dxfwriteone(str(xend));
356 gcpscad     dxfwriteone("20");
357 gcpscad     dxfwriteone(str(yend));
358 gcpscad     dxfpl(tn,xbegin,ybegin,xend,yend);
359 gcpscad     }
360 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.

```
362 gcpscad module dxfa(tn,xcenter,ycenter,radius,anglebegin,endangle) {
363 gcpscad     dxfwrite(tn,"0");
364 gcpscad     dxfwrite(tn,"ARC");
365 gcpscad     dxfwrite(tn,"10");
366 gcpscad     dxfwrite(tn,str(xcenter));
367 gcpscad     dxfwrite(tn,"20");
368 gcpscad     dxfwrite(tn,str(ycenter));
369 gcpscad     dxfwrite(tn,"40");
370 gcpscad     dxfwrite(tn,str(radius));
371 gcpscad     dxfwrite(tn,"50");
372 gcpscad     dxfwrite(tn,str(anglebegin));
373 gcpscad     dxfwrite(tn,"51");
374 gcpscad     dxfwrite(tn,str(endangle));
375 gcpscad }
376 gcpscad
377 gcpscad module dxfarc(xcenter, ycenter, radius, anglebegin, endangle, tn) {
378 gcpscad if (generatedxf == true) {
379 gcpscad     dxfwriteone("0");
380 gcpscad     dxfwriteone("ARC");
381 gcpscad     dxfwriteone("10");
382 gcpscad     dxfwriteone(str(xcenter));
383 gcpscad     dxfwriteone("20");
384 gcpscad     dxfwriteone(str(ycenter));
385 gcpscad     dxfwriteone("40");
386 gcpscad     dxfwriteone(str(radius));
387 gcpscad     dxfwriteone("50");
388 gcpscad     dxfwriteone(str(anglebegin));
389 gcpscad     dxfwriteone("51");
390 gcpscad     dxfwriteone(str(endangle));
391 gcpscad     dxfa(tn,xcenter,ycenter,radius,anglebegin,endangle);
392 gcpscad     }
393 gcpscad }
```

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

```
395 gcpscad module dxfbpl(tn,bx,by) {
396 gcpscad     dxfwrite(tn,"0");
397 gcpscad     dxfwrite(tn,"POLYLINE");
398 gcpscad     dxfwrite(tn,"8");
399 gcpscad     dxfwrite(tn,"default");
400 gcpscad     dxfwrite(tn,"66");
401 gcpscad     dxfwrite(tn,"1");
402 gcpscad     dxfwrite(tn,"70");
403 gcpscad     dxfwrite(tn,"0");
404 gcpscad     dxfwrite(tn,"0");
405 gcpscad     dxfwrite(tn,"VERTEX");
406 gcpscad     dxfwrite(tn,"8");
407 gcpscad     dxfwrite(tn,"default");
408 gcpscad     dxfwrite(tn,"70");
409 gcpscad     dxfwrite(tn,"32");
410 gcpscad     dxfwrite(tn,"10");
411 gcpscad     dxfwrite(tn,str(bx));
412 gcpscad     dxfwrite(tn,"20");
413 gcpscad     dxfwrite(tn,str(by));
414 gcpscad }
415 gcpscad
416 gcpscad module beginpolyline(bx,by,bz) {
417 gcpscad if (generatedxf == true) {
418 gcpscad     dxfwriteone("0");
419 gcpscad     dxfwriteone("POLYLINE");
420 gcpscad     dxfwriteone("8");
421 gcpscad     dxfwriteone("default");
422 gcpscad     dxfwriteone("66");
423 gcpscad     dxfwriteone("1");
424 gcpscad     dxfwriteone("70");
425 gcpscad     dxfwriteone("0");
426 gcpscad     dxfwriteone("0");
427 gcpscad     dxfwriteone("VERTEX");
428 gcpscad     dxfwriteone("8");
429 gcpscad     dxfwriteone("default");
430 gcpscad     dxfwriteone("70");
431 gcpscad     dxfwriteone("32");
432 gcpscad     dxfwriteone("10");
```

```
433 gcpscad      dxfwriteone(str(bx));
434 gcpscad      dxfwriteone("20");
435 gcpscad      dxfwriteone(str(by));
436 gcpscad      dxfbpl(current_tool(),bx,by);}
437 gcpscad }
438 gcpscad
439 gcpscad module dxfapl(tn,bx,by) {
440 gcpscad      dxfwriteone("0");
441 gcpscad      dxfwrite(tn,"VERTEX");
442 gcpscad      dxfwrite(tn,"8");
443 gcpscad      dxfwrite(tn,"default");
444 gcpscad      dxfwrite(tn,"70");
445 gcpscad      dxfwrite(tn,"32");
446 gcpscad      dxfwrite(tn,"10");
447 gcpscad      dxfwrite(tn,str(bx));
448 gcpscad      dxfwrite(tn,"20");
449 gcpscad      dxfwrite(tn,str(by));
450 gcpscad }
451 gcpscad
452 gcpscad module addpolyline(bx,by,bz) {
453 gcpscad if (generatedxf == true) {
454 gcpscad //    dxfwrite(tn,"0");
455 gcpscad      dxfwriteone("VERTEX");
456 gcpscad      dxfwriteone("8");
457 gcpscad      dxfwriteone("default");
458 gcpscad      dxfwriteone("70");
459 gcpscad      dxfwriteone("32");
460 gcpscad      dxfwriteone("10");
461 gcpscad      dxfwriteone(str(bx));
462 gcpscad      dxfwriteone("20");
463 gcpscad      dxfwriteone(str(by));
464 gcpscad      dxfapl(current_tool(),bx,by);
465 gcpscad      }
466 gcpscad }
467 gcpscad
468 gcpscad module dxfcpl(tn) {
469 gcpscad      dxfwrite(tn,"0");
470 gcpscad      dxfwrite(tn,"SEQEND");
471 gcpscad }
472 gcpscad
473 gcpscad module closepolyline() {
474 gcpscad   if (generatedxf == true) {
475 gcpscad      dxfwriteone("0");
476 gcpscad      dxfwriteone("SEQEND");
477 gcpscad      dxfcpl(current_tool());
478 gcpscad   }
479 gcpscad }
480 gcpscad
481 gcpscad module writecomment(comment) {
482 gcpscad   if (generategcode == true) {
483 gcpscad      owritecomment(comment);
484 gcpscad   }
485 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 `closedxffile` information, depending on the file format.

```
208 gcpy def pclosegcodefile():
209 gcpy     f.close()
210 gcpy
211 gcpy def pclosedxffile():
212 gcpy     dxf.close()
213 gcpy
214 gcpy def pclosedxflgblfile():
215 gcpy     dxflgbl.close()
216 gcpy
217 gcpy def pclosedxflgsqfile():
218 gcpy     dxflgsq.close()
219 gcpy
220 gcpy def pclosedxflgVfile():
221 gcpy     dxflgV.close()
222 gcpy
223 gcpy def pclosedxfsmblfile():
224 gcpy     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

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

```
487 gcpscad module closegcodefile() {
488 gcpscad   if (generategcode == true) {
489 gcpscad     owriteone("M05");
490 gcpscad     owriteone("M02");
491 gcpscad     oclosegcodefile();
492 gcpscad   }
493 gcpscad }
494 gcpscad
495 gcpscad module dxfpostamble(arg) {
496 gcpscad     dxfwrite(arg,"0");
497 gcpscad     dxfwrite(arg,"ENDSEC");
498 gcpscad     dxfwrite(arg,"0");
499 gcpscad     dxfwrite(arg,"EOF");
500 gcpscad }
501 gcpscad
502 gcpscad module closedxffile() {
503 gcpscad   if (generatedxf == true) {
504 gcpscad     dxfwriteone("0");
505 gcpscad     dxfwriteone("ENDSEC");
506 gcpscad     dxfwriteone("0");
```

```
507 gcpscad     dxfwriteone("EOF");
508 gcpscad     oclosedxffile();
509 gcpscad     echo("CLOSING");
510 gcpscad     if (large_ball_tool_no >  0) {    dxfpostamble(
                    large_ball_tool_no);
511 gcpscad       oclosedxflgblfile();
512 gcpscad     }
513 gcpscad     if (large_square_tool_no >  0) {    dxfpostamble(
                    large_square_tool_no);
514 gcpscad       oclosedxflgsqfile();
515 gcpscad     }
516 gcpscad     if (large_V_tool_no >  0) {    dxfpostamble(large_V_tool_no);
517 gcpscad       oclosedxflgVfile();
518 gcpscad     }
519 gcpscad     if (small_ball_tool_no >  0) {    dxfpostamble(
                    small_ball_tool_no);
520 gcpscad       oclosedxfsmblfile();
521 gcpscad     }
522 gcpscad     if (small_square_tool_no >  0) {    dxfpostamble(
                    small_square_tool_no);
523 gcpscad       oclosedxfsmsqfile();
524 gcpscad     }
525 gcpscad     if (small_V_tool_no >  0) {    dxfpostamble(small_V_tool_no);
526 gcpscad       oclosedxfsmVfile();
527 gcpscad     }
528 gcpscad     if (DT_tool_no >  0) {    dxfpostamble(DT_tool_no);
529 gcpscad       oclosedxfDTfile();
530 gcpscad     }
531 gcpscad     if (KH_tool_no >  0) {    dxfpostamble(KH_tool_no);
532 gcpscad       oclosedxfKHfile();
533 gcpscad     }
534 gcpscad   }
535 gcpscad }
```

2.5 Movement and Cutting

otm
ocut
orapid
rapid
rapidbx

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

```
537 gcpscad module otm(ex, ey, ez, r,g,b) {
538 gcpscad color([r,g,b]) hull(){
539 gcpscad     translate([xpos(), ypos(), zpos()]){
540 gcpscad       select_tool(current_tool());
541 gcpscad     }
542 gcpscad     translate([ex, ey, ez]){
543 gcpscad       select_tool(current_tool());
544 gcpscad     }
545 gcpscad   }
546 gcpscad oset(ex, ey, ez);
547 gcpscad }
548 gcpscad
549 gcpscad module ocut(ex, ey, ez) {
550 gcpscad   //color([0.2,1,0.2]) hull(){
551 gcpscad   otm(ex, ey, ez, 0.2,1,0.2);
552 gcpscad }
553 gcpscad
554 gcpscad module orapid(ex, ey, ez) {
555 gcpscad   //color([0.93,0,0]) hull(){
556 gcpscad   otm(ex, ey, ez, 0.93,0,0);
557 gcpscad }
558 gcpscad
559 gcpscad module rapidbx(bx, by, bz, ex, ey, ez) {
560 gcpscad   //    writeln("G0 X",bx," Y", by, "Z", bz);
561 gcpscad   if (generategcode == true) {
562 gcpscad     writecomment("rapid");
563 gcpscad     owritesix("G0 X",str(ex)," Y", str(ey), " Z", str(ez));
564 gcpscad   }
565 gcpscad     orapid(ex, ey, ez);
566 gcpscad }
567 gcpscad
568 gcpscad module rapid(ex, ey, ez) {
569 gcpscad   //    writeln("G0 X",bx," Y", by, "Z", bz);
570 gcpscad   if (generategcode == true) {
571 gcpscad       writecomment("rapid");
```

```
572 gcpscad       owritesix("G0 X",str(ex)," Y", str(ey), " Z", str(ez));
573 gcpscad   }
574 gcpscad   orapid(ex, ey, ez);
575 gcpscad }
576 gcpscad
577 gcpscad module movetosafez() {
578 gcpscad   //this should be move to retract height
579 gcpscad   if (generategcode == true) {
580 gcpscad       writecomment("Move to safe Z to avoid workholding");
581 gcpscad       owriteone("G53G0Z-5.000");
582 gcpscad   }
583 gcpscad   orapid(getxpos(), getypos(), retractheight+55);
584 gcpscad }
585 gcpscad
586 gcpscad module begintoolpath(bx,by,bz) {
587 gcpscad   if (generategcode == true) {
588 gcpscad     writecomment("PREPOSITION FOR RAPID PLUNGE");
589 gcpscad     owritefour("G0X", str(bx), "Y",str(by));
590 gcpscad     owritetwo("Z", str(bz));
591 gcpscad   }
592 gcpscad   orapid(bx,by,bz);
593 gcpscad }
594 gcpscad
595 gcpscad module movetosafeheight() {
596 gcpscad   //this should be move to machine position
597 gcpscad   if (generategcode == true) {
598 gcpscad   //    writecomment("PREPOSITION FOR RAPID PLUNGE");Z25.650
599 gcpscad   //G1Z24.663F381.0 ,"F",str(plunge)
600 gcpscad     if (zeroheight == "Top") {
601 gcpscad       owritetwo("Z",str(retractheight));
602 gcpscad     }
603 gcpscad   }
604 gcpscad     orapid(getxpos(), getypos(), retractheight+55);
605 gcpscad }
606 gcpscad
607 gcpscad module cutoneaxis_setfeed(axis,depth,feed) {
608 gcpscad   if (generategcode == true) {
609 gcpscad   //    writecomment("PREPOSITION FOR RAPID PLUNGE");Z25.650
610 gcpscad   //G1Z24.663F381.0 ,"F",str(plunge) G1Z7.612F381.0
611 gcpscad     if (zeroheight == "Top") {
612 gcpscad       owritefive("G1",axis,str(depth),"F",str(feed));
613 gcpscad     }
614 gcpscad   }
615 gcpscad   if (axis == "X") {setxpos(depth);
616 gcpscad     ocut(depth, getypos(), getzpos());}
617 gcpscad     if (axis == "Y") {setypos(depth);
618 gcpscad       ocut(getxpos(), depth, getzpos());
619 gcpscad     }
620 gcpscad       if (axis == "Z") {setzpos(depth);
621 gcpscad         ocut(getxpos(), getypos(), depth);
622 gcpscad       }
623 gcpscad }
624 gcpscad
625 gcpscad module cut(ex, ey, ez) {
626 gcpscad   //    writeln("G0 X",bx," Y", by, "Z", bz);
627 gcpscad   if (generategcode == true) {
628 gcpscad      owritesix("G1 X",str(ex)," Y", str(ey), " Z", str(ez));
629 gcpscad   }
630 gcpscad   //if (generatesvg == true) {
631 gcpscad   //    owritesix("G1 X",str(ex)," Y", str(ey), " Z", str(ez));
632 gcpscad   //    orapid(getxpos(), getypos(), retractheight+5);
633 gcpscad   //    writesvgline(getxpos(),getypos(),ex,ey);
634 gcpscad   //}
635 gcpscad   ocut(ex, ey, ez);
636 gcpscad }
637 gcpscad
638 gcpscad module cutwithfeed(ex, ey, ez, feed) {
639 gcpscad   //    writeln("G0 X",bx," Y", by, "Z", bz);
640 gcpscad   if (generategcode == true) {
641 gcpscad   //    writecomment("rapid");
642 gcpscad     owriteeight("G1 X",str(ex)," Y", str(ey), " Z", str(ez),"F",str
                (feed));
643 gcpscad   }
644 gcpscad   ocut(ex, ey, ez);
645 gcpscad }
646 gcpscad
647 gcpscad module endtoolpath() {
648 gcpscad   if (generategcode == true) {
```

```
649 gcpscad    //Z31.750
650 gcpscad    //     owriteone("G53G0Z-5.000");
651 gcpscad      owritetwo("Z",str(retractheight));
652 gcpscad    }
653 gcpscad    orapid(getxpos(),getypos(),retractheight);
654 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 2.3.1) which will need to be included in the projects which will make use of said features until such time as they are added into the main `gcodepreview.scad` file.

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

In the TUG presentation/paper: http://tug.org/TUGboat/tb40-2/tb125adams-3d.pdf a list of 2D shapes was put forward — which of these will need to be created, or if some more general solution will be put forward is uncertain. For the time being, shapes will be implemented on an as-needed basis, as modified by the interaction with the requirements of toolpaths.

The program Carbide Create has toolpath types and options which are as follows:

- Contour — No Offset — the default, this is already supported in the existing code
- Contour — Outside Offset
- Contour — Inside Offset
- Pocket — such toolpaths/geometry should include the rounding of the tool at the corners, c.f., cutrectangledxf
- Drill — note that this is implemented as the plunging of a tool centered on a circle and normally that circle is the same diameter as the tool which is used.
- Keyhole — also beginning from a circle, a nice feature for this would be to include/model the areas which should be cleared for the sake of reducing wear on the tool and ensuring chip clearance

Some further considerations:

- relationship of geometry to toolpath — arguably there should be an option for each toolpath (we will use Carbide Create as a reference implementation) which is to be supported. Note that there are several possibilities: modeling the tool movement, describing the outline which the tool will cut, modeling a reference shape for the toolpath
- tool geometry — it should be possible to include support for specialty tooling such as dovetail cutters and to get an accurate 3D model, esp. for tooling which undercuts since they cannot be modeled in Carbide Create.
- feeds and speeds — if outputting G-code it would be nice to be able to import feeds and speeds from external files such as the .csv files used for user tool libraries in Carbide Create
- 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

- 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) — cutrectangledxf, cutoutrectangledxf, rectangleoutlinedxf
  - parallelogram
  - rhombus
  - trapezoid/trapezium
  - kite
  - ring/annulus segment (straight line, concave arc, straight line, convex arc)
  - astroid (four concave arcs)
  - salinon (four semicircles)
  - three straight lines and one concave arc

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

- two straight edges, one concave arcoddly, an asymmetric form (hyperbolic sector) has a name, but not the symmetrical—while the colloquial/prosaic arrowhead was considered, it was rejected as being better applied to the shape below. (Its also the shape used for the spaceship in the game Asteroids (or Hyperspace), but that is potentially confusing with astroid.) At the conference, Dr. Knuth suggested dart as a suitable term.
- two convex, one concave arcwith the above named, the term arrowhead is freed up to use as the name for this shape.
- three straight lines and one concave arc.

The first in particular is sorely needed for this project (its the result of inscribing a circle in a square or other regular geometric shape). Do these shapes have names in any other languages which might be used instead?

- cutarcSECC
- cutarcNECW
- cutarcNECC
- cutcircleCW — while it wont matter for generating a DXF, when G-code is implemented direction of cut will be a consideration for that
- cutcircleCCdxf

It will be necessary to have two separate representations of arcs — the DXF may be easily and directly supported with a single command, but representing the matching tool movement in OpenSCAD will require a series of short line movements which approximate the arc. At this time, the current version of Carbide Create only imports circles in DXF as curves, any other example is converted into polylines — unfortunately, the implementation of this is not such as would allow directly matching that representation. A work-around to import a DXF as curves is to convert the arc into a reasonable number of line segments so as to approximate the arc.

Note that there are the following representations/interfaces for representing an arc:

- G-code — G2 (clockwise) and G3 (counter-clockwise) arcs may be specified, and since the endpoint is the positional requirement, it is most likely best to use the offset to the center (I and J), rather than the radius parameter (K) `G2/3 ...`
- DXF — `dxfarc(xcenter, ycenter, radius, anglebegin, endangle, tn)`
- approximation of arc using lines (OpenSCAD) — note that this may also be used in DXF so as to sidestep the question of how many line segments there would be for a given arc representation

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

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

- `tn`
- `ex`
- `ey`
- `ez` — allowing a different Z position will make possible threading and similar helical toolpaths
- `xcenter` — the center position will be specified as an absolute position which will require calculating the offset when it is used for G-code's `IJ`, for which `xctr/yctr` are suggested
- `ycenter`
- `radius` — while this could be calculated, passing it in as a parameter is both convenient and acts as a check on the other parameters

`arcloop` `narcloop`

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):

```
656 gcpscad module arcloop(barc,earc, xcenter, ycenter, radius) {
657 gcpscad   for (i = [barc : abs(1) : earc]) {
658 gcpscad         cut(xcenter + radius * cos(i),
659 gcpscad         ycenter + radius * sin(i),
660 gcpscad         getzpos()-(gettzpos())
661 gcpscad         );
662 gcpscad     setxpos(xcenter + radius * cos(i));
663 gcpscad     setypos(ycenter + radius * sin(i));
664 gcpscad   }
665 gcpscad }
666 gcpscad
667 gcpscad module narcloop(barc,earc, xcenter, ycenter, radius) {
668 gcpscad   for (i = [barc : -1 : earc]) {
669 gcpscad         cut(xcenter + radius * cos(i),
670 gcpscad         ycenter + radius * sin(i),
671 gcpscad         getzpos()-(gettzpos())
672 gcpscad         );
673 gcpscad     setxpos(xcenter + radius * cos(i));
```

```
674 gcpscad     setypos(ycenter + radius * sin(i));
675 gcpscad   }
676 gcpscad }
```

The various textual versions are quite obvious:

```
678 gcpscad module cutarcNECCdxf(ex, ey, ez, xcenter, ycenter, radius, tn) {
679 gcpscad   dxfarc(xcenter,ycenter,radius,0,90, tn);
680 gcpscad   settzpos((getzpos()-ez)/90);
681 gcpscad     arcloop(1,90, xcenter, ycenter, radius);
682 gcpscad }
683 gcpscad
684 gcpscad module cutarcNWCCdxf(ex, ey, ez, xcenter, ycenter, radius, tn) {
685 gcpscad   dxfarc(xcenter,ycenter,radius,90,180, tn);
686 gcpscad   settzpos((getzpos()-ez)/90);
687 gcpscad     arcloop(91,180, xcenter, ycenter, radius);
688 gcpscad }
689 gcpscad
690 gcpscad module cutarcSWCCdxf(ex, ey, ez, xcenter, ycenter, radius, tn) {
691 gcpscad   dxfarc(xcenter,ycenter,radius,180,270, tn);
692 gcpscad   settzpos((getzpos()-ez)/90);
693 gcpscad     arcloop(181,270, xcenter, ycenter, radius);
694 gcpscad }
695 gcpscad
696 gcpscad module cutarcSECCdxf(ex, ey, ez, xcenter, ycenter, radius, tn) {
697 gcpscad   dxfarc(xcenter,ycenter,radius,270,360, tn);
698 gcpscad   settzpos((getzpos()-ez)/90);
699 gcpscad     arcloop(271,360, xcenter, ycenter, radius);
700 gcpscad }
701 gcpscad
702 gcpscad module cutarcNECWdxf(ex, ey, ez, xcenter, ycenter, radius, tn) {
703 gcpscad   dxfarc(xcenter,ycenter,radius,0,90, tn);
704 gcpscad   settzpos((getzpos()-ez)/90);
705 gcpscad     narcloop(89,0, xcenter, ycenter, radius);
706 gcpscad }
707 gcpscad
708 gcpscad module cutarcSECWdxf(ex, ey, ez, xcenter, ycenter, radius, tn) {
709 gcpscad   dxfarc(xcenter,ycenter,radius,270,360, tn);
710 gcpscad   settzpos((getzpos()-ez)/90);
711 gcpscad     narcloop(359,270, xcenter, ycenter, radius);
712 gcpscad }
713 gcpscad
714 gcpscad module cutarcSWCWdxf(ex, ey, ez, xcenter, ycenter, radius, tn) {
715 gcpscad   dxfarc(xcenter,ycenter,radius,180,270, tn);
716 gcpscad   settzpos((getzpos()-ez)/90);
717 gcpscad     narcloop(269,180, xcenter, ycenter, radius);
718 gcpscad }
719 gcpscad
720 gcpscad module cutarcNWCWdxf(ex, ey, ez, xcenter, ycenter, radius, tn) {
721 gcpscad   dxfarc(xcenter,ycenter,radius,90,180, tn);
722 gcpscad   settzpos((getzpos()-ez)/90);
723 gcpscad     narcloop(179,90, xcenter, ycenter, radius);
724 gcpscad }
```

3.2 Keyhole toolpath and undercut tooling

cutkeyhole toolpath

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

Tooling for such toolpaths is defined at paragraph 2.3.1.2

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

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

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

```
726 gcpscad module cutkeyhole_toolpath(kh_start_depth, kh_max_depth,
        kht_direction, kh_distance, kh_tool_no) {
727 gcpscad if (kht_direction == "N") {
728 gcpscad   cutKH_toolpath_degrees(kh_start_depth, kh_max_depth, 90,
            kh_distance, kh_tool_no);
729 gcpscad    } else if (kht_direction == "S") {
730 gcpscad   cutKH_toolpath_degrees(kh_start_depth, kh_max_depth, 270,
            kh_distance, kh_tool_no);
731 gcpscad    } else if (kht_direction == "E") {
732 gcpscad   cutKH_toolpath_degrees(kh_start_depth, kh_max_depth, 0,
            kh_distance, kh_tool_no);
733 gcpscad    } else if (kht_direction == "W") {
734 gcpscad   cutKH_toolpath_degrees(kh_start_depth, kh_max_depth, 180,
            kh_distance, kh_tool_no);
735 gcpscad    }
736 gcpscad }
```

cutKH toolpath degrees

The original version of the command, `cutKH toolpath degrees` retains an interface which allows calling it for arbitrary beginning and ending points of an arc. Note that code is still present for the partial calculation of one quadrant (for the case of all nodes within the quadrant).

The first task is to place a circle at the origin which is invariant of angle:

```
738 gcpscad module cutKH_toolpath_degrees(kh_start_depth, kh_max_depth,
        kh_angle, kh_distance, kh_tool_no) {
739 gcpscad dxfarc(getxpos(),getypos(),tool_diameter(KH_tool_no, (kh_max_depth
        +4.36))/2,0,90, KH_tool_no);
740 gcpscad dxfarc(getxpos(),getypos(),tool_diameter(KH_tool_no, (kh_max_depth
        +4.36))/2,90,180, KH_tool_no);
741 gcpscad dxfarc(getxpos(),getypos(),tool_diameter(KH_tool_no, (kh_max_depth
        +4.36))/2,180,270, KH_tool_no);
742 gcpscad dxfarc(getxpos(),getypos(),tool_diameter(KH_tool_no, (kh_max_depth
        +4.36))/2,270,360, KH_tool_no);
```

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

```
744 gcpscad   if (kh_angle == 0) {
745 gcpscad dxfarc(getxpos(),getypos(),tool_diameter(KH_tool_no, (kh_max_depth)
        )/2,180,270, KH_tool_no);
746 gcpscad dxfarc(getxpos(),getypos(),tool_diameter(KH_tool_no, (kh_max_depth)
        )/2,90,180, KH_tool_no);
747 gcpscad dxfarc(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, KH_tool_no);
748 gcpscad dxfarc(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)), KH_tool_no);
749 gcpscad dxfarc(getxpos()+kh_distance,getypos(),tool_diameter(KH_tool_no, (
        kh_max_depth+4.36))/2,0,90, KH_tool_no);
750 gcpscad dxfarc(getxpos()+kh_distance,getypos(),tool_diameter(KH_tool_no, (
        kh_max_depth+4.36))/2,270,360, KH_tool_no);
751 gcpscad dxfpolyline(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_distance, getypos()+tool_diameter(KH_tool_no, (
        kh_max_depth+4.36))/2, KH_tool_no);
752 gcpscad dxfpolyline(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_distance, getypos()-tool_diameter(KH_tool_no, (
        kh_max_depth+4.36))/2, KH_tool_no);
753 gcpscad dxfpolyline(getxpos(),getypos(),getxpos()+kh_distance,getypos(),
        KH_tool_no);
754 gcpscad cutwithfeed(getxpos()+kh_distance,getypos(),-kh_max_depth,feed);
755 gcpscad setxpos(getxpos()-kh_distance);
756 gcpscad   } else if (kh_angle > 0 && kh_angle < 90) {
757 gcpscad echo(kh_angle);
758 gcpscad   dxfarc(getxpos(),getypos(),tool_diameter(KH_tool_no, (
            kh_max_depth))/2,90+kh_angle,180+kh_angle, KH_tool_no);
759 gcpscad   dxfarc(getxpos(),getypos(),tool_diameter(KH_tool_no, (
            kh_max_depth))/2,180+kh_angle,270+kh_angle, KH_tool_no);
760 gcpscad dxfarc(getxpos(),getypos(),tool_diameter(KH_tool_no, (kh_max_depth)
        )/2,kh_angle+asin((tool_diameter(KH_tool_no, (kh_max_depth+4.36)
        )/2)/(tool_diameter(KH_tool_no, (kh_max_depth))/2)),90+kh_angle,
```

```
             KH_tool_no);
761 gcpscad dxfarc(getxpos(),getypos(),tool_diameter(KH_tool_no, (kh_max_depth)
            )/2,270+kh_angle,360+kh_angle-asin((tool_diameter(KH_tool_no, (
            kh_max_depth+4.36))/2)/(tool_diameter(KH_tool_no, (kh_max_depth)
            )/2)), KH_tool_no);
762 gcpscad dxfarc(getxpos()+(kh_distance*cos(kh_angle)),
763 gcpscad   getypos()+(kh_distance*sin(kh_angle)),tool_diameter(KH_tool_no, (
              kh_max_depth+4.36))/2,0+kh_angle,90+kh_angle, KH_tool_no);
764 gcpscad dxfarc(getxpos()+(kh_distance*cos(kh_angle)),getypos()+(kh_distance
            *sin(kh_angle)),tool_diameter(KH_tool_no, (kh_max_depth+4.36))
            /2,270+kh_angle,360+kh_angle, KH_tool_no);
765 gcpscad dxfpolyline( getxpos()+tool_diameter(KH_tool_no, (kh_max_depth))/2*
            cos(kh_angle+asin((tool_diameter(KH_tool_no, (kh_max_depth+4.36)
            )/2)/(tool_diameter(KH_tool_no, (kh_max_depth))/2))),
766 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))),
767 gcpscad  getxpos()+(kh_distance*cos(kh_angle))-((tool_diameter(KH_tool_no,
             (kh_max_depth+4.36))/2)*sin(kh_angle)),
768 gcpscad  getypos()+(kh_distance*sin(kh_angle))+((tool_diameter(KH_tool_no,
             (kh_max_depth+4.36))/2)*cos(kh_angle)), KH_tool_no);
769 gcpscad echo("a",tool_diameter(KH_tool_no, (kh_max_depth+4.36))/2);
770 gcpscad echo("c",tool_diameter(KH_tool_no, (kh_max_depth))/2);
771 gcpscad echo("Aangle",asin((tool_diameter(KH_tool_no, (kh_max_depth+4.36))
            /2)/(tool_diameter(KH_tool_no, (kh_max_depth))/2)));
772 gcpscad echo(kh_angle);
773 gcpscad  cutwithfeed(getxpos()+(kh_distance*cos(kh_angle)),getypos()+(
             kh_distance*sin(kh_angle)),-kh_max_depth,feed);
774 gcpscad  setxpos(getxpos()-(kh_distance*cos(kh_angle)));
775 gcpscad  setypos(getypos()-(kh_distance*sin(kh_angle)));
776 gcpscad   } else if (kh_angle == 90) {
777 gcpscad dxfarc(getxpos(),getypos(),tool_diameter(KH_tool_no, (kh_max_depth)
            )/2,180,270, KH_tool_no);
778 gcpscad dxfarc(getxpos(),getypos(),tool_diameter(KH_tool_no, (kh_max_depth)
            )/2,270,360, KH_tool_no);
779 gcpscad dxfarc(getxpos(),getypos(),tool_diameter(KH_tool_no, (kh_max_depth)
            )/2,0,90-asin(
780 gcpscad     (tool_diameter(KH_tool_no, (kh_max_depth+4.36))/2)/(
                 tool_diameter(KH_tool_no, (kh_max_depth))/2)), KH_tool_no);
781 gcpscad dxfarc(getxpos(),getypos(),tool_diameter(KH_tool_no, (kh_max_depth)
            )/2,90+asin(
782 gcpscad     (tool_diameter(KH_tool_no, (kh_max_depth+4.36))/2)/(
                 tool_diameter(KH_tool_no, (kh_max_depth))/2)),180,
                 KH_tool_no);
783 gcpscad  dxfpolyline(getxpos(),getypos(),getxpos(),getypos()+kh_distance);
784 gcpscad dxfarc(getxpos(),getypos()+kh_distance,tool_diameter(KH_tool_no, (
            kh_max_depth+4.36))/2,0,90, KH_tool_no);
785 gcpscad dxfarc(getxpos(),getypos()+kh_distance,tool_diameter(KH_tool_no, (
            kh_max_depth+4.36))/2,90,180, KH_tool_no);
786 gcpscad  dxfpolyline(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_distance, KH_tool_no);
787 gcpscad  dxfpolyline(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_distance, KH_tool_no);
788 gcpscad  cutwithfeed(getxpos(),getypos()+kh_distance,-kh_max_depth,feed);
789 gcpscad  setypos(getypos()-kh_distance);
790 gcpscad   } else if (kh_angle == 180) {
791 gcpscad dxfarc(getxpos(),getypos(),tool_diameter(KH_tool_no, (kh_max_depth)
            )/2,0,90, KH_tool_no);
792 gcpscad dxfarc(getxpos(),getypos(),tool_diameter(KH_tool_no, (kh_max_depth)
            )/2,270,360, KH_tool_no);
793 gcpscad dxfarc(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)), KH_tool_no);
794 gcpscad dxfarc(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, KH_tool_no)
            ;
795 gcpscad dxfarc(getxpos()-kh_distance,getypos(),tool_diameter(KH_tool_no, (
            kh_max_depth+4.36))/2,90,180, KH_tool_no);
796 gcpscad dxfarc(getxpos()-kh_distance,getypos(),tool_diameter(KH_tool_no, (
            kh_max_depth+4.36))/2,180,270, KH_tool_no);
797 gcpscad dxfpolyline(getxpos()-sqrt((tool_diameter(KH_tool_no, (kh_max_depth
```

```
            ))/2)^2-(tool_diameter(KH_tool_no, (kh_max_depth+4.36))/2)^2),
798 gcpscad  getypos()+tool_diameter(KH_tool_no, (kh_max_depth+4.36))/2,
799 gcpscad  getxpos()-kh_distance,
800 gcpscad  getypos()+tool_diameter(KH_tool_no, (kh_max_depth+4.36))/2,
              KH_tool_no);
801 gcpscad dxfpolyline( getxpos()-sqrt((tool_diameter(KH_tool_no, (
             kh_max_depth))/2)^2-(tool_diameter(KH_tool_no, (kh_max_depth
             +4.36))/2)^2),
802 gcpscad  getypos()-tool_diameter(KH_tool_no, (kh_max_depth+4.36))/2,
803 gcpscad  getxpos()-kh_distance,
804 gcpscad  getypos()-tool_diameter(KH_tool_no, (kh_max_depth+4.36))/2,
              KH_tool_no);
805 gcpscad  dxfpolyline(getxpos(),getypos(),getxpos()-kh_distance,getypos(),
              KH_tool_no);
806 gcpscad  cutwithfeed(getxpos()-kh_distance,getypos(),-kh_max_depth,feed);
807 gcpscad  setxpos(getxpos()+kh_distance);
808 gcpscad   } else if (kh_angle == 270) {
809 gcpscad dxfarc(getxpos(),getypos(),tool_diameter(KH_tool_no, (kh_max_depth)
             )/2,0,90, KH_tool_no);
810 gcpscad dxfarc(getxpos(),getypos(),tool_diameter(KH_tool_no, (kh_max_depth)
             )/2,90,180, KH_tool_no);
811 gcpscad dxfarc(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, KH_tool_no)
             ;
812 gcpscad dxfarc(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)), KH_tool_no)
             ;
813 gcpscad dxfarc(getxpos(),getypos()-kh_distance,tool_diameter(KH_tool_no, (
             kh_max_depth+4.36))/2,180,270, KH_tool_no);
814 gcpscad dxfarc(getxpos(),getypos()-kh_distance,tool_diameter(KH_tool_no, (
             kh_max_depth+4.36))/2,270,360, KH_tool_no);
815 gcpscad  dxfpolyline(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_distance, KH_tool_no);
816 gcpscad  dxfpolyline(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_distance, KH_tool_no);
817 gcpscad  dxfpolyline(getxpos(),getypos(),getxpos(),getypos()-kh_distance,
              KH_tool_no);
818 gcpscad  cutwithfeed(getxpos(),getypos()-kh_distance,-kh_max_depth,feed);
819 gcpscad  setypos(getypos()+kh_distance);
820 gcpscad   }
821 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`.

```
823 gcpscad module begincutdxf(rh, ex, ey, ez, fr) {
824 gcpscad   rapid(getxpos(),getypos(),rh);
```

```
825 gcpscad    cutwithfeed(ex,ey,ez,fr);
826 gcpscad }
```

```
828 gcpscad module continuecutdxf(ex, ey, ez, fr) {
829 gcpscad    cutwithfeed(ex,ey,ez,fr);
830 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:

```
832 gcpscad module cutrectangledxf(bx, by, bz, rwidth, rheight, rdepth, rtn)
        {//passes
833 gcpscad    movetosafez();
834 gcpscad    hull(){
835 gcpscad      //  for (i = [0 : abs(1) : passes]) {
836 gcpscad      //      rapid(bx+tool_radius(rtn)+i*(rwidth-tool_diameter(
                    current_tool()))/passes,bx+tool_radius(rtn),1);
837 gcpscad      //      cutwithfeed(bx+tool_radius(rtn)+i*(rwidth-tool_diameter
                    (current_tool()))/passes,by+tool_radius(rtn),bz-rdepth,feed)
                    ;
838 gcpscad      //      cutwithfeed(bx+tool_radius(rtn)+i*(rwidth-tool_diameter
                    (current_tool()))/passes,by+rheight-tool_radius(rtn),bz-
                    rdepth,feed);
839 gcpscad
840 gcpscad      cutwithfeed(bx+tool_radius(rtn),by+tool_radius(rtn),bz-rdepth,
                    feed);
841 gcpscad      cutwithfeed(bx+rwidth-tool_radius(rtn),by+tool_radius(rtn),bz-
                    rdepth,feed);
842 gcpscad      cutwithfeed(bx+rwidth-tool_radius(rtn),by+rheight-tool_radius(
                    rtn),bz-rdepth,feed);
843 gcpscad      cutwithfeed(bx+tool_radius(rtn),by+rheight-tool_radius(rtn),bz-
                    rdepth,feed);
844 gcpscad    }
845 gcpscad    //dxfarc(xcenter,ycenter,radius,anglebegin,endangle, tn)
846 gcpscad    dxfarc(bx+tool_radius(rtn),by+tool_radius(rtn),tool_radius(rtn)
                    ,180,270, rtn);
847 gcpscad    //dxfpolyline(xbegin,ybegin,xend,yend, tn)
848 gcpscad    dxfpolyline(bx,by+tool_radius(rtn),bx,by+rheight-tool_radius(rtn)
                    , rtn);
849 gcpscad    dxfarc(bx+tool_radius(rtn),by+rheight-tool_radius(rtn),
                    tool_radius(rtn),90,180, rtn);
850 gcpscad    dxfpolyline(bx+tool_radius(rtn),by+rheight,bx+rwidth-tool_radius(
                    rtn),by+rheight, rtn);
851 gcpscad    dxfarc(bx+rwidth-tool_radius(rtn),by+rheight-tool_radius(rtn),
                    tool_radius(rtn),0,90, rtn);
852 gcpscad    dxfpolyline(bx+rwidth,by+rheight-tool_radius(rtn),bx+rwidth,by+
                    tool_radius(rtn), rtn);
853 gcpscad    dxfarc(bx+rwidth-tool_radius(rtn),by+tool_radius(rtn),tool_radius
                    (rtn),270,360, rtn);
854 gcpscad    dxfpolyline(bx+rwidth-tool_radius(rtn),by,bx+tool_radius(rtn),by,
                     rtn);
855 gcpscad }
```

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

```
857 gcpscad module cutrectangleoutlinedxf(bx, by, bz, rwidth, rheight, rdepth,
        rtn) {//passes
858 gcpscad    movetosafez();
859 gcpscad    cutwithfeed(bx+tool_radius(rtn),by+tool_radius(rtn),bz-rdepth,
                feed);
860 gcpscad    cutwithfeed(bx+rwidth-tool_radius(rtn),by+tool_radius(rtn),bz-
```

```
                rdepth,feed);
861 gcpscad    cutwithfeed(bx+rwidth-tool_radius(rtn),by+rheight-tool_radius(rtn
                ),bz-rdepth,feed);
862 gcpscad    cutwithfeed(bx+tool_radius(rtn),by+rheight-tool_radius(rtn),bz-
                rdepth,feed);
863 gcpscad    dxfarc(bx+tool_radius(rtn),by+tool_radius(rtn),tool_radius(rtn)
                ,180,270, rtn);
864 gcpscad    dxfpolyline(bx,by+tool_radius(rtn),bx,by+rheight-tool_radius(rtn)
                , rtn);
865 gcpscad    dxfarc(bx+tool_radius(rtn),by+rheight-tool_radius(rtn),
                tool_radius(rtn),90,180, rtn);
866 gcpscad    dxfpolyline(bx+tool_radius(rtn),by+rheight,bx+rwidth-tool_radius(
                rtn),by+rheight, rtn);
867 gcpscad    dxfarc(bx+rwidth-tool_radius(rtn),by+rheight-tool_radius(rtn),
                tool_radius(rtn),0,90, rtn);
868 gcpscad    dxfpolyline(bx+rwidth,by+rheight-tool_radius(rtn),bx+rwidth,by+
                tool_radius(rtn), rtn);
869 gcpscad    dxfarc(bx+rwidth-tool_radius(rtn),by+tool_radius(rtn),tool_radius
                (rtn),270,360, rtn);
870 gcpscad    dxfpolyline(bx+rwidth-tool_radius(rtn),by,bx+tool_radius(rtn),by,
                 rtn);
871 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):

```
873 gcpscad module rectangleoutlinedxf(bx, by, bz, rwidth, rheight, rtn) {
874 gcpscad   dxfpolyline(bx,by,bx,by+rheight, rtn);
875 gcpscad   dxfpolyline(bx,by+rheight,bx+rwidth,by+rheight, rtn);
876 gcpscad   dxfpolyline(bx+rwidth,by+rheight,bx+rwidth,by, rtn);
877 gcpscad   dxfpolyline(bx+rwidth,by,bx,by, rtn);
878 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:

```
880 gcpscad module cutoutrectangledxf(bx, by, bz, rwidth, rheight, rdepth, rtn)
                {
881 gcpscad   movetosafez();
882 gcpscad   cutwithfeed(bx-tool_radius(rtn),by-tool_radius(rtn),bz-rdepth,
                feed);
883 gcpscad   cutwithfeed(bx+rwidth+tool_radius(rtn),by-tool_radius(rtn),bz-
                rdepth,feed);
884 gcpscad   cutwithfeed(bx+rwidth+tool_radius(rtn),by+rheight+tool_radius(rtn
                ),bz-rdepth,feed);
885 gcpscad   cutwithfeed(bx-tool_radius(rtn),by+rheight+tool_radius(rtn),bz-
                rdepth,feed);
886 gcpscad   cutwithfeed(bx-tool_radius(rtn),by-tool_radius(rtn),bz-rdepth,
                feed);
887 gcpscad   dxfpolyline(bx,by,bx,by+rheight, rtn);
888 gcpscad   dxfpolyline(bx,by+rheight,bx+rwidth,by+rheight, rtn);
889 gcpscad   dxfpolyline(bx+rwidth,by+rheight,bx+rwidth,by, rtn);
890 gcpscad   dxfpolyline(bx+rwidth,by,bx,by, rtn);
891 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
```

```
gcptmpl use <gcodepreview.py>;
gcptmpl use <pygcodepreview.scad>;
gcptmpl include <gcodepreview.scad>;
gcptmpl
gcptmpl $fa = 2;
gcptmpl $fs = 0.125;
gcptmpl
gcptmpl /* [Export] */
gcptmpl Base_filename = "export";
gcptmpl /* [Export] */
gcptmpl generatedxf = true;
gcptmpl /* [Export] */
gcptmpl generategcode = true;
gcptmpl ///* [Export] */
gcptmpl //generatesvg = false;
gcptmpl
gcptmpl /* [CAM] */
gcptmpl toolradius = 1.5875;
gcptmpl /* [CAM] */
gcptmpl large_ball_tool_no = 0; // [0:0,111:111,101:101,202:202]
gcptmpl /* [CAM] */
gcptmpl large_square_tool_no = 0; // [0:0,112:112,102:102,201:201]
gcptmpl /* [CAM] */
gcptmpl large_V_tool_no = 0; // [0:0,301:301,690:690]
gcptmpl /* [CAM] */
gcptmpl small_ball_tool_no = 0; // [0:0,121:121,111:111,101:101]
gcptmpl /* [CAM] */
gcptmpl small_square_tool_no = 102; // [0:0,122:122,112:112,102:102]
gcptmpl /* [CAM] */
gcptmpl small_V_tool_no = 0; // [0:0,390:390,301:301]
gcptmpl /* [CAM] */
gcptmpl KH_tool_no = 0; // [0:0,375:375]
gcptmpl /* [CAM] */
gcptmpl DT_tool_no = 0; // [0:0,814:814]
gcptmpl
gcptmpl /* [Feeds and Speeds] */
gcptmpl plunge = 100;
gcptmpl /* [Feeds and Speeds] */
gcptmpl feed = 400;
gcptmpl /* [Feeds and Speeds] */
gcptmpl speed = 16000;
gcptmpl /* [Feeds and Speeds] */
gcptmpl square_ratio = 1.0; // [0.25:2]
gcptmpl /* [Feeds and Speeds] */
gcptmpl small_V_ratio = 0.75; // [0.25:2]
gcptmpl /* [Feeds and Speeds] */
gcptmpl large_V_ratio = 0.875; // [0.25:2]
gcptmpl
gcptmpl /* [Stock] */
gcptmpl stocklength = 219;
gcptmpl /* [Stock] */
gcptmpl stockwidth = 150;
gcptmpl /* [Stock] */
gcptmpl stockthickness = 8.35;
gcptmpl /* [Stock] */
gcptmpl zeroheight = "Top"; // [Top, Bottom]
gcptmpl /* [Stock] */
gcptmpl stockorigin = "Center"; // [Lower-Left, Center-Left, Top-Left,
          Center]
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);
```

```
79 gcptmpl beginpolyline(0,0,0.25);
80 gcptmpl
81 gcptmpl cutoneaxis_setfeed("Z",0,plunge*square_ratio);
82 gcptmpl
83 gcptmpl cutwithfeed(stocklength/2,stockwidth/2,-stockthickness,feed);
84 gcptmpl addpolyline(stocklength/2,stockwidth/2,-stockthickness);
85 gcptmpl
86 gcptmpl endtoolpath();
87 gcptmpl closepolyline();
88 gcptmpl }
89 gcptmpl
90 gcptmpl closegcodefile();
91 gcptmpl closedxffile();
```

Note that the line:

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

may be commented out — whether or no it is actually necessary will need to be decided upon.

4.1 G-code and modules and commands

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

Command/Module	G-code

`opengcodefile(...); setupstock(...)`	`(export.nc)`
	`(stockMin: -109.5, -75mm, -8.35mm)`
	`(stockMax:109.5mm, 75mm, 0.00mm)`
	`(STOCK/BLOCK, 219, 150, 8.35, 109.5, 75, 8.35)`
	`G90`
	`G21`

`movetosafez()`	`(Move to safe Z to avoid workholding)` `G53G0Z-5.000`

`toolchange(...);`	`(TOOL/MILL,3.17, 0.00, 0.00, 0.00)`
	`M6T102`
	`M03S16000`

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

`closegcodefile();`	`M05` `M02`

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

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

4.2 DXF

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

5 Future

5.1 Images

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

5.2 Generalized DXF creation

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

5.3 Import G-code

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

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

5.4 Bézier curves in 2 dimensions

Take a Bézier curve definition and approximate it as arcs and write them into a DXF?
https://pomax.github.io/bezierinfo/
c.f., https://linuxcnc.org/docs/html/gcode/g-code.html#gcode:g5

5.5 Bézier curves in 3 dimensions

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

- concise — a given Bézier curve should be represented by just the point coordinates, so two on-curve points, two off-curve points, each with a pair of coordinates
- For a given shape/region it will need to be possible to have a matching definition exactly match up with it so that one could piece together a larger more complex shape from smaller/simpler regions
- similarly it will be necessary for it to be possible to sub-divide a defined region — for example it should be possible if one had 4 adjacent regions, then the four quadrants at the intersection of the four regions could be used to construct a new region — is it possible to derive a new Bézier curve from half of two other curves?

For the three planes:

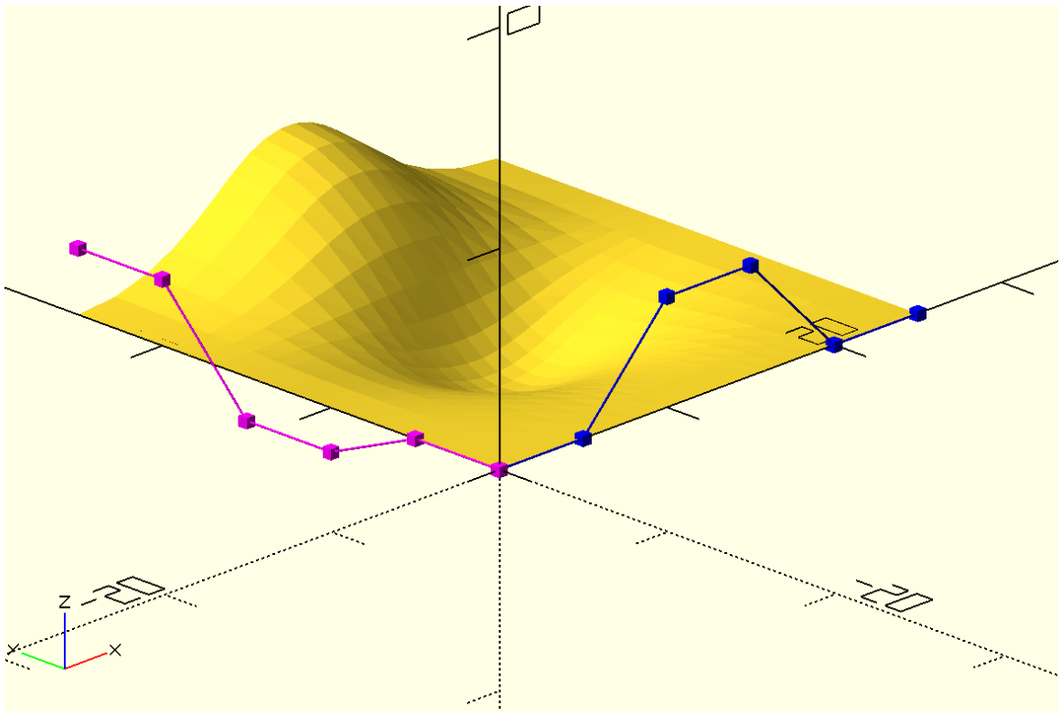
- XY
- XZ
- ZY

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

```
3 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

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

Index

Routines

Variables