Design into 3D: A system for customizable project designs

William Adams

Abstract

Design into 3D is a system for modeling parametric projects for manufacture using CNC machines. It documents using OpenSCAD to allow a user to instantly see a 3D rendering of the result of adjusting a parameter in the Customizer interface, saving parameters as JSON files which are then read into a LuaLaTeX file which creates a PDF as a cut list/setup sheet/assembly instructions and uses MetaPost to create SVG files which may be loaded into a CAM tool. A further possibility is using a tool such as TPL (Tool Path Language) to make files which are ready to cut.

1 iTEX

It has been almost ten years since Prof. Knuth made the earthshaking announcement of iTEX (see fig. 1; my thanks to Robin Laakso, executive director who kept track of her keepsake as I did not). For the folks who were not fortunate enough to be able to attend: `youtube.com/watch?v=eKaI78K_rgA` (from `tug.org/tug2010/program.html`).

The announcement posited a successor to TEX which would among other things, support 3D, and output to:

- lasercutters
- embroidering machines
- 3D printers
- plasma cutters

all of which are examples of Computer Numeric Control (CNC) machines. Presumably other machines such as mills and routers would also have been supported. While 3D printers have a straightforward mechanism for creating parts (load a 3D file into a "slicing" application), and laser and plasma cutters are limited to 2D (with the possibility of repeated passes for lasers), mills and routers afford the limitation of a 2.5D movement of the tool over and around the part, and the flexibility of using tooling with different shapes which will allow efficient cutting of surfaces with finishes not readily achieved with other tools. They also afford the possibility of loading stock larger than the working area and either cutting it incrementally (known as tiling) or cutting only a small portion of the stock (e.g., when cutting joinery into the end of a board).

Figure 1: iTEX keepsake

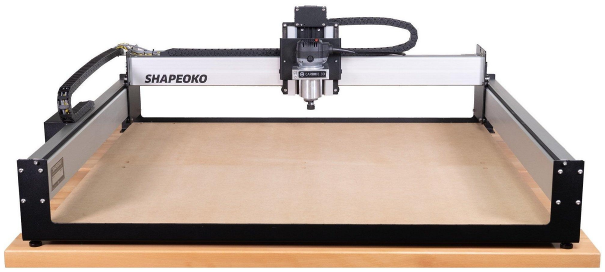


Figure 2: Shapeoko 3 (XXL)

2 CNC machines

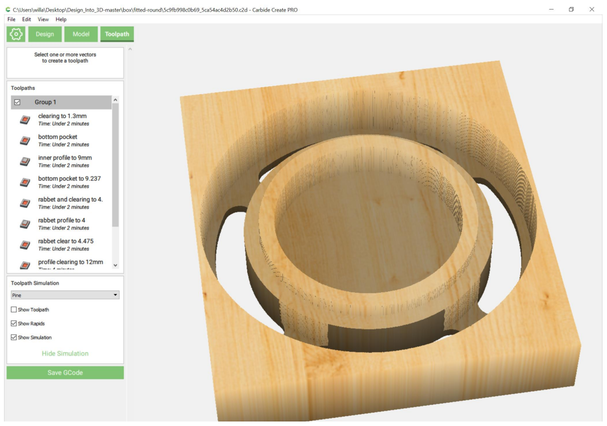
Since then, CNC machines have become far more affordable and accessible, mostly due to the open sourcing of the Enhanced Machine Controller,[1] and the development of Grbl which runs on the inexpensive Arduino,[2] with one early machine on its third iteration[3] (see fig. 2).

I happened to pick up a Shapeoko 1 (an open source hobbyist CNC machine based on Bart Dring's MakerSlide,[4] which uses the open source G-Code interpreter Grbl running on an Arduino) early on, and became involved in the project doing documentation and so forth, and now work for the company as off-site tech support.

3 CAD/CAM

CNC is driven by Computer Aided Design (CAD), and Computer Aided Manufacturing (CAM). Most applications thus far developed for this follow the same basic concept: Draw a design or shape, select elements of it and assign appropriate toolpaths to those elements. This works, but can be tedious and repetitive, especially when a design needs some

[1] `www.nist.gov/publications/use-open-source-distribution-machine-tool-controller`
[2] `bengler.no/grbl`
[3] `carbide3d.com/shapeoko`
[4] `www.kickstarter.com/projects/93832939/makerslide-open-source-linear-bearing-system`



C:\Users\willa\Desktop\Design_Into_3D-master\box\fitted-round\5c9fb998c0b69_5ca54ac4d2b50.c2d - Carbide Create PRO

Select one or more vectors to create a toolpath

Group 1
- clearing to 1.3mm — Time: Under 2 minutes
- bottom pocket — Time: Under 2 minutes
- inner profile to 9mm — Time: Under 2 minutes
- bottom pocket to 9.237 — Time: Under 2 minutes
- rabbet and clearing to 4. — Time: Under 2 minutes
- rabbet profile to 4 — Time: Under 2 minutes
- rabbet clear to 4.475 — Time: Under 2 minutes
- profile clearing to 12mm

Pine
- [ ] Show Toolpath
- [✓] Show Rapids
- [✓] Show Simulation

Figure 3: Carbide Create user interface

adjustment such as size or the inclusion or removal of a feature.

Even a simple, small, round box scrolls off the interface when enumerating all of its toolpath settings, making it tedious to transfer said settings to a different project, let alone set them up in the first place. (Fig. 3 shows Carbide Create,[5] a freely available CAD/CAM program).

Some 3D CAM tools do afford options for exporting settings and loading them into different projects, but then one is restricted to the toolpaths which a 3D CAM tool can create, and must work up a 3D model of the project in question. While the latter may not be much of a limitation, the former certainly is.

4 Tagging vs. parameters

TeX works from the idea of a manuscript, assigning to it macros/tagging/markup which then allow the text to be typeset. Moreover, (LA)TeX typically doesn't describe a document as fully as would be needed to make it into a finished object, omitting considerations such as signatures, binding method, and usually the design of a physical cover or dust jacket. Unfortunately, the CAD/CAM workflow doesn't allow for the sort of free-flowing narrative which even a rigorous scientific paper would allow. For example, it may be possible to define a potential project concisely:

- Project type: Box
- Shape: Round
- Lid style: Fitted
- Number of compartments: 1
- Box dimensions:
  - Diameter 50.8mm
  - Height 16.175mm

[5] carbide3d.com/carbidecreate

but there are no readily accessible tools for taking such specifications (or parameters) and directly and immediately creating the design in a format a computer can work with. Parametric tools do allow one to create such designs, but the design has to be created (or programmed) in such a tool.

5 Parametric CAD

That last does indicate a class of tool which is suited for this sort of work: Parametric CAD applications allow one to use numbers, formulae, and algorithms directly to define a design. Commercial examples:

- Autodesk Fusion 360/Inventor
- CATIA V5
- NX
- Onshape
- Pro Engineer
- Rhino 3D (when using the Grasshopper plug-in)
- Solid Edge
- SolidWorks

Many open source applications have also been developed which afford this style of design:

- FreeCAD — unfortunately somewhat limited in the calculations which may be performed; using a spreadsheet is advocated as a work-around[6], and importing OpenSCAD files is also an option.
- NaroCAD
- OpenVSP (Vehicle Sketch Pad from NASA)
- SolveSpace — fully graphical, with parameter alteration requiring selection.
- Varkon

But the most notable implementations are those which are programmatic in nature. Arguably there are too many to name (especially as any programming language can be one), but of special note are:

- Antimony — regrettably available for only GNU/Linux and Mac OS X; previous versions were the subject of the developer, Matt Keeter's, academic thesis.
- Maker.JS — a Microsoft Garage Project, this tool supports 2D design, but requires special effort to create a 3D file or preview.
- OpenSCAD — the most popular tool, widely used for 3D printing, and is notable for support on the popular project-sharing site Thingiverse which inaugurated the "Customizer" feature.
- PLaSM
- Tool Path Language (TPL) — a relatively recent development, this is a JavaScript variant supporting creation of G-Code to control the machine.

[6] floatingcam.com/blog/freecad-parametric-design

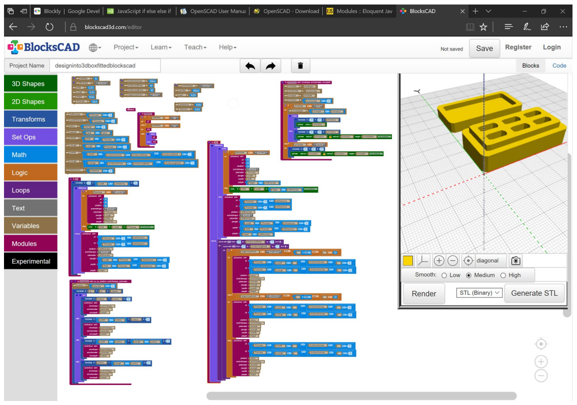


Figure 4: Blockly interface and graphical code of the initial prototype

This attempt at a representative sampling includes the most popular implementation, OpenSCAD, which was used for the implementation of this project.

6 BlocksCAD

Initial development was done using the Blockly implementation of OpenSCAD BlocksCAD (see fig. 4).[7] There are a number of similar tools, with varying tradeoffs, compromises, and difficulties. A better tool, with better graphical integration (specifically, the ability to select nodes, edges, or faces and drag them) would make for even easier development.

BlocksCAD allows one to save a project as an XML file, and to export to OpenSCAD. Similar tools include OpenJSCAD and Flood Editor.

7 OpenSCAD

BlocksCAD allowed a rapid development without worrying about the trivialities of coding such as the placement of semi-colons and an easy conversion into the textual OpenSCAD.

More important for the project is where the Customizer features (unfortunately unsupported by BlocksCAD) were implemented; see fig. 5.

8 Presets

Once a design has been worked up using the customization interface, the parameters must be passed to other tools. Fortunately, OpenSCAD implements saving design settings as “presets” in a JSON file:

```
{
    "parameterSets": {
        "export": {
            "$fn": "45",
            "Boxshape": "0",
            "Clearance": "0.01",
            "Height": "13.5",
            "Length": "66.675",
            "PARTNO": "0",
            "Thickness": "6.35",
            "Width": "209.55",
            "depth": "25.4",
            "diameter": "3.175",
            "dividers_lengthwise": "1",
            "dividers_thickness": "3.175",
            "dividers_widthwise": "2",
            "endmillshape": "1",
            "largecompartment": "2",
            "partspacing": "12.7"
        }
    },
    "fileFormatVersion": "1"
}
```

[7] www.blockscad3d.com/editor

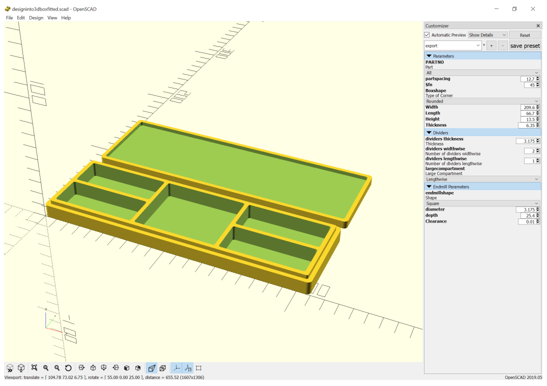


Figure 5: OpenSCAD design with Customizer

It is then a matter of loading the JSON data into variables. The first tool which makes use of this is LuaLaTeX, as well as the embedded METAPOST interpreter. Fortunately, the Lua scripting language has a tool available for importing JSON data.[8] Also, Henri Menke (at the conference) demonstrated an elegant system for reading in JSON which merits investigation (see pp. 129–135 in this issue).

```
\newcommand{\boxspecification}{export}
%\typein[\boxspecification]{What preset to use?}
\begin{luacode}
function read(file)
    local handler = io.open(file, "rb")
    local content = handler:read("*all")
    handler:close()
    return content
end
JSON = (loadfile "JSON.lua")()
local table = JSON:decode(read(
                   "designinto3dboxfitted.json"))
```

[8] regex.info/blog/lua/json

First, define a macro for each value which may then be redefined at need:

```
%           "PARTNO": "0",
\newcommand{\PARTNO}{\relax}
\newcommand{\definePARTNO}[1]
  {\renewcommand{\PARTNO}{#1}}
```

Then read in each variable from the selected preset (in this case, assigned to the LaTeX macro `\boxspecification`):

```
PARTNO = (table['parameterSets']
               ['\boxspecification']['PARTNO'])
```

Define the contents of the matching TeX macro:

```
\definePARTNO{\directlua{tex.print(PARTNO)}}
```

9 Drawing

Once one has all the numbers loaded, it's a matter of defining macros (the actual path definitions are quite lengthy):

```
def rp (expr x,y,z,w,l,t,d) = draw ⟨outer path⟩;
enddef;
def rpf (expr x,y,z,w,l,t,d,f) =
  fill ⟨inner block⟩ cycle withgreyscale f;
enddef;
def rpu (expr x,y,z,w,l,t,d) =
  unfill ⟨boundary⟩ -- cycle; enddef;
```

and then using them to draw:

```
beginfig(1);
rpf(-diam,-diam,0, Width*u+diam*2, Length*u
   +diam*2, Thickness*u, diameter*u,0.0);
rpu(0,0,0,Width*u, Length*u, Thickness*u, diam);
rpf(Thickness/2*u-halfclearance*u, Thickness/2*u
   -halfclearance*u, 0, Width*u-Thickness*u
   +Clearance*u, Length*u-Thickness*u+Clearance*u,
   Thickness *u-Thickness/4*u, diam,0.5);
endfig;
```

and to fill in the project description:

```
\sbox{\projectdescription}{\vtop{PARTNO:
                                  \dltw{PARTNO}\par
Boxshape: \dltw{Boxshape}\par
Clearance: \dltw{Clearance}\par
Height: \dltw{Height}\par
Length: \dltw{Length}\par
Thickness: \dltw{Thickness}\par
Width: \dltw{Width}\par
depth: \dltw{depth}\par
diameter: \dltw{diameter}\par
dividers:\par
\quad lengthwise: \dltw{dividerslengthwise}\par
\quad thickness: \dltw{dividersthickness}\par
\quad widthwise: \dltw{dividerswidthwise}\par
```

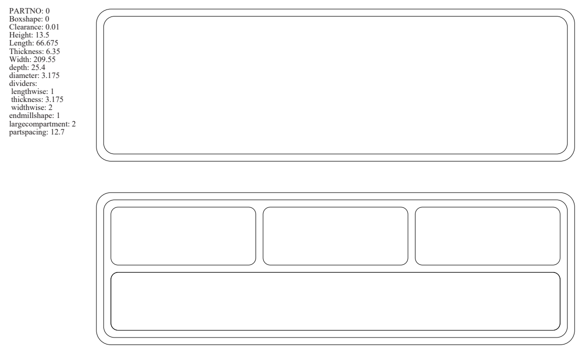


Figure 6: Typeset project plans and parameters

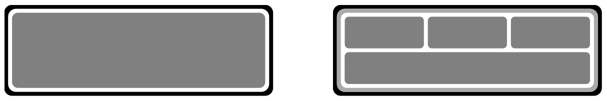


Figure 7: MetaPost output

```
endmillshape: \dltw{cuttershape}\par
largecompartment: \dltw{largecompartment}\par
partspacing: \dltw{partspacing}}}%
```

to end up with what's shown in fig. 6.

This is a basic representation — it would be possible to elaborate on that, adding colour and a depth mapping notation, and identify the parts (in this case, lid and base), and for more complex instructions, generate assembly instructions. It is also possible to add geometry and colour code such images so that they can be cut out directly.

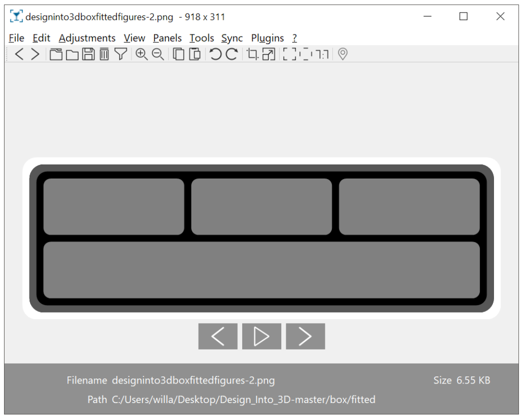
The system includes code for making SVG files which may be directly imported into a CAM tool.

```
outputtemplate := "%j-%c.svg";
prologues := 3;
outputformat := "svg";
input designinto3dboxfittedpreamble;
input designinto3dboxfittedfigure1;
input designinto3dboxfittedfigure2;
input designinto3dboxfittedpostamble;
```

The preamble and postamble files have macros and code for cleaning things up. The final drawings are shown in fig. 7.

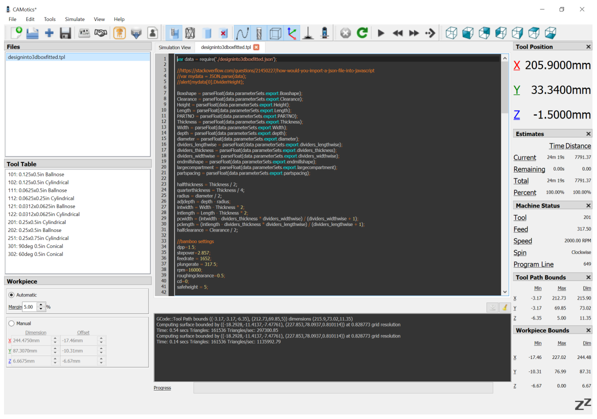
Creating SVG files allows one to use METAPOST only on the drawings, which is quick and efficient, and to use an SVG viewer (here, nomacs from Image Lounge (`nomacs.org`), shown in fig. 8) to interactively edit and remake the files, adjusting until things are as desired.

Once the files were ready, they could be imported into a CAM tool (in this case, the free Carbide Create) and toolpaths assigned so as to prepare the


designinto3dboxfittedfigures-2.png - 918 x 311
File Edit Adjustments View Panels Tools Sync Plugins ?

Filename designinto3dboxfittedfigures-2.png Size 6.55 KB
Path C:/Users/willa/Desktop/Design_Into_3D-master/box/fitted

Figure 8: nomacs interactive interface



```
var data = require('./designinto3dboxfitted.json');

//https://stackoverflow.com/questions/21450227/how-would-you-import-a-json-file-into-javascript
//var mydata = JSON.parse(data);
//alert(mydata[0].DividerHeight);

Boxshape = parseFloat(data.parameterSets.export.Boxshape);
Clearance = parseFloat(data.parameterSets.export.Clearance);
Height = parseFloat(data.parameterSets.export.Height);
Length = parseFloat(data.parameterSets.export.Length);
PARTNO = parseFloat(data.parameterSets.export.PARTNO);
Thickness = parseFloat(data.parameterSets.export.Thickness);
Width = parseFloat(data.parameterSets.export.Width);
depth = parseFloat(data.parameterSets.export.depth);
diameter = parseFloat(data.parameterSets.export.diameter);
dividers_lengthwise = parseFloat(data.parameterSets.export.dividers_lengthwise);
dividers_thickness = parseFloat(data.parameterSets.export.dividers_thickness);
dividers_widthwise = parseFloat(data.parameterSets.export.dividers_widthwise);
endmillshape = parseFloat(data.parameterSets.export.endmillshape);
largecompartment = parseFloat(data.parameterSets.export.largecompartment);
partspacing = parseFloat(data.parameterSets.export.partspacing);

halfthickness = Thickness / 2;
quarterthickness = Thickness / 4;
radius = diameter / 2;
adjdepth = depth - radius;
intwidth = Width - Thickness * 2;
intlength = Length - Thickness * 2;
pcwidth = (intwidth - dividers_thickness * dividers_widthwise) / (dividers_widthwise + 1);
pclength = (intlength - dividers_thickness * dividers_lengthwise) / (dividers_lengthwise + 1);
halfclearance = Clearance / 2;

//bamboo settings
dpp=1.5;
stepover=2.857;
feedrate = 1652;
plungerate = 317.5;
rpm=16000;
roughingclearance=0.5;
cd=0;
safeheight = 5;
```

```
GCode::Tool Path bounds ((-3.17,-3.17,-6.35), (212.73,69.85,5)) dimensions (215.9,73.02,11.35)
Computing surface bounded by ((-18.2928,-11.4137,-7.47761), (227.853,78.0937,0.810114)) at 0.828773 grid resolution
Time: 0.54 secs Triangles: 161536 Triangles/sec: 297300.85
Computing surface bounded by ((-18.2928,-11.4137,-7.47761), (227.853,78.0937,0.810114)) at 0.828773 grid resolution
Time: 0.14 secs Triangles: 161536 Triangles/sec: 1135992.79
```

Figure 10: Code for making toolpaths

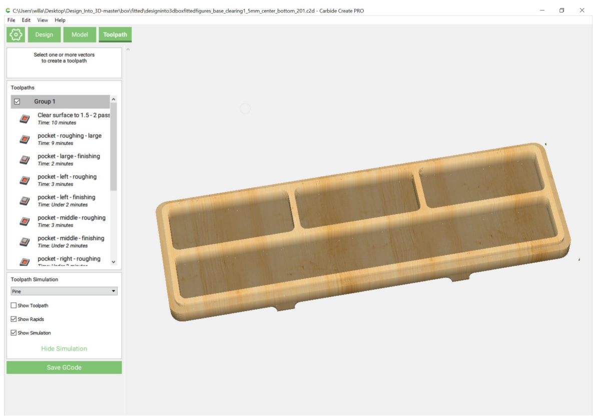
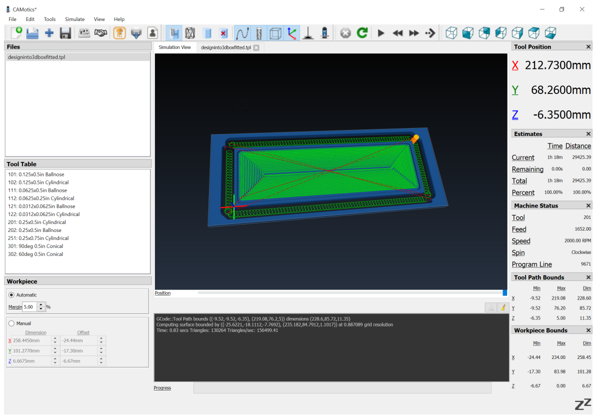


Figure 9: Toolpaths in Carbide Create



```
GCode::Tool Path bounds ((-9.52,-9.52,-6.35), (219.08,76.2,5)) dimensions (228.6,85.72,11.35)
Computing surface bounded by ((-25.6221,-18.1112,-7.7692), (235.182,84.7912,1.1017)) at 0.887089 grid resolution
Time: 0.83 secs Triangles: 130264 Triangles/sec: 156499.41
```

Figure 11: Preview of toolpaths

project for cutting, as shown in fig. 9. This however, limits one to the capabilities of the program in question, and requires a fair bit of manual effort.

10 Coding

The normal output for toolpaths is G-Code (RS-274), developed by the Electronic Industries Alliance in the early 1960s. For lack of a CAM tool which will directly map such vector greyscale images to efficient toolpaths we have instead chosen to work up a program based on CAMotics (`camotics.org`) which will import the JSON data parameters and directly create the toolpaths which will allow the design to be cut out, shown in fig. 10. The results of running this are in fig. 11.

In addition to reading in the parameters, ideally this tool would create optimal toolpaths using advanced features such as:

- ramping in — moving into a cut on a diagonal, or in a helical motion, rather than a straight vertical plunge — endmills are four times better at side-to-side cutting than they are at drilling.
- trochoidal toolpaths — shown in the curlicue paths around the perimeter of the part in fig. 11, trochoidal toolpaths allow efficient removal of material in a narrow slot by reducing tooling engagement, avoiding the full engagement of the machine attempting to move directly through material which has not yet been cut away.
- adaptive clearing — similar to trochoidal toolpaths, this is an optimized motion to clear an area, minimizing redundant motion while keeping tooling engagement at or near optimum.
- roughing clearance and finishing passes — the best finish and most precise/accurate parts are achieved by allowing the machine to remove a minimal amount of material at the end of a cut — much of the complexity shown in the toolpaths shown above were the result of manually implementing these.

11 Cutting

Once toolpaths are created, whether programmatically or using a typical CAM tool, the project may then be cut on the machine (fig. 12).

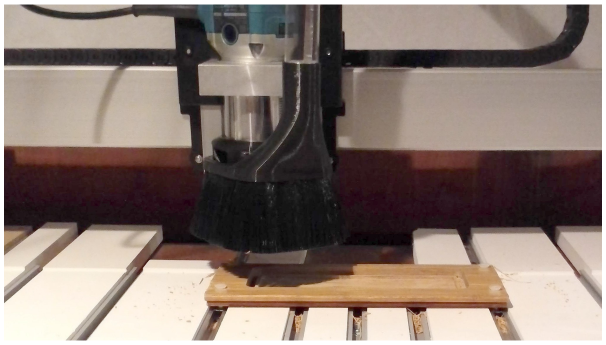


Figure 12: Project cutting

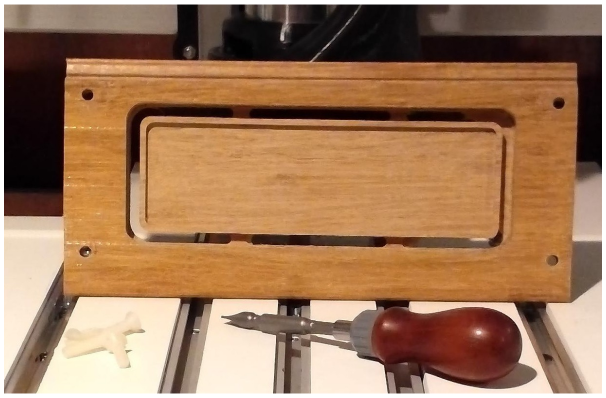


Figure 13: Post-processing after the cut

Once cut, the part will usually require some sort of post-processing (fig. 13) — at a minimum sanding, but possibly cutting tabs to release it from surrounding stock, or cutting away material at the bottom of the profile which was not completely removed.

But once, post-processed, one has a completed project (fig. 14).

12 Concepts

The Tool Path Language program proves the concept of beginning-to-end automation, but raises further questions, and leaves much room for improvement:

- Each project design must be worked up as a collection of specific programs — is there some way to have a more general design language which allows a more natural description of designs?
- The OpenSCAD customization interface is quite limited — an early version attempted to implement a natural switching between Imperial and metric units, but this was so awkward that it was abandoned — using another tool to develop a front-end would seem better.
- It requires that the user download, install, and use a number of tools (OpenSCAD, LuaLaTeX, CAMotics/Tool Path Language) — this on top of the normal program(s) required to run the machine.

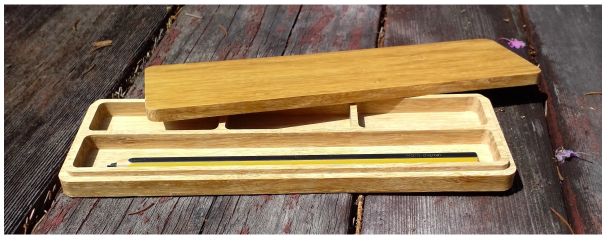


Figure 14: Completed project

12.1 Shapes

The above code, rather simplistically, only requires clearing rounded corner pockets. More complex projects will require macros/functions for additional shapes, and names for them. Arranging them by the number of points, we find that all but a few have an accepted single word nomenclature (or suitably concise description):

- 0
  - circle
  - ellipse (requires some sort of non-arc curve)
    - egg-shaped (oval)
  - 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[9]
  - two convex, one concave arc
- 4
  - rectangle (including square)
  - parallelogram
  - rhombus
  - trapezoid/trapezium
  - kite
  - ring/annulus segment (straight line, concave arc, straight line, convex arc)
  - astroid (four concave arcs)
  - salinon (four semicircles)
  - three straight lines and one concave arc

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

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

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

A final consideration: It has been said that there are two types of furniture — the system fails to take that into account or to leverage on it.

12.2 Two types of furniture

What are the two types of furniture?

- Boxes
- Platforms

This first project has involved making two-piece boxes out of solid materials, simply removing what is not needed for the design. While this works for small pieces, it is necessarily limited to the degree to which it can be scaled up, and quickly becomes profligately wasteful of material. The traditional solution for this is joinery, of which there are many sorts, and thus far for CNC, usually involve complicated fixtures and jigs and multiple setups.

12.3 Further steps

Developing a solution which could incorporate joinery efficiently is one obvious next step. All of the pockets assume 2.5D cutting on a single plane — the ability to make cuts at an angle would afford a welcome flexibility which is needed in some sorts of joinery. Similarly, the ability to make cuts using arbitrary endmill shapes may enable designs as yet undreamed of. Possibilities:

- Joinery
- General purpose design frameworks/grammars
- Special purpose tools — there are many extant project generators for various sorts of boxes, furniture (chairs and workbenches) gears, geography, clocks, even houses — other possibilities include telescopes, cribbage boards, &c.
- Would it be possible to create a font where a series of letters would describe discrete aspects of a design, assign toolpaths to the appropriate letters using that font, and then to change the design by just changing the text?
- Ornamentation — that's next year's Kickstarter and presentation — ideas include Sheridan (traditional Western floral leatherworking), Celtic knots and letters, and Arabesques, as well as various arrangements of text.

13 Continuing work

This was initially a (funded) Kickstarter.[10] It is being developed as a wiki page on the Shapeoko project[11] with code on GitHub.[12] A number of sample files and projects have already been made[13,14,15]; and this is tied into a Thingiverse project[16] and an online box generator.[17]

◇ William Adams
`willadams (at) aol dot com`

[9] `en.wikipedia.org/wiki/Hyperbolic_sector` and `www.reddit.com/r/Geometry/comments/bkbzgh/is_there_a_name_for_a_3_pointed_figure_with_two`

[10] `kickstarter.com/projects/designinto3d/design-into-3d-a-book-of-customizable-project-desi`

[11] `wiki.shapeoko.com/index.php/Design_into_3D`

[12] `github.com/WillAdams/Design_Into_3D`

[13] `cutrocket.com/p/5c9fb998c0b69`

[14] `cutrocket.com/p/5cb536396c281`

[15] `cutrocket.com/p/5cba77918bb4b`

[16] `www.thingiverse.com/thing:3575705`

[17] `chaunax.github.io/projects/twhl-box/twhl.html`