Working With the Machine

A Maker’s Journey into Clojure

This talk

- My Story
  - My fascination with making
  - Curiosity with Computers as Creative Tools
  - Excitement about Clojure
- Encouraging and (maybe, just a little) Inspiring

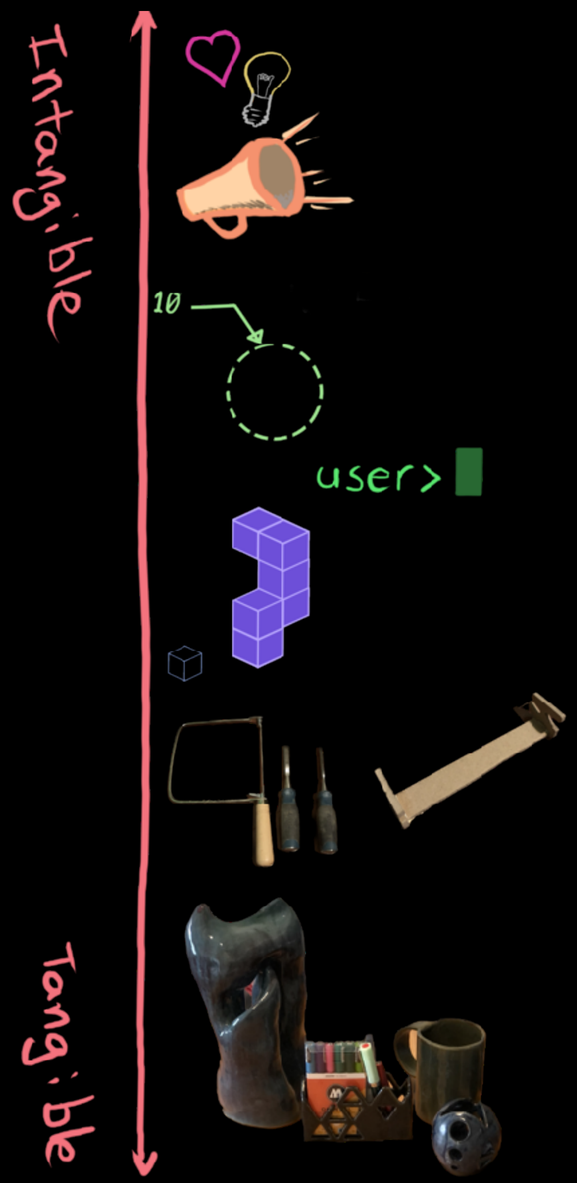
Me (briefly)

- Grew up in Southern Ontario
- Family Business -> Greenhouses
- ‘The Shop’ -> work, play, building
- Mechanical Engineering

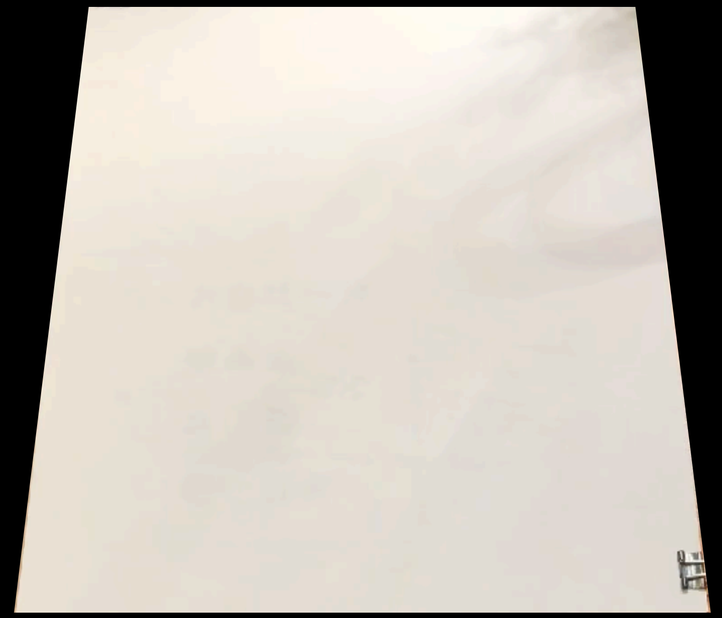
Concepts

Tangibility

"Let me see that"

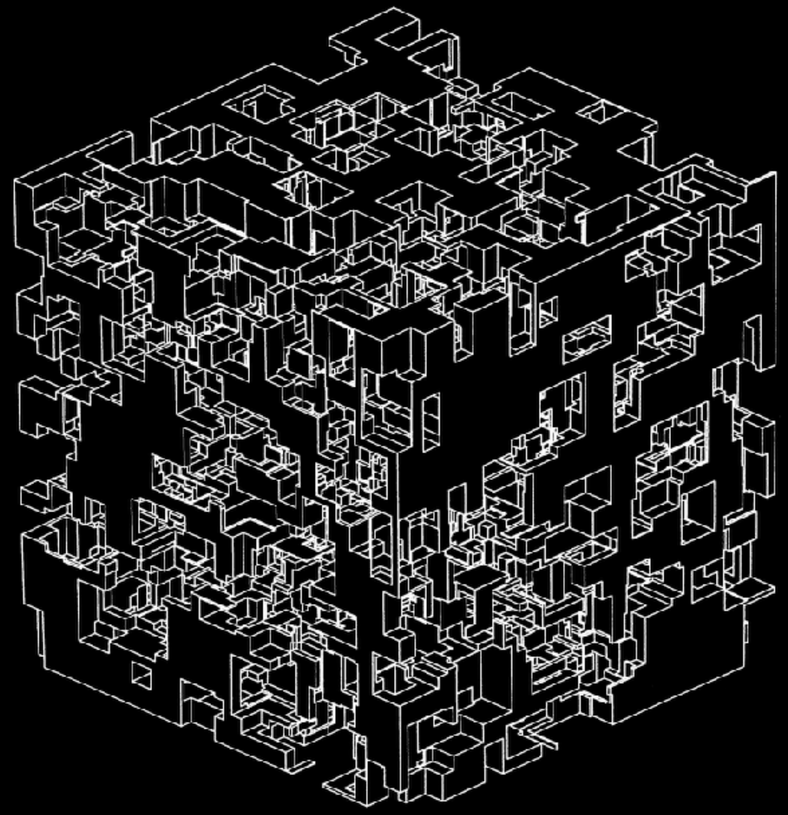


- Desires, insights
- Ideas and Concepts
- Verbal/Textual Communication
- Designs, Drawings
- Models (digital, scale models)
- Objects (eg. Prototypes)



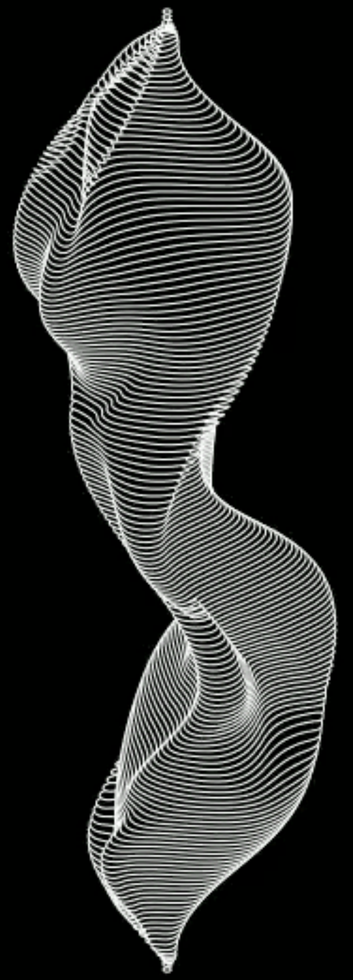
Garbage Collection

- Don’t be afraid to try stuff
- Explore your way towards solutions
- Make stuff even if others can do it better
- Makers make things. There are no shortcuts



Motivation

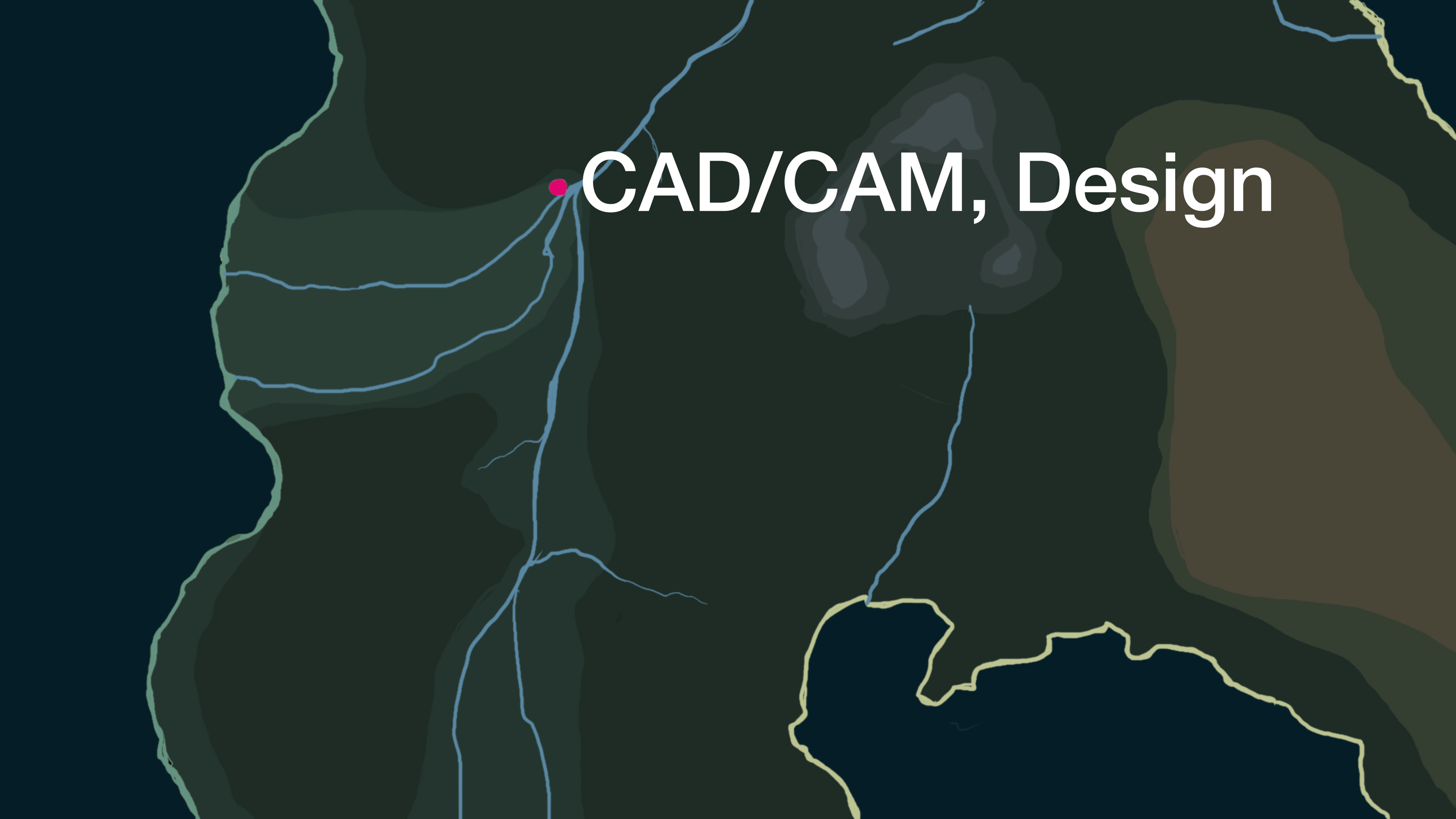
- Curiosity
- Sense of Accomplishment
- Competence/Confidence
- Desire to Help
- Love



Being a Maker

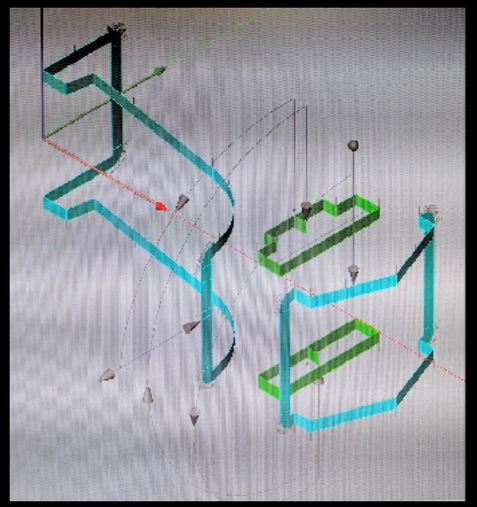
- One who makes
- Turns ideas into tangible objects
- More than ‘person with a 3D printer’

The Journey Begins


CAD/CAM, Design

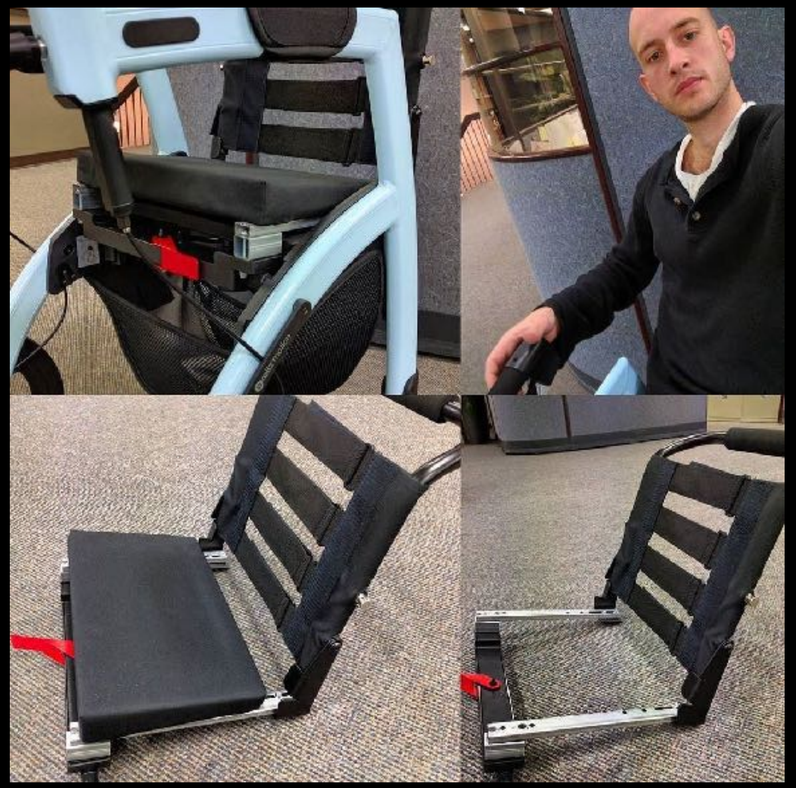
CAD Programs

- Solidworks
  - 3D, ‘Feature Tree’, Sketch/Constraint Based Modelling
- AutoCAD
  - 2D, sort of 3D, command based, Direct Modelling
- Proprietary CAM per machine
  - 2D nesting programs
  - Tool Path adjustment

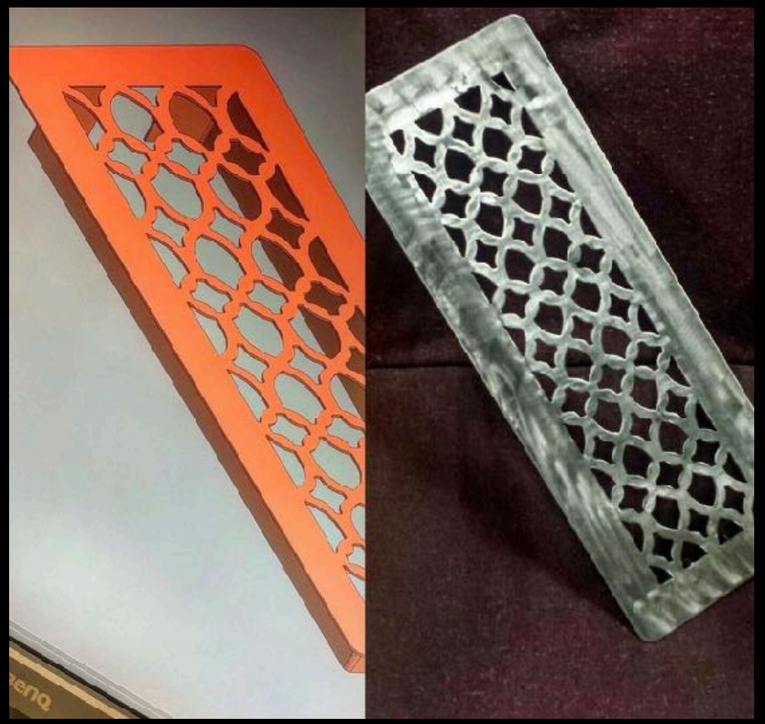


Some Early Projects

Laser cut Metal Tube ‘Scene’

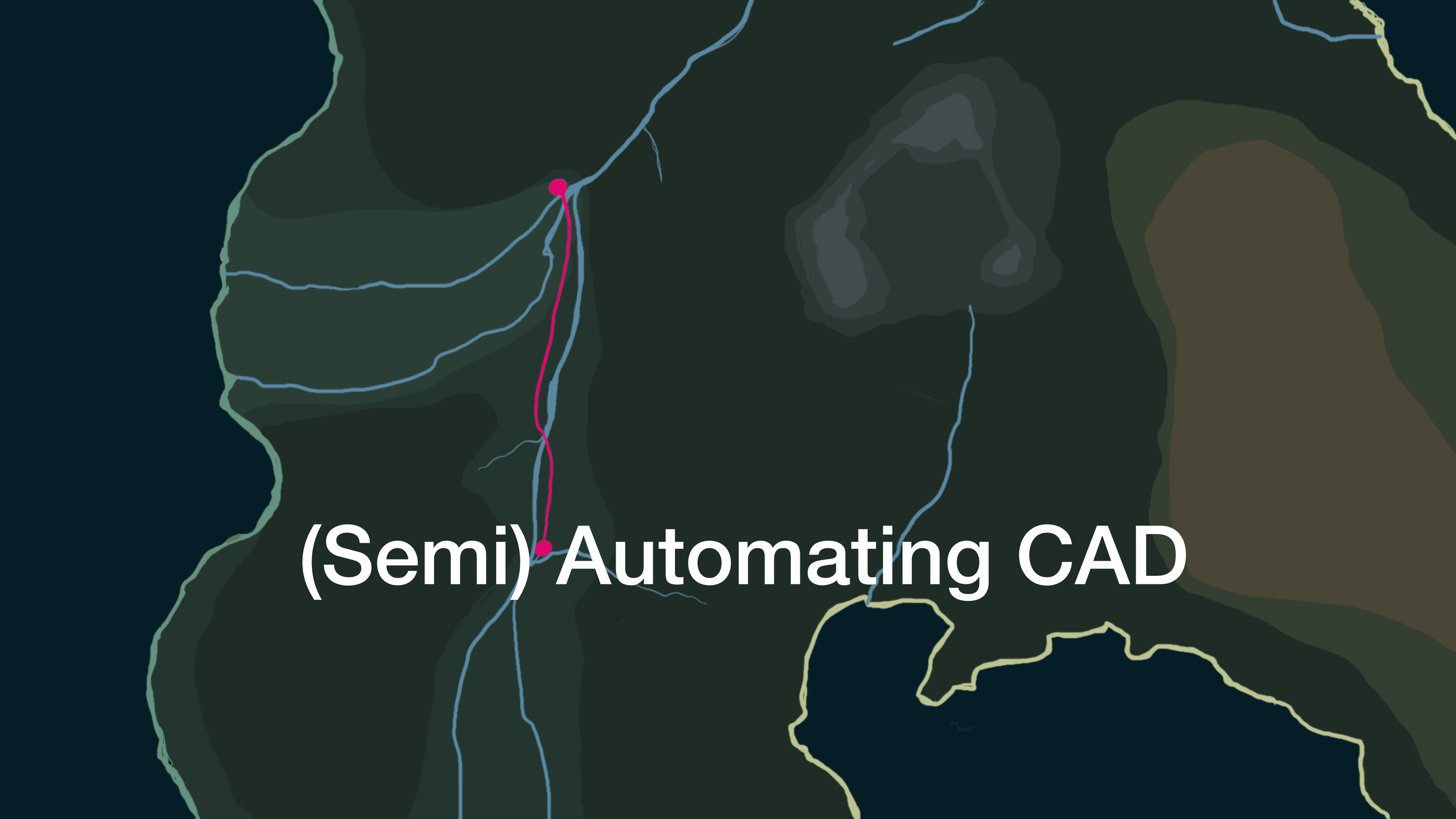


Walker modification for Oma

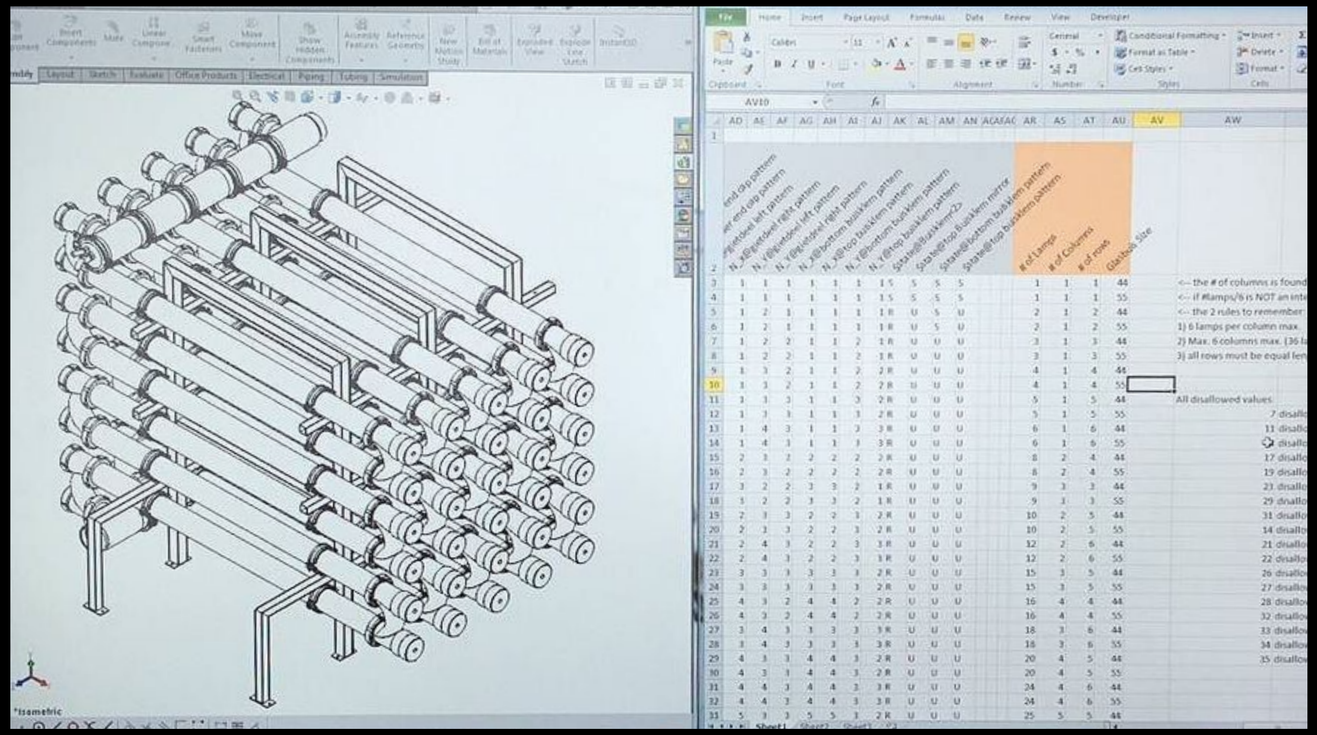


Vent cover for my sister

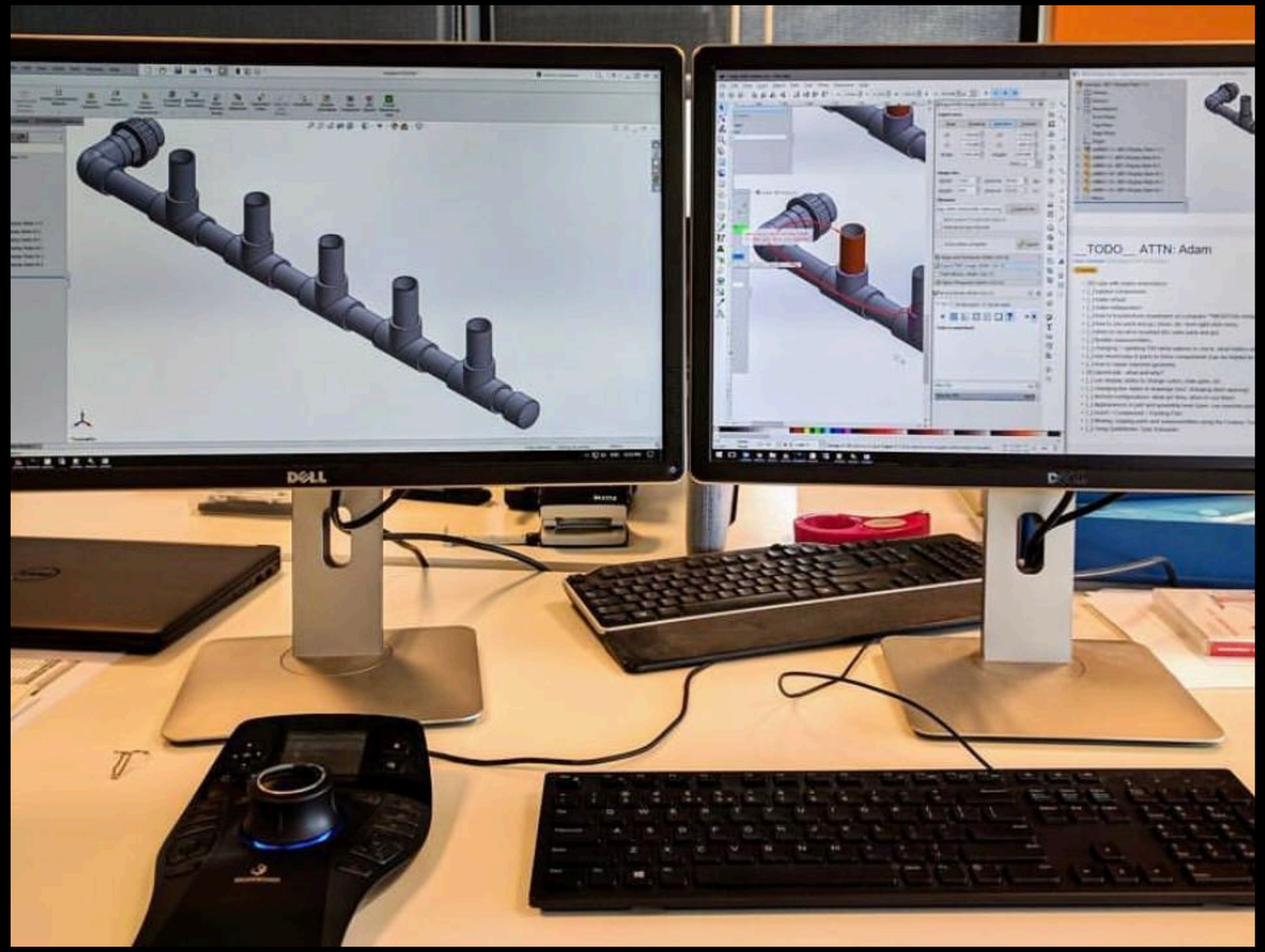
Experimental Furniture


(Semi) Automating CAD

Configurations



- SolidWorks ‘configurations’ used embedded Excel Sheets
- Attributes in your model as column heading
  - Values for the attributes in the rows
  - Each row = 1 configuration
- Excel formulas and macros all work


__TODO__ ATTN: Adam

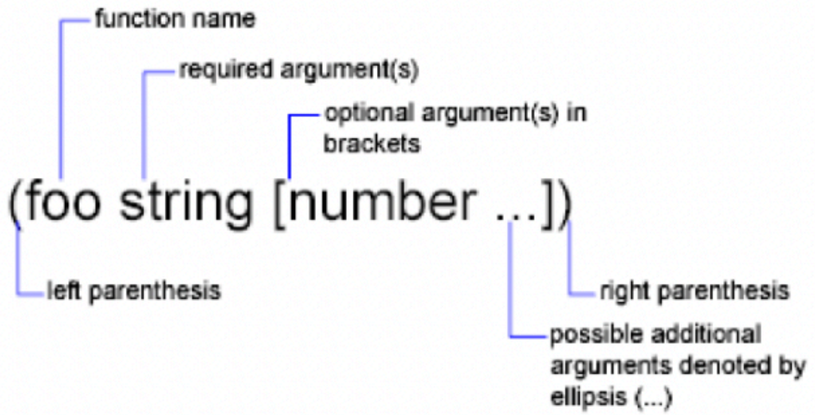
Problems with Configurations

- Logic of models hidden in embedded sheets
- Can have logic in SW formulas that directly contradicts logic in a configuration spreadsheet
- Feature toggling can break constraints
- High learning curve and difficult to communicate structure/proper usage
- Can reduce part/sub-assembly reuse
- Inflexible to product changes over time

Auto-LISP

AutoLISP Function Syntax

In this guide, the following conventions describe the syntax for AutoLISP functions:



Nil Variables

An AutoLISP variable that has not been assigned a value is said to be `nil`. This is different from blank, which is considered a character string, and different from 0, which is a number. So, in addition to checking a variable for its current value, you can test to determine if the variable has been assigned a value.

Each variable consumes a small amount of memory, so it is good programming practice to reuse variable names or set variables to `nil` when their values are no longer needed. Setting a variable to `nil` releases the memory used to store that variable's value. If you no longer need the `val` variable, you can release its value from memory with the following expression:

```
(setq val nil)

nil
```

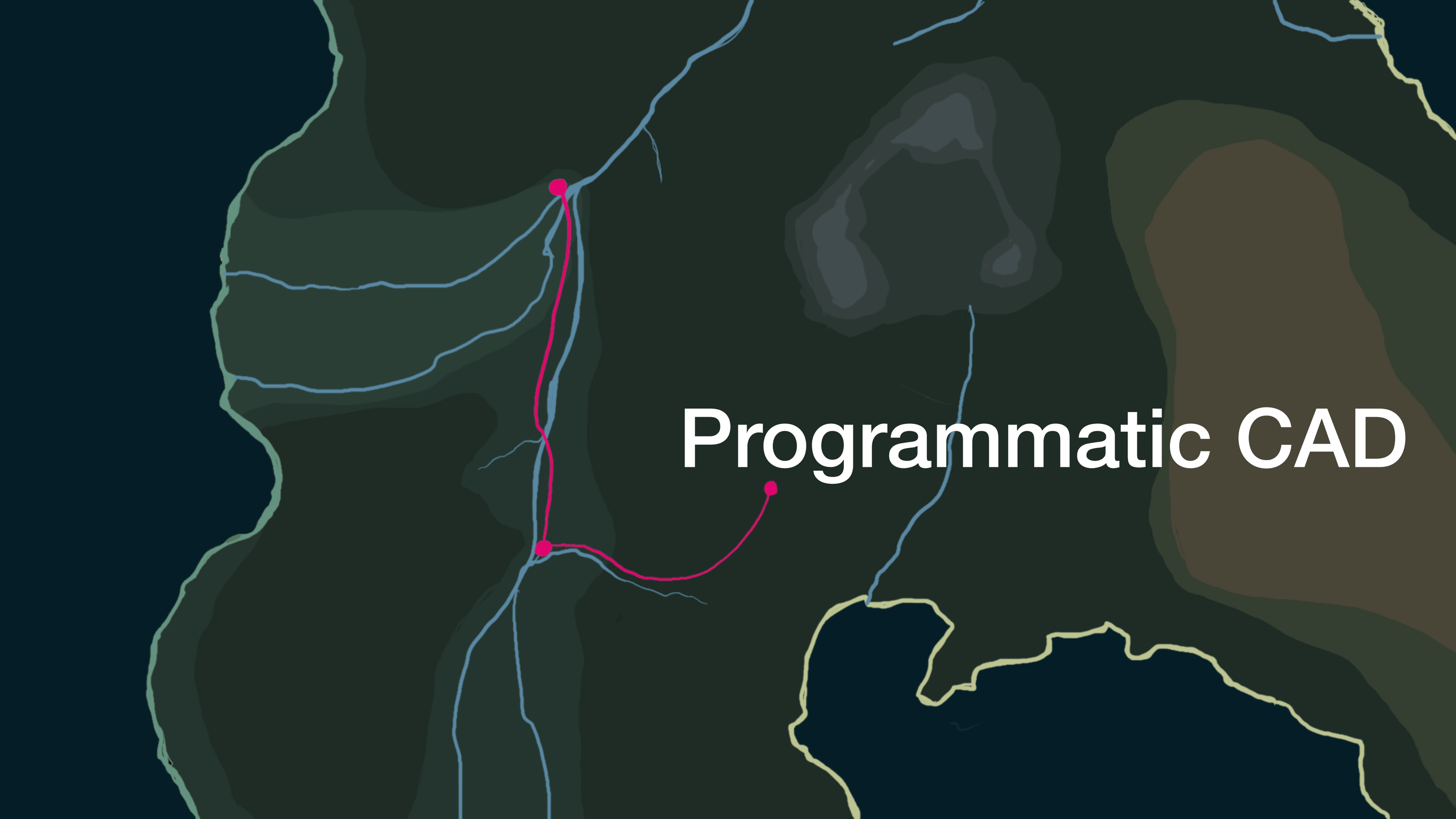
Another efficient programming practice is to use local variables whenever possible. See Local Variables in Functions (page 34) on this topic.

http://docs.autodesk.com/ACDMAC/2013/ENU/PDFs/acdmac_2013_autolisp_developers_guide.pdf

Visual Basic for Applications

```
Dim swApp As SldWorks.SldWorks
Dim swModel As SldWorks.ModelDoc2

Sub main()

    Set swApp = Application.SldWorks

    Set swModel = swApp.ActiveDoc

    If Not swModel Is Nothing Then

        Dim swSelMgr As SldWorks.SelectionMgr

        Set swSelMgr = swModel.SelectionManager

        Dim swView As SldWorks.View

        Set swView = swSelMgr.GetSelectedObject6(1, -1)

        If Not swView Is Nothing Then

            Dim swRefDoc As SldWorks.ModelDoc2
            Set swRefDoc = swView.ReferencedDocument

            If swRefDoc Is Nothing Then
                Err.Raise vbError, "", "Drawing view model is not loaded"
            End If
```

https://www.codestack.net/solidworks-api/document/drawing/open-referenced-model/


Programmatic CAD

Why Programmatic CAD?

- Code as Documentation
- Literate Programming as documentation:
  - For the Code
  - for the Domain Logic of the stuff being manufactured
- Textual version control
- Open Tools = No vendor lock-in

Some examples

- OpenSCAD
- Cadquery
- Freecad Python Scripting
- Honourable mention: Inkscape

Jupyter Notebook Extension ‘CQNB’

File Edit View Insert Cell Kernel Widgets Help

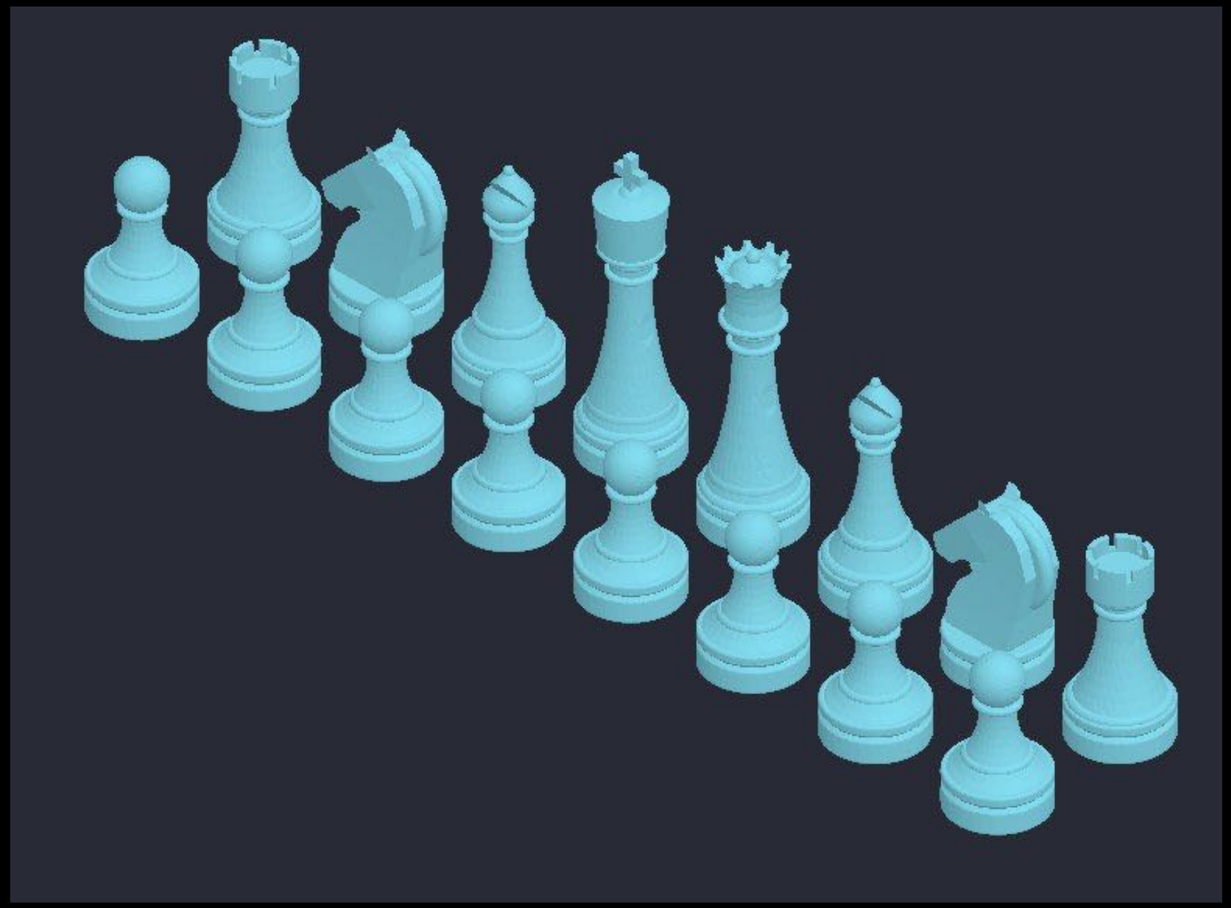
Trusted | Python 2

In [ ]:

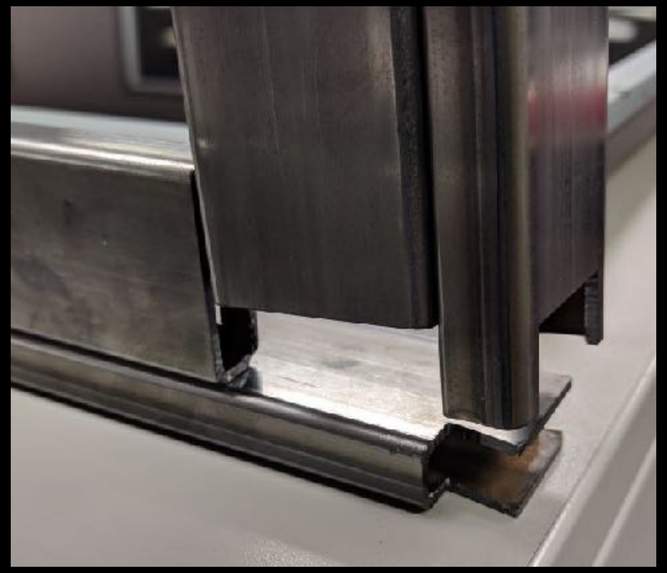
```
import cadquery as cq
# Set up the length, width, and thickness
(L, w, t) = (20.0, 6.0, 3.0)
s = cq.Workplane("XY")
# Draw half the profile of the bottle and extrude it
p = s.center(-L / 2.0, 0).vLine(w / 2.0)
     .threePointArc((L / 2.0, w / 2.0 + t), (L, w / 2.0)).vLine(-w / 2.0)
     .mirrorX().extrude(30.0, True)
# Make the neck
p.faces(">Z").workplane().circle(3.0).extrude(2.0, True)
result = p.faces(">Z").shell(0.3)
# Displays the result of this script
show_object(result)
```

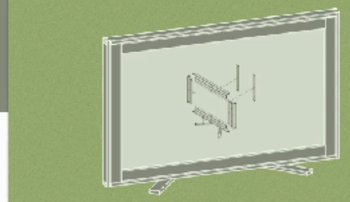
Writing A Model with CQ

Illustrations in Inkscape



Chess Set Designed entirely with Cadquery





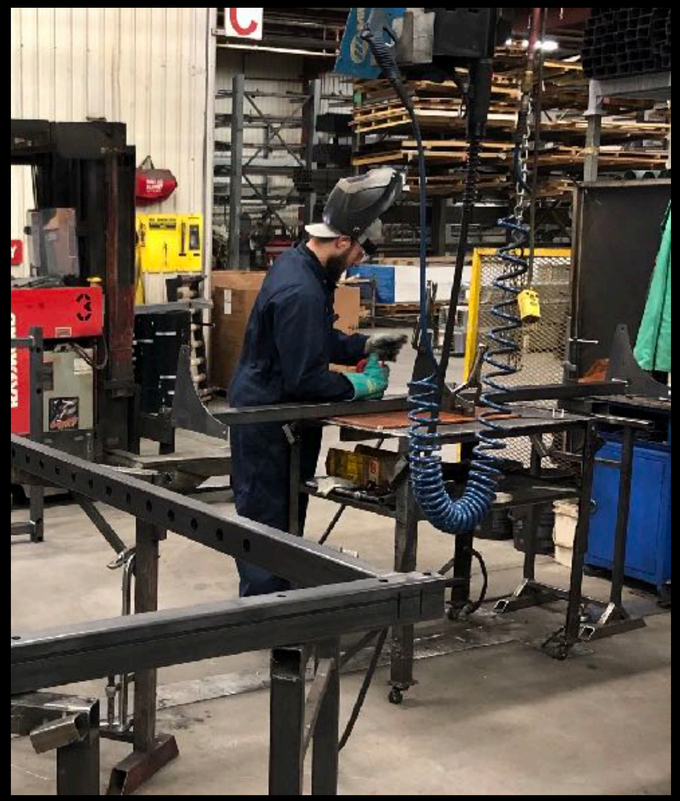
Building a TV Frame

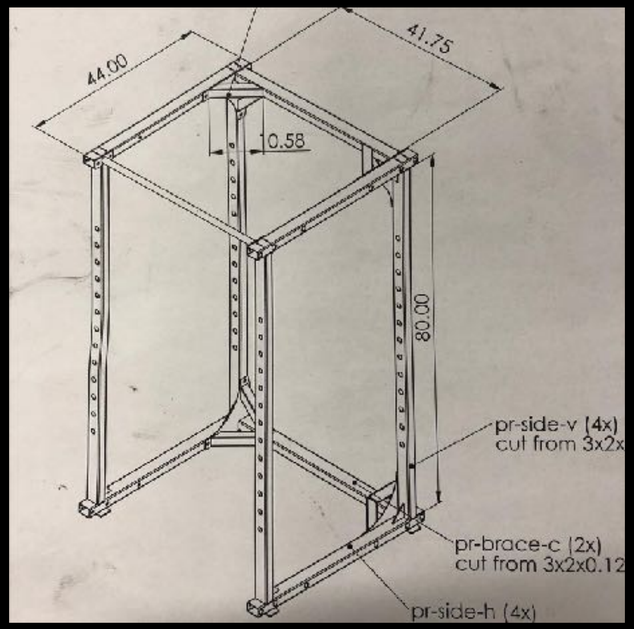
> If you're not keen on reading through my design work, that's a bit of a shame, but I understand; we're busy people. Feel free to check out the downloads at the bottom of the article, though!

Several years ago I built a weird looking desk that had a PC and TV built right in. It never progressed beyond 'usable prototype', and was eventually disassembled. The TV had been partially stripped down to fit nicely in the desk's surface, and the old parts were discarded.

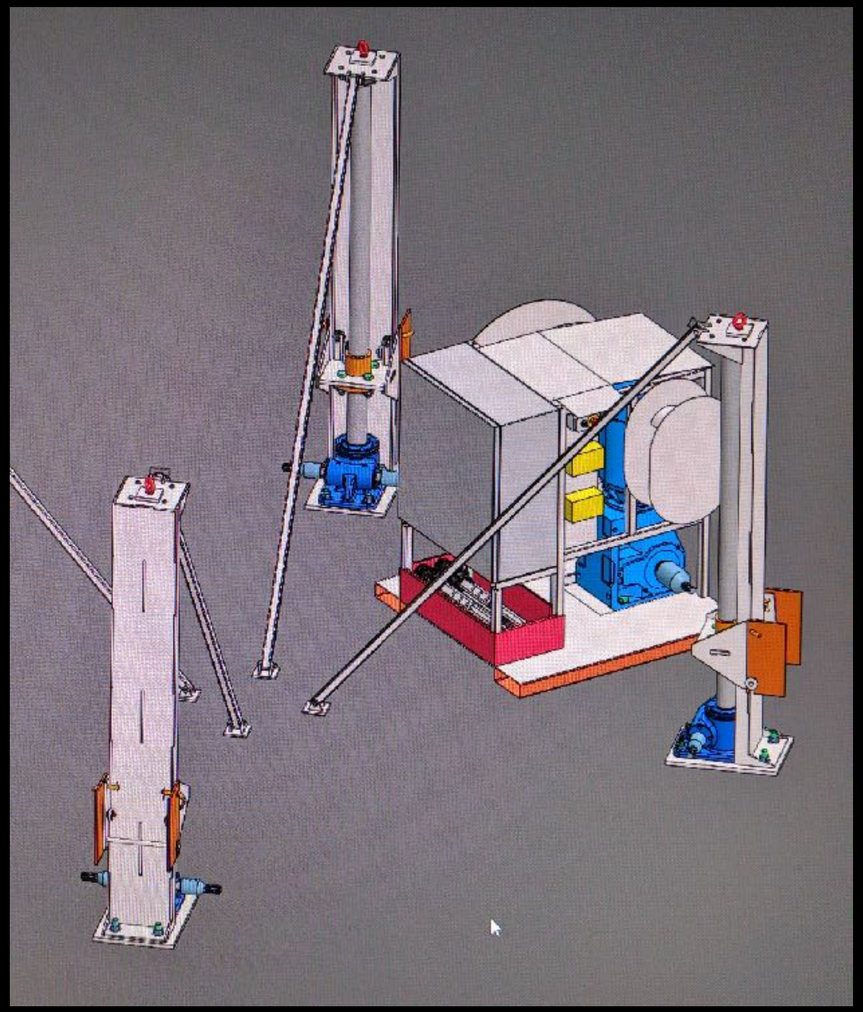
Laser cut Chess Set

Pandemic Gains With my Bro





An Actual Work Project




Scratching the Creative Itch
Going All-in with Clojure

Why Clojure?

- Functional programming made more sense in my brain
- scad-clj -> I could actually understand the whole library
- REPL and code tangibility
- Older code still works, making examples more consistently valuable
- People who use it seem to know what they're doing

```
(defn sphere 4 references
  [r]
  (fn [pt]
    (- (u/distance pt [0 0 0]) r)))
```

Augmenting My Tattoo

<- Tracing on iPad

|
V

.png to .svg with tracer

|
V

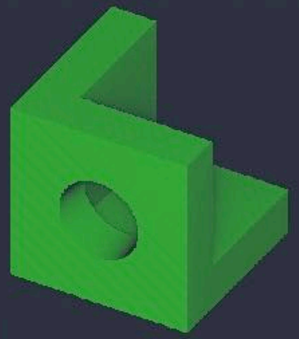
<- 3D model with OpenSCAD

|
V

Animate with ffmpeg ->

Naive (really bad) Ray Marching

```
#'forge.frep/cider-show-3d
forge.frep> (cider-show-3d asdf)
#object[java.io.File 0x68ccf591 "_imgtmp.png"]
```



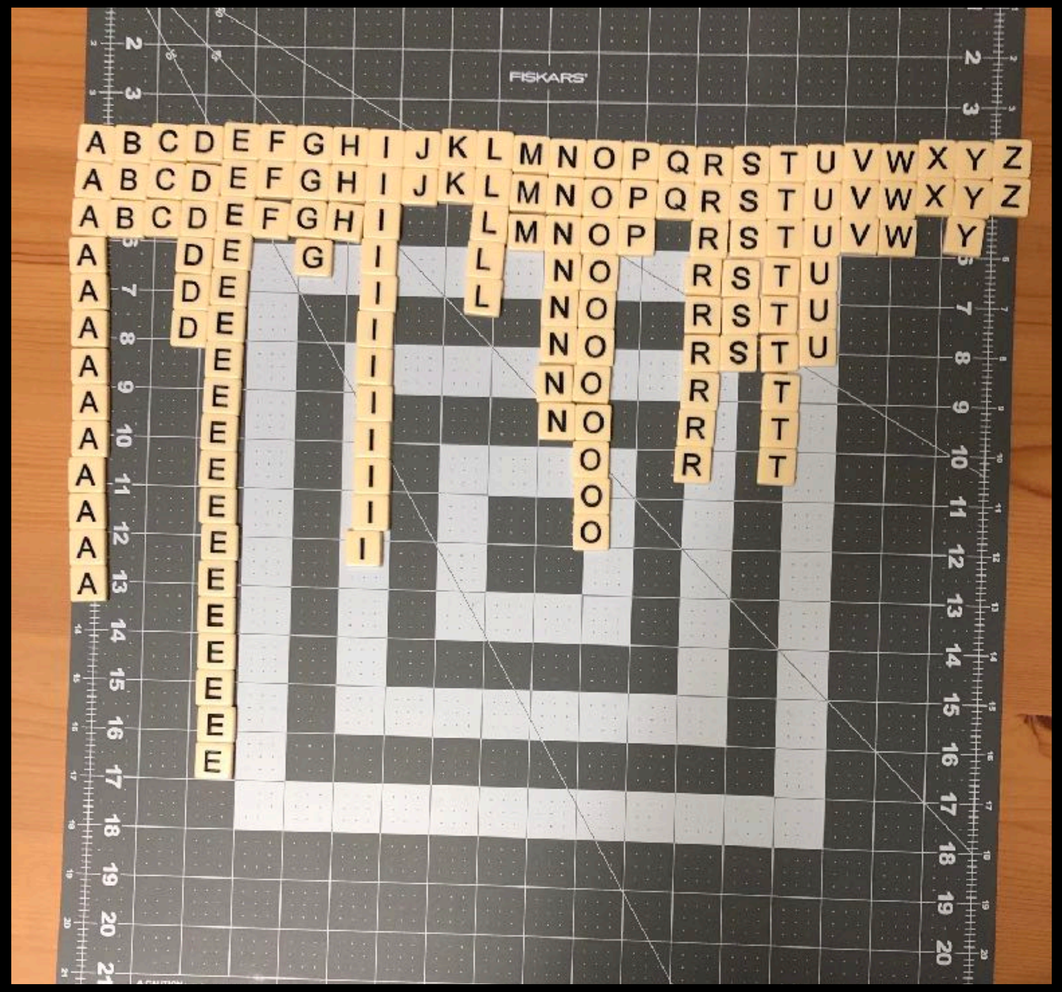
```
forge.frep>
```

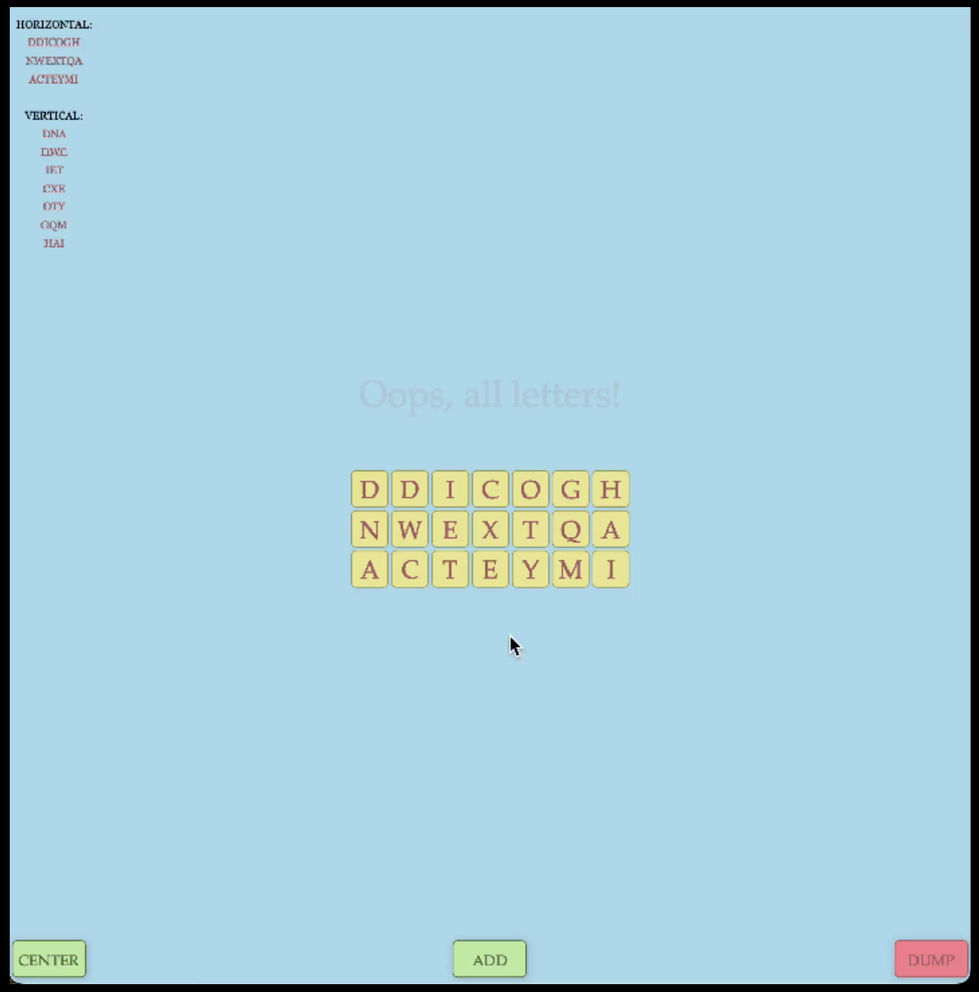
```
(defn march
  [f [x y]]
  (let [frep (isometric-xf f)
        from (conj [x y] -200)
        dir [0 0 1]
        max-steps 50
        min-dist 0.0001]
    (loop [n-steps 0
           total-dist 0]
      (let [pt (utils/v+ from [0 0 total-dist])
            d (utils/round (frep pt) 4)]
        (if (or (> n-steps max-steps)
                (< d min-dist))
          (- 1.0 (/ n-steps max-steps))
          (recur (inc n-steps) (+ total-dist d)))))))

(defn render-pixel
  [[pt a]]
  (when (> a 0.001)
    (-> (svg/rect 1 1)
        (tf/translate [0.5 0.5])
        (tf/translate pt)
        (tf/style {:fill "limegreen"
                   :opacity a}))))

(defn render-frep
  [x1 y1 x2 y2 f]
  (cp/pfor 10 [x (range x1 (inc x2))
               y (range y1 (inc y2))]
           (render-pixel [[x y] (f [x y])])))

(defn cider-show-3d
  [frep]
  (let [f (partial march frep)
        render (remove nil? (render-frep -200 -200 200 200 f))]
    (when (not (empty? render))
      (svg-clj.tools/cider-show render))))

(def asdf
  (let [base (box 200 200 200)]
    (-> (difference
         (difference
          base
          (-> base (translate [50 50 50])))
         (-> (cylinder 45 250) (rotate [0 90 0])))
        (rotate [0 0 180]))))
```

```
(defn block
  [w]
  (let [w2 (/ w 2.0) ;; half width
        chs (/ w 20.0) ;; chamfer size
        cr (* (Math/sqrt 2) (- w2)) ;; corner radius
        ch (->> (cube (* 2 w) chs chs)
                (rotate [(* PI 0.25) 0 0])
                (translate [0 w2 w2])) ;; chamfer component
        hpos (- w2 (* 2 chs))
        hole (->> (cylinder (/ chs 2) chs)
                  (translate [hpos hpos w2]))
        sch (union hole
                   (rotate [0 0 (* PI 1)] hole)
                   (rotate [0 0 (* PI 0.5)] hole)
                   (rotate [0 0 (* PI -0.5)] hole)
                   ch
                   (rotate [0 0 (* PI 0.5)] ch)
                   (rotate [0 0 (* PI 1.5)] ch))]
    (->> (difference (cube w w w)
                     sch
                     (->> sch (rotate [(* PI 0.5) 0 0]))
                     (->> sch (rotate [0 (* PI 0.5) 0]))
                     (->> sch (rotate [(* PI 1) 0 0]))
                     (->> sch (rotate [(* PI 1.5) 0 0]))
                     (->> sch (rotate [0 (* PI 1.5) 0])))
         (color [(/ 253 255) (/ 170 255) (/ 59 255) 1]))))

(union (block 100)
       (bolts 100)
       (for [rt [0 0.5 1.0 1.5]]
         (->> (f? 3 7.25)
              (translate [0 -50 0])
              (rotate [0 0 (* PI rt)]))))
```

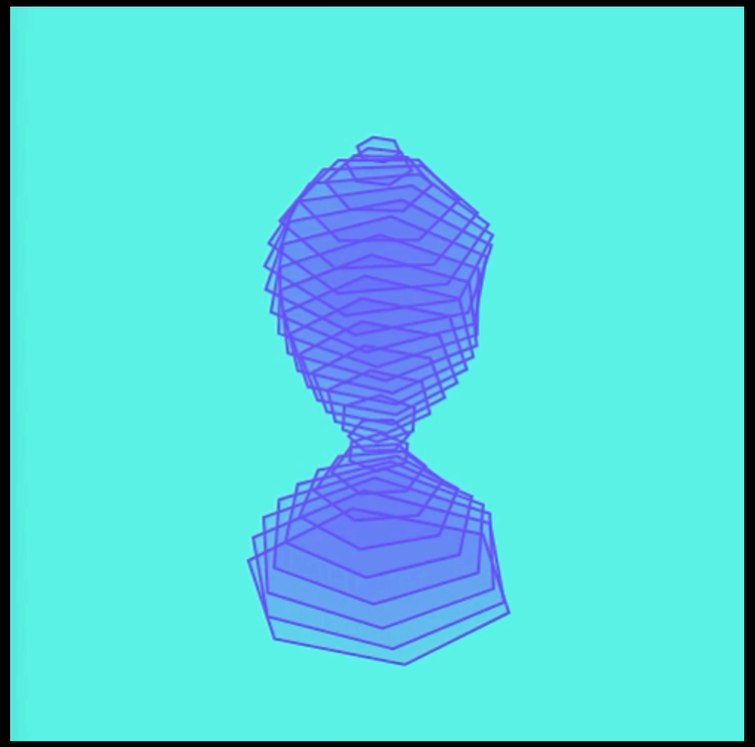
Fun with scad-cli

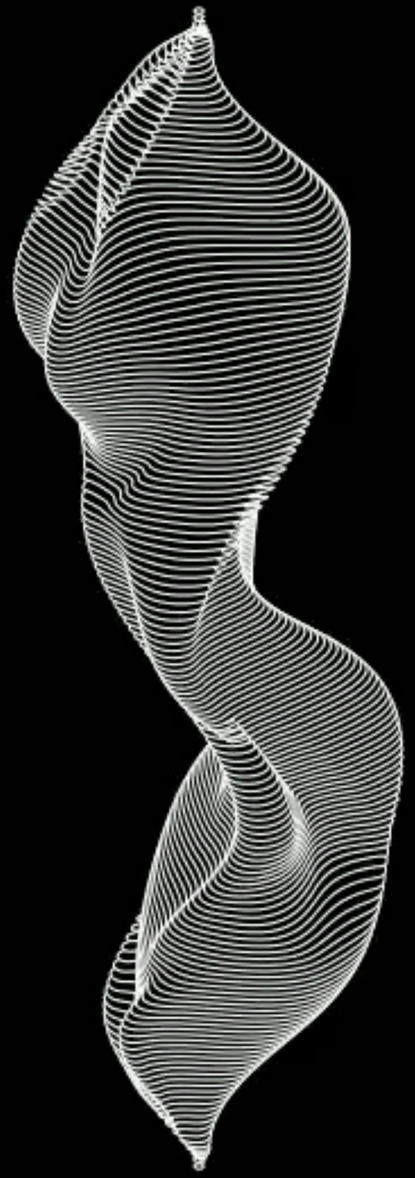
Recreating Bananagrams in Clojurescript

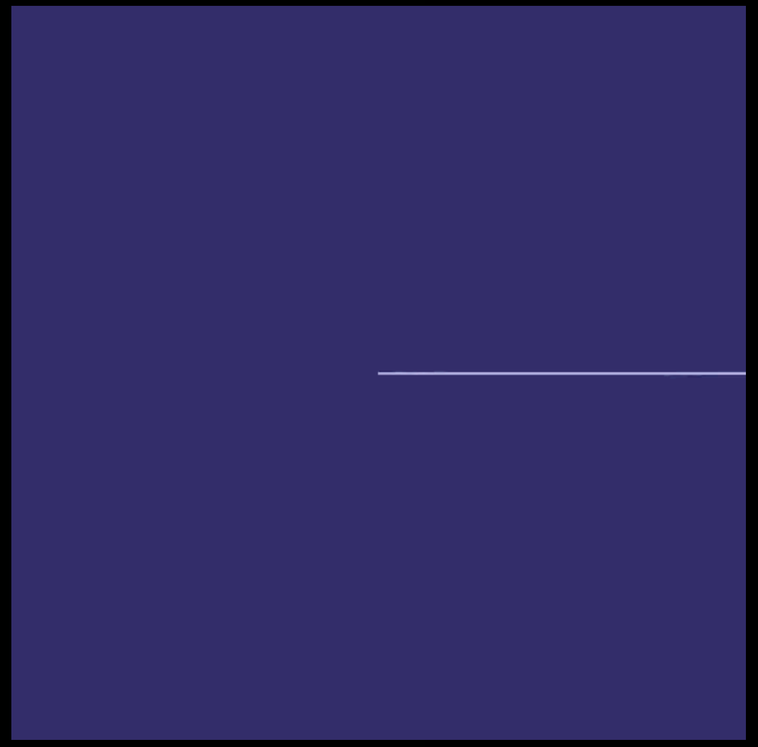


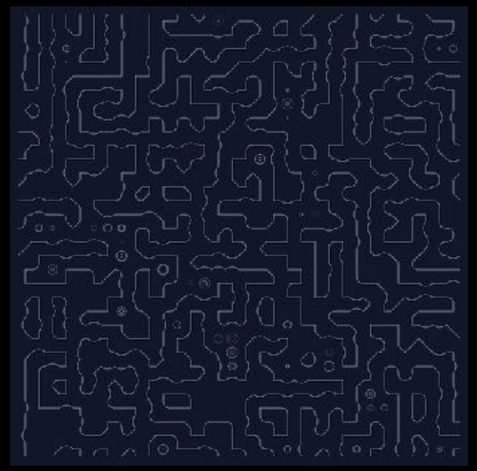


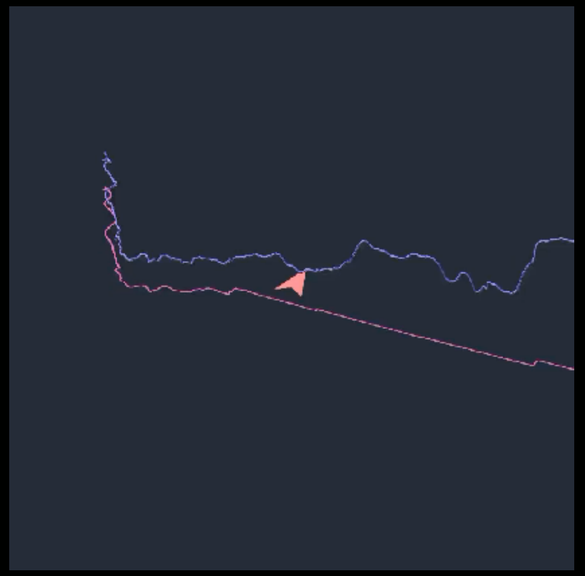
Prototype Hydroponics System

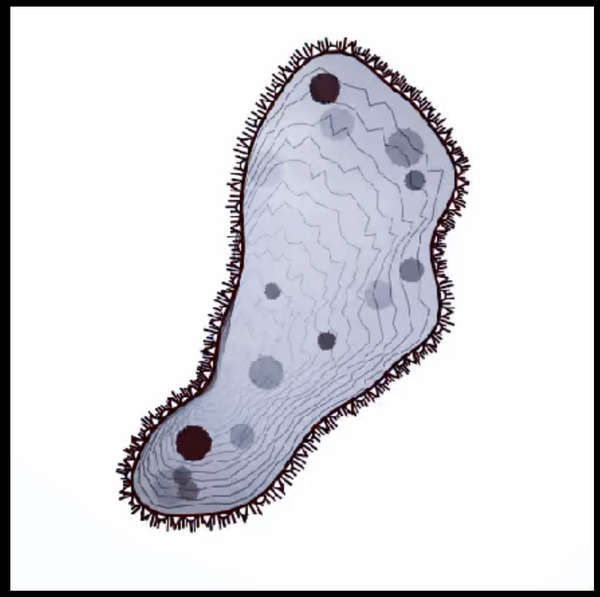


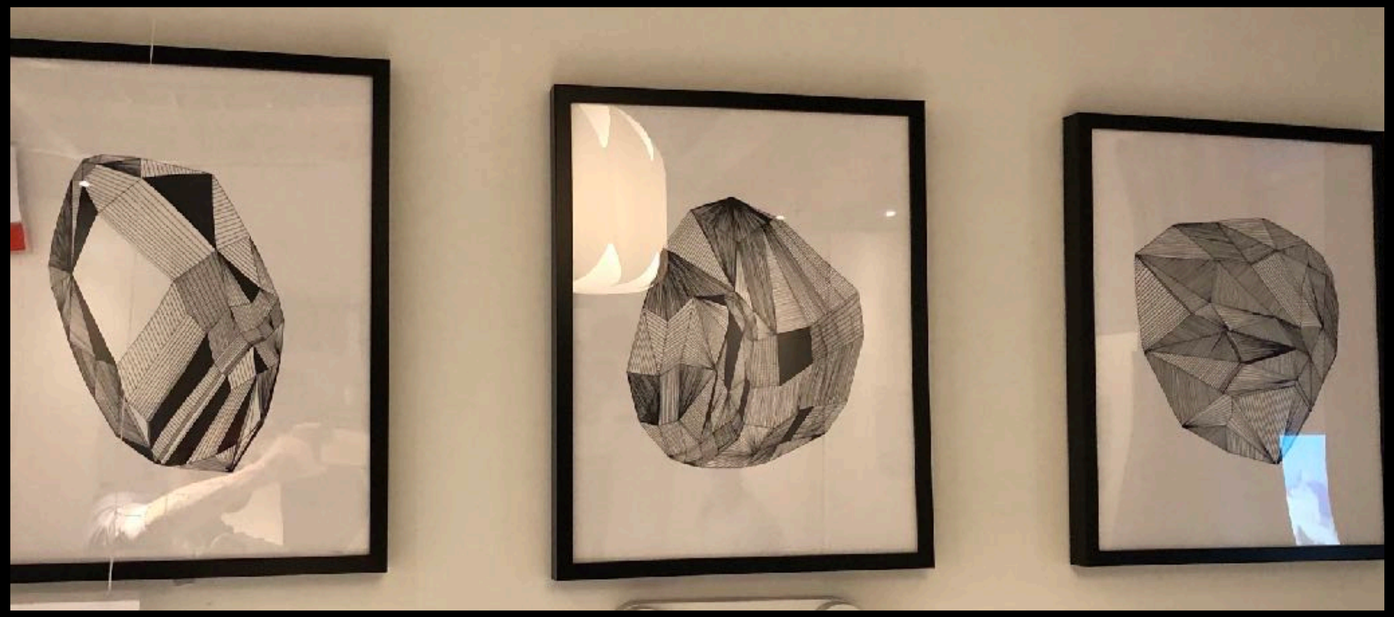


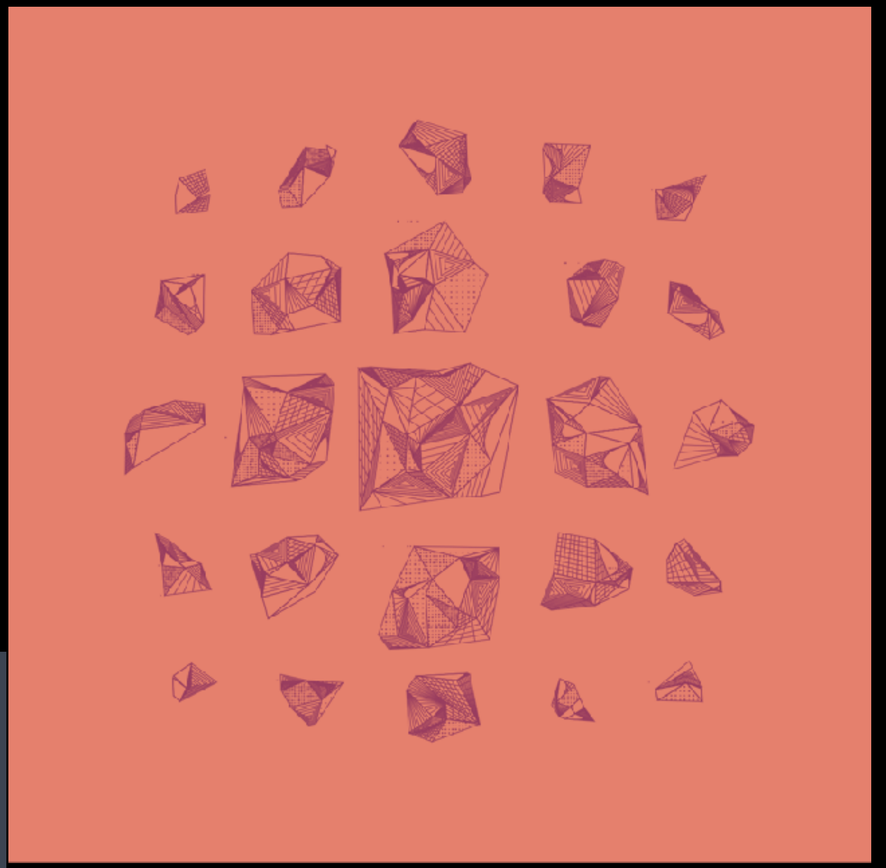




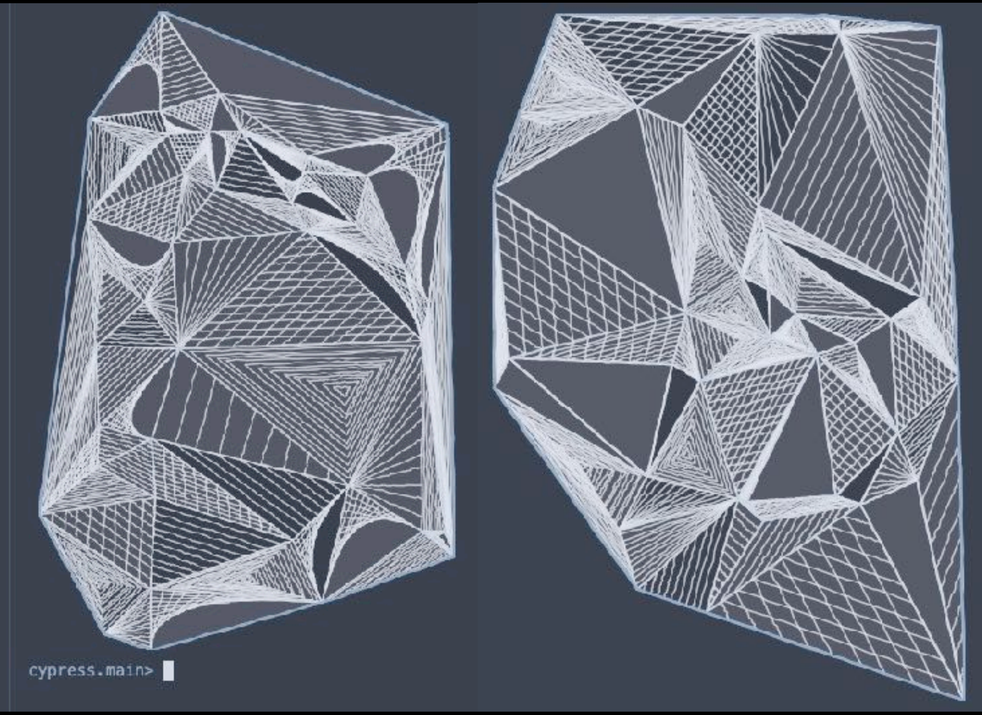








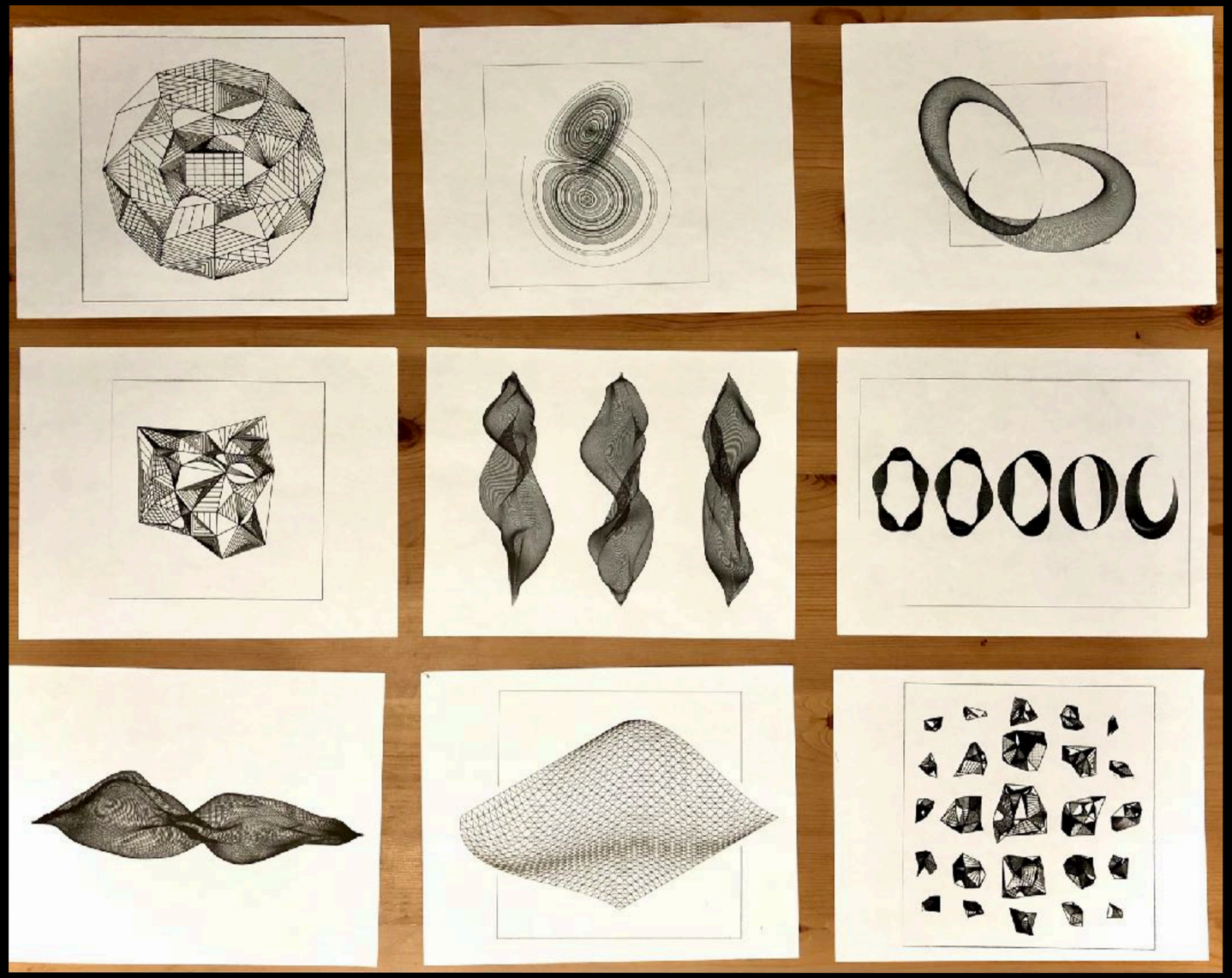
```
** art-gen
#+begin_src clojure
(defn fill-tri
  [tri]
  (let [trif (get tri-fill-strategies (rand-int (count
tri-fill-strategies)))
        lines (trif tri (+ 9 (rand-int 7)))
        f (fn [pts]
            (let [sk (if (< 2 (count pts))
                       sketch-polygon
                       #(apply sketch-line %))]
              (-> pts sk (tf/style a-style))))]
    (el/g
     (apply el/g (map f lines)))))

(defn gen-art
  []
  (let [pts (random-pts 400 600 40)
        tris (:triangles
(delaunay-triangulation.core/triangulate pts))
        hull (map :pt (hull pts))
        trifa #(-> % sketch-polygon (tf/style a-style))
        trifb #(-> % sketch-polygon (tf/style b-style))
        [trioa triob] (split-at (* 0.25 (count tris)) (vec
(shuffle tris)))]
    (el/g
     (apply el/g (map fill-tri tris))
     (apply el/g (map trifa tris))
     (apply el/g (map trifb triob))
     (-> hull sketch-polygon (tf/style c-style)))))

#+end_src
```



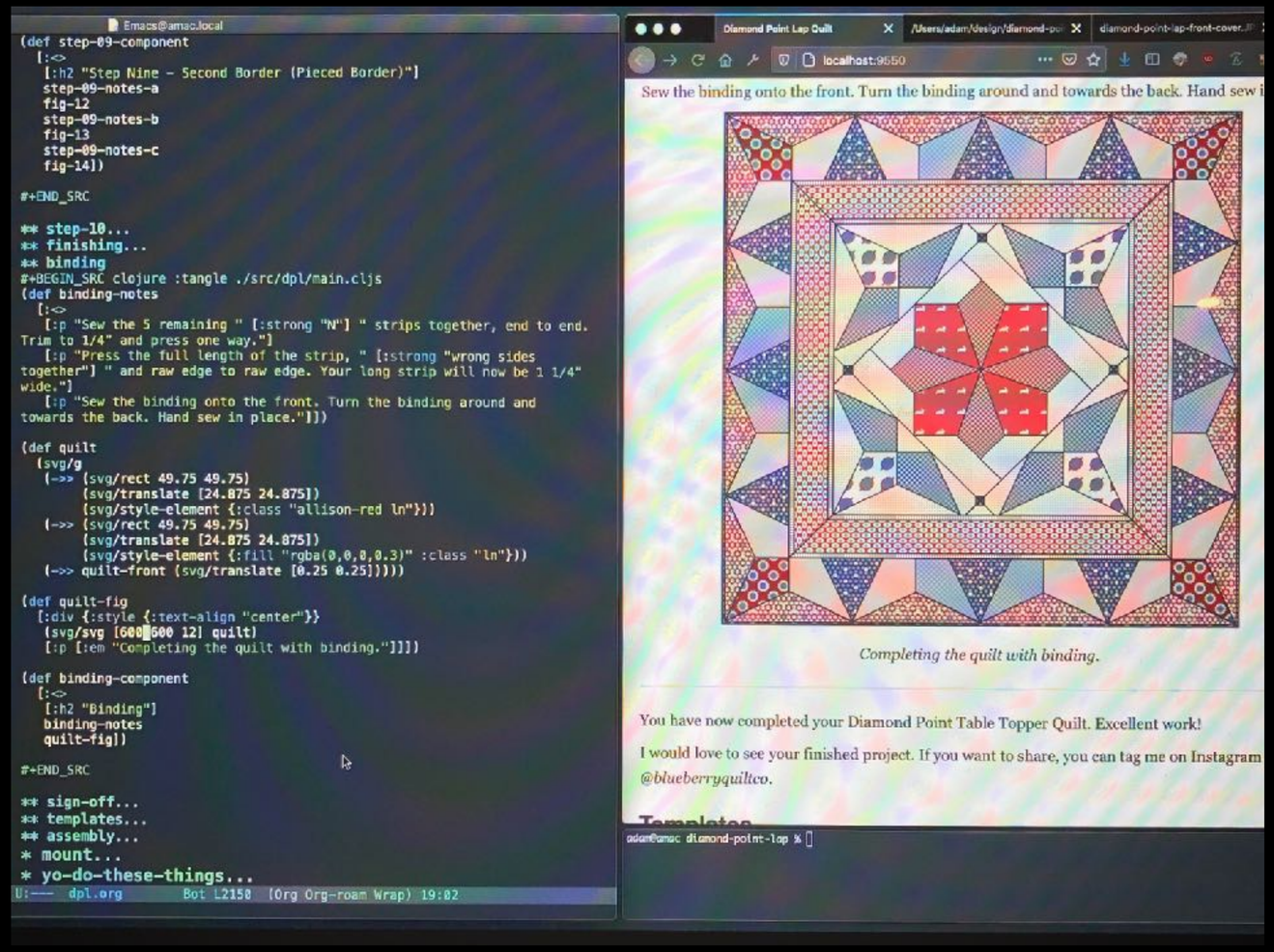
cypress.main>

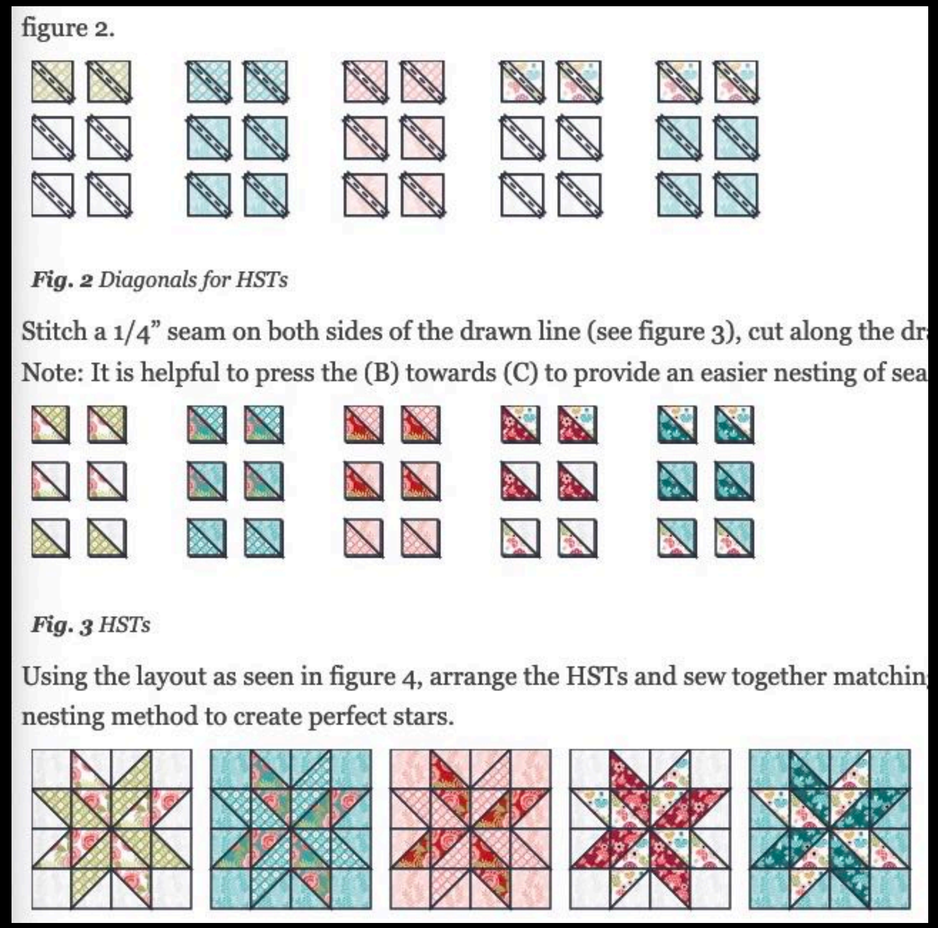
Generative Art

Pen Plotting

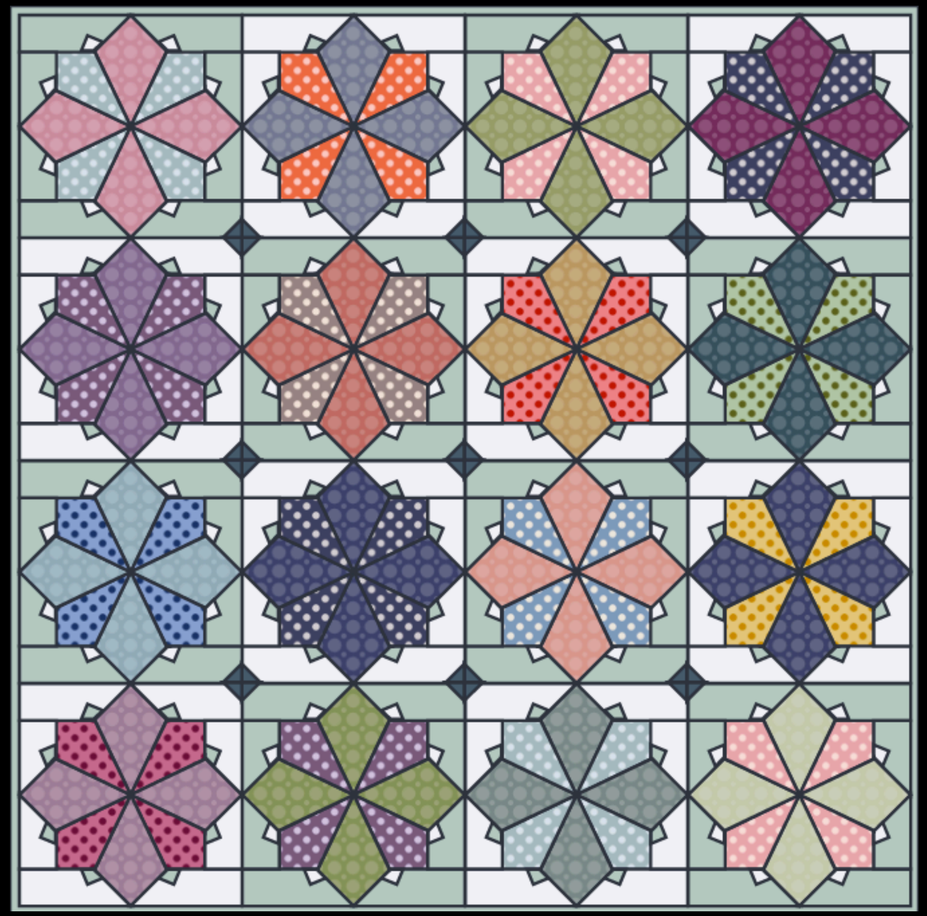


Quilt Patterns

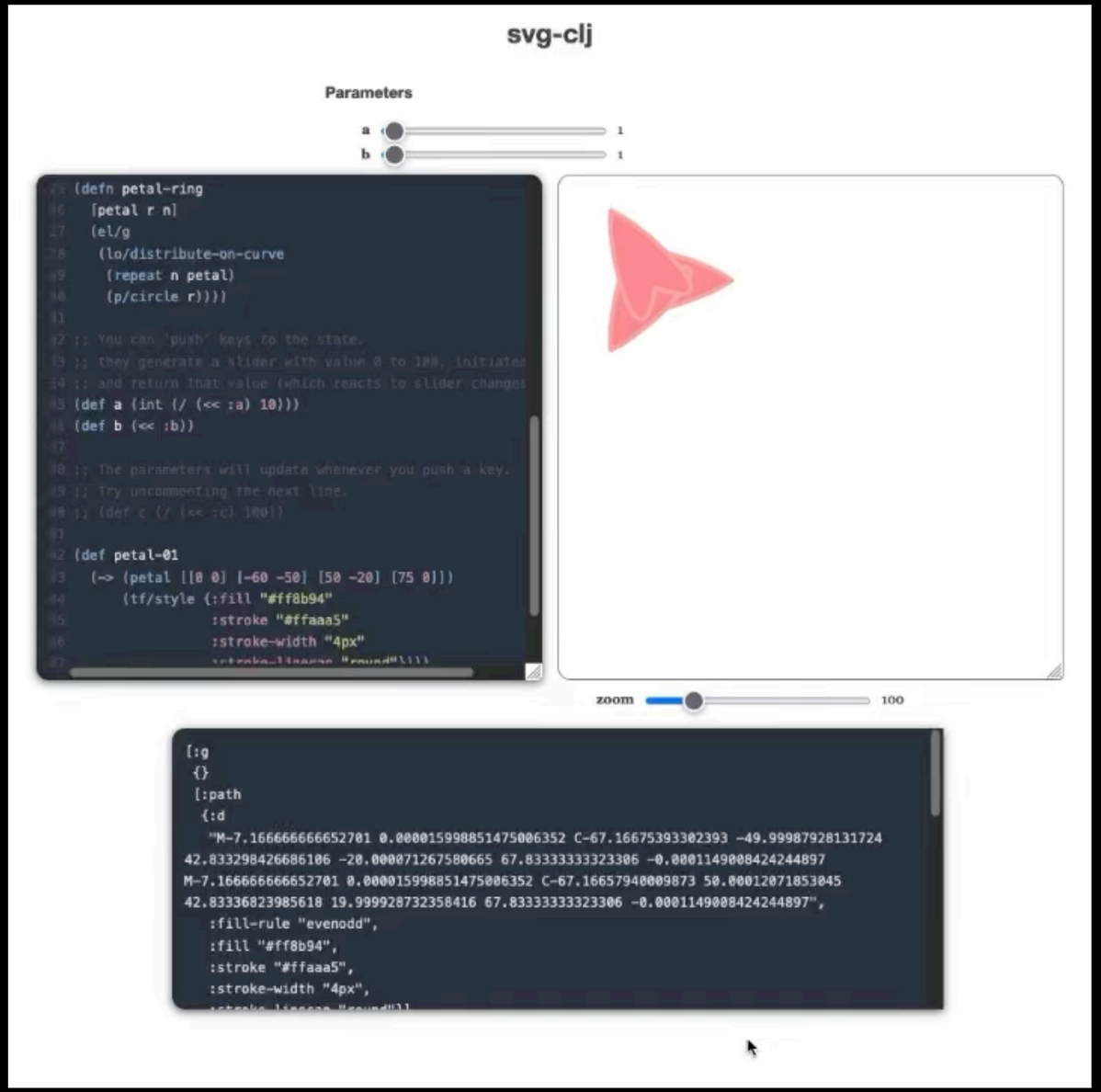




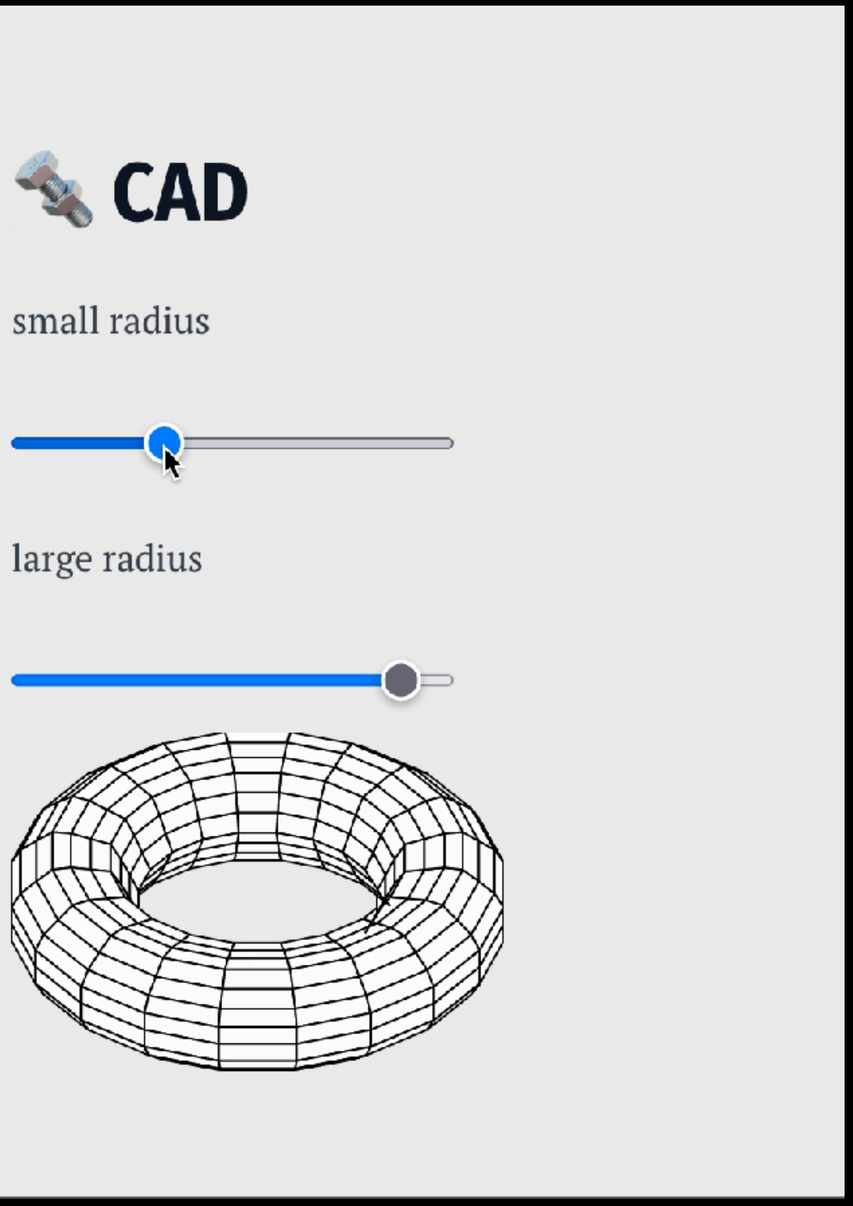
Quilt Patterns




Experiments with Interactivity

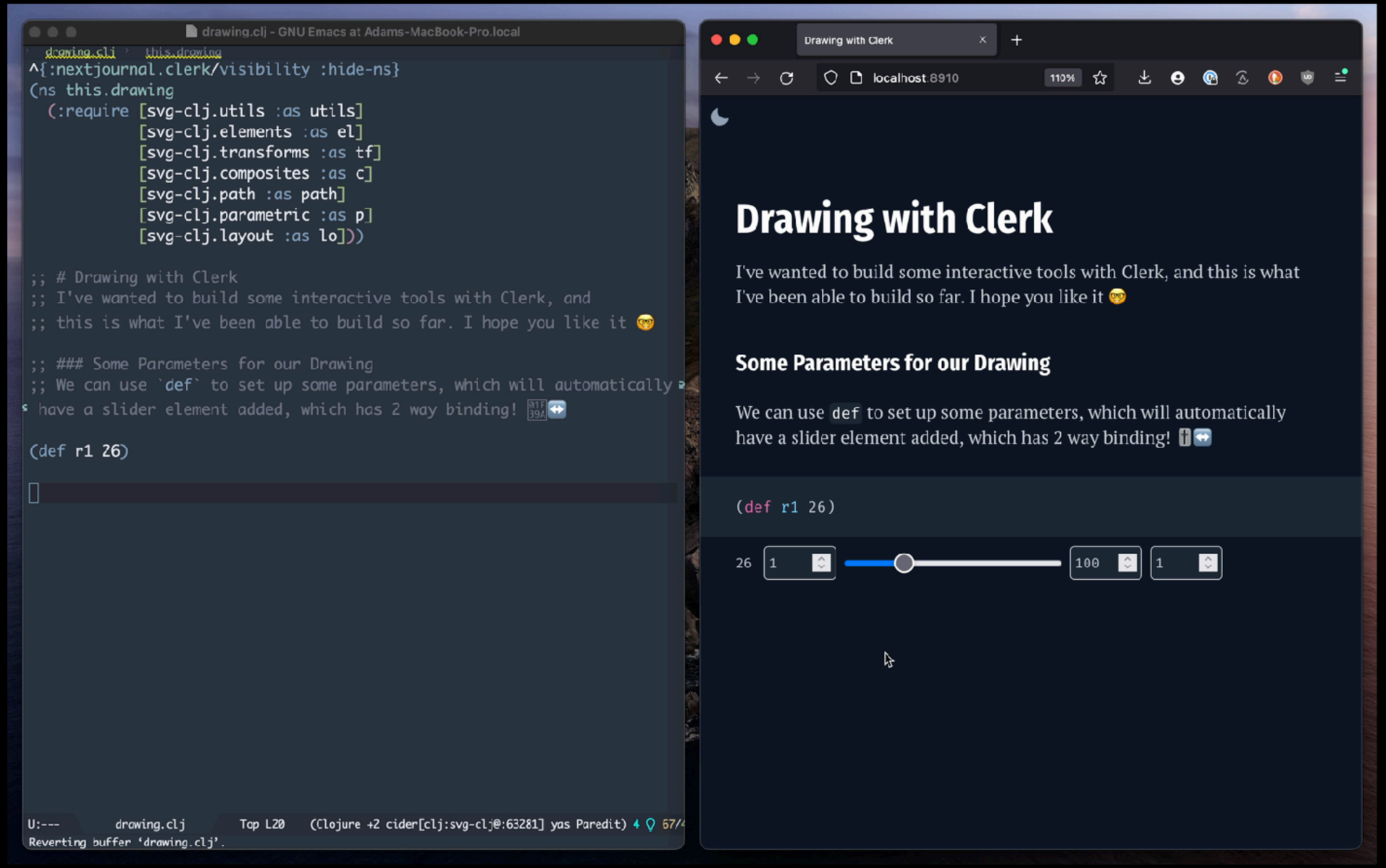


Svg-clj + sci + code mirror



Clerk notebooks

Bidirectional Clerk Notebook



Demo Time

Parting Thoughts

- Make stuff
  - For yourself
  - For others
  - For the joy of it
- Learn tools and skills by using them
- Have fun!
  - Making things and writing Clojure can be a joyful experience.
  - Fun is valuable and worthwhile