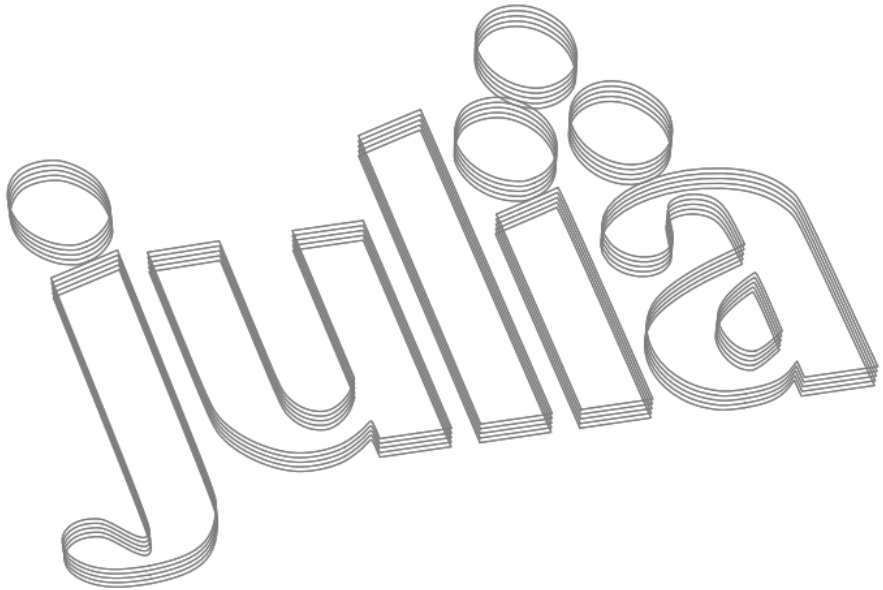


3D Printing with Julia

Presenting Euclid, a new high performance multi-material slicer

Jack Minardi
Co-Founder and Software Lead at Voxel8

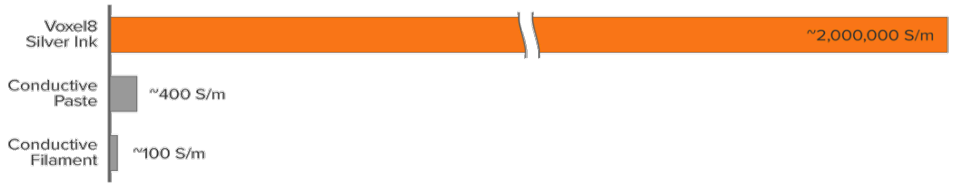
The Voxel8 Developer's Kit

EDISON AWARDS

"ONE OF THE 9 BEST IDEAS AT CES"
- FASTCOMPANY

New Platform for Prototyping 3D Electronics

- Dual modular print heads for multi-material printing
- Room temperature curable conductive ink, 20,000x more conductive than conductive filaments for 3D printing
- Integrated pneumatic dispensing allowing wide materials pallet
- Auto bed leveling and nozzle alignment with laser profilometer



What is Euclid?

Euclid is a high performance path-planner for 3D printing written in Julia.

Path-planning (a.k.a. “Slicing”) is the process of converting a 3D mesh file into a series of X, Y, Z moves that when executed will produce the desired part.

©2015 Voxel8 - 3D Printed Electronics

Why Julia?

- Path planning is computationally expensive.
- The expressiveness of a language matters.
- `PyCall` provides any missing “batteries”.
- I am young and naïve.

©2015 Voxel8 - 3D Printed Electronics

VOXEL8

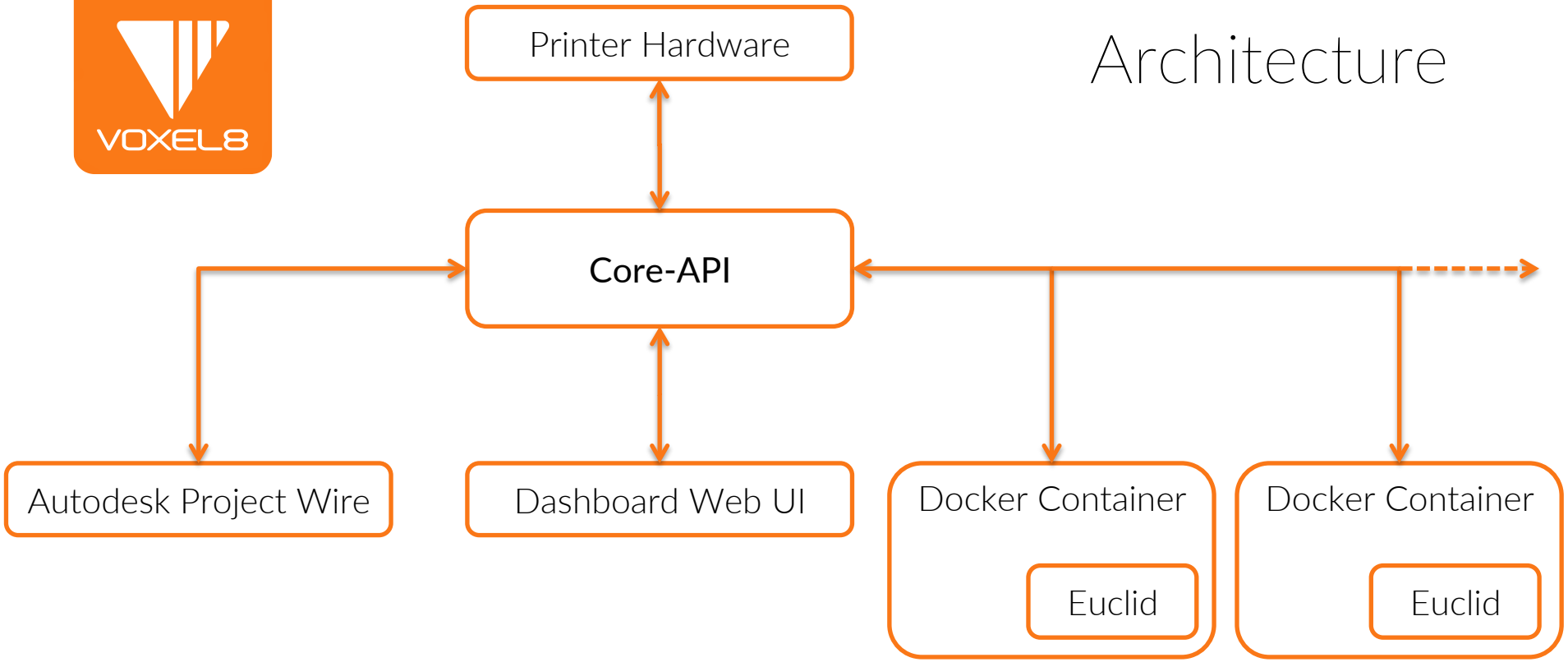
Why Julia?

"A company that gets software written faster and better will, all other things being equal, put its competitors out of business."

– Paul Graham on why Lisp was his secret weapon

©2015 Voxel8 - 3D Printed Electronics

Architecture



©2015 Voxel8 - 3D Printed Electronics

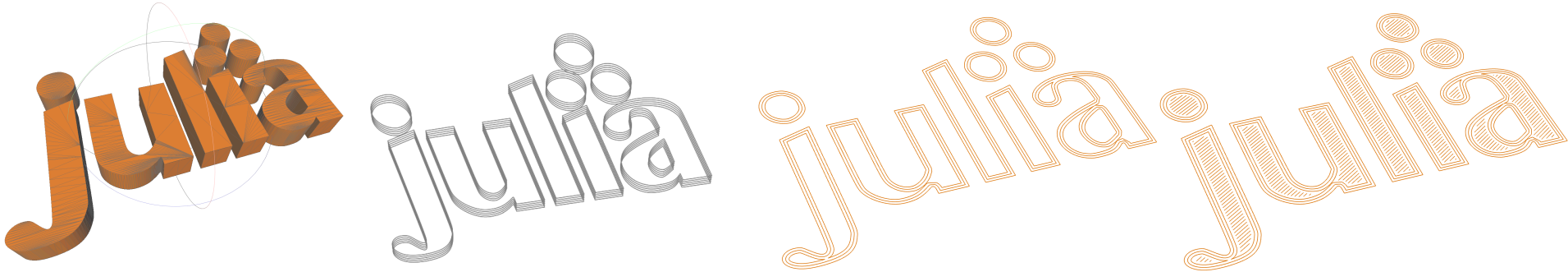
Euclid REQUIRE

ArgParse
BoundingBoxes
Compat
Coverage
Dates
Distances
Graphs
ImmutableArrays
JSON
Lint
Meshes
PyCall
ZMQ

©2015 Voxel8 - 3D Printed Electronics

Euclid Pipeline

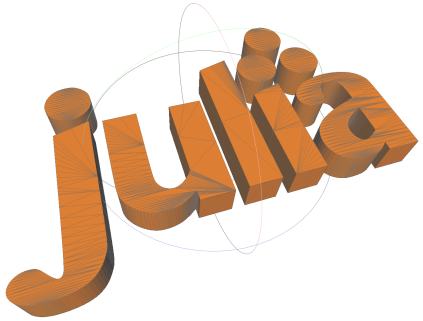
Design Model → Slice Mesh → Offset → Infill → Optimize Path Order → Output GCode



©2015 Voxel8 - 3D Printed Electronics

Design

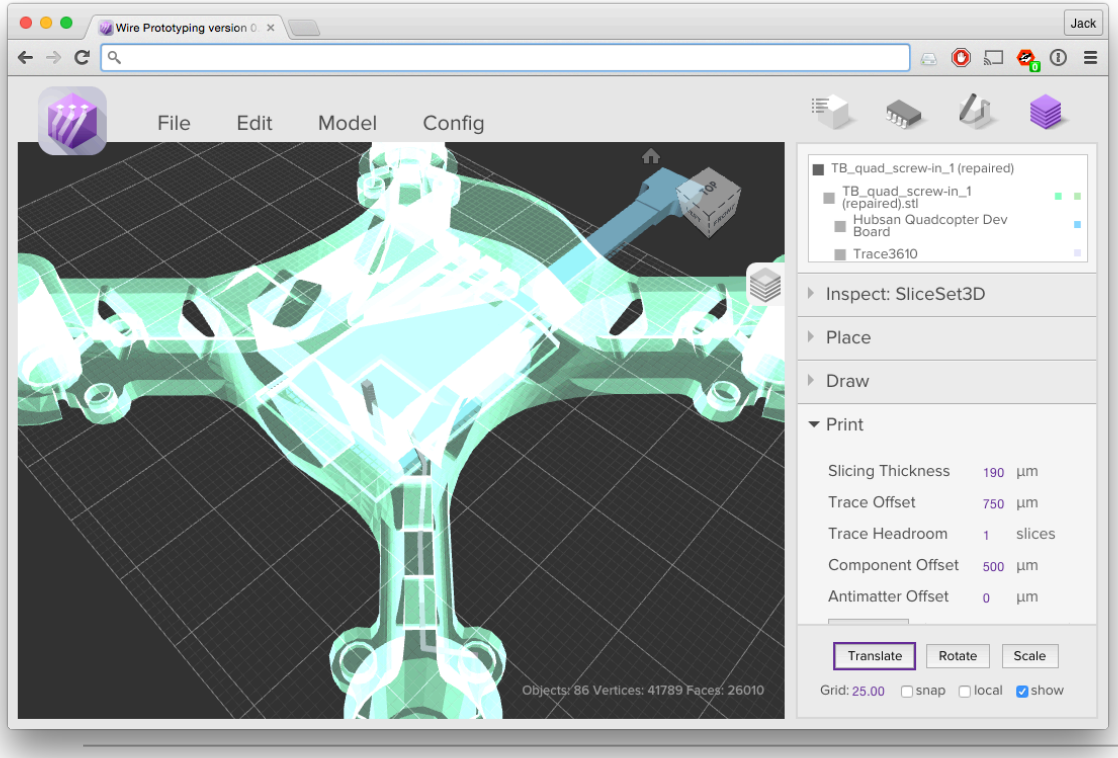
Design Model → Slice Mesh → Offset → Infill → Optimize Path Order → Output GCode



©2015 Voxel8 - 3D Printed Electronics

Design

Autodesk
Project Wire



©2015 Voxel8 - 3D Printed Electronics

Common Mesh File Formats

STL

```
facet normal ni nj nk
    outer loop
        vertex v1x v1y v1z
        vertex v2x v2y v2z
        vertex v3x v3y v3z
    endloop
endfacet
…
```

OBJ

```
v v1x v1y v1z
v v2x v2y v2z
v v3x v3y v3z
…
f 1 2 3
…
```

©2015 Voxel8 - 3D Printed Electronics

Meshes.jl

github.com/JuliaGeometry/Meshes.jl

The three main types in the Meshes.jl package are Vertex, Face, and Mesh.

The Mesh type is internally stored similarly to an OBJ file.

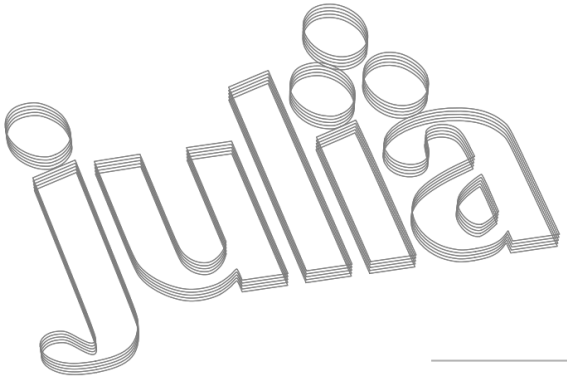
```
typealias Vertex Vector3{Float64}

immutable Face{T}
    v1::T
    v2::T
    v3::T
end

type Mesh{V, F} <: AbstractMesh{V, F}
    vertices::Vector{V}
    faces::Vector{F}
end
```

©2015 Voxel8 - 3D Printed Electronics

Slicing



©2015 Voxel8 - 3D Printed Electronics

Slicing

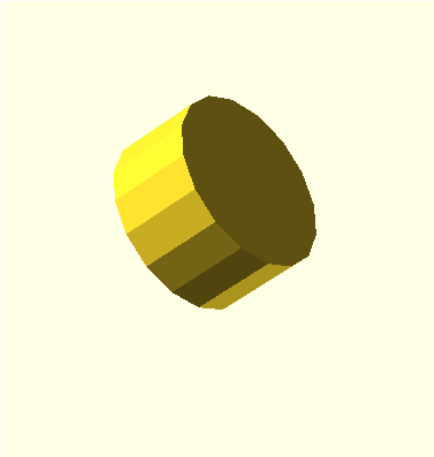
The mesh file is sliced with Meshes.jl

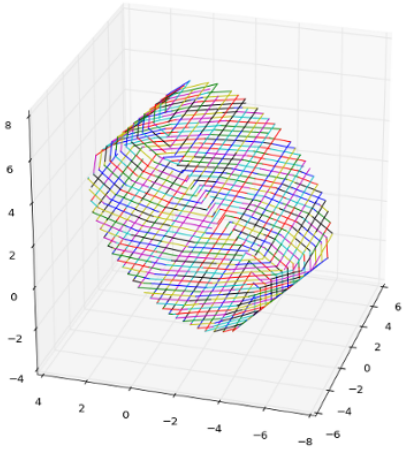
```
using Meshes

m = mesh("/path/to/mesh.stl")
heights = [1, 2, 3]
slices = slice(m, heights)
```

©2015 Voxel8 - 3D Printed Electronics

Slicing

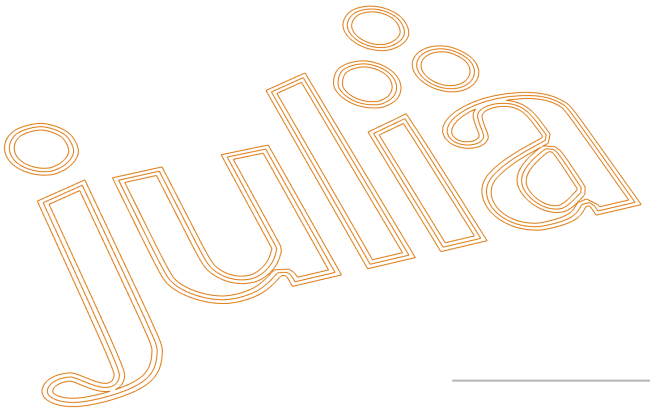




©2015 Voxel8 - 3D Printed Electronics

Offsetting Polygons

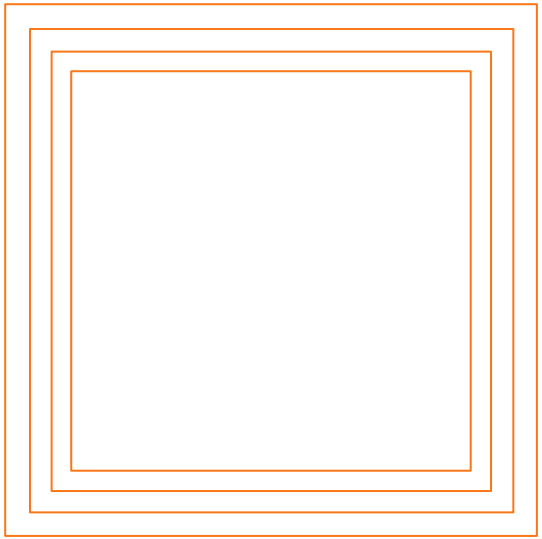
Design Model → Slice Mesh → **Offset** → Infill → Optimize Path Order → Output GCode



©2015 Voxel8 - 3D Printed Electronics

Offsetting Polygons

The inner offset with a distance **d** of a polygon **P** is the boundary of the set of points each of which is contained in **P** and has a distance greater than **d** from the boundary of **P**. [1]



©2015 Voxel8 - 3D Printed Electronics

1. Chen, McMains - Polygon Offsetting by Computing Winding Numbers. Proceedings of IDETC/CIE 2005

Offsetting Polygons

©2015 Voxel8 - 3D Printed Electronics

Offsetting Attempt 1

Homegrown solution – Polygons.jl

- Greiner-Hormann Algorithm for clipping
- Hormann-Agathos Algorithm for “Point in Polygon” tests

Abandoned because it was too slow and hard to debug.

©2015 Voxel8 - 3D Printed Electronics

Offsetting Attempt 2

Wrap Angus Johnson's Clipper Library

- Clipper is the gold standard library used by many other slicers
- Written in C++, our wrapper used Cxx.jl
- See our progress here: github.com/voxel8/Clipper.jl

Abandoned because it requires unstable Julia, unstable Cxx, unstable LLVM.

©2015 Voxel8 - 3D Printed Electronics

Offsetting Attempt 3

Use pyclipper with PyCall

- pyclipper is a cython wrapper for Clipper
- PyCall makes this very easy to interop with

This is the method we are currently using.

©2015 Voxel8 - 3D Printed Electronics

Offsetting Polygons

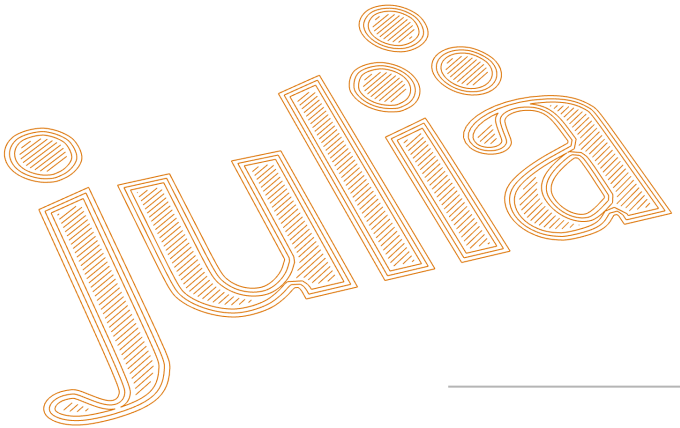
Ultimately when Cxx.jl is stable we will use it to interact with the Clipper C++ library.

©2015 Voxel8 - 3D Printed Electronics

VOXEL8

Infill

Design Model → Slice Mesh → Offset → **Infill** → Optimize Path Order → Output GCode

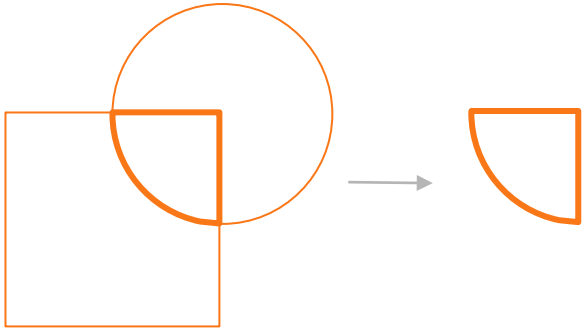


©2015 Voxel8 - 3D Printed Electronics

Infill

Infill utilizes clipping, which is finding the intersection of two polygons.

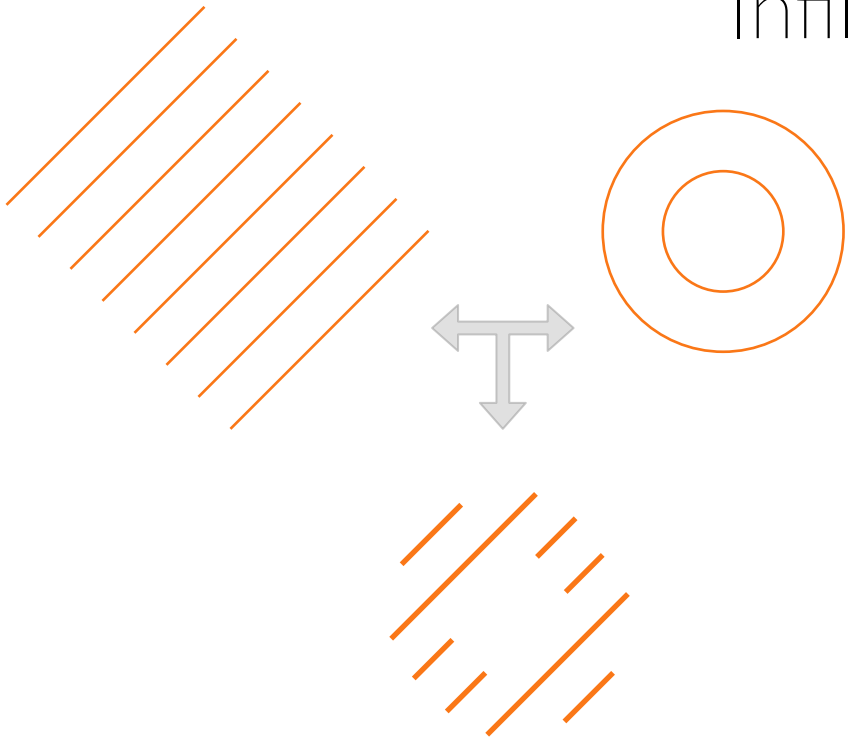
We also use the Clipper library here.



©2015 Voxel8 - 3D Printed Electronics

Infill

Infill is generated by clipping a rectilinear pattern against the inner-most offset.



©2015 Voxel8 - 3D Printed Electronics

Optimize Path Order

©2015 Voxel8 - 3D Printed Electronics

Optimize Path Order

At this point we have all the lines we need to execute to build the part, we just need to choose the order in which we get to them.

This is essentially the traveling salesman problem

Goals

- Minimize print time
- Maximize print quality
- Minimize travel (moves w/o extrusion)
- Minimize “retractions”

©2015 Voxel8 - 3D Printed Electronics

GCode

©2015 Voxel8 - 3D Printed Electronics

GCode

GCode is essentially a series of X, Y, Z moves, how fast to execute them, and whether or not to extrude material while moving.

©2015 Voxel8 - 3D Printed Electronics

GCode

What is GCode?

- It was invented in the 1950s right here a MIT
- Conceived as a numerical control language for CNCs, has been adopted by the 3D printer community
- There are various flavors and implementations but they all share a common core.

©2015 Voxel8 - 3D Printed Electronics

GCode

```
G0 E-0.3 F960
G0 X12.5 Y70.23 F9000
G0 E0.32 F960
G1 X121.69 Y70.44 E0.08 F3000
G0 X121.69 Y70.75 F9000
```

©2015 Voxel8 - 3D Printed Electronics

Euclid Command Line Interface

```
$ euclid \
    --config /path/to/config.json \
    /path/to/plastic.stl \
    /path/to/conductor.stl
```

©2015 Voxel8 - 3D Printed Electronics

Dockerizing Euclid

What is Docker?

"Docker allows you to package an application with all of its dependencies into a standardized unit for software development."

Why?

- No dependency on AWS's tech
  - .ebextensions
  - Amazon Linux
- Easy to test the environment and deploy anywhere Docker runs.

©2015 Voxel8 - 3D Printed Electronics

Dockerfile

```
# Start from a basic installation of Julia v0.3 on Ubuntu.
FROM julia-on-ubuntu

# Install package dependencies.
RUN apt-get update && \
  apt-get install -y \
    python-pip \
    python-scipy && \
  pip install pyclipper==0.9.3b0

# Julia package installs write to the home directory and shouldn't be
# run as root.
USER julia

# Install dependencies and euclid.
RUN julia -e 'Pkg.clone("git@github.com:Voxel8/Polygons.jl.git"); \
                  Pkg.clone("git@github.com:Voxel8/Euclid.git"); \
                  Pkg.build("Euclid")'

# Expose euclid's listen port.
EXPOSE 2014
```

©2015 Voxel8 - 3D Printed Electronics

Core-API

We've built a Ruby on Rails server that handles incoming requests and sends them to the Docker container.

The Rails server provides a RESTful API to Euclid.

Endpoint	Methods
/jobs	POST
/jobs/:id	GET
/configs	GET, POST
/configs/:id	GET, PUT, DELETE

©2015 Voxel8 - 3D Printed Electronics

Core-API

Client Workflow

- Client POSTs to jobs/ with a config_id and the model data.
- Core-API returns a Job ID
- Client can request details of any job by Job ID

©2015 Voxel8 - 3D Printed Electronics

Core-API

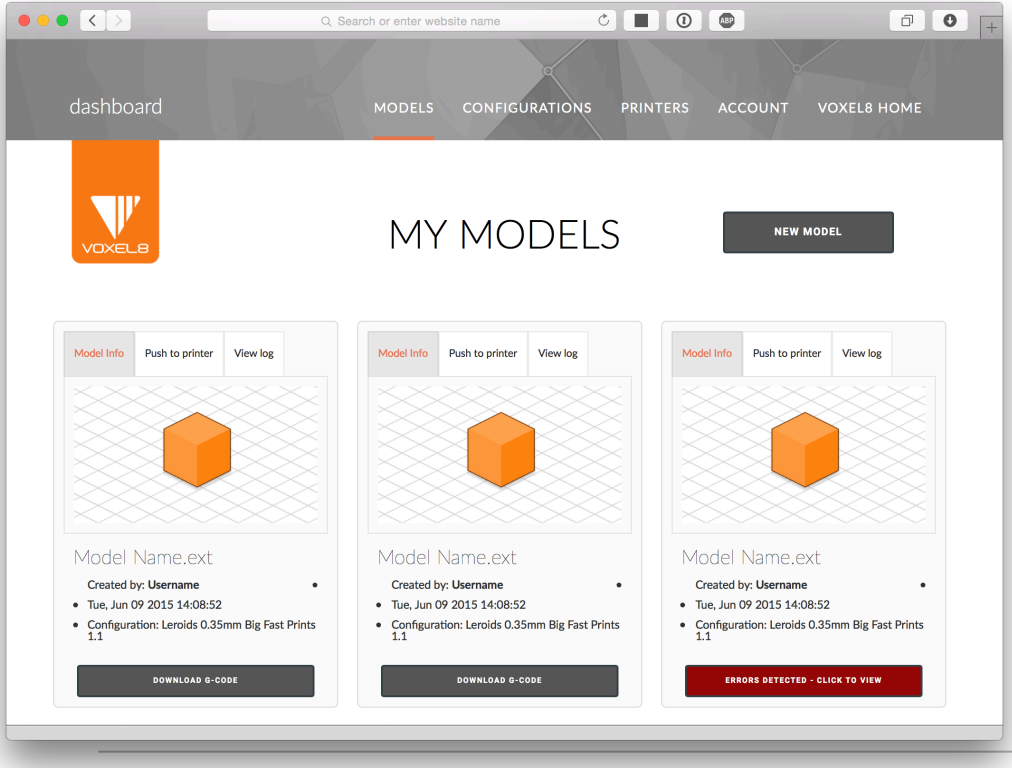
To start the slice job we simply run the following command from Rails:

```
$ docker exec '#{container_id}' euclid --config '#{config}' -o '#{outfile}' #{infiles}
```

When Euclid finishes it writes the GCode file to a shared directory that is accessible from Rails. This file is then uploaded to an S3 bucket.

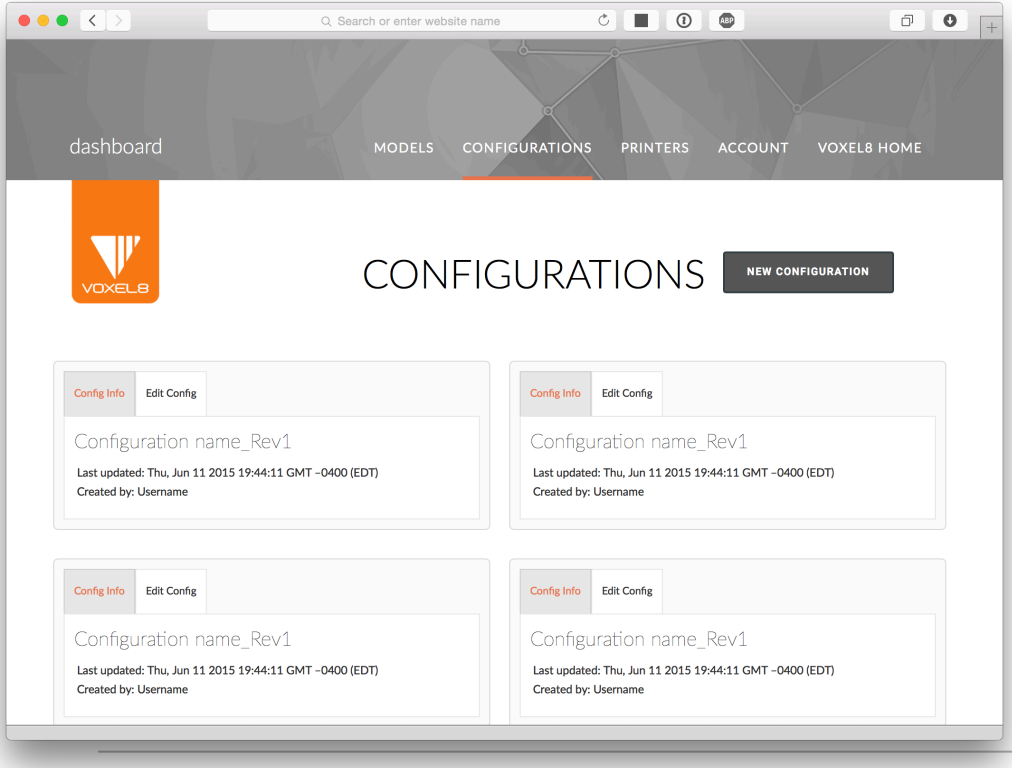
©2015 Voxel8 - 3D Printed Electronics

Dashboard



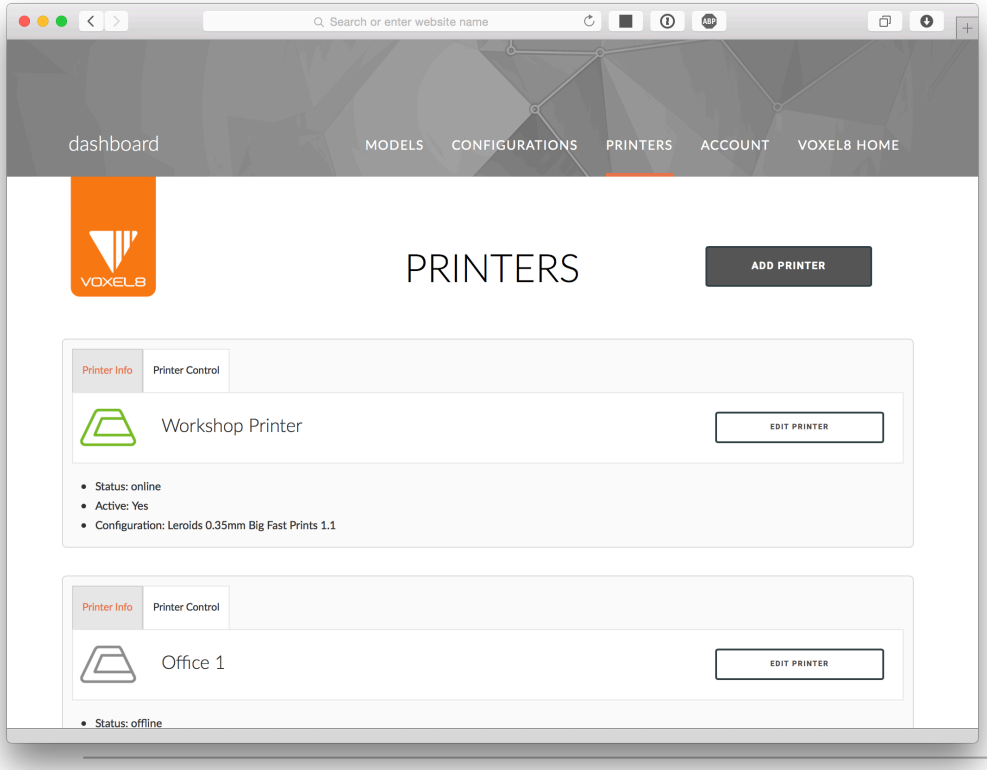
©2015 Voxel8 - 3D Printed Electronics

Dashboard



©2015 Voxel8 - 3D Printed Electronics

Dashboard



©2015 Voxel8 - 3D Printed Electronics

Conclusion

- The speed at which we can add features has been great for productivity.
- Even though Julia is young, it has a growing number of useful packages.
- Core language has been stable and fast.
- Most packages (that claim stability) have been very stable.
- It is easy to install and manage packages (as they are just git repos)
- If there is no package available, PyCall always works and has plenty of "batteries".
- Even though Julia wasn't designed for the web, Docker makes it easy to put in the cloud.

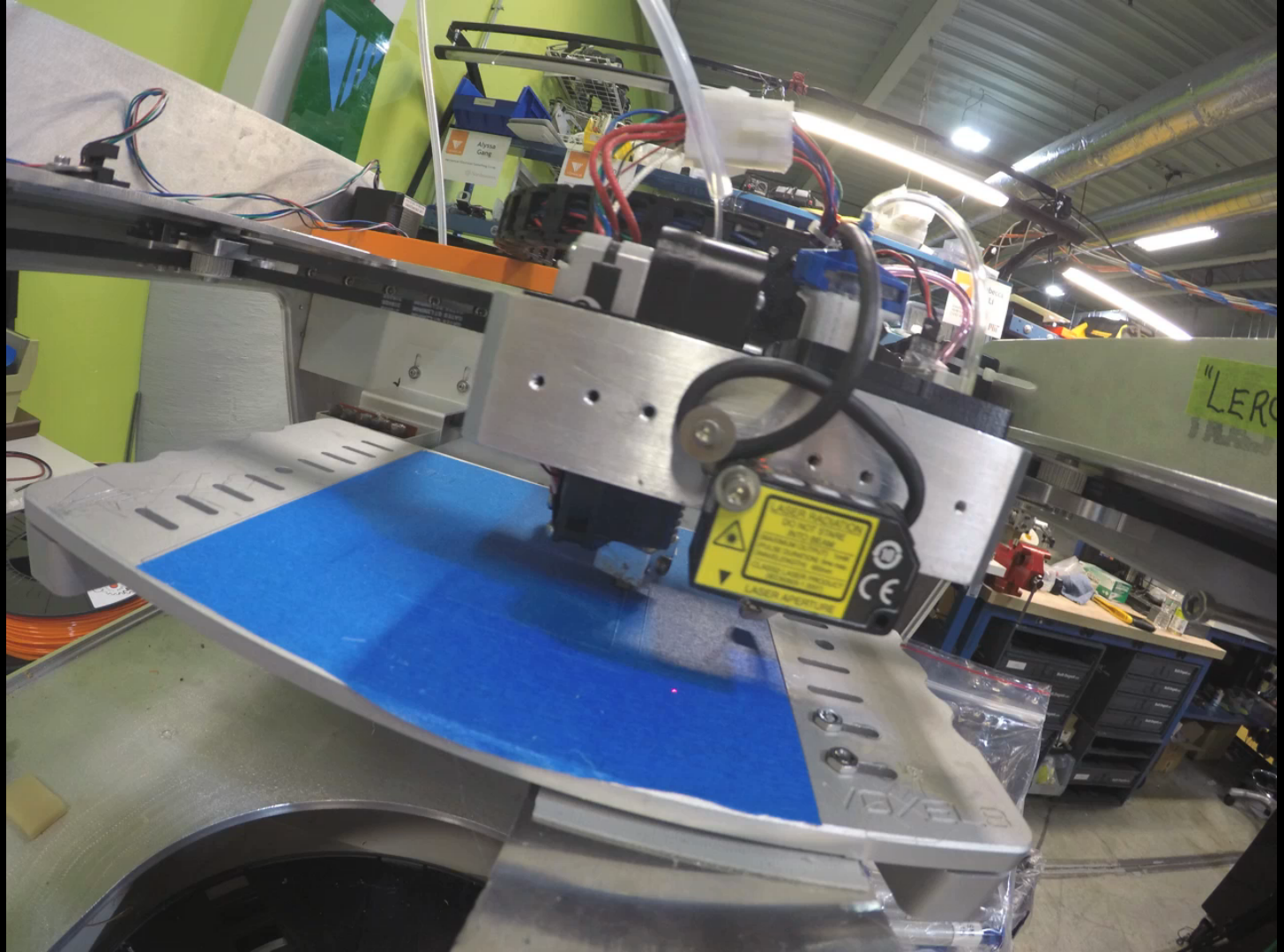
©2015 Voxel8 - 3D Printed Electronics

Conclusion

Wishlist

- Julia 0.4 stable release
- Package pre-compilation
- Docstrings
- Stable Cxx.jl
- Debugger

©2015 Voxel8 - 3D Printed Electronics


LASER RADIATION
DO NOT STARE INTO BEAM
LASER APERTURE
CE

Questions?

Jack Minardi
Voxel8

@jackminardi
@voxel8co