

# ImplicitCAD: Haskell all of the Things

Julia Longtin

ImplicitCAD Project

*julia.longtin@gmail.com*

February 28, 2020

## Source Code

- <https://github.com/colah/ImplicitCAD/>

# Who am I?

- Free Software Developer for 20+ years
  - Contributor to OpenEMR, FAI, TinTin++, LinuxPMI,...
- Work at Wire.com
  - The presented work has no affiliation with my employer
- Maintainer of ImplicitCAD
- Author of HSlice

# Who am I?

- Have run two Hackerspaces



- Spent 8+ years teaching 10 hours a week
- Tested ImplicitCAD on the members at HacDC

# Who am I?

- Transhumanist, 3D printer aficionado
  - ... would love to be bio-printing ...
- Scratch building mutant printers for 10+ years
- Created method for converting 3D prints to aluminium with microwaves (see: 31c3)

# What is ImplicitCAD?

- Programmatic 3D Modeling System
- Originally written by Christopher Olah in 2011
  - I Took over as maintainer in 2014
- Licensed under the AGPLv3+
- Written in Haskell

## Source Code

- <https://github.com/colah/ImplicitCAD/>

## Online Editor

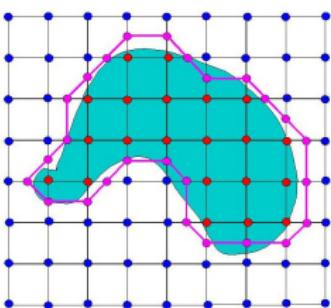
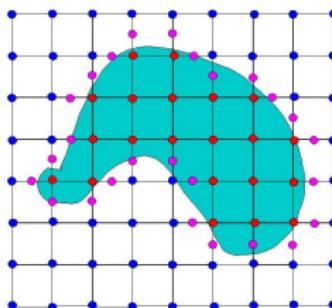
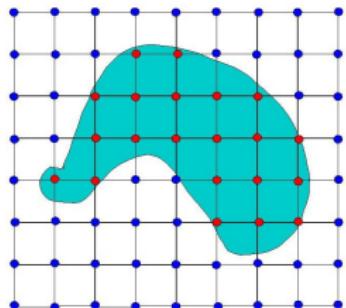
- <http://implicitcad.org/editor/>

# What is ImplicitCAD?

- Three Components:
  - 3D/2D Rendering Engine
  - SCAD Execution Engine
  - Library (Haskell DSL)

# ImplicitCAD's Rendering Engine

- Renders 3D: STL, OBJ... “Triangles on the outside”.
  - Uses marching cubes algorithm for triangulation functions
- Renders 2D: SVG, SCAD, PNG, DXF...
  - Uses modified marching squares, or ray tracing depending on format

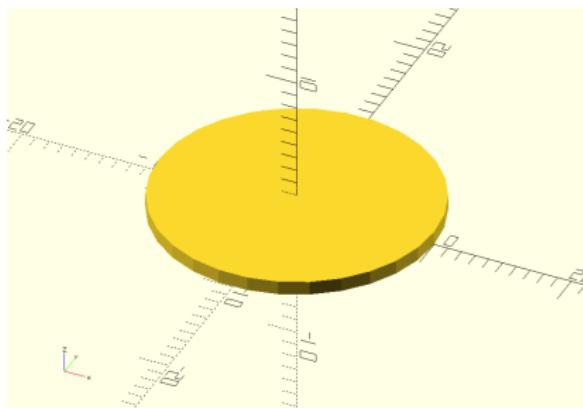


- Highly parallelized, uses all CPUs

# ImplicitCAD's SCAD Execution Engine

- OpenSCAD like, not quite OpenSCAD compatible
  - Lack of a real SCAD standard allows us to experiment with the language
- Leans toward primitive elements with more power

```
cylinder(h=1,r=10);
```



```
cylinder(h=1,r=10);
```



# ImplicitCAD's Haskell Library

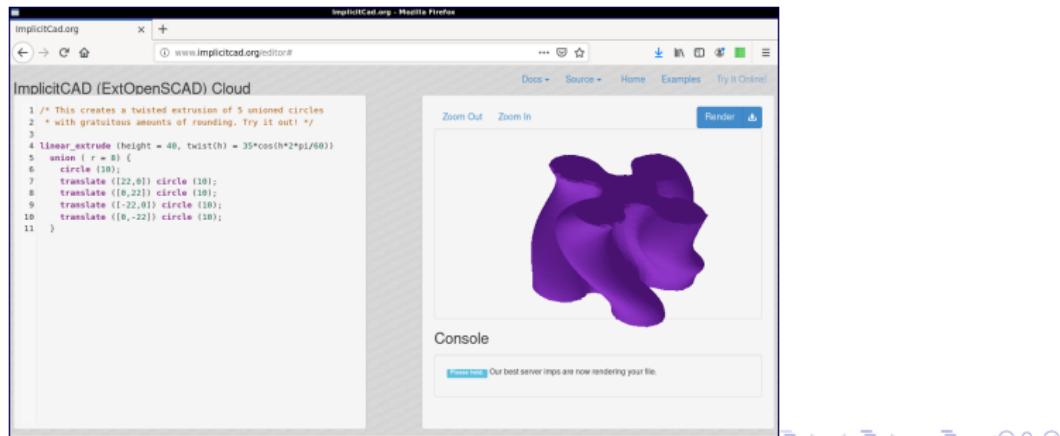
- Usable from simple Haskell programs (Haskell DSL!) to generate objects
- Capable of SCAD execution without modeling
- Separable expression executor

# ImplicitCAD Executables

- extopenscad: SCAD engine in command line form

```
[juri@localhost]$ extopenscad cylinder_example.escad
Loading File.
Processing File.
Rendering 3D object from cylinder_example.escad to cylinder_example.stl with resolution
0.3349119544218533 in box ((-10.0,-10.0,0.0),(10.0,10.0,1.0))
ExtrudeR 0.0 (Circle 10.0) 1.0
[juri@localhost]$ ]
```

- ImplicitSNAP: Engine backing the web site.



# Other SCAD Engines

## OpenSCAD - <https://openscad.org>

- Graphical Interface
- GPLv2 License

## OpenJSCAD - <https://openjscad.org>

- Web based User Interface
- MIT License

## Curv - <https://curv3d.org>

- Graphical Interface, GPU required
- Apache License
- Uses maths based on ImplicitCAD

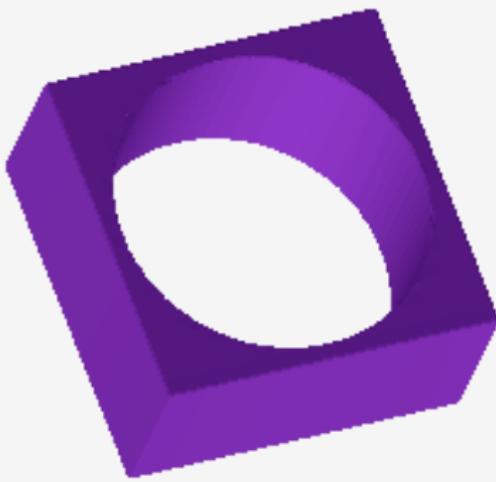
# Python SCAD engines?

- At least three rewrites of ImplicitCAD in Python
  - All Non-Free, one in use by military...

# Why ImplicitCAD?

- Simple SCAD Language

```
1 linear_extrude (height = 10)
2   difference () {
3     translate (-10, -10) square (20);
4     circle (r=9);
5 }
```



# Why ImplicitCAD?

- Simple Haskell DSL

```
-- A simple cylinder
import Prelude ((()))
import Graphics.Implicit(writeSTL, cylinder2)

main = writeSTL 0.2 "cylinder_example.stl" $ cylinder2 10 10 1
```



# Why ImplicitCAD?

- Nearly Haskell-Native intermediate state

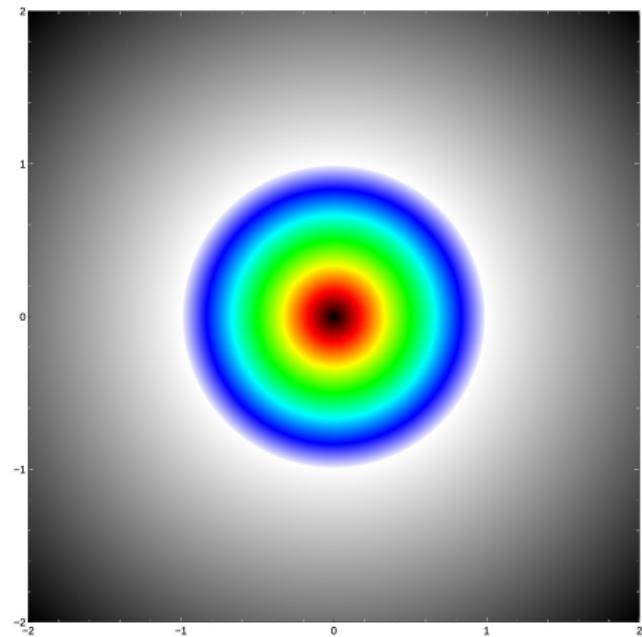
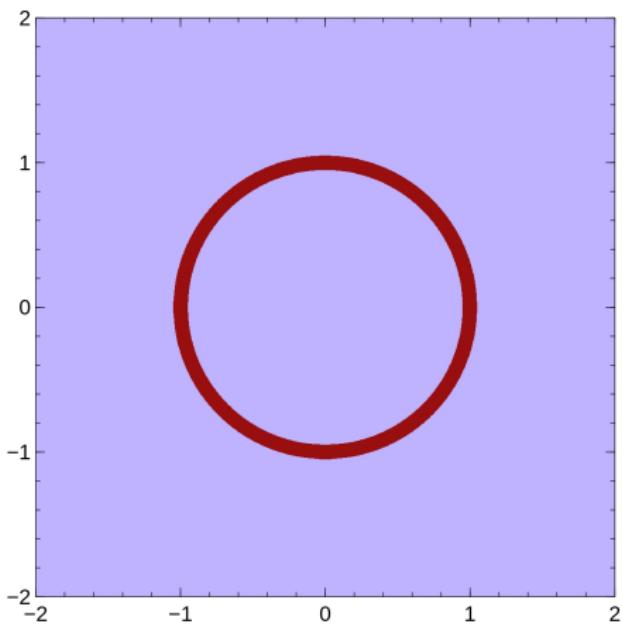
```
[juri@localhost]$ extopencad example.escad | grep ExtrudeR
ExtrudeR 0.0 (DifferenceR 0.0 [Translate2 (-10.0,-10.0) (RectR 0.0 (0.0,0.0)
(20.0,20.0)),Circle 9.0]) 10.0
[juri@localhost]$ ]
```

```
-- A simple cylinder
import Prelude ((()))
import Graphics.Implicit

main = writeSTL 0.2 "example.stl" $
    extrudeR 0.0 (differenceR 0.0 [translate (-10.0,-10.0) $
        rectR 0.0 (0.0,0.0) (20.0,20.0),
        circle 9.0]) 10.0
```

# Why ImplicitCAD?

- Implicit Constructive Solid Geometry



# Why ImplicitCAD?

- High Performance through parallel list comprehensions

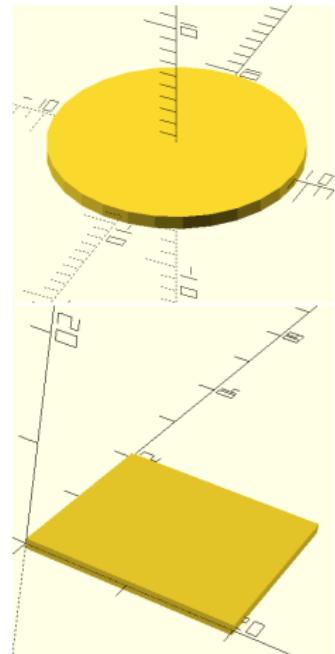
```
-- | Calculate mid points on X, and Y axis in 2D space.
midsY = [[
    interpolate (y0, objJX0Y0) (y1', objJX0Y1) (objJ $* x0) yres
    | x0 <- pks |
    | objJX0Y0 <- objJY0 | objJX0Y1 <- objJY1
] | y0 <- pVs | y1' <- tail pVs | objJY0 <- objJV | objJY1 <- tail objJV
] using parBuffer (max 1 $ div (fromN m) forcesteps) rdeepseq

midsX = [[
    interpolate (x0, objJX0Y0) (x1', objJX1Y0) (objJ *$ y0) xres
    | x0 <- pks | x1' <- tail pks | objJX0Y0 <- objJY0 | objJX1Y0 <- tail objJV
] | y0 <- pVs |
    | objJY0 <- objJV
] using parBuffer (max 1 $ div (fromN nx) forcesteps) rdeepseq

-- | Calculate segments for each side
segs = [[
    getSegs (x0,y0) (x1',y1') obj (objJX0Y0, objJX1Y0, objJX0Y1, objJX1Y1) (midR0, midR1, midB0, midB1)
    | x0<-pks | x1'<-tail pks | midR0<-nX' | midB1<-nY' | midR0<-nY'' | midR1<-tail nY''' | objJX0Y0<-objJY0 | objJX1Y0<-tail objJV | objJX0Y1<-objJY1 | objJX1Y1<-tail objJV
    | y0<-pVs | y1'<-tail pVs | midX' <-midX | midY' <-tail midX | midY'' <-midY | objJY0 <- objJV | objJY1 <- tail objJV
] using parBuffer (max 1 $ div (fromN $ nx*m) forcesteps) rdeepseq
```

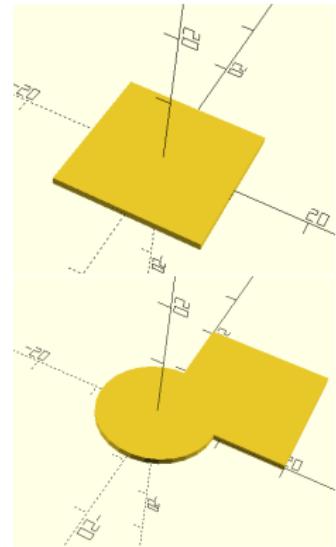
# Basic CSG in SCAD

- `circle(r=9);`
- `square(20);`



# Basic CSG in SCAD

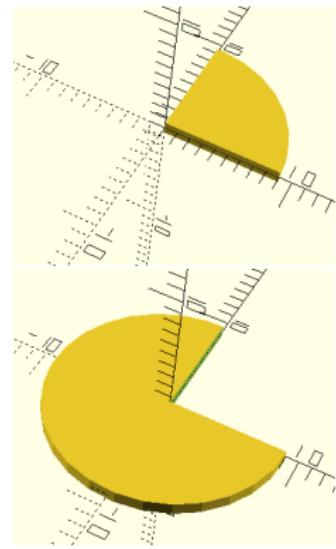
- translate([-10,-10]) square(20);



- union() { circle(r=9);  
square(20); }

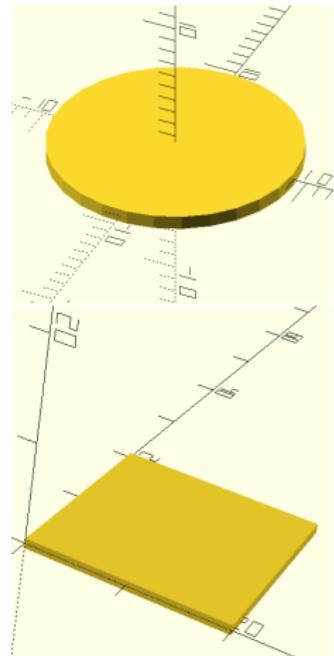
# Basic CSG in SCAD

- intersection() { circle(r=9); square(20); }
- difference() { circle(r=9); square(20); }



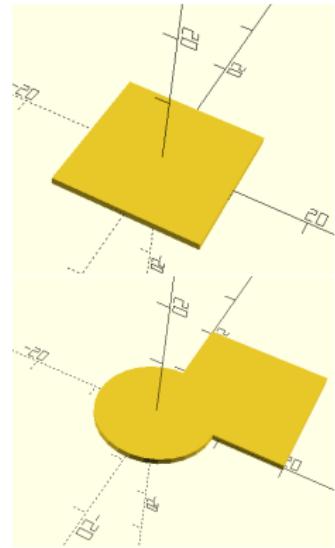
# Basic CSG in Haskell

- main = writeSVG 0.2  
“example.svg” \$ circle 9.0
- main = writeSVG 0.2  
“example.svg” \$ rectR 0 (0,0)  
(20,20)



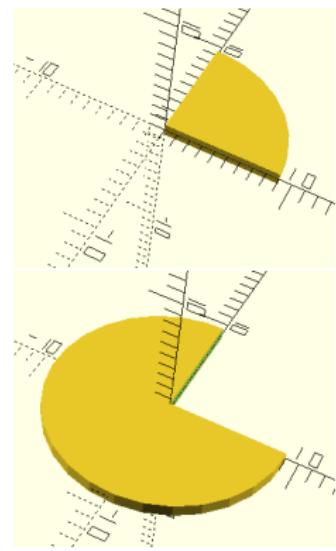
# Basic CSG in Haskell

- main = writeSVG 0.2  
“example.svg” \$ translate  
(-10,-10) \$ rectR 0 (0,0) (20,20)
- main = writeSVG 0.2  
“example.svg” \$ unionR 0  
[circle 9, rectR 0 (0,0) (20,20)]



# Basic CSG in Haskell

- main = writeSVG 0.2  
“example.svg” \\$ intersectR 0  
[circle 9, rectR 0 (0,0) (20,20)]
  
- main = writeSVG 0.2  
“example.svg” \\$ differenceR 0  
[circle 9, rectR 0 (0,0) (20,20)]



# Additional Primitives

- polygon();
- cube();
- sphere();
- cylinder();

# Additional Operations

- `scale() { }`
- `rotate() { }`
- `linear_extrude( ) { }`

# Flow Control

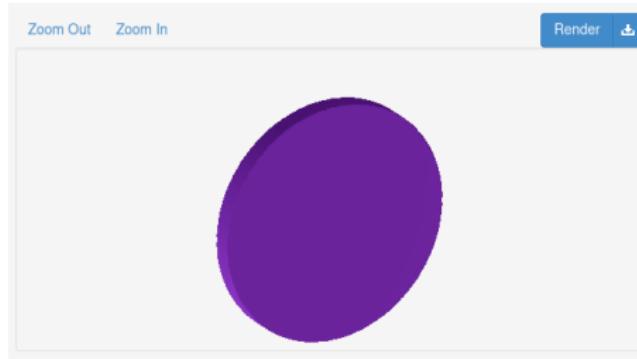
- module ( ) { }
- function ( ) ...
- if ... then ... else
- for ( ) { }
- include ...
- use ...
- echo ();

# SCAD Example: Disc



```
module disc_2d(diameter){  
    radius=diameter/2;  
    circle(r=radius);  
}  
module disc_3d(diameter, thickness){  
    linear_extrude(thickness)  
    disc_2d(diameter);  
}  
disc_3d(thickness=10, diameter=120);
```

# SCAD Example: Disc



```
module cylinder_3d(height, diameter){  
    linear_extrude(height){  
        circle(r=diameter/2);  
    }  
}  
cylinder_3d(height=10, diameter=120);
```

# SCAD Example: Bead



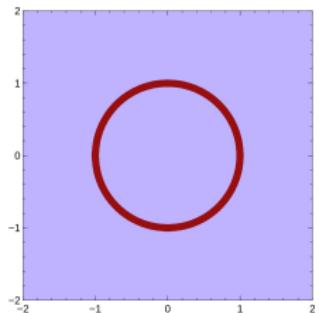
```
module bead_3d(height ,diameter ,hole_diameter) {  
    difference() {  
        cylinder(r=diameter/2, h=height);  
        cylinder(r=hole_diameter/2, h=height);  
    }  
}  
bead_3d(height=10, diameter=120, hole_diameter=20)
```

# Implicit CSG

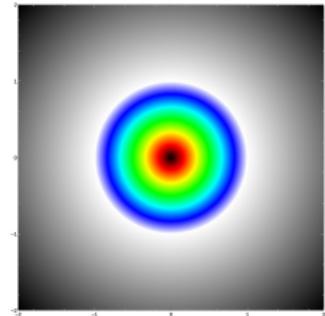
- CSG using Implicit functions
- Implicit functions are functions that define gradient to edge
- Interior of objects = negative value
- Exterior of objects = positive value

# Circles

- $x^2 + y^2 = r^2$

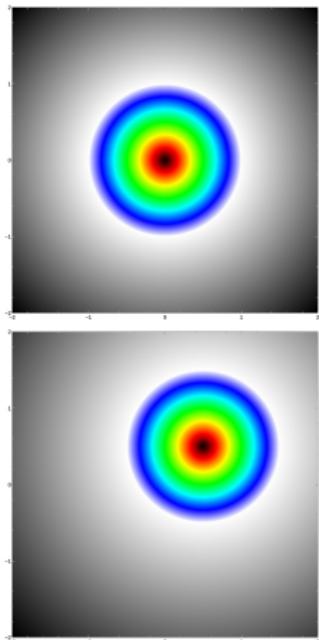


- $f(x, y) = \sqrt{x^2 + y^2} - 1$



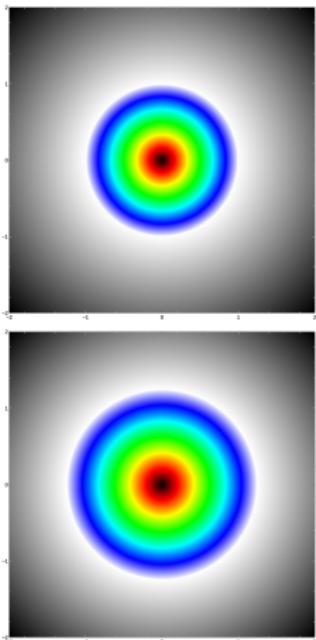
# Translation

- $f(x, y) = \sqrt{x^2 + y^2} - 1$
- $tf(x, y, tx, ty) = \sqrt{(x - tx)^2 + (y - ty)^2} - 1$



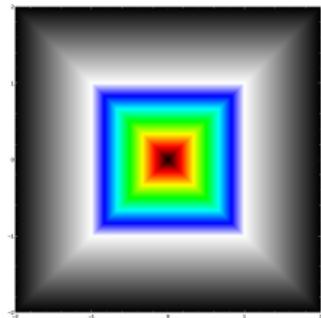
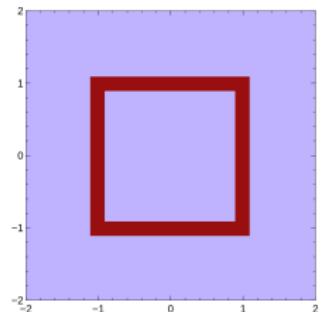
# Scaling

- $f(x, y) = \sqrt{x^2 + y^2} - 1$
- $sf(x, y, sx, sy) = \sqrt{((x/sx)^2 + (y/sy)^2)} - 1$



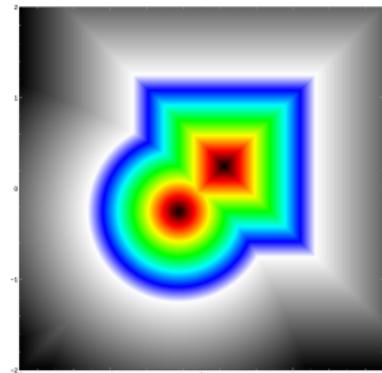
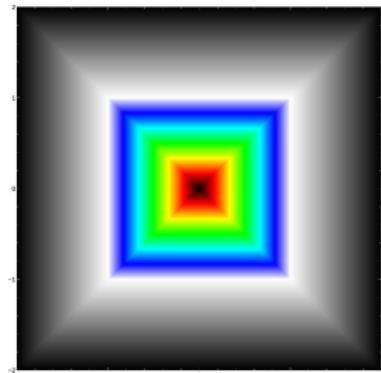
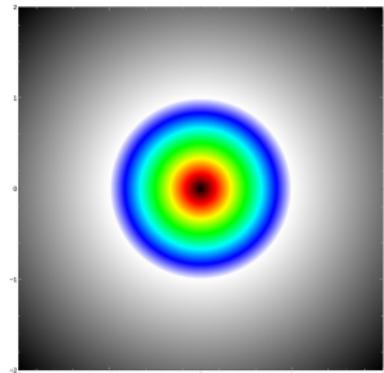
# Squares

- $\text{abs}(x) = r \wedge \text{abs}(y) < r \vee \text{abs}(y) = r \wedge \text{abs}(x) < r$
- $f(x, y) = \text{maximum}(\text{abs}(x), \text{abs}(y)) - 1$



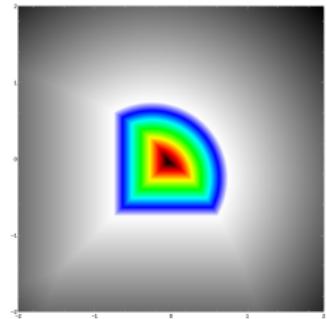
# Unioning

- $f(x, y) = \sqrt{x^2 + y^2} - 1$
- $f(x, y) = \max(\text{abs}(x), \text{abs}(y)) - 1$
- $f(x, y) = \min(\sqrt{(x+0.25)^2 + (y+0.25)^2} - 1, \max(\text{abs}(x - 0.25), \text{abs}(y - 0.25)) - 1)$



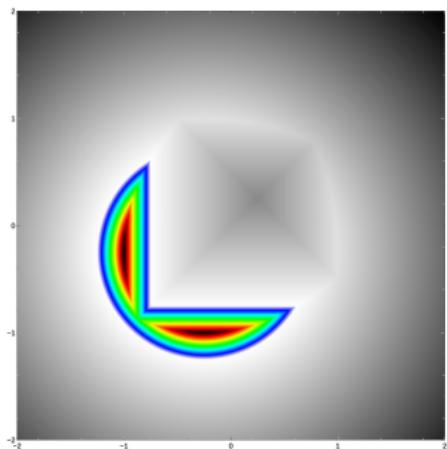
# Intersecting

- Taking the maximum of the objects
- $\max(\sqrt{(x + 0.25) * (x + 0.25) + (y + 0.25) * (y + 0.25)} - 1, \max(|x - 0.25|, |y - 0.25|)) - 1$



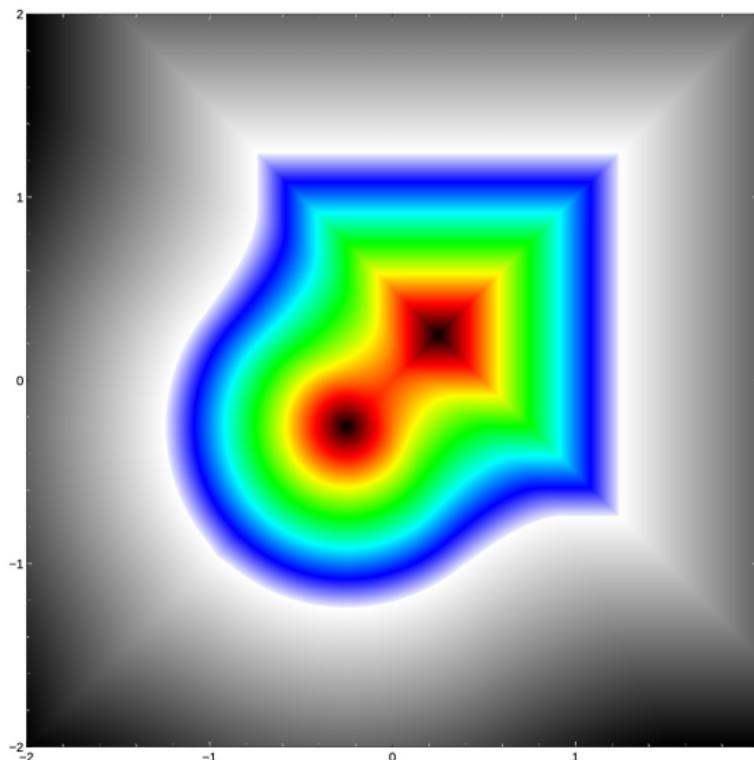
# Getting the Difference

- Taking the maximum of the item to be removed from, and the inverse of the items being removed
- $\max(\sqrt{(x+0.25)*(x+0.25)+(y+0.25)*(y+0.25)}) - 1, -(\max(\text{abs}(x-0.25), \text{abs}(y-0.25))) - 1)$



# Rounded Unions

$$\text{rmin}_r(a, b) = \begin{cases} \min(a, b) & |a - b| \geq r \\ b + r * \sin(\pi/4 + \arcsin(\frac{a-b}{r\sqrt{2}})) - r & |a - b| < r \end{cases}$$



# Implicit CSG Advantages

- Easy Rounding
- $\text{square}(x = 30, y = 30, r = 5)$
- $\text{square}(x = 30, y = 30, r = 15) == \text{Circle!}$
- Most primitives support rounding
- Rendering can be matched to your ability to print



# Implicit CSG Disadvantages

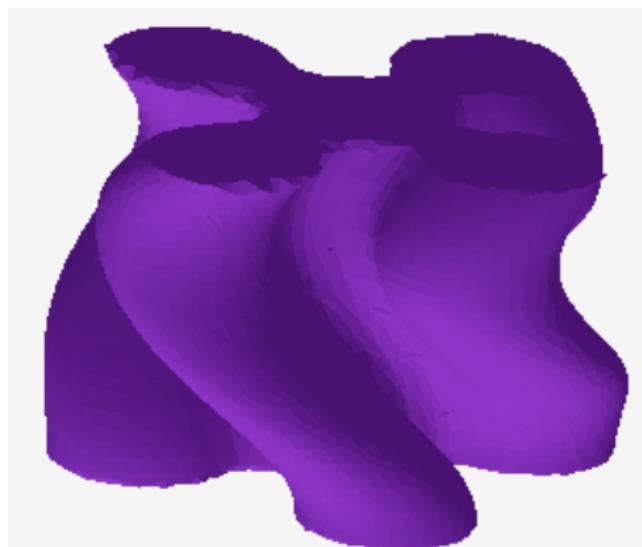
- Eats CPU
- Must track render area, as well as function stack
  - Tighter boxes mean less CPU wasted

# Haskell Implications

- Can generate and use functions
  - Hard to reason about them
  - Cannot print in intermediate form dump
- List comprehensions parallelize well
- ... Sadly little overlap between Haskellers and 3D modelers
- Always telling users to use “`+RTS -N -qg`” is painful

# Using Functions

```
linear_extrude (height = 40, twist(h) = 35*cos(h*2*pi/60))
union ( r = 8) {
    circle (10);
    translate ([22,0]) circle (10);
    translate ([0,22]) circle (10);
    translate([-22,0]) circle (10);
    translate ([0,-22]) circle (10);
}
```



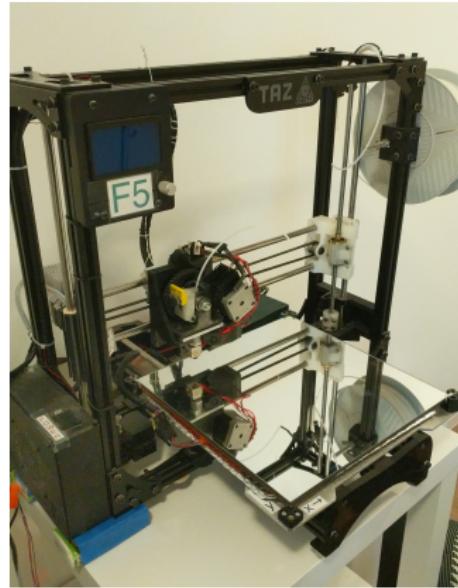
# linear\_extrude functional arguments

- $height(x, y) = value$
- $twist(h) = value$
- $scale(h) = value$
- $translate(h) = (x, y)$

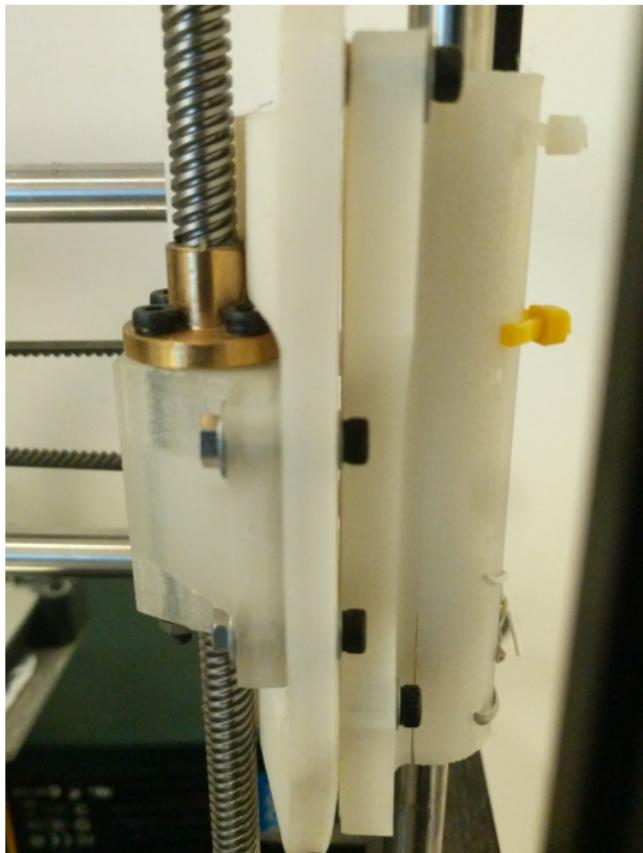
# Real World

- My 3D Printer

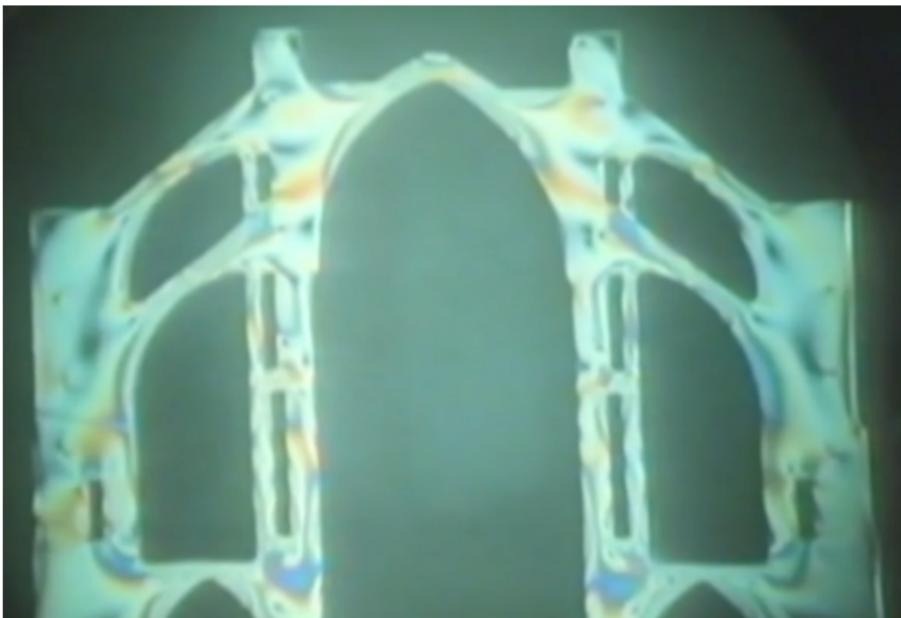
- LulzBot Taz 3-5ish
- Re-printing with ImplicitCAD
- STLs provided by LulzBot
  - STLs are not source code!



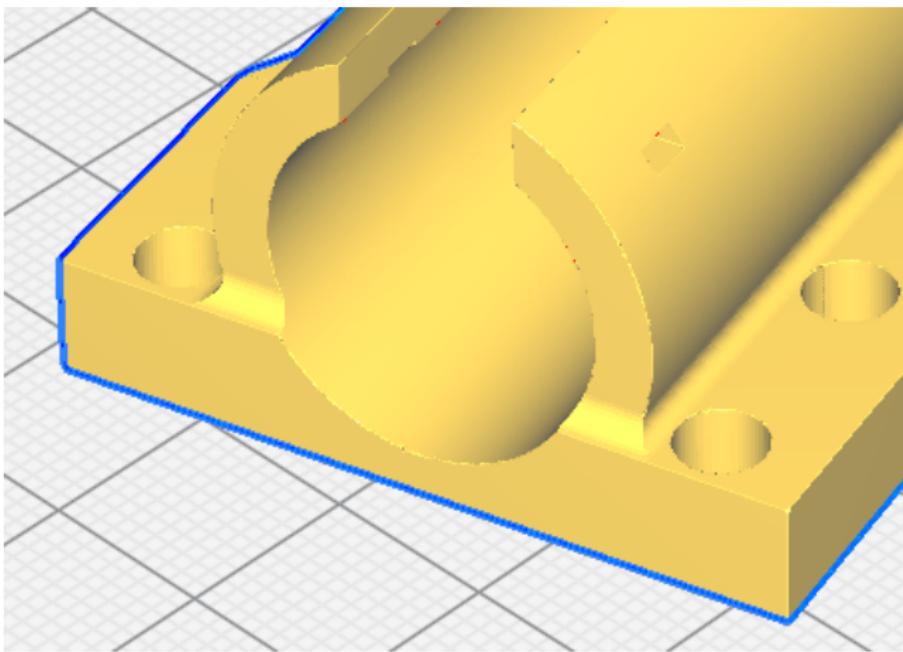
# Cracking Parts



# Stress under Pressure



# Rounding



# Future ImplicitCAD Functionality

- Functions as a type, so they can be reasoned about by the engine
- User-defined implicit operations

```
raw3D(bbox=function_that_returns_two_points ,  
      implicit(x,y,z)=function_that_returns_a_value);
```

# What is HSlice?

- Another Slicer!
- Originally written by Catherine Moresco and Noah Halford in 2016
  - I Took over as lead developer last year
- Licensed under the AGPLv3+
- Written in Haskell
- Goals:
  - Support slicing for printers with combined linear and rotational motion
  - Part of my effort to have a 100 percent haskell based 3D printing stack
- Abuses the expression evaluation from ImplicitCAD..

## Source Code

- <https://github.com/julialongtin/HSlice/>

# Functional OpenSCAD

The screenshot shows the OpenSCAD 2019.08.25 application window titled "debug.scad — OpenSCAD". The interface includes a menu bar (File, Edit, Design, View, Help), a toolbar, and a code editor pane containing the following SCAD code:

```
File Edit Design View Help
Editor
1
2 hypot = function(x, y) sqrt(x * x + y * y);
3 mean  = function(x, y) (x + y) / 2;
4
5 func  = function(x) x == 0
6   ? function(x, y) x + y
7   : function(x, y) x - y;
8
9 module m(h, x, y) {
10   echo(h = h(x, y));
11   translate([h(x, y), 0, 0]) cube(1);
12 }
13
14 echo(func0 = func(0)(8, 3));
15 echo(func1 = func(1)(8, 3));
16
17 echo(l = let(f = function(x) x) sqrt(f(25)));
18
19 echo(f = (function(x) sqrt(x))(49));
20
21 color("red") m(hypot, 3, 4);
22 color("green") m(mean, 5, 8);
23 color("blue") m(function(x, y) sqrt(x + y), 5, 6);
24 |
```

The 3D viewports show three cubes: a red cube at the origin, a green cube rotated along the x-axis, and a blue cube rotated along the y-axis.

The "Console" pane displays the following output:

```
Parsing design (AST generation)...
Compiling design (CSG Tree generation)...
ECHO: func0 = 11
ECHO: func1 = 5
ECHO: l = 5
ECHO: f = 7
ECHO: h = 5
ECHO: h = 6.5
ECHO: h = 3.31662
Compiling design (CSG Products generation)...
Geometries in cache: 1
Geometry cache size in bytes: 728
CGAL Polyhedrons in cache: 0
CGAL cache size in bytes: 0
Compiling design (CSG Products normalization)...
Normalized CSG tree has 3 elements
Compile and preview finished.
Total rendering time: 0 hours, 0 minutes, 0 seconds
```

At the bottom, the Viewport settings are shown as "translate = [ 4.03 2.06 0.19 ] rotate = [ 55.70 0.00 15.90 ] distance = 15.29 (872x388)". The status bar indicates "OpenSCAD 2019.08.25".

# ExplicitCAD

- A graphical front end for ImplicitCAD
- Originally written by Kliment
- Licensed under the GPLv3
- Written in C++
- Goals:
  - Give ImplicitCAD a better look and feel for non-command line users

## Source Code

- <https://github.com/kliment/explicitcad>

-Fin-

# Questions?