Implicit kernels for solid modeling

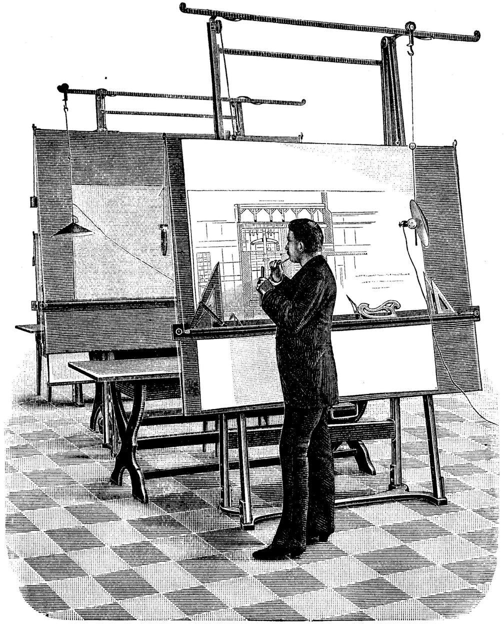
Matt Keeter
2018–01–19

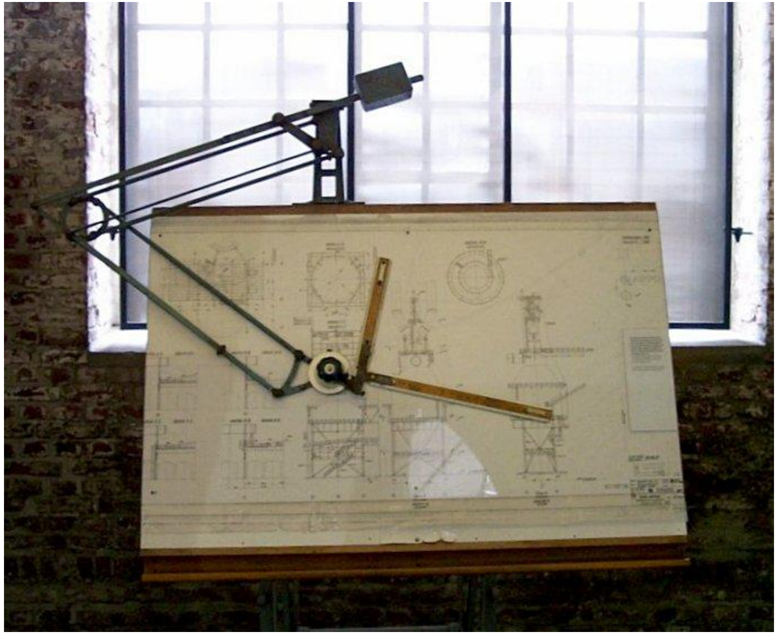
matt.j.keeter@gmail.com
mattkeeter.com

A Brief* History of CAD

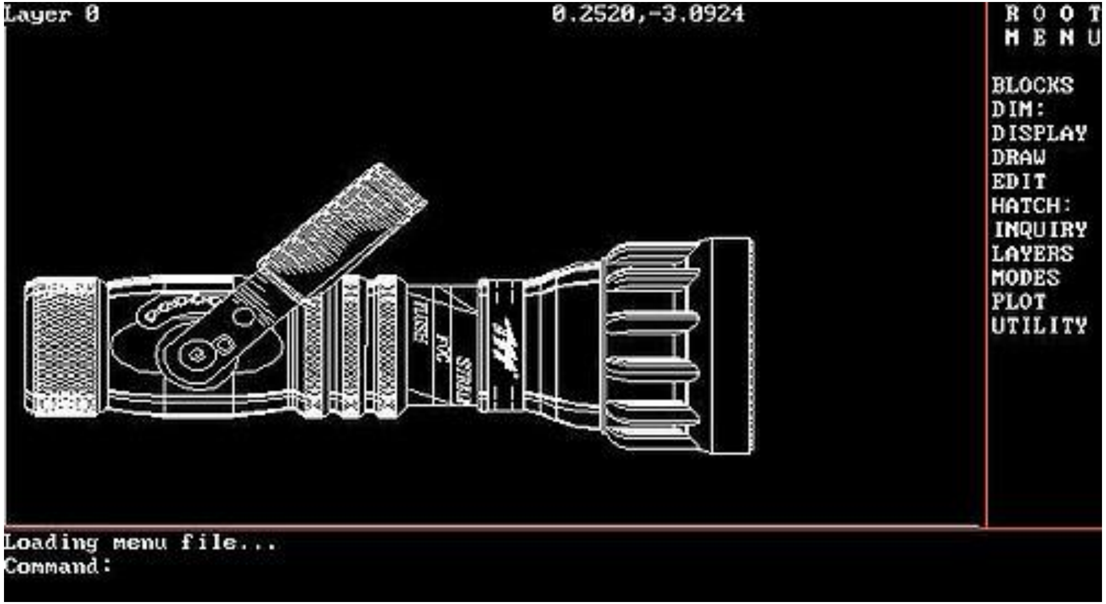
*and probably inaccurate

Drafting tables



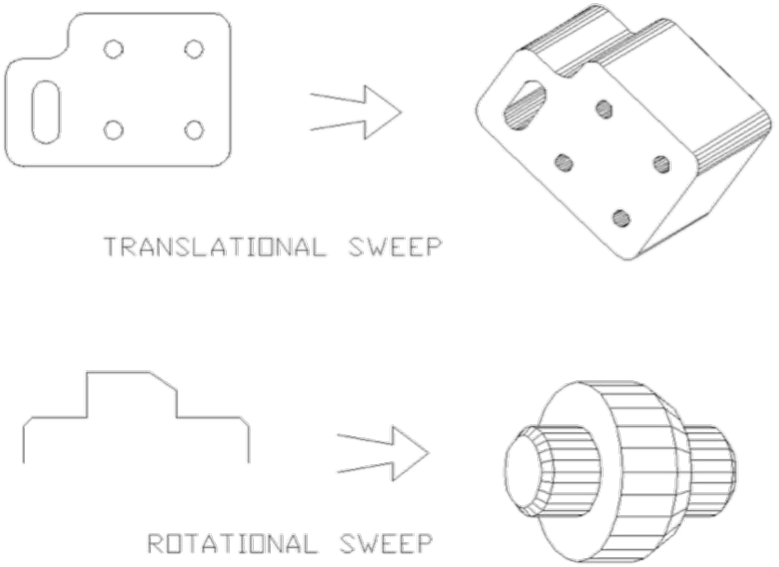


Computer-aided drafting



AutoCAD 2.18 (1985)

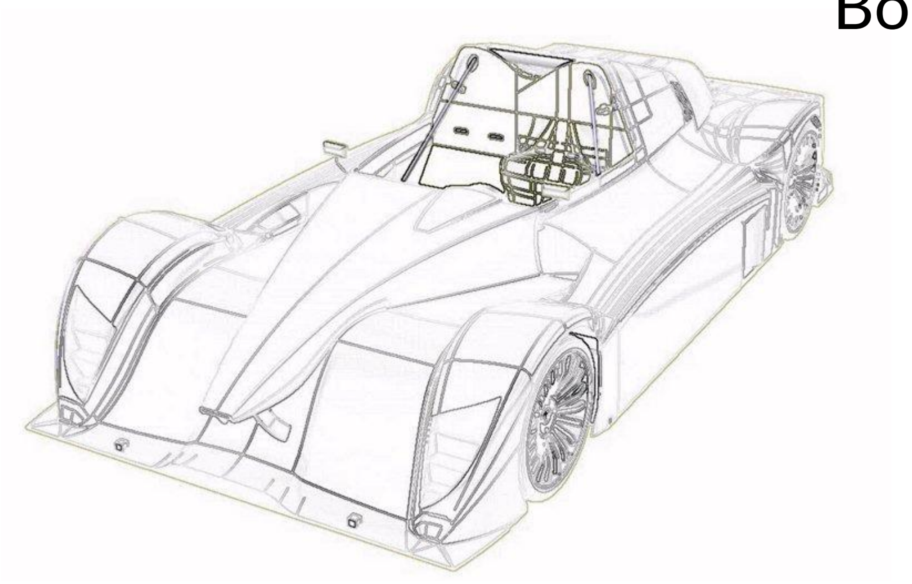
Into the 3rd Dimension



"While all drafting is 2D, and almost all users will spend all their time with AutoCAD working in 2D mode, 3D is important more from a marketing perception standpoint than a technical one."

John Walker, 1983

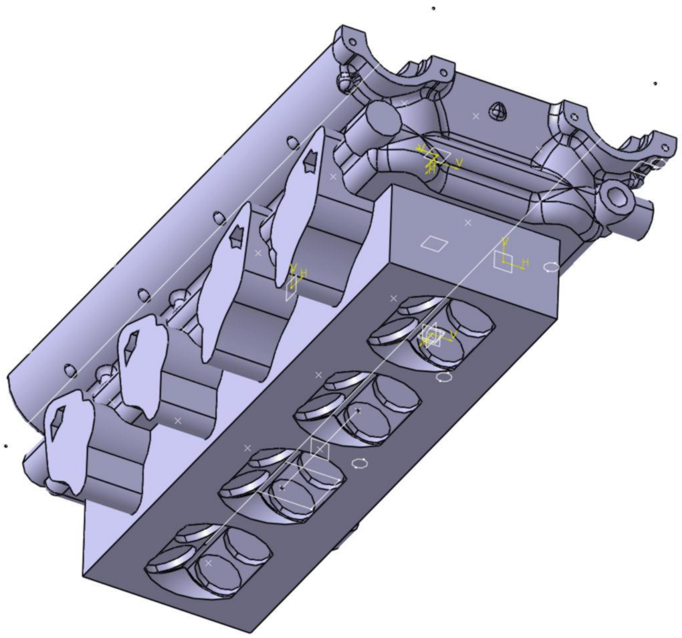
The path of least resistance



Boundary representations

- Natural extension from 2D
- Easy to render
- Fragile representation
- Geometric operations are hard!

Solid Modeling

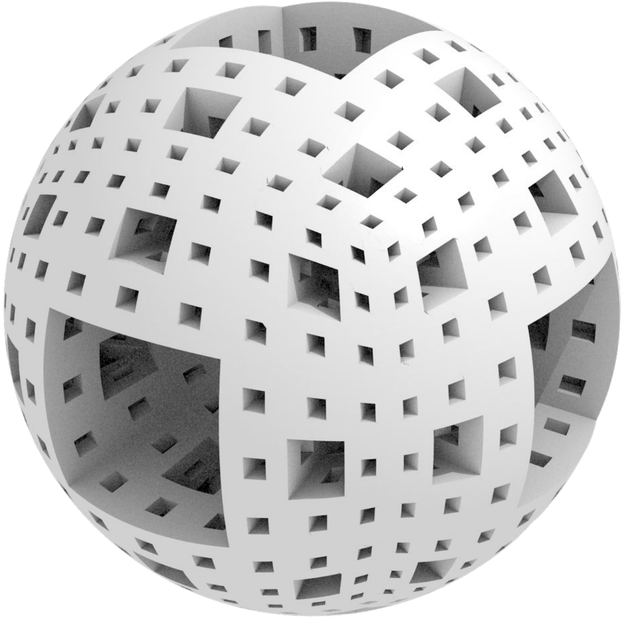


"I have seen solids modeling, and it is the future. For years the skeptics have criticized solids as impractical, compute intensive, and inflexible. 'You can't cut chips with solids,' they'd say, or 'sure they're fun, but what can you do with the model when you're done'?"

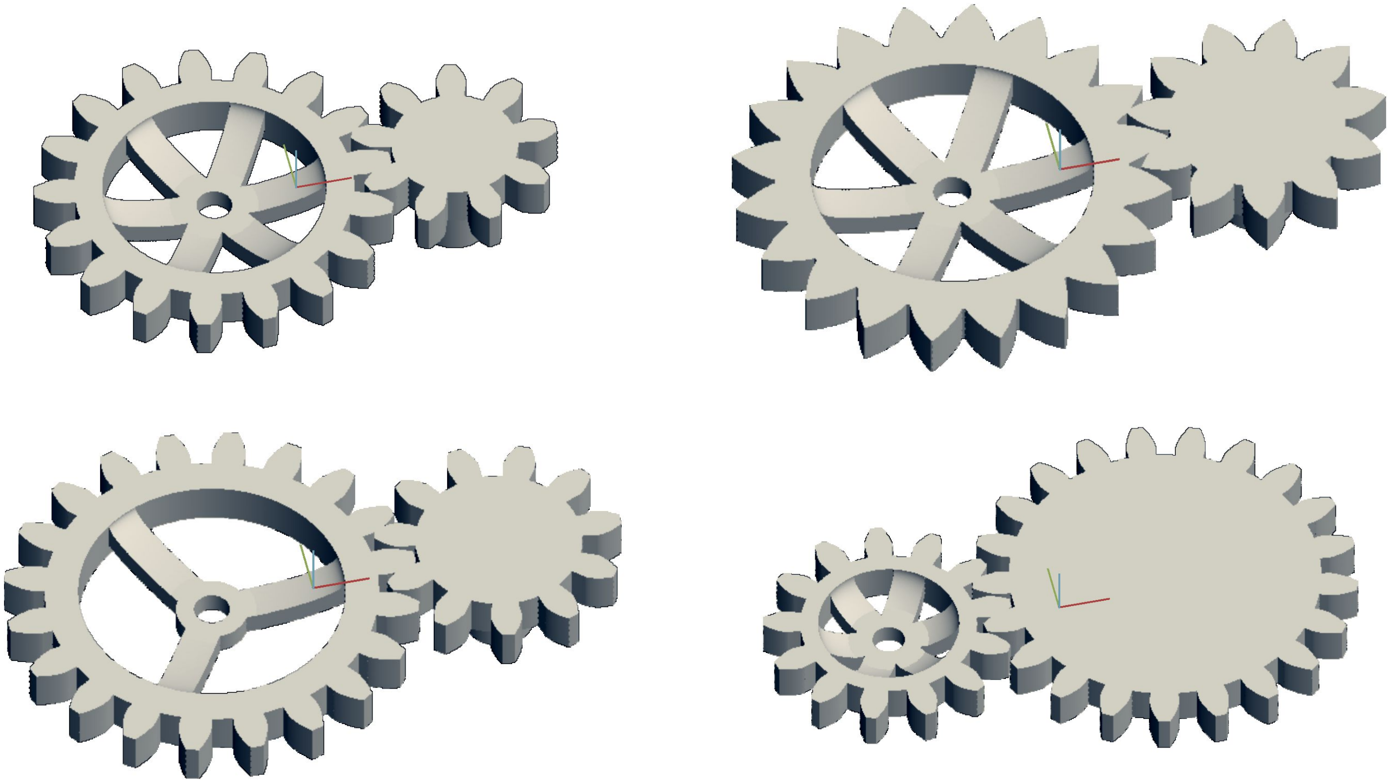
Eric Lyons, 1986

How would you make…

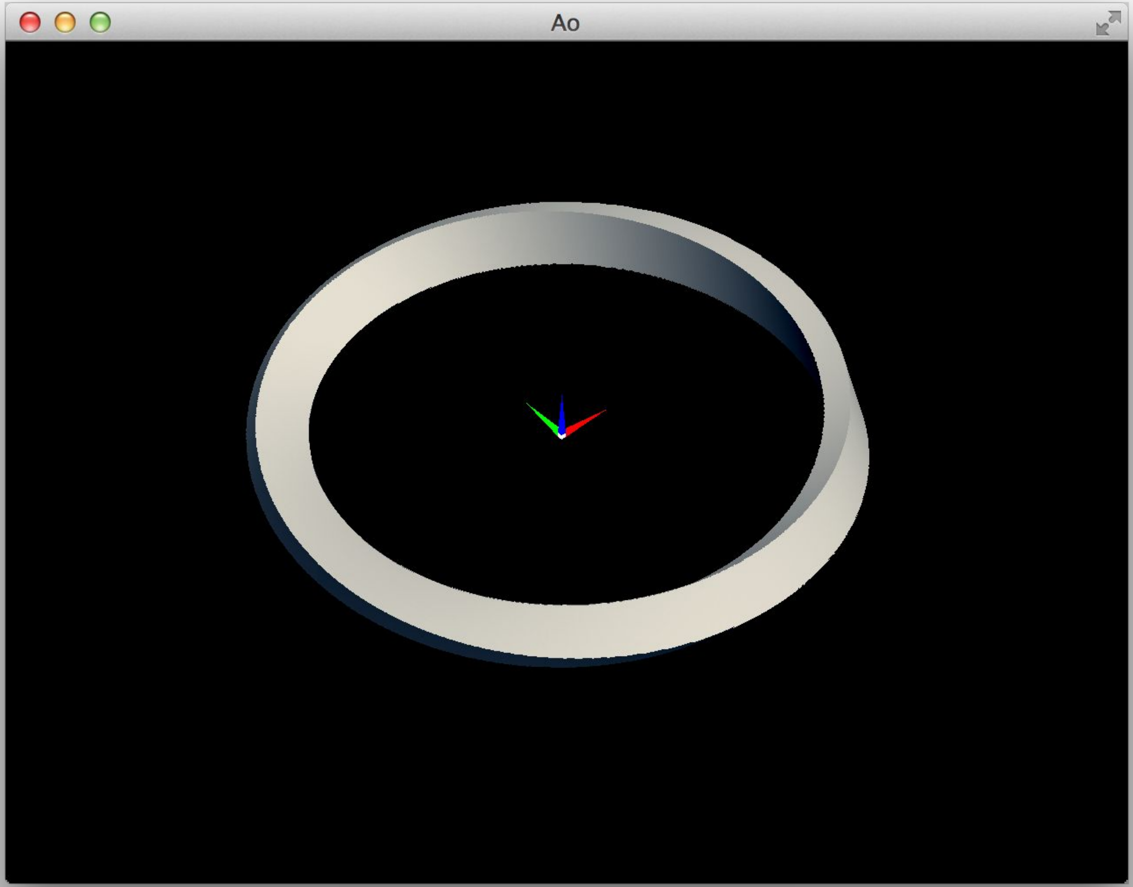
Menger Sponges



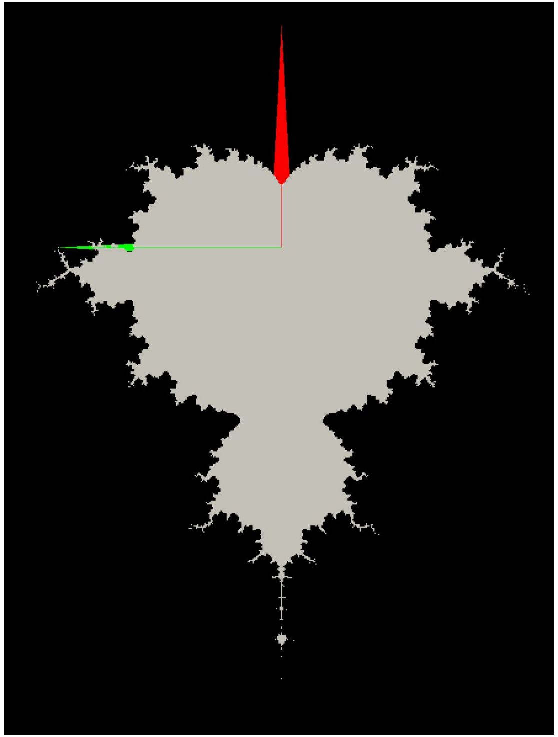
Family of gear pairs



Möbius strip



Mandelbrot Vase



Fundamentals of functional representations

Functional representations

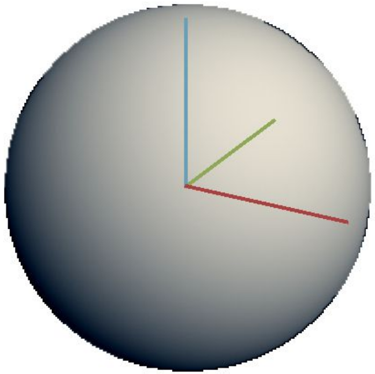
$f : \mathbf{R}^3 \rightarrow \mathbf{R}$

$f(x, y, z) \rightarrow$ Distance

Distance < 0: inside

Distance = 0: boundary

Distance > 0: outside



$f(x, y, z) = x^2 + y^2 + z^2 - r^2$

Upsides & downsides

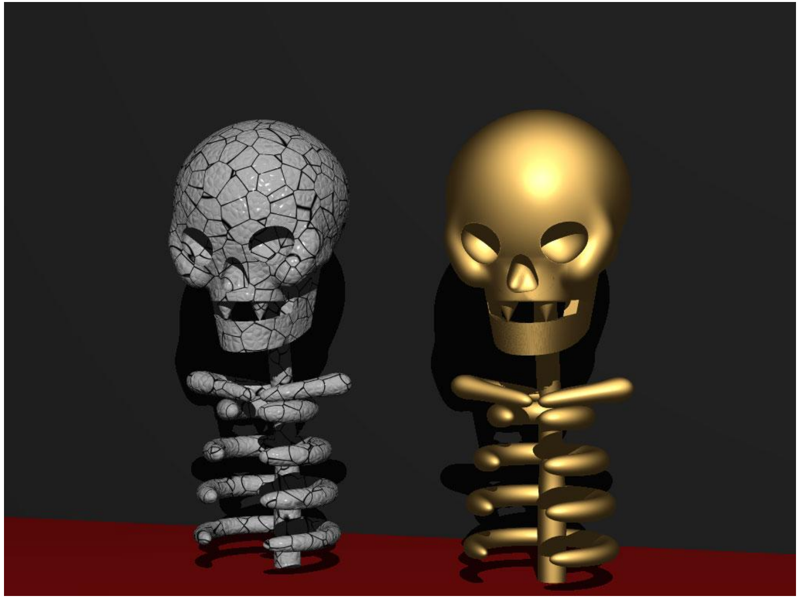
- \+ CSG (union / intersection / difference) becomes trivial
- \+ Unusual transforms become possible
- − Rendering is harder

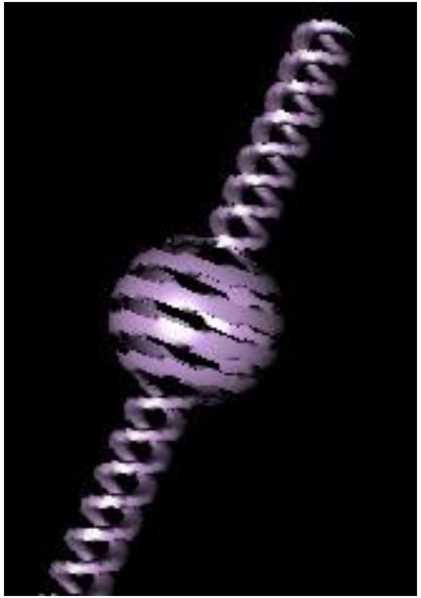
  (but computers are fast,

  and it parallelizes well)
- − Features are implicit
- − Hard to interface with existing b-rep ecosystem (meshes, NURBS, etc.)

Previous work

Hyperfun (SIGGRAPH 1999)

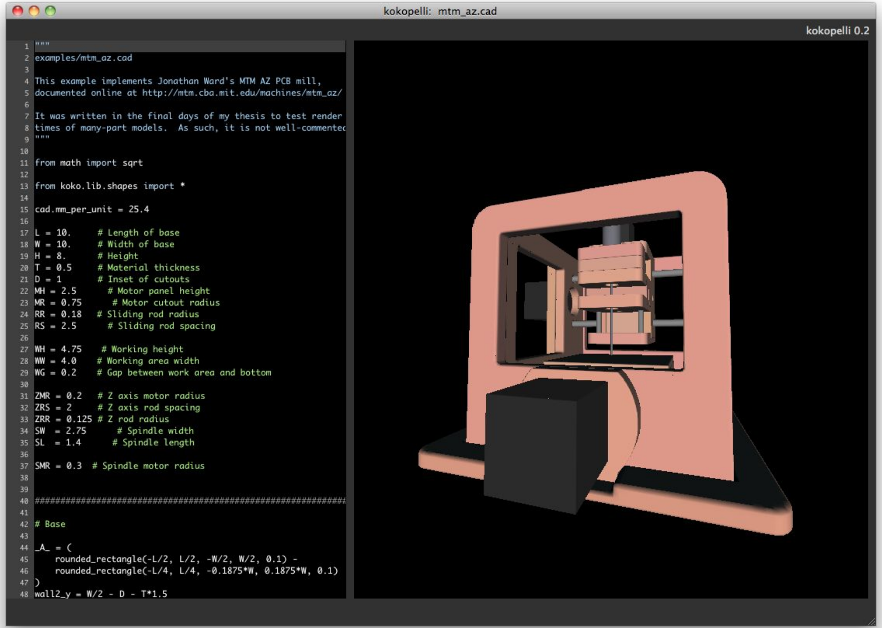


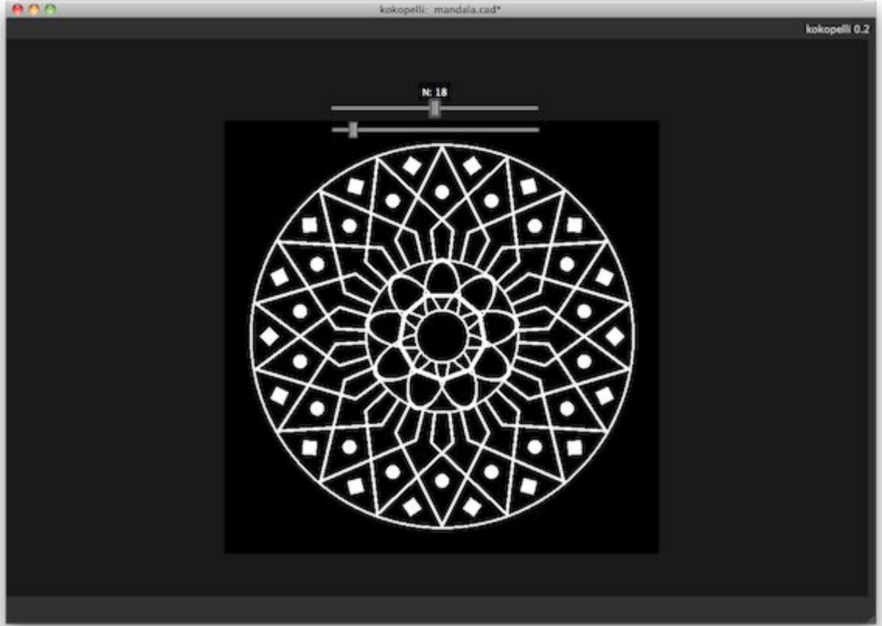


hyperfun.org

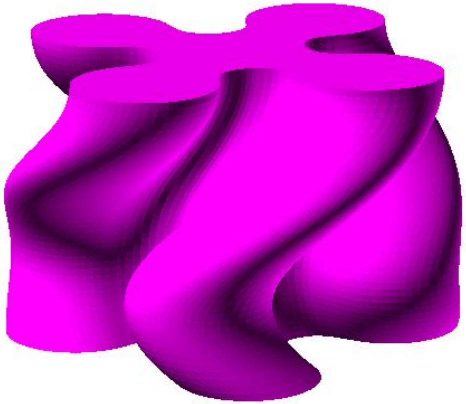
github.com/mkeeter/kokopelli

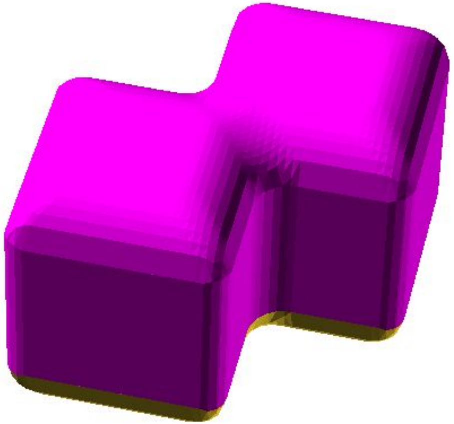
Kokopelli (2012)

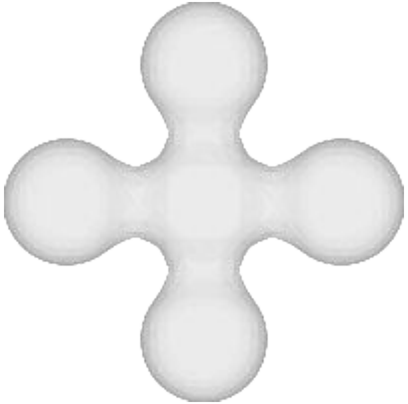




ImplicitCad (2012)



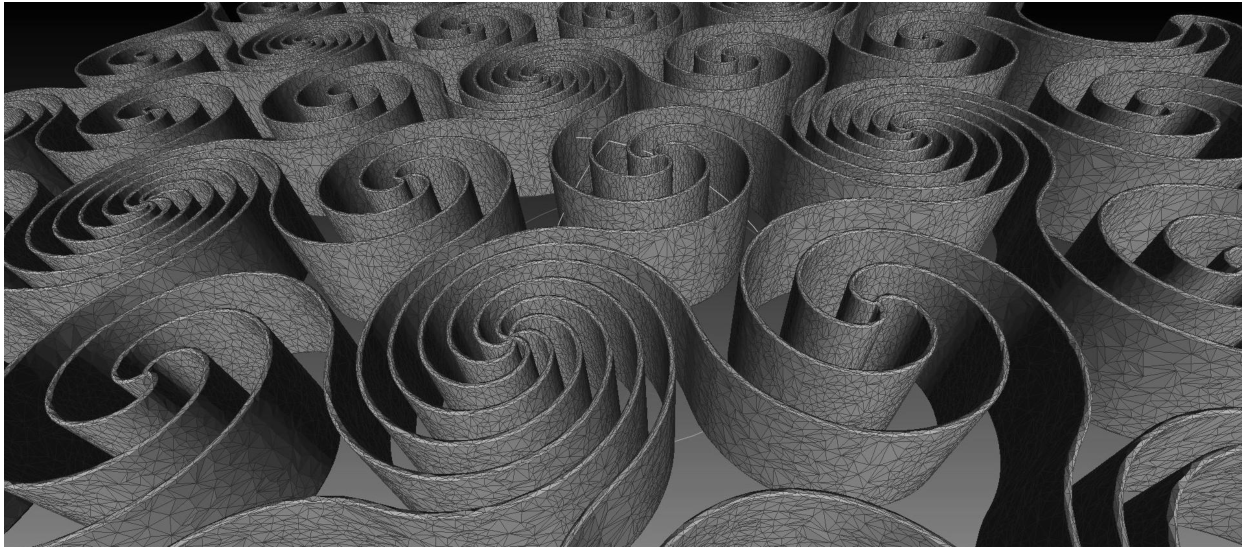




"ImplicitCAD is a project dedicated to using the power of math and computer science to get stupid design problems out of the way of the 3D printing revolution."

implicitcad.org

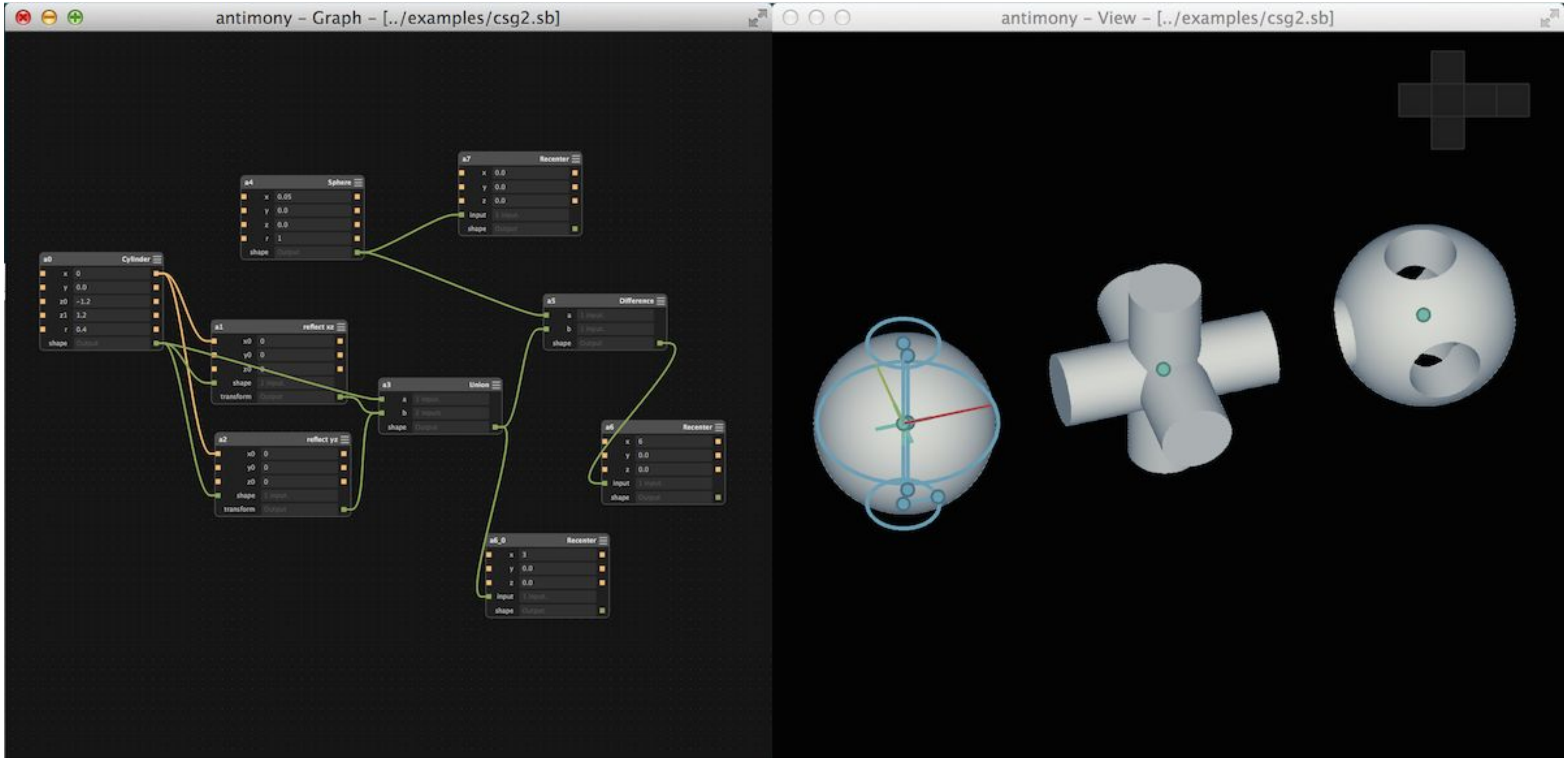
Symvol (2012)



uformia.com

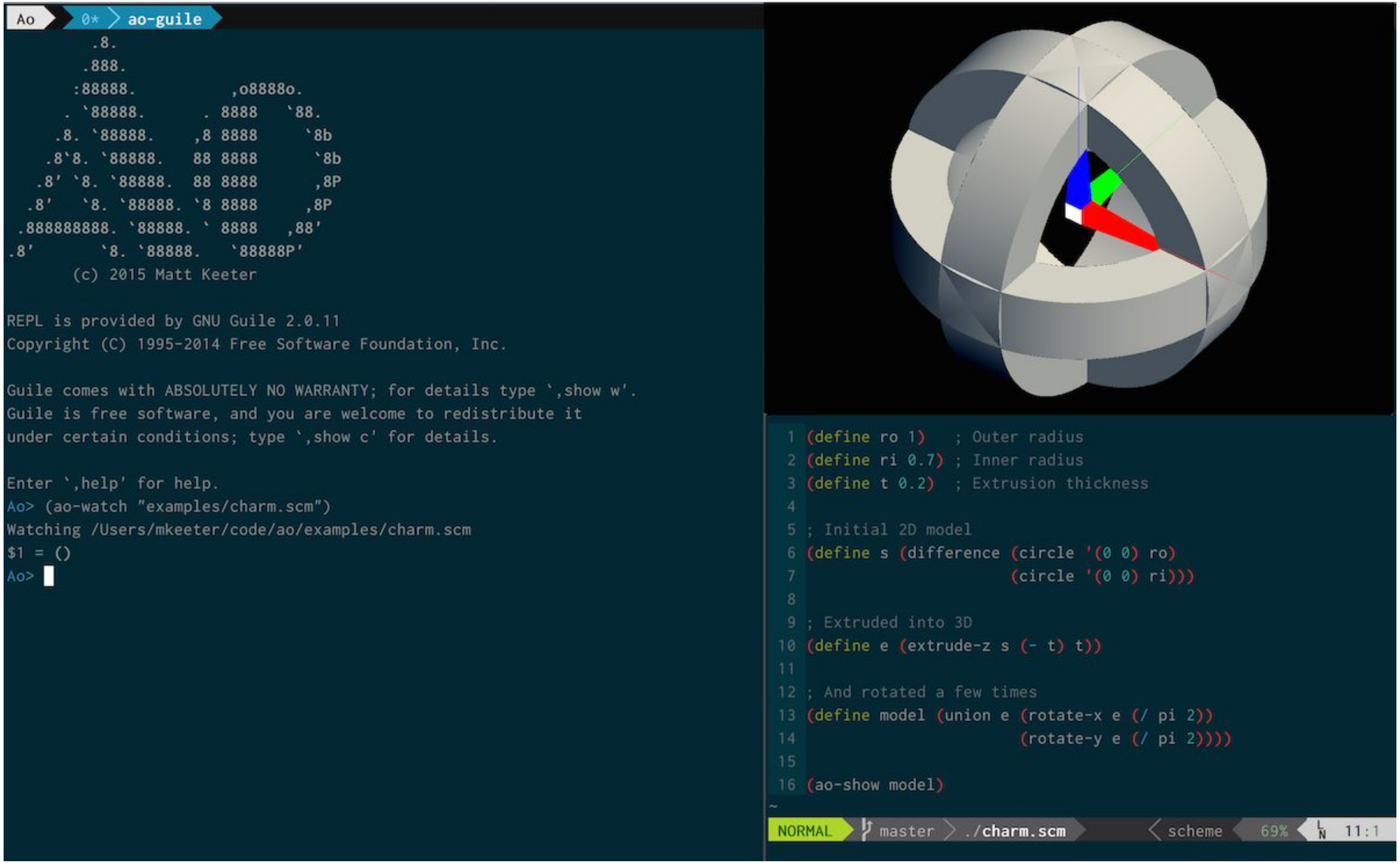
github.com/mkeeter/antimony

Antimony (2013)

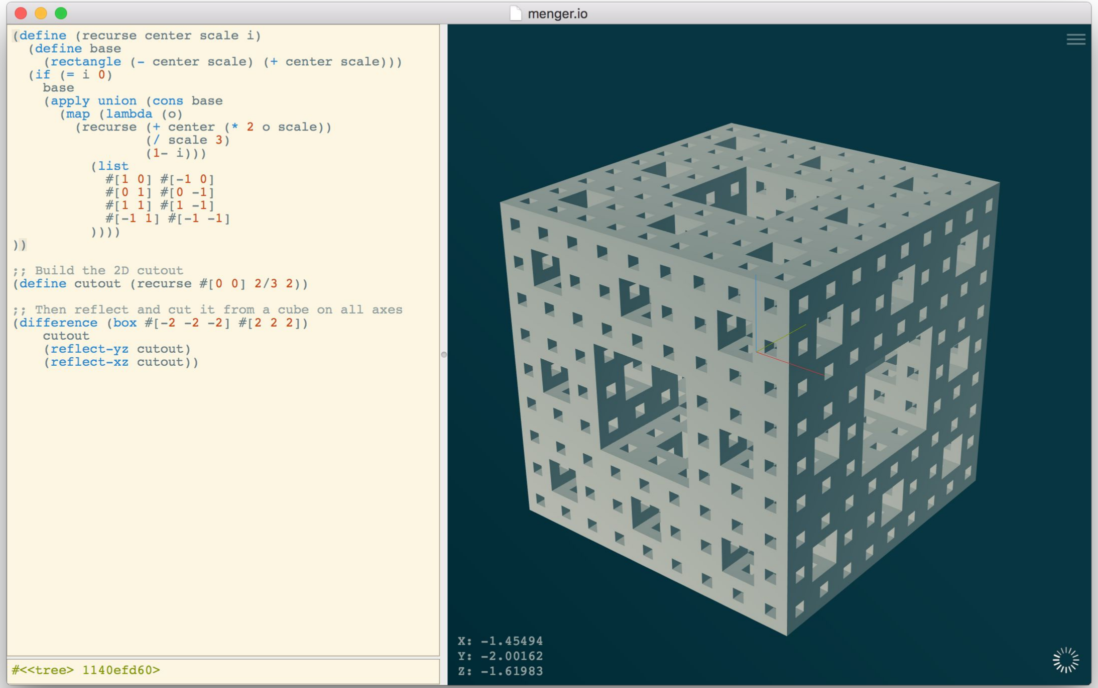


github.com/mkeeter/ao-guile-repl

Ao (2016)



libfive + Studio (2018)



libfive.com

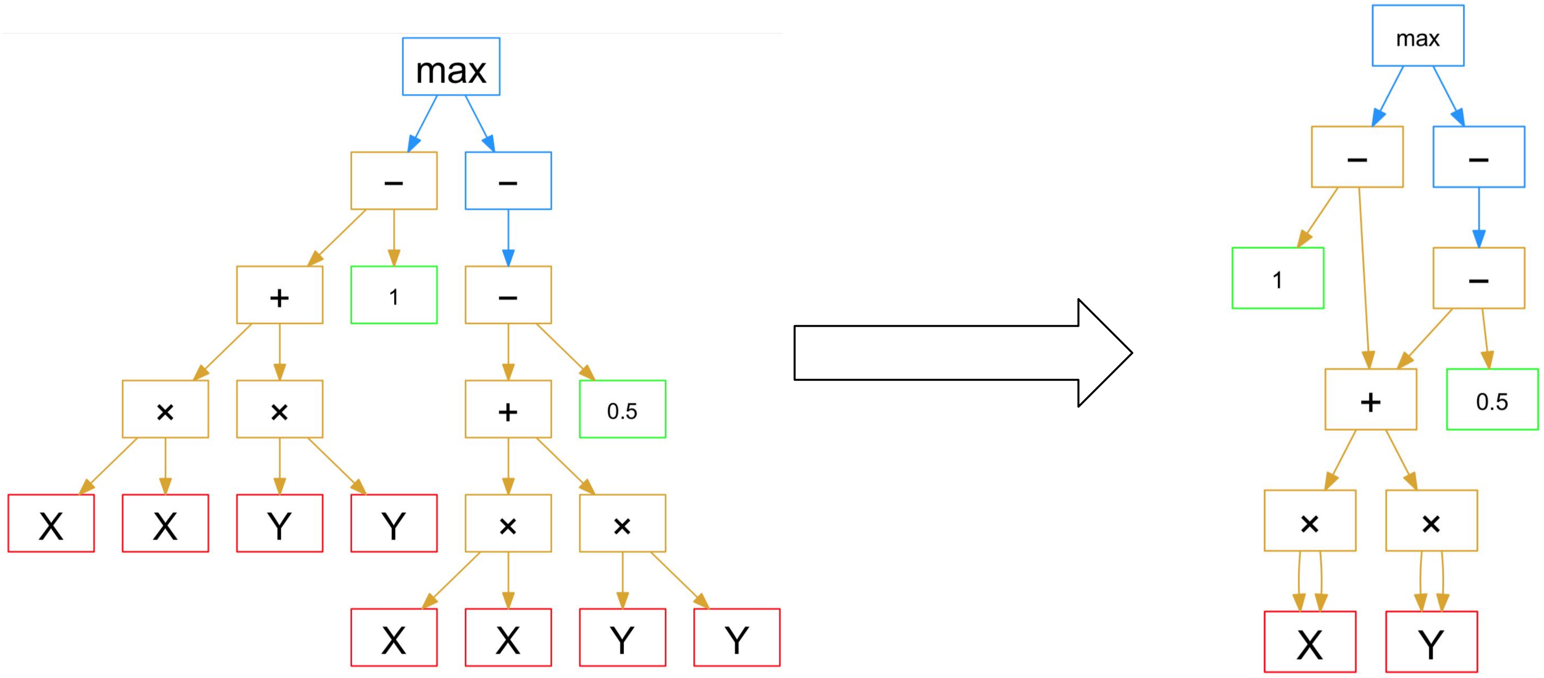
Implicit kernel design

Representation vs. Evaluation

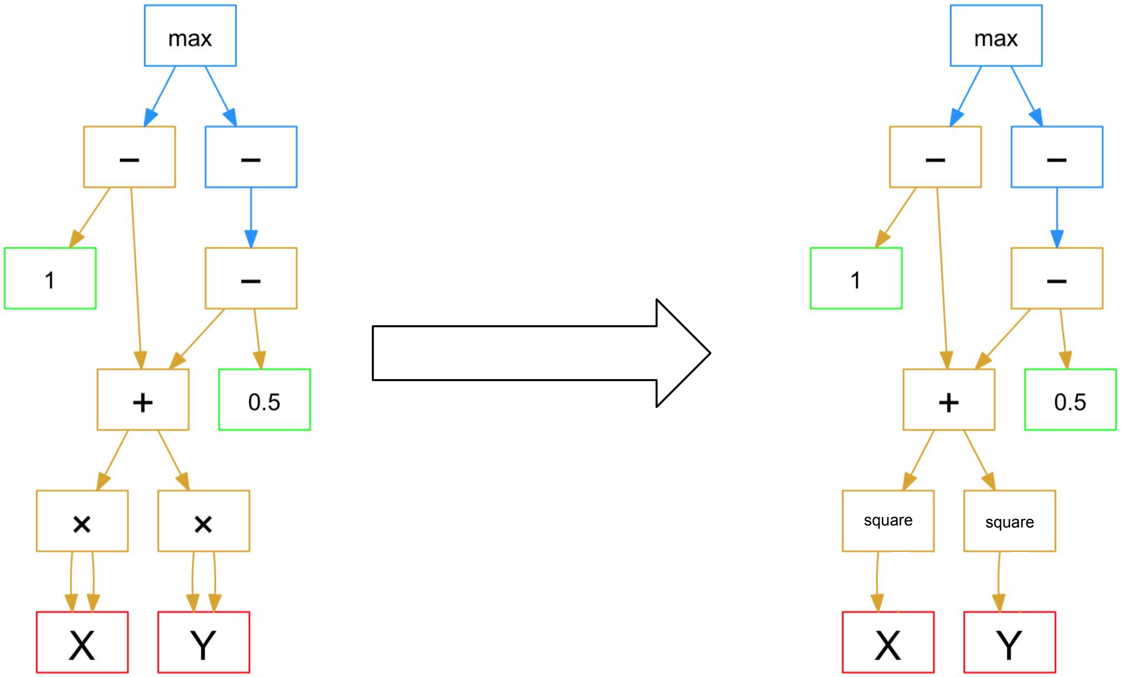
Shape representation

- Lightweight manipulation of math trees
- De-duplication of clauses
- Arithmetic identities + constant folding
- Balancing of commutative operations

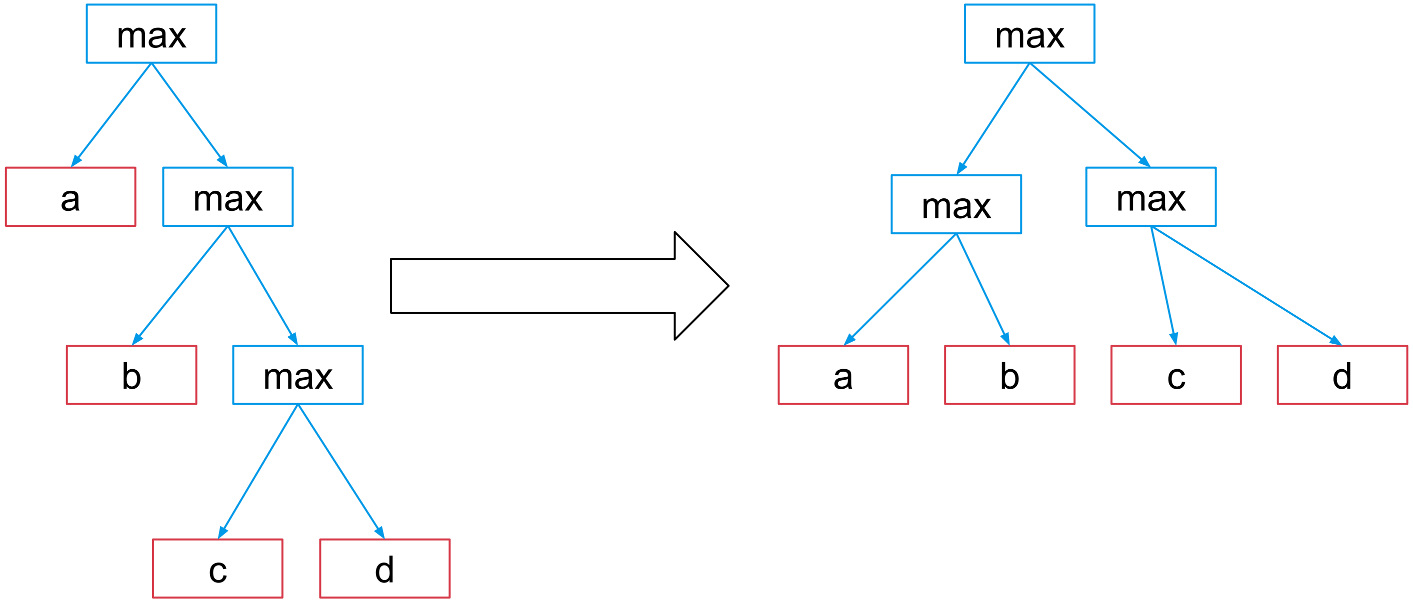
Deduplication



Arithmetic identities



Tree balancing



Representing a clause

```
/*  This is where tree data is actually stored  */
struct Tree_ {
    /*
     *  Destructor erases this Tree
     * from the global Cache
     */
    ~Tree_();

    const Opcode::Opcode op;
    const uint8_t flags;
    const unsigned rank;

    /*  Only populated for constants  */
    const float value;

    /*  Only populated for operations  */
    const std::shared_ptr<Tree_> lhs;
    const std::shared_ptr<Tree_> rhs;
};
```

Representing a clause

```cpp
/*  This is where tree data is actually stored  */
struct Tree_ {
    /*
     *  Destructor erases this Tree
     * from the global Cache
     */
    ~Tree_();

    const Opcode::Opcode op;
    const uint8_t flags;
    const unsigned rank;

    /*  Only populated for constants  */
    const float value;

    /*  Only populated for operations  */
    const std::shared_ptr<Tree_> lhs;
    const std::shared_ptr<Tree_> rhs;
};
```

```cpp
/*  Lightweight, passable-by-value handle */
class Tree {
   /*  Overload arithmetic here! */

   /*  Here's the actual Tree data */
   std::shared_ptr<Tree_> ptr;
};
```

Representing a clause

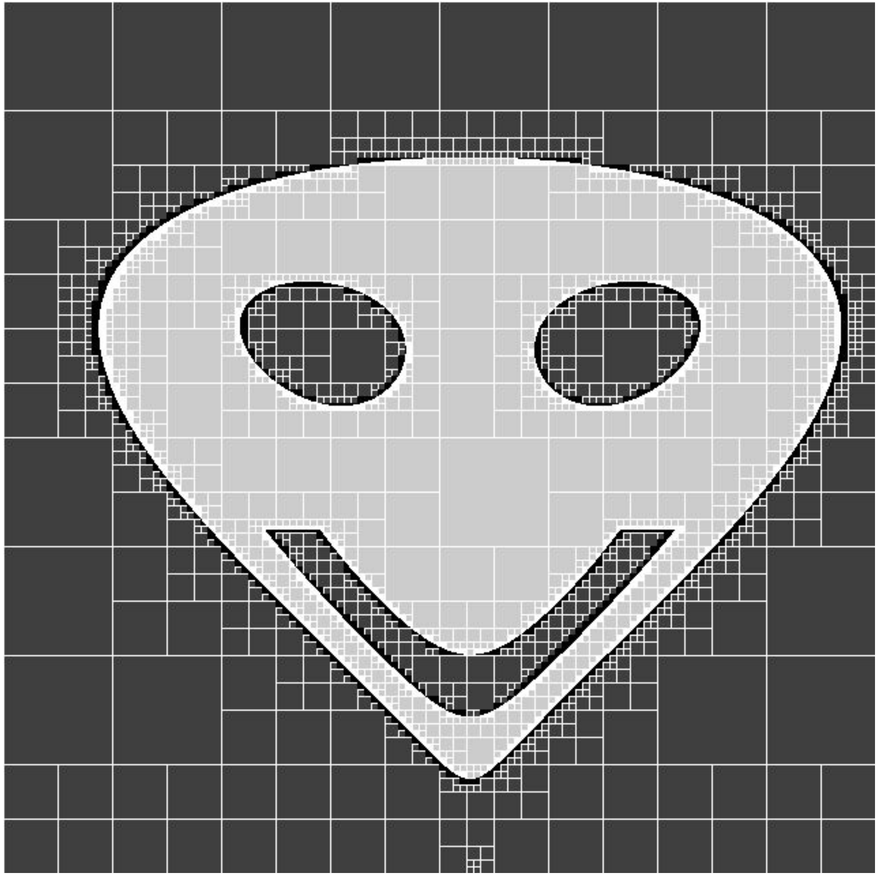
```cpp
/*  This is where tree data is actually stored  */
struct Tree_ {
    /*
     *  Destructor erases this Tree
     * from the global Cache
     */
    ~Tree_();

    const Opcode::Opcode op;
    const uint8_t flags;
    const unsigned rank;

    /*  Only populated for constants  */
    const float value;

    /*  Only populated for operations  */
    const std::shared_ptr<Tree_> lhs;
    const std::shared_ptr<Tree_> rhs;
};
```

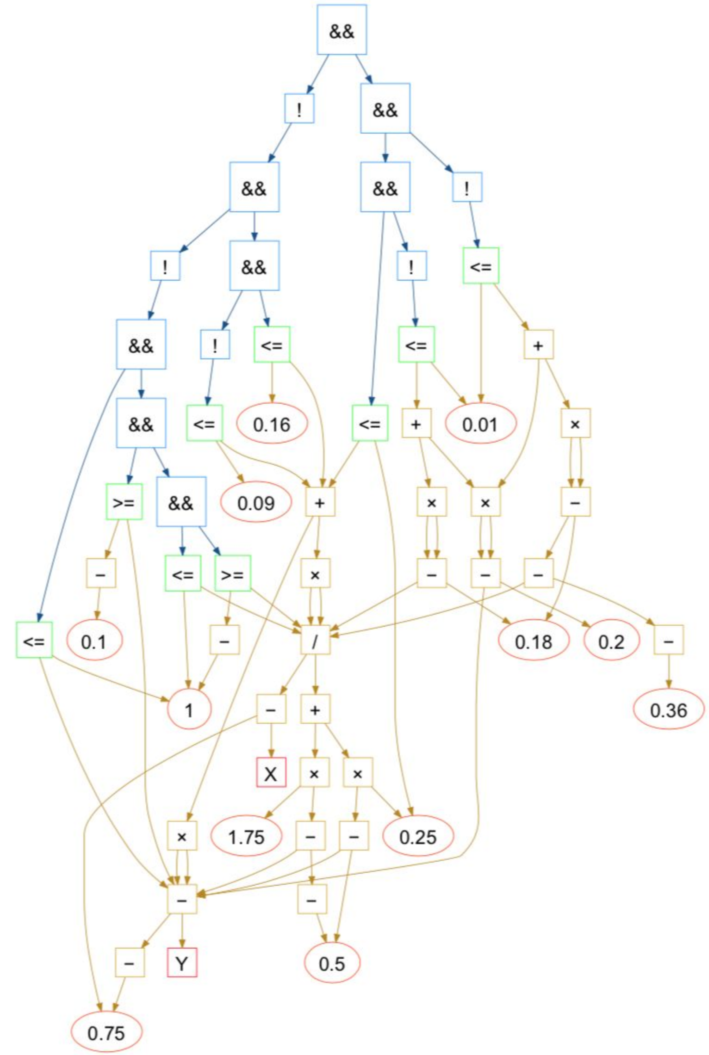
```cpp
/*  Lightweight, passable-by-value handle */
class Tree {
   /*  Overload arithmetic here! */

   /*  Here's the actual Tree data */
   std::shared_ptr<Tree_> ptr;
};


class Cache {
   /*  Functions to handle identities here! */

   typedef std::tuple<Opcode::Opcode,
                      const Tree_*, /* lhs */
                      const Tree_*  /* rhs */ > Key;
   std::map<Key, std::weak_ptr<Tree::Tree_>> ops;
   std::map<float, std::weak_ptr<Tree::Tree_>> constants;
};
```

Representation vs.
Evaluation

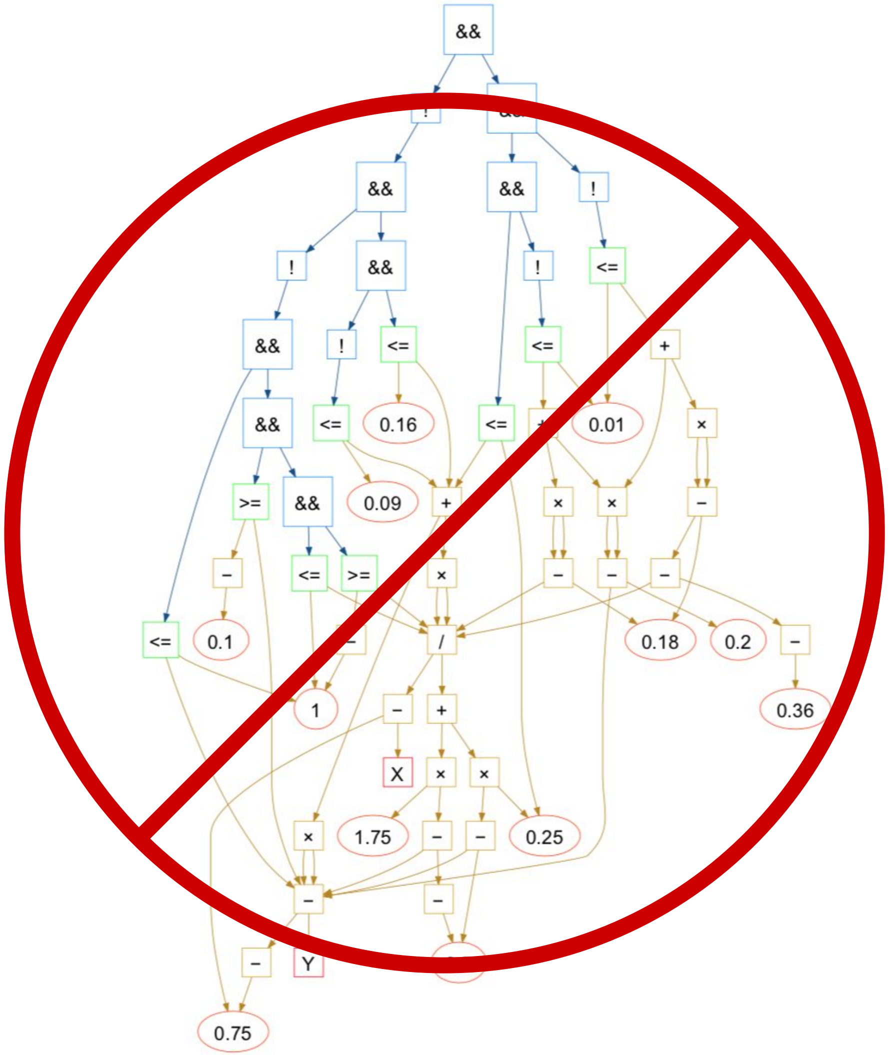
General form of algorithms

- If at minimum size, perform operations on voxel
- Otherwise, evaluate interval
    - If filled or empty, return early
- Subdivide and recurse
- After recursion is done, collapse branches if possible

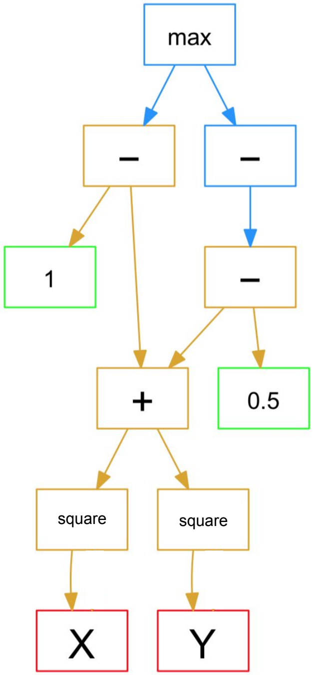




&&
!
&&
&&
&&
!
&&
!
<=
&&
!
<=
&&
<=
0.16
<=
+
&&
<=
+
0.01
×
>=
&&
0.09
+
×
×
−
−
<=
>=
×
−
−
−
<=
0.1
−
/
0.18
0.2
−
1
−
+
0.36
X
×
×
×
1.75
−
−
0.25
−
−
−
Y
0.5
0.75



&&
!
&&
&&
&&
!
&&
!
!
<=
&&
!
<=
<=
+
&&
<=
0.16
<=
0.01
×
>=
&&
0.09
+
×
×
−
−
<=
>=
×
−
−
−
<=
0.1
−
/
0.18
0.2
−
1
−
+
0.36
X
×
×
×
1.75
−
−
0.25
−
−
−
Y
0.75



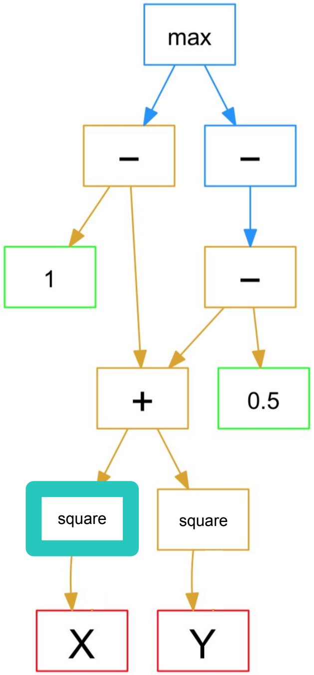
Tape

opcode	square	square	+	–	–	–	max
lhs	0	1	4	6	6	8	7
rhs			5	2	3		9
out	4	5	6	7	8	9	10

Memory

index	0	1	2	3	4	5	6	7	8	9	10
value	X	Y	1	0.5							out

Flat tapes are faster than chasing pointers!



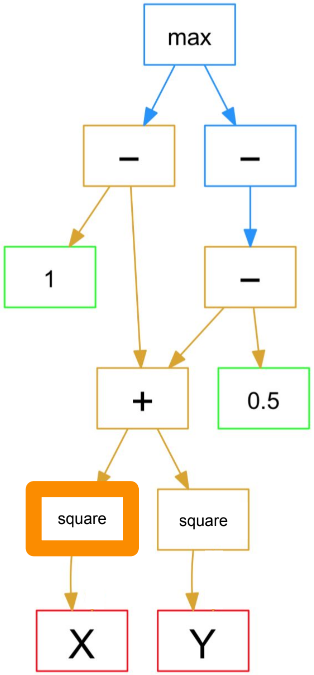
Tape

opcode	square	square	+	−	−	−	max
lhs	0	1	4	6	6	8	7
rhs			5	2	3		9
out	4	5	6	7	8	9	10

Memory

index	0	1	2	3	4	5	6	7	8	9	10
value	X	Y	1	0.5							out

Flat tapes are faster than chasing pointers!



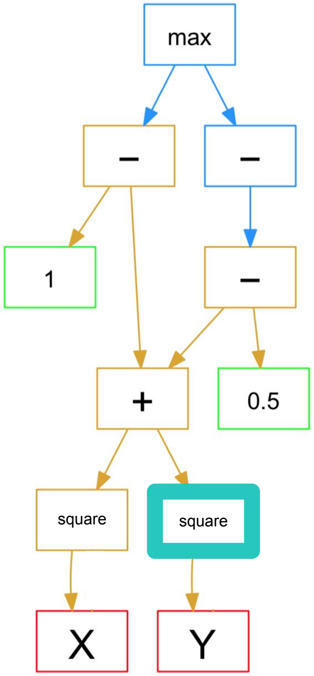
Tape

opcode	square	square	+	–	–	–	max
lhs	0	1	4	6	6	8	7
rhs			5	2	3		9
out	4	5	6	7	8	9	10

Memory

index	0	1	2	3	4	5	6	7	8	9	10
value	X	Y	1	0.5	X^2						out

Flat tapes are faster than chasing pointers!



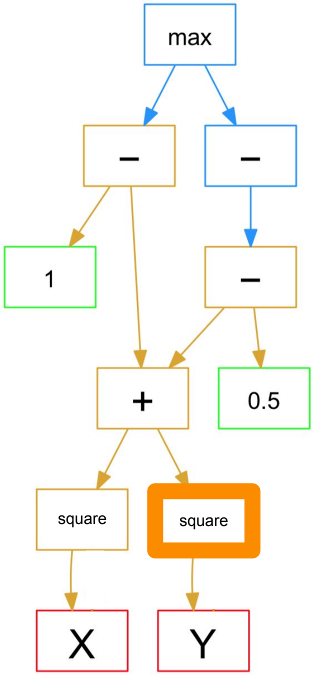
Tape

opcode	square	square	+	–	–	–	max
lhs	0	1	4	6	6	8	7
rhs			5	2	3		9
out	4	5	6	7	8	9	10

Memory

index	0	1	2	3	4	5	6	7	8	9	10
value	X	Y	1	0.5	X^2						out

Flat tapes are faster than chasing pointers!



Tape

opcode	square	square	+	–	–	–	max
lhs	0	1	4	6	6	8	7
rhs			5	2	3		9
out	4	5	6	7	8	9	10

Memory

index	0	1	2	3	4	5	6	7	8	9	10
value	X	Y	1	0.5	X^2	Y^2					out

Flat tapes are faster than chasing pointers!

Value types

Floating-point values $\qquad f(1, 0, 0) = 0$

Value types

Floating-point values	f(1, 0, 0) = 0
Arrays of floats (optimization for speed)	f({...}, {...}, {...}) = {...}

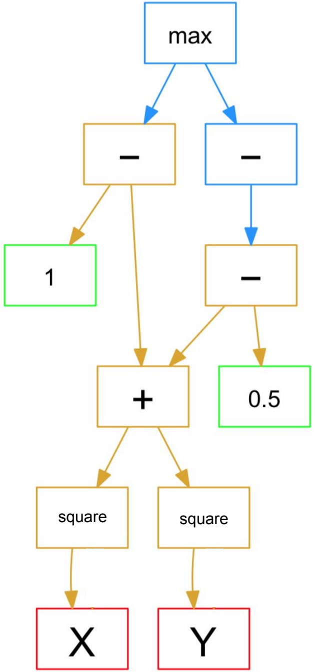
Value types

Floating-point values	f(1, 0, 0) = 0
Arrays of floats (optimization for speed)	f({...}, {...}, {...}) = {...}
Interval ranges	f([0, 1], [1, 2], [0, 0]) = [0, 4]

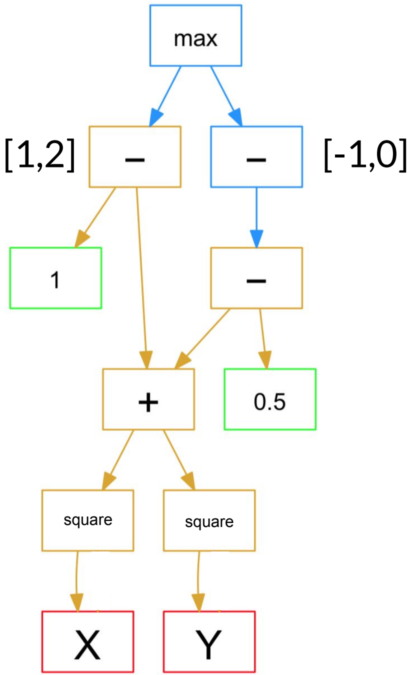
Value types

Floating-point values	f(1, 0, 0) = 0
Arrays of floats (optimization for speed)	f({...}, {...}, {...}) = {...}
Interval ranges	f([0, 1], [1, 2], [0, 0]) = [0, 4]
Derivatives (automatic differentiation)	(df/dx)(1, 0, 0) = 2 (df/dy)(1, 0, 0) = 0 (df/dz)(1, 0, 0) = 0

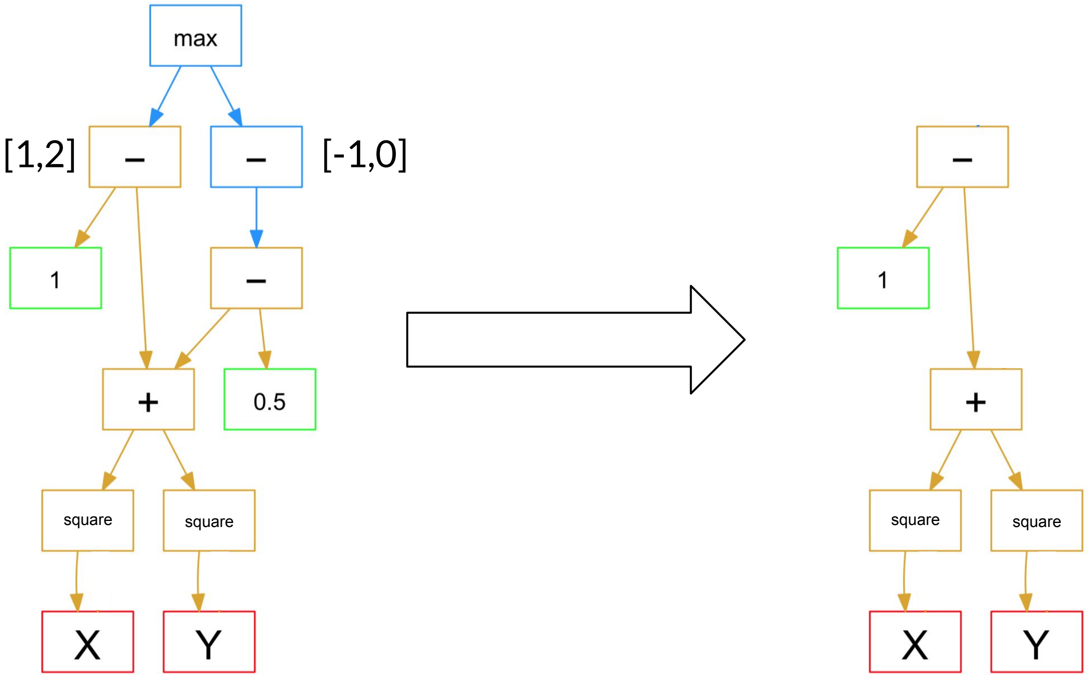
Tree pruning with intervals



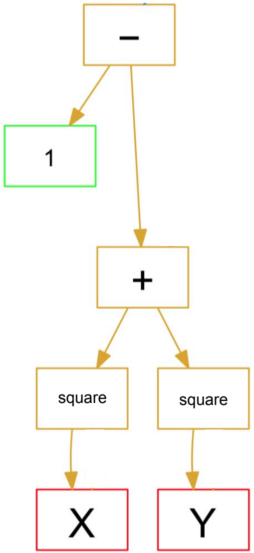
Tree pruning with intervals



Tree pruning with intervals



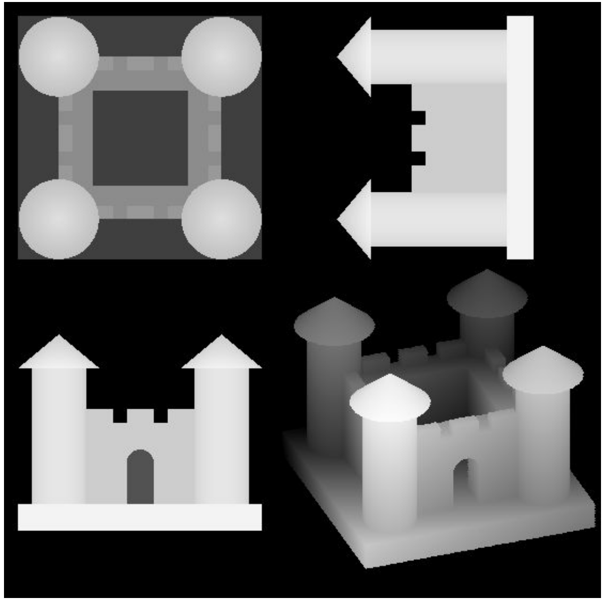
Tree pruning with intervals



opcode	square	square	+	–	~~–~~	~~–~~	~~max~~
lhs	0	1	4	6	~~6~~	~~8~~	~~7~~
rhs			5	2	~~3~~		~~9~~
out	4	5	6	7	~~8~~	~~9~~	~~10~~

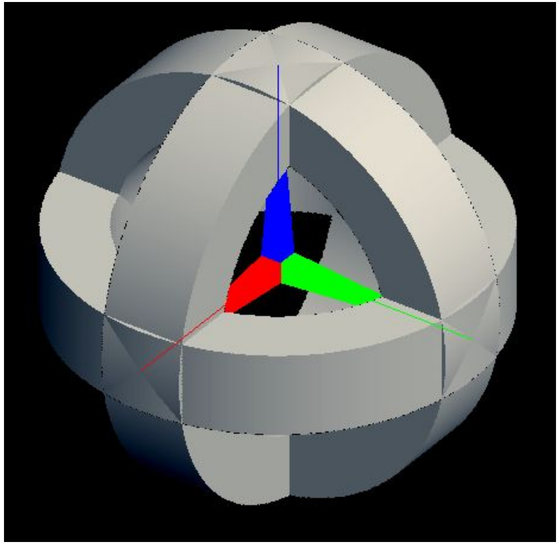
Outputs

Voxel output: heightmaps



Brightness = z-height
Useful for 2.5D machining

Voxel output: shaded



Normals are based on derivatives (df/dx, df/dy, df/dz)

B-rep output: meshes

Normals are used to position vertices on sharp edges and corners.

Real things!

Creepy Crawly Cutter
Sam Calisch

Curtain rail brackets
Paul Meyer

Rotary encoder
Matt Keeter

Unsolved problems

- GPU acceleration
- Feature-based design
- Interacting with meshes
- Constraint systems