Designing with Functional Representations: GUI and Solver

Matt Keeter
matt.keeter@cba.mit.edu

Spring 2012

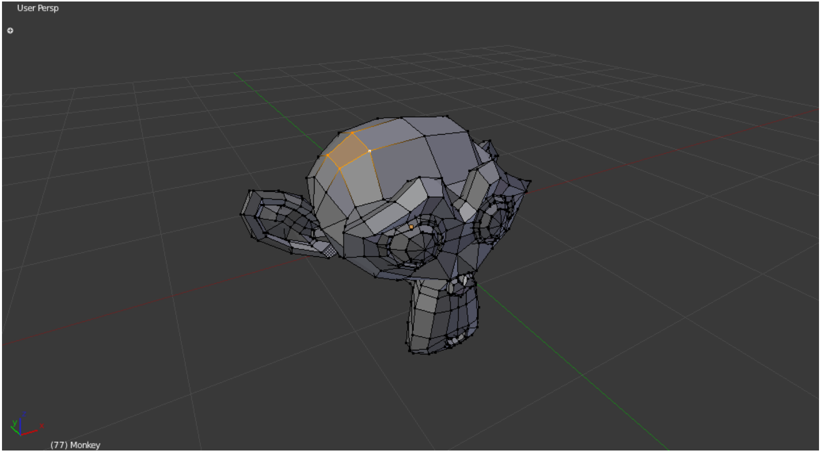
The Center for Bits and Atoms
Massachusetts Institute of Technology

Formats for Fabrication

- How do we represent objects?
- 2D areas and 3D volumes
- Design $\rightarrow$ fabrication

Boundary Representations

Data describing surface of an object



Boundary Representations

Advantages:

- Easy to render
- Long history
- Common in computer graphics

Boundary Representations

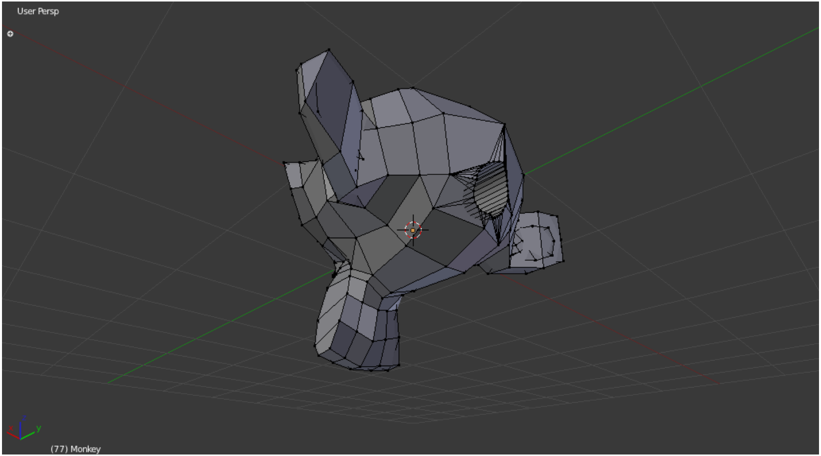
Advantages:

- Easy to render
- Long history
- Common in computer graphics

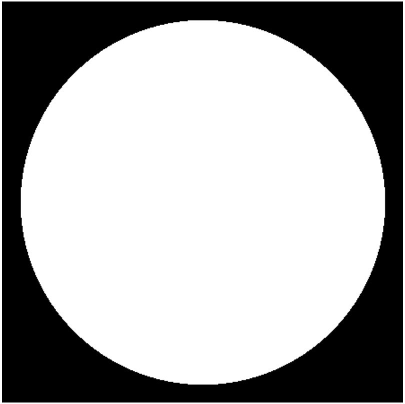
Disadvantages:

- Finite resolution
- Requires surface $\rightarrow$ volume conversion
- Constructive solid geometry is hard / messy

Boundary Representations

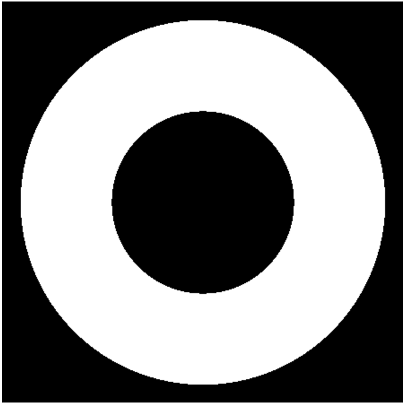


Functional Representation



```
X*X + Y*Y < 1
```

Functional Representation



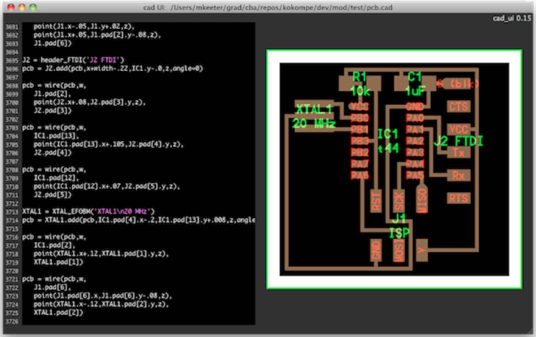
(X*X + Y*Y < 1) && (X*X + Y*Y > 0.5)

Functional Representation

- Resolution-independent
- Platform-independent
- Easy to transform and modify
- Hard to render

Design Tools

- Library of common shapes and operators
- Python scripts as design files
- Interactive GUI:

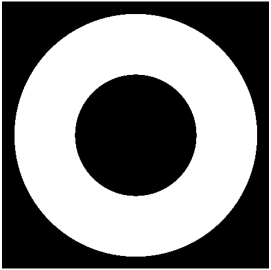


Solver Fundamentals

How to convert an expression into an image?

```
(X*X + Y*Y < 1) && (X*X + Y*Y > 0.5)
```

$$\downarrow$$



Solver Fundamentals

- Previous solver:
  - Brute-force evaluation
  - Paste expression into template C program
  - Compile & run!
  - Evaluates expression for every pixel

Solver Fundamentals

- Previous solver:
  - Brute-force evaluation
  - Paste expression into template C program
  - Compile & run!
  - Evaluates expression for every pixel
- We can do better.

Solver Architecture

- **Parser**
  - Converts string into tree structure
  - Optimizes tree structure
- Solver
  - Evaluates expression on region
  - Interval arithmetic speeds up evaluation
  - Uses caching and multithreading

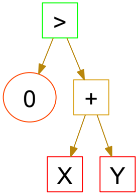
Parser

Expressions $\rightarrow$ trees

Parser

Expressions $\rightarrow$ trees

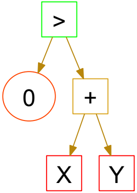
`X + Y > 0` becomes



Parser

Expressions → trees

`X + Y > 0` becomes

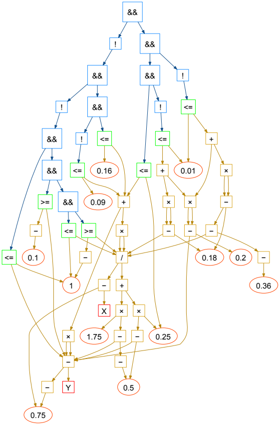


Uses shunting-yard algorithm

Tree Structure

Tree of expressions operating on

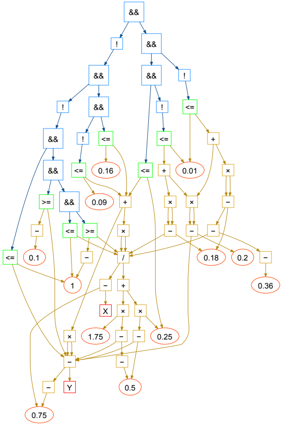
- constants
- variables
- other expressions



Tree Structure

Distinct data types:

- Floating-point value/interval
- Tri-bool (true, false, or ambiguous)
- Color (32-bit integer)



Architecture

- Parser
  - Converts string into tree structure
  - Optimizes tree structure
- **Solver**
  - Evaluates expression on region
  - Interval arithmetic speeds up evaluation
  - Uses caching and multithreading

Interval Arithmetic

- Operations are applied to regions in space

Interval Arithmetic

- Operations are applied to regions in space
- Logic operations are true, false, or ambiguous
  - $[-1, 1] < 2$ is true
  - $[-1, 1] < -2$ is false
  - $[-1, 1] < 0$ is ambiguous

Subdivision & Recursion

Solver algorithm:

- Evaluate on initial region
- If true or false, color and return
- If ambiguous, subdivide and recurse

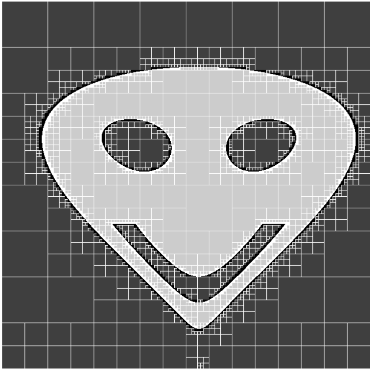
Subdivision & Recursion

Solver algorithm:

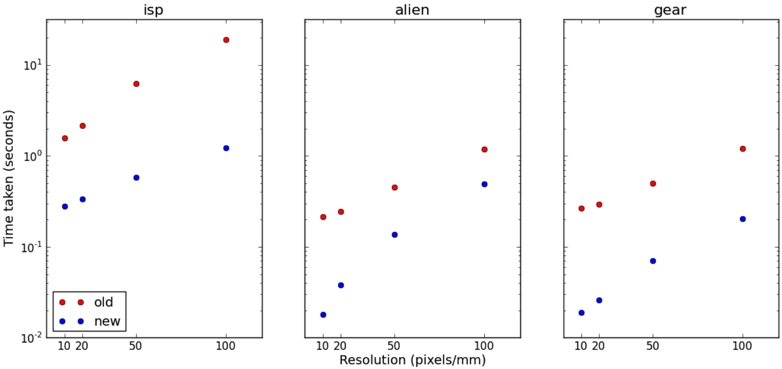
- Evaluate on initial region
- If true or false, color and return
- If ambiguous, subdivide and recurse

Regions below a minimum size are evaluated point-by-point, which improves performance.

Subdivision & Recursion



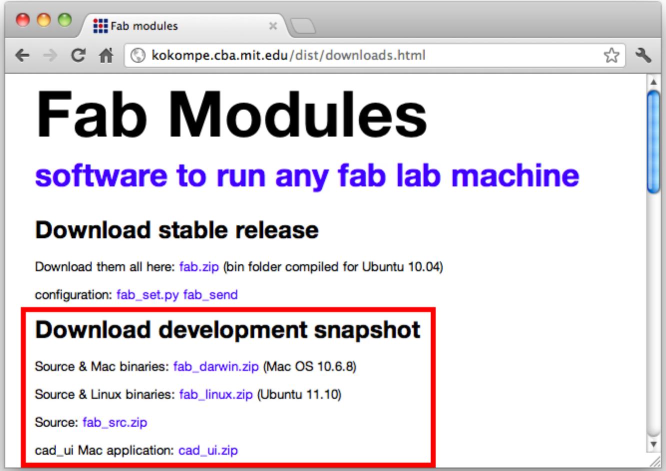
Performance



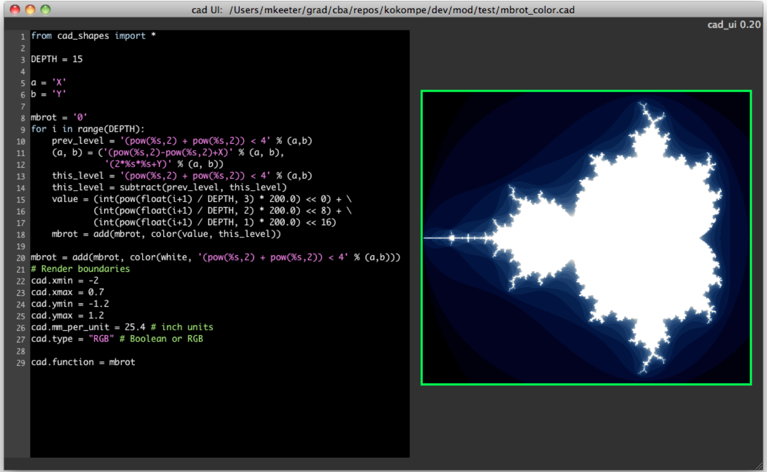
Future Work

- Improving GUI design tools
- Generating surfaces
- Improving standard library
- Possibly switching to GPU

Resources


Fab modules
kokompe.cba.mit.edu/dist/downloads.html
Fab Modules
software to run any fab lab machine
Download stable release
Download them all here: fab.zip (bin folder compiled for Ubuntu 10.04)
configuration: fab_set.py fab_send
Download development snapshot
Source & Mac binaries: fab_darwin.zip (Mac OS 10.6.8)
Source & Linux binaries: fab_linux.zip (Ubuntu 11.10)
Source: fab_src.zip
cad_ui Mac application: cad_ui.zip

Questions?


cad UI: /Users/mkeeter/grad/cba/repos/kokompe/dev/mod/test/mbrot_color.cad
cad_ui 0.20
from cad_shapes import *

DEPTH = 15

a = 'X'
b = 'Y'

mbrot = '0'
for i in range(DEPTH):
    prev_level = '(pow(%s,2) + pow(%s,2)) < 4' % (a,b)
    (a, b) = ('(pow(%s,2)-pow(%s,2)+X)' % (a, b),
              '(2*%s*%s+Y)' % (a, b))
    this_level = '(pow(%s,2) + pow(%s,2)) < 4' % (a,b)
    this_level = subtract(prev_level, this_level)
    value = (int(pow(float(i+1) / DEPTH, 3) * 200.0) << 0) + \
            (int(pow(float(i+1) / DEPTH, 2) * 200.0) << 8) + \
            (int(pow(float(i+1) / DEPTH, 1) * 200.0) << 16)
    mbrot = add(mbrot, color(value, this_level))

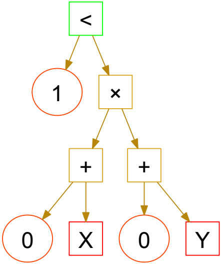
mbrot = add(mbrot, color(white, '(pow(%s,2) + pow(%s,2)) < 4' % (a,b)))
# Render boundaries
cad.xmin = -2
cad.xmax = 0.7
cad.ymin = -1.2
cad.ymax = 1.2
cad.mm_per_unit = 25.4 # inch units
cad.type = "RGB" # Boolean or RGB

cad.function = mbrot

Extra Slides

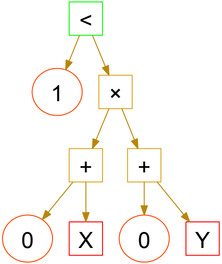
Parser-Level Optimizations

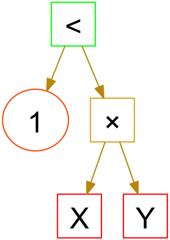
Tree Simplification




```
(X+0) * (Y+0) < 1
```

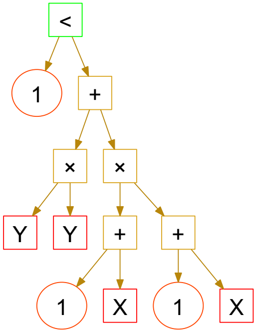
Tree Simplification





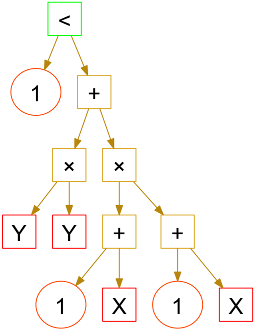

```
(X+0) * (Y+0) < 1
```

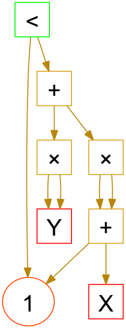
Node Combination




```
(X+1)*(X+1) + Y*Y < 1
```

Node Combination



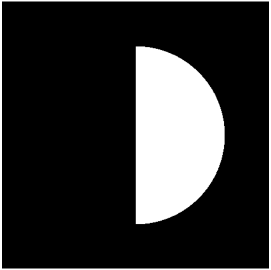



```
(X+1)*(X+1) + Y*Y < 1
```

Extra Slides

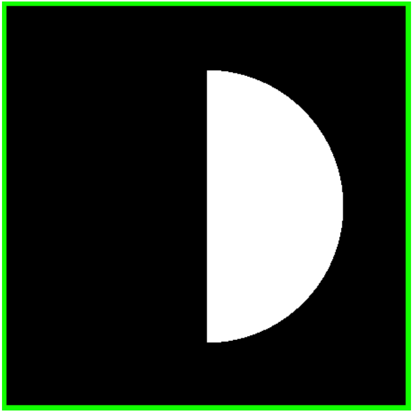
Solver-Level Optimizations

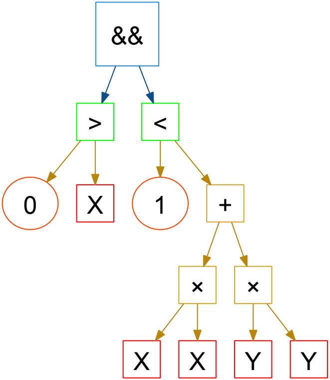
Branch Caching



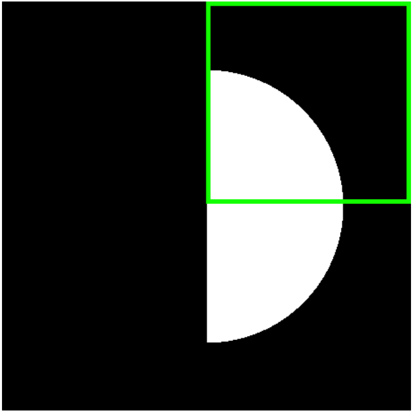
```
(X > 0) && (X*X + Y*Y < 1)
```

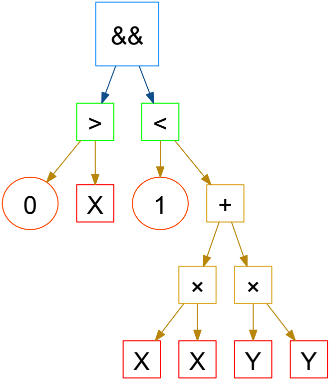
Branch Caching



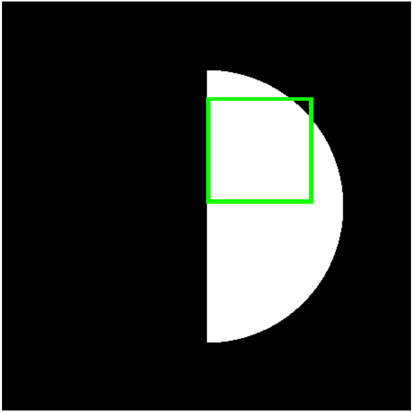


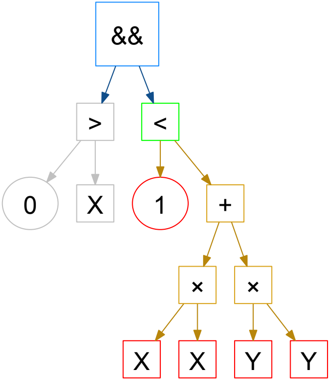
Branch Caching





Branch Caching



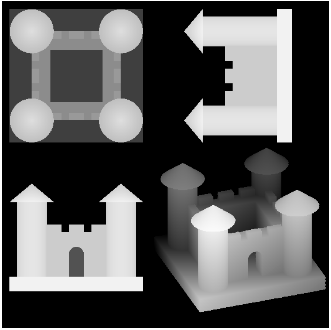


Multithreading

- Problem has parallel structure
- Distribute work over multiple cores:
  - Divide region evenly
  - Assign each core a subregion
- GPU could also be used

Z-culling

- For 3D objects, goal is height-map
- Skip evaluation if region is occluded



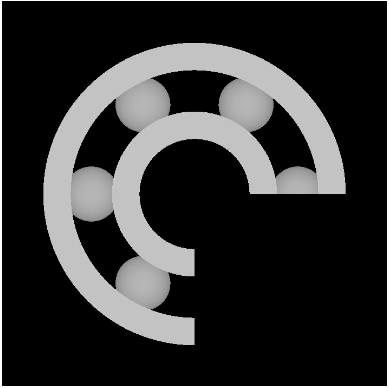
Extra Slides

Test Procedures & Results

Test Files

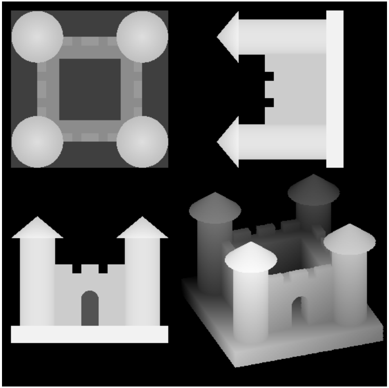
Alien

Test Files



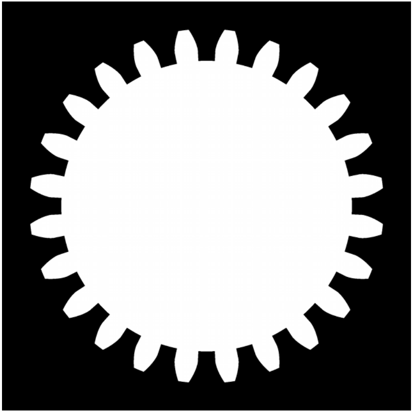
Bearing

Test Files



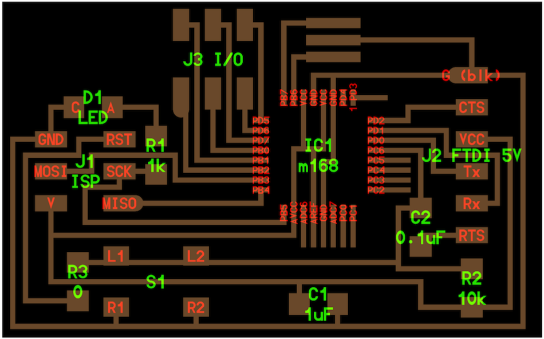
Castle

Test Files



Gear

Test Files



PCB

File Statistics

	Dimensions			Volume	File size
File	W	H	D	(MPixels)	(chars)
alien	3 555	3 555	1	12.6	1 880
bearing	711	711	237	119.8	1 414
castle	447	447	203	40.6	49 854
gear	1 904	1 904	1	3.6	8 128
pcb	2 273	1 460	1	3.3	378 743

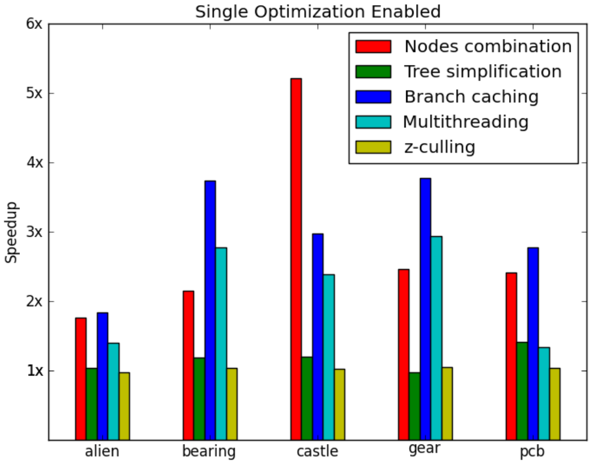
Speed Test Procedure

- Enable/disable one optimization (with all others optimizations disabled/enabled)
- Run 10x
- Find average run time
- Calculate speedup/slowdown

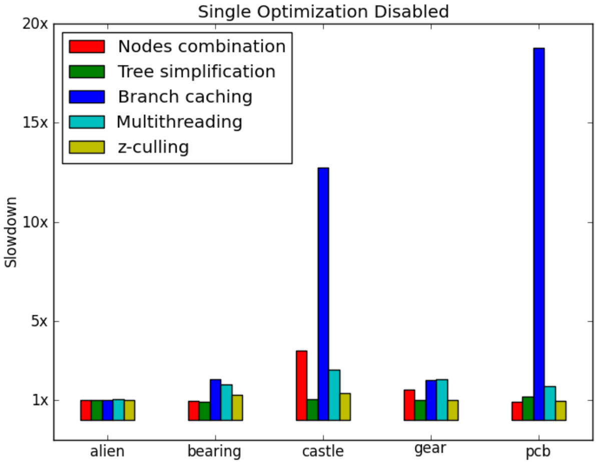
Caveat:

Behavior is sensitive to the selected resolution

Results


Single Optimization Enabled
Speedup
6x
5x
4x
3x
2x
1x
alien
bearing
castle
gear
pcb
Nodes combination
Tree simplification
Branch caching
Multithreading
z-culling

Results


Single Optimization Disabled
Nodes combination
Tree simplification
Branch caching
Multithreading
z-culling
Slowdown
20x
15x
10x
5x
1x
alien
bearing
castle
gear
pcb

Implementation & Code Details

Implementation Details

- 4,370 lines of C++.
- Inheritance is used for Node classes
- Parent class `Node` is derived into
    - `NonaryNode`
    - `UnaryNode`
    - `BinaryNode`

  (which are further derived into operator classes)

Evaluation Procedure

- Two solve functions:
  - Float (single point)
  - Interval (region)
- Nodes store results of evaluation locally
- Nodes with children look up children's locally stored results
- Children must be evaluated before parents!

Tree Data Structure

- Lists of nodes, sorted by weight into levels
  - Variables and constants: `weight = 0`
  - Others: `weight = max(child weights) + 1`
- Evaluate nodes with `weight = 0`, then nodes with `weight = 1`, then nodes with `weight = 2`, etc.
- This order of evaluation ensures that children are evaluated before parents.

Branch Cache Implementation

- Each level keeps a count of “active nodes”
- “Push” (recursing on sub-interval):
  - Swap unambiguous nodes to the back of the list
  - Deactivate children of unambiguous nodes
  - Decrement active node count.
  - Save the number of cached nodes
- “Pop” (returning from recursion):
  - Increment active node count
  - Revive cached nodes
  - Activate children of revived nodes