Formats for Fabrication

- How do we represent objects?
- 2D areas and 3D volumes
- Design → 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 → volume conversion
- Constructive solid geometry is hard / messy
Boundary Representations
Functional Representation

\[ X^2 + Y^2 < 1 \]
Functional Representation

\((X^2 + Y^2 < 1) \land (X^2 + Y^2 > 0.5)\)
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:
How to convert an expression into an image?

\[(X^2 + Y^2 < 1) \land (X^2 + Y^2 > 0.5)\]
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

$X + Y > 0$ becomes

Uses shunting-yard algorithm
Expressions $\rightarrow$ trees

$X + Y > 0$ becomes

```
>  
0 +
X Y
```
Expressions $\rightarrow$ trees

$X + Y > 0$ becomes

Uses shunting-yard algorithm
Tree of expressions operating on constants, variables, and other expressions.
Tree Structure

Distinct data types:
- Floating-point value/interval
- Tri-bool (true, false, or ambiguous)
- Color (32-bit integer)
Architectures

- **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

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

- isp
- alien
- gear

Time taken (seconds)
Resolution (pixels/mm)
Future Work

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

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?
Extra Slides

Parser-Level Optimizations
Tree Simplification

\[(X+0) \times (Y+0) < 1\]
Tree Simplification

\[(X+0) \times (Y+0) < 1\]
(X+1)*(X+1) + Y*Y < 1
Node Combination

\((X+1) \times (X+1) + Y \times Y < 1\)
Extra Slides

Solver-Level Optimizations
(X > 0) && (X*X + Y*Y < 1)
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

<table>
<thead>
<tr>
<th>File</th>
<th>Dimensions</th>
<th>Volume (MPixels)</th>
<th>File size (chars)</th>
</tr>
</thead>
<tbody>
<tr>
<td>alien</td>
<td>W 3555</td>
<td>H 3555</td>
<td>D 1</td>
</tr>
<tr>
<td></td>
<td>711</td>
<td>711</td>
<td>237</td>
</tr>
<tr>
<td>castle</td>
<td>447</td>
<td>447</td>
<td>203</td>
</tr>
<tr>
<td>gear</td>
<td>1904</td>
<td>1904</td>
<td>1</td>
</tr>
<tr>
<td>pcb</td>
<td>2273</td>
<td>1460</td>
<td>1</td>
</tr>
</tbody>
</table>
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

- **Nodes combination**
- **Tree simplification**
- **Branch caching**
- **Multithreading**
- **z-culling**

- alien
- bearing
- castle
- gear
- pcb

Speedup

6x
5x
4x
3x
2x
1x
Results
Extra Slides

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