Optimizing Tradeoffs of Non-Functional Properties in Software

Jonathan Dorn
July 20, 2017
Implementation Combinations

![Diagram showing points in a scatter plot with axes labeled 'Visual Inaccuracies' and 'Computation Complexity'.]
Implementation Combinations

![Graph showing the relationship between Visual Inaccuracies and Computation Complexity. The graph includes a line and several scattered data points.]
Implementation Combinations

Visual Inaccuracies vs. Computation Complexity
Thesis

*Search-based* software engineering techniques applying *local* software transformations can automatically and effectively explore *tradeoffs* between a variety of measurable *non-functional properties* in existing software artifacts with indicative workloads across application domains.
Non-Functional Properties

• Not “what” a program does, but “how well.”
  • “More” or “less;” “higher” or “lower.”

• Characterize implementations by how much of a property they posses.

• Often interact via tradeoffs.
  • E.g., performance vs. maintainability.
Optimization Philosophy

Program Transformations

• Un-annotated *source code*.
  • “Raw” C, Java, assembly.

• *Local* transformations.
  • E.g., change one function call or one line.
  • Likely to be independent.

Program Properties

• *Retain* functionality.

• Improvement *correlated* with human perception.

• Estimate properties *automatically*. 
Insights

• Adapt program repair.
  • *Evolutionary search*: Modify an existing “nearly correct” implementation.
  • *Regression testing*: Only consider programs that retain functionality.

• Adapt profile-guided optimization.
  • *Indicative workloads*: Short runs can indicate important opportunities.
Search-Based Optimization Framework

Input

- Evolutionary search
- Regression testing
- Indicative workloads

Output

Independent local transformations
Search-Based Optimization Framework
Outline

Overview

Application Domains

- **Graphics**: Run Time and Visual Quality
- **Data Centers**: Output Accuracy and Energy Use
- **Unit Tests**: Readability and Test Coverage

Concluding Thoughts
Outline

Overview

Application Domains

**Graphics:** Run Time and Visual Quality

**Data Centers:** Output Accuracy and Energy Use

**Unit Tests:** Readability and Test Coverage

Concluding Thoughts
Computer Generated Imagery

• Video games topped $90B in 2015.*
• Diagnostic imaging projected to top $30B by 2021.**
• Applications demand:
  • High-quality visuals.
  • Interactive performance.

Aliasing Example

Credit “Moire pattern of bricks” by Colin M.L. Burnett, via Wikimedia Commons, licensed under CC BY-SA 3.0.
Project Overview

• Goal:
  • Reduce aliasing (= improve *visual quality*) and retain interactive *run times*.

• Approach:
  • Replace expressions that cause aliasing with non-aliasing expressions.
Search-Based Optimization Framework

- **Input**
- **Search**
- **Evaluation**
- **Transformation**
- **Output**
Aliasing

• Caused when samples (pixels) are \textit{widely spaced} relative to details.
Aliasing

• Caused when samples (pixels) are *widely spaced* relative to details.
  • Reduce spacing (e.g., add more pixels = expensive!).
Aliasing

• Caused when samples (pixels) are \textit{widely spaced} relative to details.
  • Reduce spacing (e.g., add more pixels = expensive!).
  • Remove details (e.g., smoothing or “\textit{band-limiting}”).
Nyquist Limit

• Formally, aliasing is defined in terms of the Fourier transform of the image function.

• Nyquist-Shannon Sampling Theorem: Aliasing occurs when the image has frequencies greater than or equal to half the sampling frequency.
  • Band-limiting retains frequencies within a desired band.
Nyquist Limit

• Formally, aliasing is defined in terms of the Fourier transform of the image function.

\[ \text{Fourier transform} \]

• **Nyquist-Shannon Sampling Theorem**: Aliasing occurs when the image has frequencies greater than or equal to half the sampling frequency.
  
  • Band-limiting retains frequencies within a desired band.
Convolution Theorem

• *Product* of Fourier transforms of $f$ and $g$ is equal to the Fourier transform of the *convolution* of $f$ and $g$:

$$\mathcal{F}[f] \cdot \mathcal{F}[g] = \mathcal{F}[f \ast g]$$

$$f \ast g = \int_{-\infty}^{\infty} f(x - x')g(x') \, dx'$$
Band-Limiting

- **Convolve** the image with a filter *before sampling.*

\[ \hat{f}(x, w) = \int_{-\infty}^{\infty} f(x - x') g(x', w) \, dx \]
Band-Limiting

• *Convolve* the image with a filter *before sampling*.

\[ \hat{f}(x, w) = \int_{-\infty}^{\infty} f(x - x')g(x', w) \, dx \]

• Convolving shader programs.
  • Insight: *compose* band-limited sub-components.
## Our Band-Limiting Transformation

<table>
<thead>
<tr>
<th>$f(x)$</th>
<th>$\hat{f}(x,w)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x$</td>
<td>$x$</td>
</tr>
<tr>
<td>$x^2$</td>
<td>$x^2 + w^2$</td>
</tr>
<tr>
<td>$\text{frac}_1(x)$</td>
<td>$\frac{1}{2} - \sum_{n=1}^{\infty} \frac{\sin(2\pi n x)}{\pi n} e^{-2n^2 \pi^2 n^2}$</td>
</tr>
<tr>
<td>$\text{frac}_2(x)$</td>
<td>$\frac{1}{2w} \left( \text{frac}^2 \left( x + \frac{w}{2} \right) + \left[ x + \frac{w}{2} \right] - \text{frac}^2 \left( x - \frac{w}{2} \right) - \left[ x - \frac{w}{2} \right] \right)$</td>
</tr>
<tr>
<td>$\text{frac}_3(x)$</td>
<td>$\frac{1}{12w^2} \left[ f'(x-w) + f'(x+w) - 2f'(x) \right]$</td>
</tr>
</tbody>
</table>

where $f'(t) = 3t^2 + 2\text{frac}_3(t) - 3\text{frac}_2(t) + \text{frac}(t) - t$

| $|x|$ | $\text{erf} \left( \frac{x}{w \sqrt{2}} \right) + \sqrt{\frac{2}{\pi}} e^{-\frac{x^2}{2w^2}}$ |
| $|x|$ | $x - \text{frac}(x,w)$ |
| $[x]$ | $\text{floor}(x,w) + 1$ |
| $\cos x$ | $\cos x e^{-\frac{x^2}{2}}$ |
| $\text{saturation}(x)$ | $\frac{1}{2} \left( \text{erf} \left( \frac{x}{w \sqrt{2}} \right) - (x-1) \text{erf} \left( \frac{x-1}{w \sqrt{2}} \right) + \sqrt{\frac{2}{\pi}} \left( e^{-\frac{x^2}{2w^2}} - e^{-\frac{(w-1)^2}{2w^2}} \right) + 1 \right)$ |
| $\sin x$ | $\sin x e^{-\frac{x^2}{2}}$ |
| $\text{step}(a,x)$ | $\frac{1}{2} \left( 1 + \text{erf} \left( \frac{x-a}{w \sqrt{2}} \right) \right)$ |
| $\text{trunc}(x)$ | $\text{floor}(x,w) - \text{step}(x,w) + 1$ |

- Table of band-limited built-in functions.
- One-time manual effort.
- See appendix.

**Transformation:**

- Replace function call with band-limited function call.
Search-Based Optimization Framework

Input

Evaluation

Search

Genetic Algorithm

Transformation

Replace with band-limited function

Error & Run time

Output
Evaluation

• **Benchmarks**: 11 programs used in previous work on antialiasing.

• Compare against *16x supersampling*.

• **Metrics**:
  • *Error* relative to 2000x supersampling.
  • *Run time*.
Results: Checkerboard

Target Image  No Antialiasing  16x Supersampling  Our Approach
Results: Checkerboard

Target Image

No Antialiasing

16x Supersampling

Our Approach

Error heatmap
$L^2$ in RGB
Results: Checkerboard

- **4x faster** than super-sampling.
- **2x less $L^2$ (RGB) error** than supersampling.
Results: Brick and Wood

Target Image  No Antialiasing  16x Supersampling  Our Approach

5x faster, 3x more $L^2$ error than supersampling.

6x faster, 2x less $L^2$ error than supersampling.
Runtime Results

![Bar chart comparing Super-Sampling and Our Approach for various patterns like step, ridges, pulse, noise1, checker, circles1, wood, brick, noise2, circles2, and perlin. The x-axis represents different patterns, and the y-axis represents the normalized runtime. The chart shows that Super-Sampling generally has a higher normalized runtime compared to Our Approach.]
Error Results

![Error Results Graph]

- Normalized Error
- Super-Sampling
- Our Approach

- Images:
  - step
  - ridges
  - pulse
  - noise1
  - checker
  - circles1
  - wood
  - brick
  - noise2
  - circles2
  - perlin
Aliasing Reduction Summary

- Developed anti-aliasing approach for programs.
  - Derived and published band-limited expression for common programming language primitives.

- Added new Pareto non-dominated points to the design space.
  - In many cases, we dominate existing approach.

Outline

Overview

Application Domains

- **Graphics**: Run Time and Visual Quality
- **Data Centers**: Output Accuracy and Energy Use
- **Unit Tests**: Readability and Test Coverage

Concluding Thoughts
Data Center Energy Use

Percentages of US electricity use in a given year

<table>
<thead>
<tr>
<th>Year</th>
<th>Infrastructure</th>
<th>Communications</th>
<th>Storage</th>
<th>High-end servers</th>
<th>Mid-range servers</th>
<th>Volume servers</th>
</tr>
</thead>
<tbody>
<tr>
<td>2000</td>
<td>0.82%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2005</td>
<td>1.53%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2010</td>
<td>2.78%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Reproduced from [Koomey 2011]
Approximate Computing Applications

• “Correct” answer is unknown or not well defined.
  • Recommendation systems.
  • Search systems.
  • Prediction systems.
Project Overview

• Goal:
  • Reduce *energy* retaining human-acceptable output.

• Approach:
  • Optimize energy use and output error.
  • Identify largest energy reduction below error threshold.
Search-Based Optimization Framework

- Input
- Search
- Transformation
- Evaluation
- Output
# Measuring Program Energy

## Considerations

- Performance / response time
- Precision and accuracy
- Disaggregation
  - Workload setup and cleanup
  - Daemon processes
- System configuration
  - Core allocation
  - Device sleep

## Mechanisms
# Measuring Program Energy

## Considerations

- Performance / response time
- Precision and accuracy
- Disaggregation
  - Workload setup and cleanup
  - Daemon processes
- System configuration
  - Core allocation
  - Device sleep

## Mechanisms
Measuring Program Energy

**CONSIDERATIONS**

- Performance / response time
- Precision and accuracy
- Disaggregation
  - Workload setup and cleanup
  - Daemon processes
- System configuration
  - Core allocation
  - Device sleep

**MECHANISMS**

- Simulation
  - gem5
- Power model
  - Intel Power Gadget
  - Mac Activity Monitor
- Physical
  - Commodity energy meter
  - Phasor Measurement Unit
  - Custom-built
Measuring Program Energy

**CONSIDERATIONS**

- Performance / response time
- Precision and accuracy
- Disaggregation
  - Workload setup and cleanup
  - Daemon processes
- System configuration
  - Core allocation
  - Device sleep

**MECHANISMS**

- Simulation
  - gem5
- Power model
  - Intel Power Gadget
  - Mac Activity Monitor
- Physical
  - Commodity energy meter
  - Phasor Measurement Unit
  - Custom-built
Measuring Program Energy

**Considerations**

- Performance / response time
- Precision and accuracy
- Disaggregation
  - Workload setup and cleanup
  - Daemon processes
- System configuration
  - Core allocation
  - Device sleep

**Mechanisms**

- Simulation
  - gem5
- Power model
  - Intel Power Gadget
  - Mac Activity Monitor
- Physical
  - Commodity energy meter
  - Phasor Measurement Unit
  - Custom-built

*Slow*  

*Inaccurate*
Measuring Program Energy

**Considerations**

- Performance / response time
- Precision and accuracy
- Disaggregation
  - Workload setup and cleanup
  - Daemon processes
- System configuration
  - Core allocation
  - Device sleep

**Mechanisms**

- Simulation
  - gem5
- Power model
  - Intel Power Gadget
  - Mac Activity Monitor
- Physical
  - Commodity energy meter
  - Phasor Measurement Unit
  - Custom-built
Measuring Program Energy

**CONSIDERATIONS**
- Performance / response time
- Precision and accuracy
- Disaggregation
  - Workload setup and cleanup
  - Daemon processes
- System config
  - Core allocation
  - Device sleep

**MECHANISMS**
- Simulation
  - gem5
- Power model
  - Intel Power Gadget
  - Mac Activity Monitor
- Physical
  - Commodity energy meter
  - Phasor Measurement Unit
  - Custom-built

- Slow
- Inaccurate
- Coarse-grained
- Cost prohibitive
Fast and Accurate Physical Energy Measurement

- Sampling rate:
  - Internal: 1200 Hz
  - External: 10-20 Hz
- Variance < 1W on 100W load.
- $100 per system monitored.
Search-Based Optimization Framework

Input

Search

Evalulation

Insert, Delete, & Swap

Transformation

Genetic Algorithm

Energy & Error

Output
Evaluation

- **Benchmarks**: PARSEC suite, large data center applications.

- Compare against “loop perforation.”

- **Metrics**:
  - *Energy use*.
  - *Error* (application-specific, relative to original).
# Data Center Benchmarks (PARSEC)

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Application Domain</th>
<th>Error Metric</th>
</tr>
</thead>
<tbody>
<tr>
<td>blackscholes</td>
<td>Financial analysis</td>
<td>RMSE</td>
</tr>
<tr>
<td>bodytrack</td>
<td>Computer vision</td>
<td>RMSE</td>
</tr>
<tr>
<td>ferret</td>
<td>Similarity search</td>
<td>Kendall’s $\tau$</td>
</tr>
<tr>
<td>fluidanimate</td>
<td>Animation</td>
<td>Hamming Distance</td>
</tr>
<tr>
<td>freqmine</td>
<td>Data mining</td>
<td>RMSE</td>
</tr>
<tr>
<td>swaptions</td>
<td>Financial analysis</td>
<td>RMSE</td>
</tr>
<tr>
<td>vips</td>
<td>Media processing</td>
<td>Image Similarity</td>
</tr>
<tr>
<td>x264</td>
<td>Media processing</td>
<td>Image Similarity</td>
</tr>
</tbody>
</table>
## Data Center Benchmarks

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Application Domain</th>
<th>Error Metric</th>
</tr>
</thead>
<tbody>
<tr>
<td>blackscholes</td>
<td>Financial analysis</td>
<td>RMSE</td>
</tr>
<tr>
<td>bodytrack</td>
<td>Computer vision</td>
<td>RMSE</td>
</tr>
<tr>
<td>ferret</td>
<td>Similarity search</td>
<td>Kendall’s $\tau$</td>
</tr>
<tr>
<td>fluidanimate</td>
<td>Animation</td>
<td>Hamming Distance</td>
</tr>
<tr>
<td>freqmine</td>
<td>Data mining</td>
<td>RMSE</td>
</tr>
<tr>
<td>swapoptions</td>
<td>Financial analysis</td>
<td>RMSE</td>
</tr>
<tr>
<td>vips</td>
<td>Media processing</td>
<td>Image Similarity</td>
</tr>
<tr>
<td>x264</td>
<td>Media processing</td>
<td>Image Similarity</td>
</tr>
<tr>
<td>blender</td>
<td>3D renderer</td>
<td>Image Similarity</td>
</tr>
<tr>
<td>libav</td>
<td>Media processing</td>
<td>Image Similarity</td>
</tr>
</tbody>
</table>
## Data Center Benchmarks

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Application Domain</th>
<th>Error Metric</th>
</tr>
</thead>
<tbody>
<tr>
<td>blackscholes</td>
<td>Financial analysis</td>
<td>RMSE</td>
</tr>
<tr>
<td>bodytrack</td>
<td>Computer vision</td>
<td>RMSE</td>
</tr>
<tr>
<td>ferret</td>
<td>Similarity search</td>
<td>Kendall’s $\tau$</td>
</tr>
<tr>
<td>fluidanimate</td>
<td>Animation</td>
<td>Hamming Distance</td>
</tr>
<tr>
<td>freqmine</td>
<td>Data mining</td>
<td>RMSE</td>
</tr>
<tr>
<td>swapoptions</td>
<td>Financial analysis</td>
<td></td>
</tr>
<tr>
<td>vips</td>
<td>Media processing</td>
<td></td>
</tr>
<tr>
<td>x264</td>
<td>Media processing</td>
<td>Image Similarity</td>
</tr>
<tr>
<td>blender</td>
<td>3D renderer</td>
<td>Image Similarity</td>
</tr>
<tr>
<td>libav</td>
<td>Media processing</td>
<td>Image Similarity</td>
</tr>
</tbody>
</table>

Order of magnitude larger. Evaluate scalability.
Acceptable Error

• Highly subjective and domain-specific.

• Protocol:
  • Noticeable distortion on casual viewing (blender, bodytrack, libav, vips, x264).
  • All values within 5% of original (blackscholes, freqmine, swaptions).
  • At least half of search results in common (ferret).
  • No acceptable error (fluidanimate).
# Energy Reduction Results (%)

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>No Error</th>
<th>Acceptable Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>blackscholes</td>
<td>92</td>
<td>92</td>
</tr>
<tr>
<td>bodytrack</td>
<td>0</td>
<td>59</td>
</tr>
<tr>
<td>ferret</td>
<td>0</td>
<td>30</td>
</tr>
<tr>
<td>fluidanimate</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>freqmine</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>swaptions</td>
<td>39</td>
<td>68</td>
</tr>
<tr>
<td>vips</td>
<td>21</td>
<td>29</td>
</tr>
<tr>
<td>x264</td>
<td>0</td>
<td>65</td>
</tr>
<tr>
<td>blender</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>libav</td>
<td>3</td>
<td>92</td>
</tr>
</tbody>
</table>
PARSEC Results

Our technique
Loop perforation

bodytrack

ferret

Error

Joules

Error

Joules
PARSEC Results

- **Our technique**
- **Loop perforation**

### swaptions

![Graph for swaptions](image1)

- **Error**
- **Joules**

### x264

![Graph for x264](image2)

- **Error**
- **Joules**
Can You Spot the Difference?
Can You Spot the Difference?

65% lower energy
Energy Optimization Summary

- Designed and built cost-effective energy meter.
  - Sub-second accuracy.
  - HW and SW designs are open-source.
- 41% average energy reduction with human-acceptable error.
- Submitted to TSE (*Reviewed and revised*).
Outline

Overview

Application Domains

Graphics: Run Time and Visual Quality
Data Centers: Output Accuracy and Energy Use
Unit Tests: Readability and Test Coverage

Concluding Thoughts
Expensive Testing Failures

• Mars Spirit Rover ($1B).
  • Almost lost mission due to filesystem bug.*

• Knight Capital trading glitch ($440M).
  • Development software released into production.

• Inadequate testing costs the US over $60B.***

Test Coverage

• Approximate measure of test suite quality.
  • Lines, branches, conditions, etc.
  • Mutation testing.

• Many standards and organizations mandate particular thresholds.
  • DO-178B (avionics software)
  • ANSI/IEEE Std 1008-1987 (software unit testing)
Developer Time in IDEs

Adapted from [Beller, et al. 2015]
Project Overview

• Goal:
  • Generate *readable, high-coverage* test suites.

• Approach:
  1. Model test readability.
  2. Optimize coverage and readability.
  3. Validate with human study.
Search-Based Optimization Framework

Input

Evaluation

Search

Transformation

Output
Readability Models

• Extract **features** from source code.
  • E.g., average line length, total unique identifiers.

• Conduct **human study** to collect ratings.
  • Java familiarity quiz.

• **Linear regression** model.
Generating Test Suites

- Extend **EVO**S**UITE** test suite generator for Java.
- Optimizes coverage objectives via evolutionary search.

```java
CharRange charRange0 = CharRange.isNot('#');
Character character0 = Character.valueOf('#');
CharRange charRange1 =
    CharRange.isNotIn('"', (char) character0);
char char0 = charRange1.getStart(); assertEquals('"', char0);

boolean boolean0 = charRange0.contains('"');
assertTrue(boolean0);
```
Generating Test Suites

• Extend **EVO**SUIT**E** test suite generator for Java.
  • Optimizes coverage objectives via evolutionary search.

• Extend fitness function with readability model.
Generating Test Suites

• Extend **EvoSuite** test suite generator for Java
  • Optimizes coverage objectives via evolutionary search.

• Extend fitness function with readability model.
  1. **EvoSuite** uses redundant instructions for diversity.
     • Converted to additional coverage in later generations.
  2. Redundant instructions reduce readability.
  3. Redundancy eliminated before being exploited.
Generating Test Suites

• Extend **EvoSuite** test suite generator for Java
  • Optimizes coverage objectives via evolutionary search.

• **Extend fitness function with readability model.**

• Optimize coverage, then readability.
  • Two-phase optimization.
  • Transformation should maintain coverage.
Readability Transformation

• Transformation:
  • Replace RHS of assignment with same-type expression.
  • Remove dead code.

```java
Foo foo = new Foo();
Bar bar = new Bar("Some parameter", 17);
foo.setBar(bar);
assertTrue(foo.isBar());
```
Readability Transformation

• Transformation:
  • Replace RHS of assignment with same-type expression.
  • Remove dead code.

```java
Foo foo = new Foo();
Bar bar = new Bar("Some parameter", 17);
foo.setBar(bar);
assertTrue(foo.isBar());
```

```java
Foo foo = new Foo();
Bar bar = new Bar("Some parameter", 17);
foo.setBar(null);
assertTrue(foo.isBar());
```
Search-Based Optimization Framework

2-stages: Genetic Algorithm & Hill Climbing

Replace with same-type expression.

Readability Metric & Coverage

Transformation

Input

Search

Evaluation

Output
Evaluation

- **Benchmarks**: 30 Java classes taken from 10 open-source projects.

- Fitness metrics (for search):
  - Coverage.
  - Readability metric.

- Real-world validation:
  - Human ratings of readability.
  - Human understanding of generated tests.
Head-to-Head Comparison

Test Case A

class CommandLine_ESTest {
    @Test
    public void test0() throws Throwable {
        CommandLine commandLine0 = new CommandLine();
        boolean boolean0 = commandLine0.hasOption("-V");
        String string0 = commandLine0.getOptionValue("-V");
        Option option0 = new Option((String) null, "-V");
        commandLine0.addOption(option0);
        boolean boolean1 = commandLine0.hasOption("-V");
        assertFalse(boolean1 == boolean0);
        assertTrue(boolean1);
    }
}

Test Case B

class CommandLine_ESTest {
    @Test
    public void test0() throws Throwable {
        CommandLine commandLine0 = new CommandLine();
        Option option0 = new Option("", false, "");
        commandLine0.addOption(option0);
        boolean boolean0 = commandLine0.hasOption("-");
        assertTrue(boolean0);
    }
}
Human Preference Results

Average: 69%
Test Understanding

```java
package org.apache.commons.cli;

import static org.junit.Assert.*;
import org.junit.Test;
import org.apache.commons.cli.Option;

public class Option_ESTest {

    @Test
    public void test0() throws Throwable {
        Option option0 = new Option((String) null, " ");
        // Undeclared exception!
        try {
            int int0 = option0.getId();
            fail("Expecting exception: NullPointerException");
        } catch(NullPointerException e) {
            //
            // no message in exception (getMessage() returned null
            //
        }
    }
}
```
Test Understanding Results

Time to Answer

Minutes

Optimized
Not Optimized

Avg
Readable Test Suite Summary

• Developed effective readability model for tests.
• Algorithm to optimize readability and coverage.
• Empirical evaluation of test readability on human performance.
• Distinguished Paper at ESEC-FSE 2015.
Outline

Overview

Application Domains

- **Graphics**: Run Time and Visual Quality
- **Data Centers**: Output Accuracy and Energy Use
- **Unit Tests**: Readability and Test Coverage

Concluding Thoughts
Contributions

• Representations, transformations, and search strategies for optimizing non-functional properties.
• Empirical evaluations of evolutionary optimization of non-functional properties in three application domains.
• First project to automatically band-limit procedural shaders.
• Derivations for band-limiting shading language primitives.
• Demonstration of optimizations enabled by relaxing requirement of bitwise output equivalence.
• Demonstration of impact of readability of maintenance activities.


Optimizing Tradeoffs of Non-Functional Properties in Software
BACKUP
Results: Brick and Wood

Target Image  No Antialiasing  16x Supersampling  Our Approach

5x faster, 3x more $L^2$ error than supersampling.

6x faster, 2x less $L^2$ error than supersampling.
Results: Noise1 and Noise2

Target Image | No Antialiasing | 16x Supersampling | Our Approach
---|---|---|---

7x faster, same $L^2$ error as supersampling.

6x faster, sane $L^2$ error as supersampling.
### Results: Circles2 and Perlin

<table>
<thead>
<tr>
<th>Target Image</th>
<th>No Antialiasing</th>
<th>16x Supersampling</th>
<th>Our Approach</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.jpg" alt="Image" /></td>
<td><img src="image2.jpg" alt="Image" /></td>
<td><img src="image3.jpg" alt="Image" /></td>
<td><img src="image4.jpg" alt="Image" /></td>
</tr>
</tbody>
</table>

**32x faster, 2x more $L^2$ error than supersampling.**

<table>
<thead>
<tr>
<th>Target Image</th>
<th>No Antialiasing</th>
<th>16x Supersampling</th>
<th>Our Approach</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image5.jpg" alt="Image" /></td>
<td><img src="image6.jpg" alt="Image" /></td>
<td><img src="image7.jpg" alt="Image" /></td>
<td><img src="image8.jpg" alt="Image" /></td>
</tr>
</tbody>
</table>

**18x faster, 2x more $L^2$ error than supersampling.**
Assembly Optimization Example

.L23:
  ...
  cmpl  %r13d, 40(%rsp)

  movq  16(%rsp), %r9
  movsd  %xmm0, (%r9)
  je    .L9
  ...
  call  _Z12CumNormalInvd
Assembly Optimization Example

.L23: Top of one unrolling of inner loop
  ...
  cmp %r13d, 40(%rsp) Loop condition check

  movq 16(%rsp), %r9
  movsd %xmm0, (%r9)
  je .L9 Jumps out of loop
  ...
  call _Z12CumNormalInvd
Assembly Optimization Example

.L23:

...  
      cmpl %r13d, 40(%rsp)
      xorl %eax, %eax  \[Resets condition flags\]
      movq 16(%rsp), %r9
      movsd %xmm0, (%r9)
      je   .L9 \[Always exits loop!\]
...  
call _Z12CumNormalInvd
Assembly Optimization Example

.L23:

...  
cmpl %r13d, 40(%rsp)
xorl %eax, %eax
movq 16(%rsp), %r9
movsd %xmm0, (%r9)
je .L9
...
call _Z12CumNormalInvd

• *No change* in observed behavior.

• Skipped iterations increase precision.

• Fixed number of digits in output.
Energy and Runtime

Energy Reduction

-20% 0% 20% 40% 60% 80% 100%

Energy
Runtime

blackscholes
bodytrack
ferret
fluidanimate
freqmine
swaptions
vips
x264
Feature Predictive Power

- Max line length
- Total identifier length
- Avg identifier length
- Total constructor calls
- Total line length
- Max identifier length
- Total unique identifiers
- Total identifiers
- Total distinct methods
- Token entropy
- Total floats
- Max nulls
- Avg string length
- Total numbers
- Avg loops
- Avg nulls
- Identifier ratio
- Has exceptions
- Total assertions
- Avg branches
- Max characters
- Avg arithmetic operators
- Method ratio
- Has assertions

0 0.1 0.2 0.3 0.4 0.5