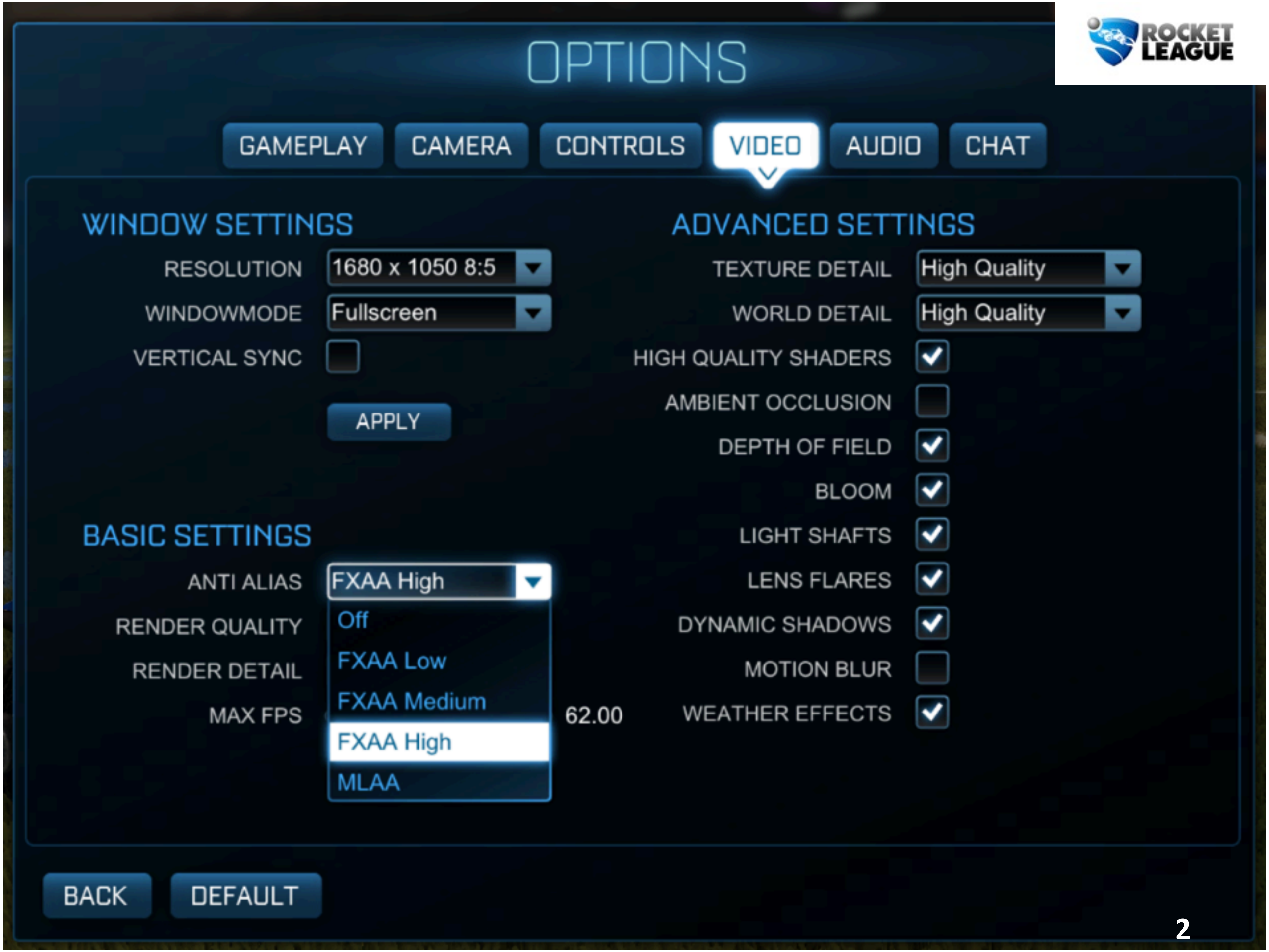
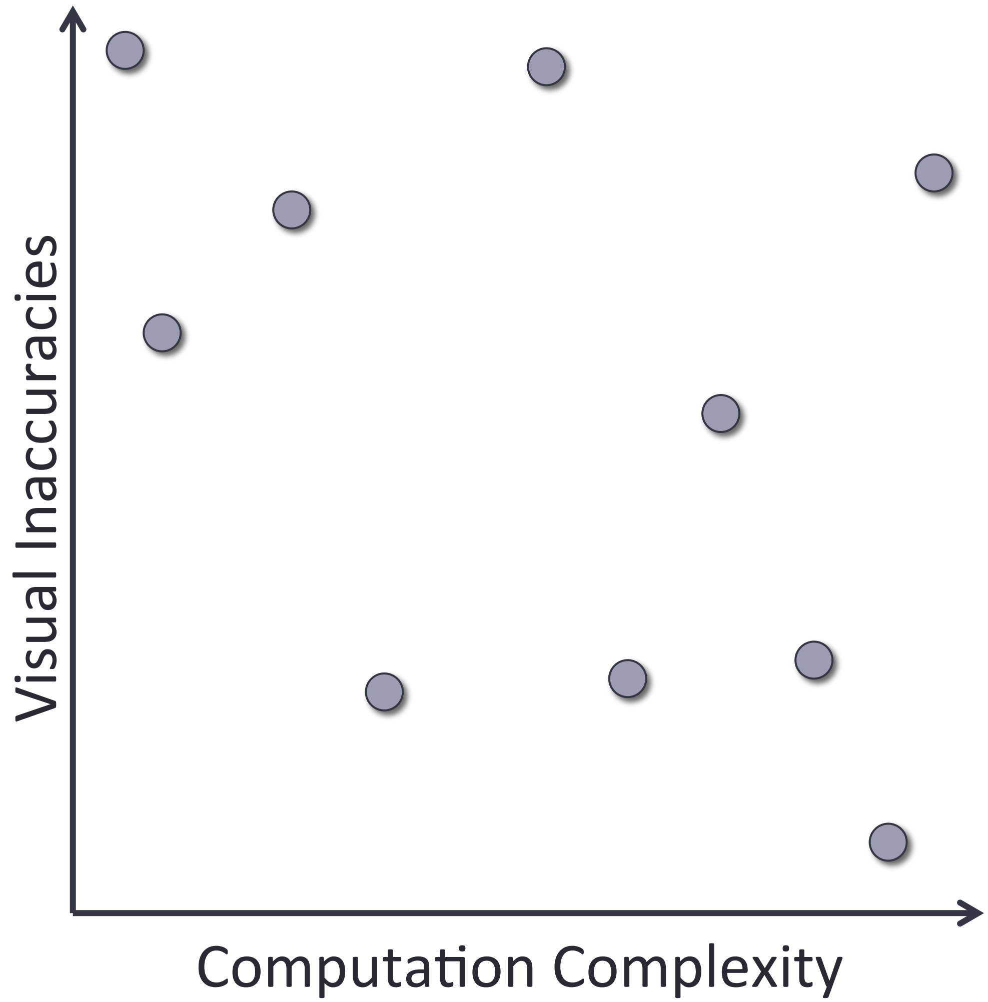
Optimizing Tradeoffs of Non-Functional Properties in Software

Jonathan Dorn

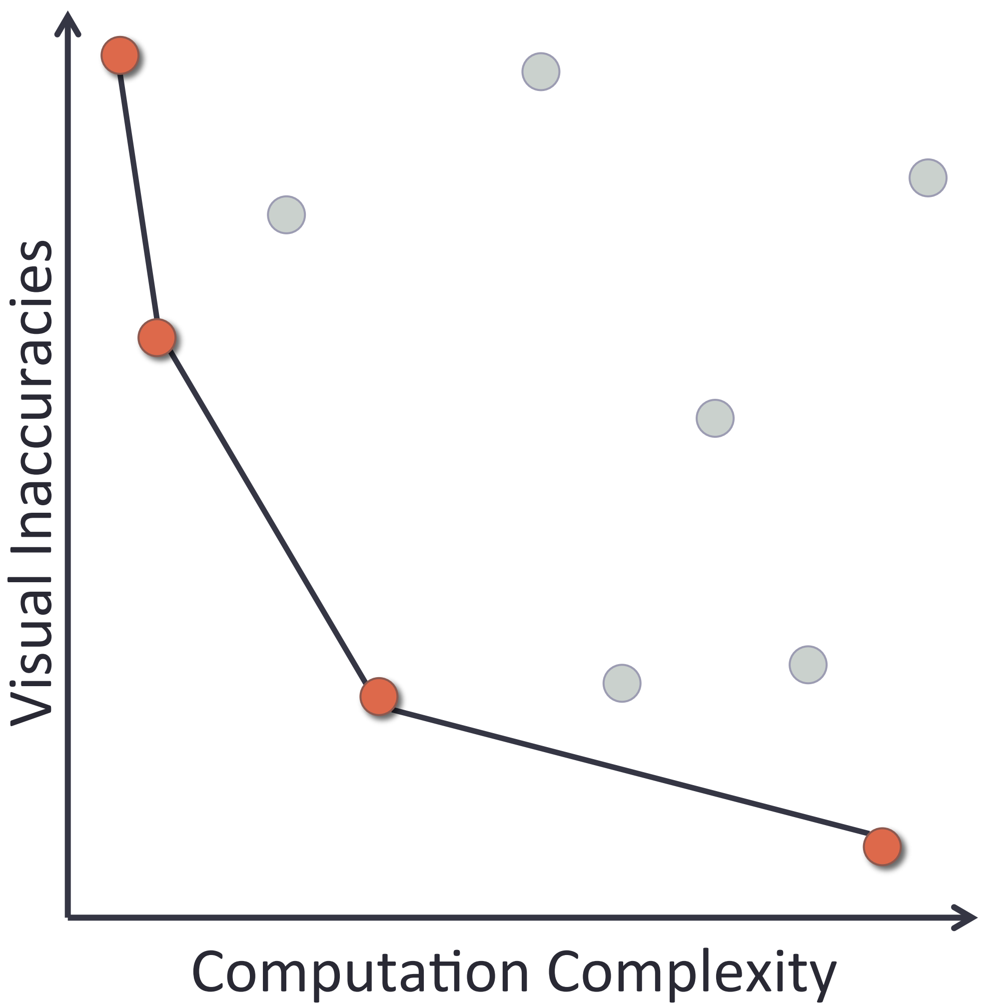
July 20, 2017


OPTIONS
GAMEPLAY
CAMERA
CONTROLS
VIDEO
AUDIO
CHAT
WINDOW SETTINGS
RESOLUTION
1680 x 1050 8:5
WINDOWMODE
Fullscreen
VERTICAL SYNC
APPLY
BASIC SETTINGS
ANTI ALIAS
FXAA High
RENDER QUALITY
RENDER DETAIL
MAX FPS
Off
FXAA Low
FXAA Medium
FXAA High
MLAA
62.00
ADVANCED SETTINGS
TEXTURE DETAIL
High Quality
WORLD DETAIL
High Quality
HIGH QUALITY SHADERS
AMBIENT OCCLUSION
DEPTH OF FIELD
BLOOM
LIGHT SHAFTS
LENS FLARES
DYNAMIC SHADOWS
MOTION BLUR
WEATHER EFFECTS
BACK
DEFAULT

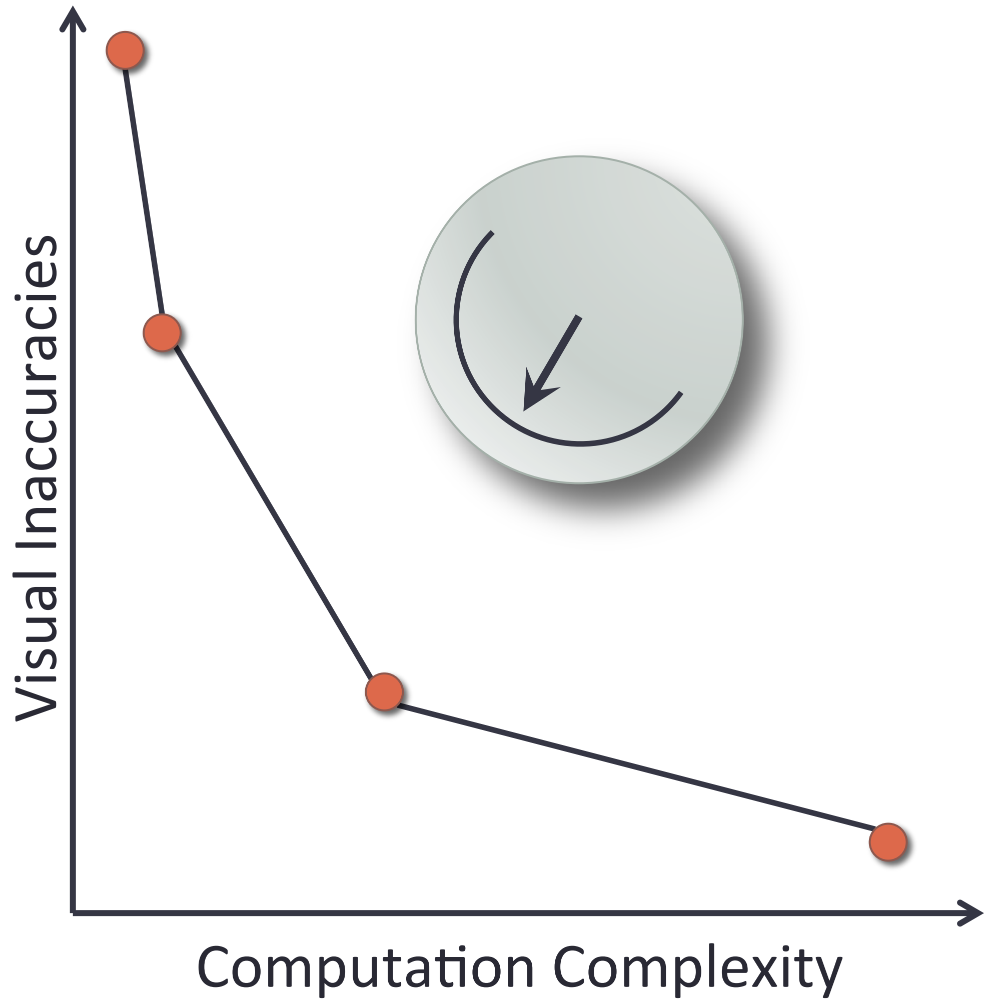
Implementation Combinations



Implementation Combinations



Implementation Combinations



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.

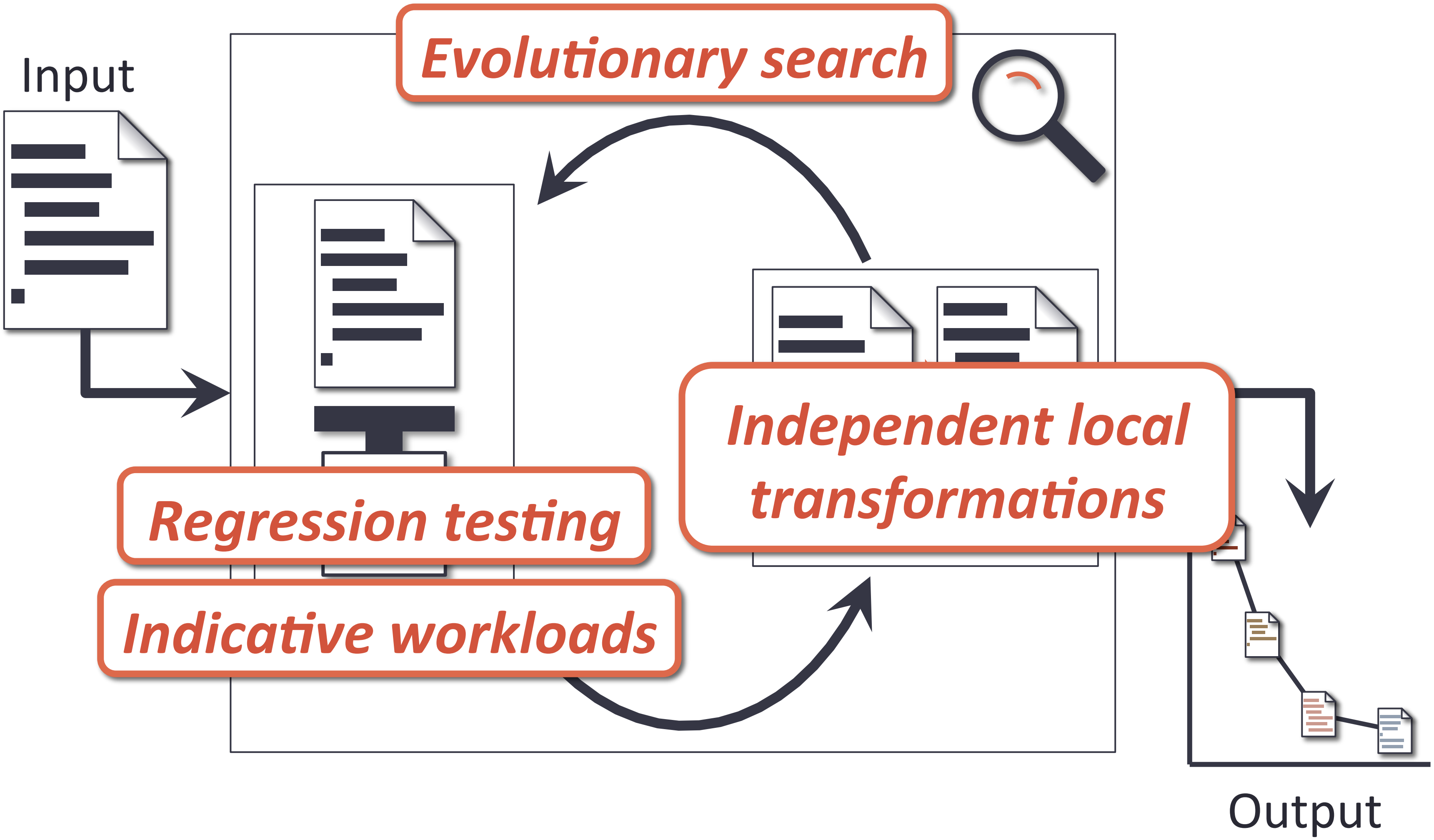
Program Properties

- ***Retain*** functionality.
- Improvement ***correlated*** with human perception.
- Estimate properties ***automatically***.

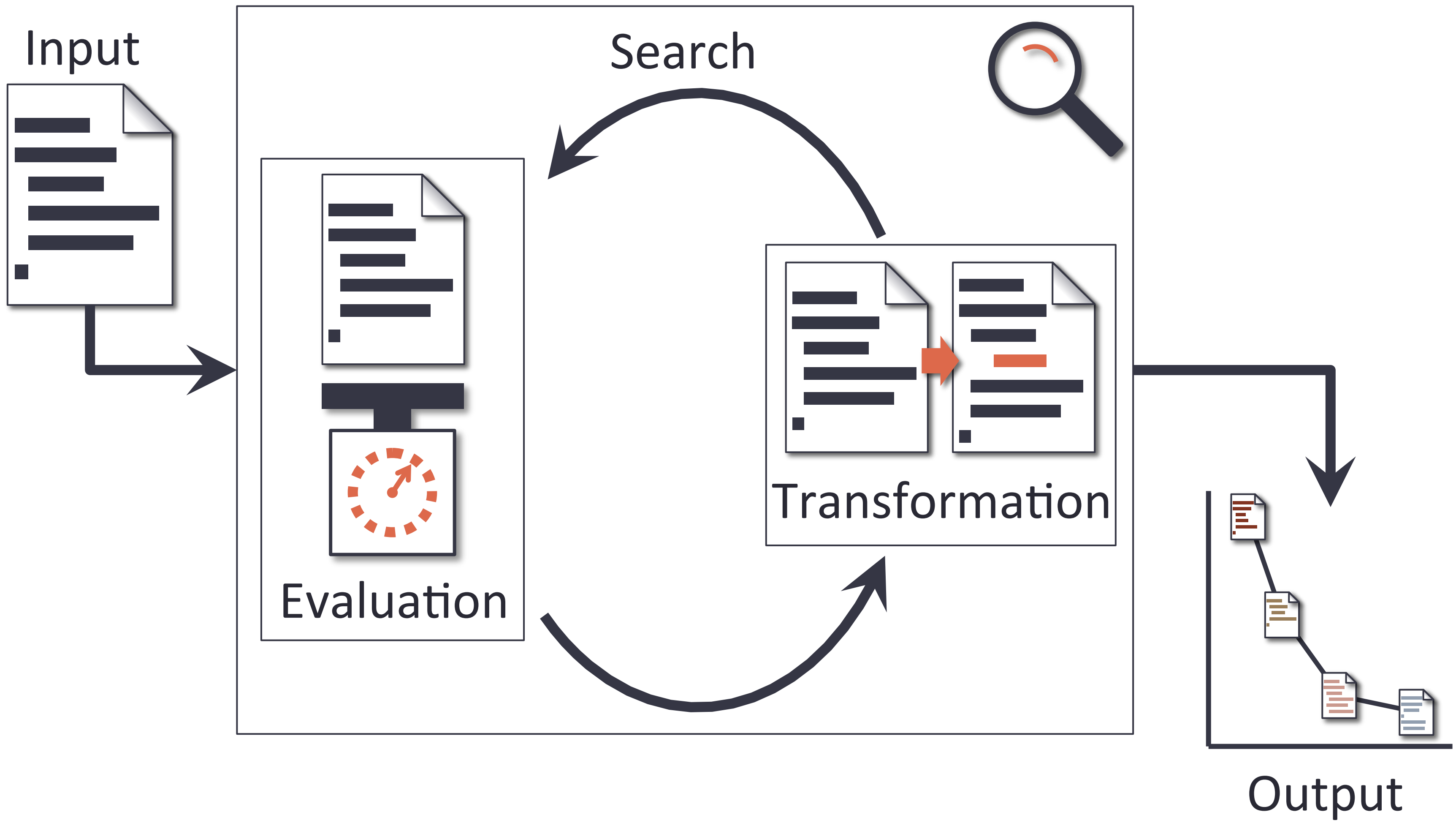
Insights

- 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



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

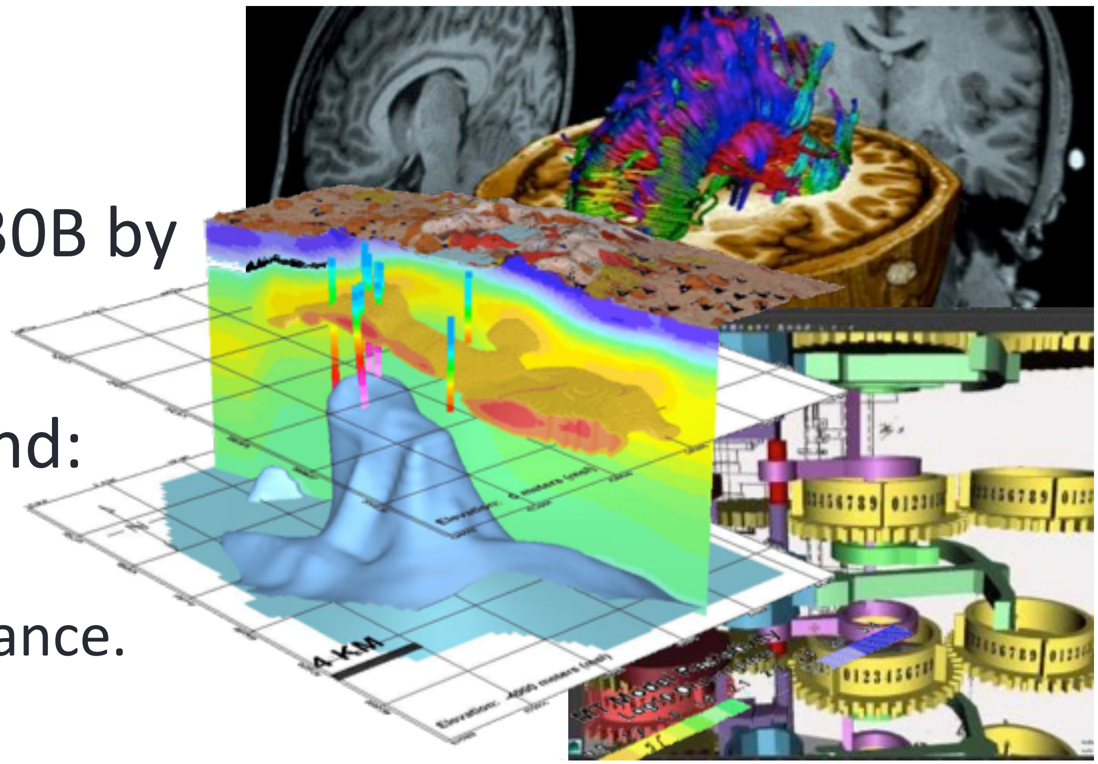
Application Domains

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

Concluding Thoughts

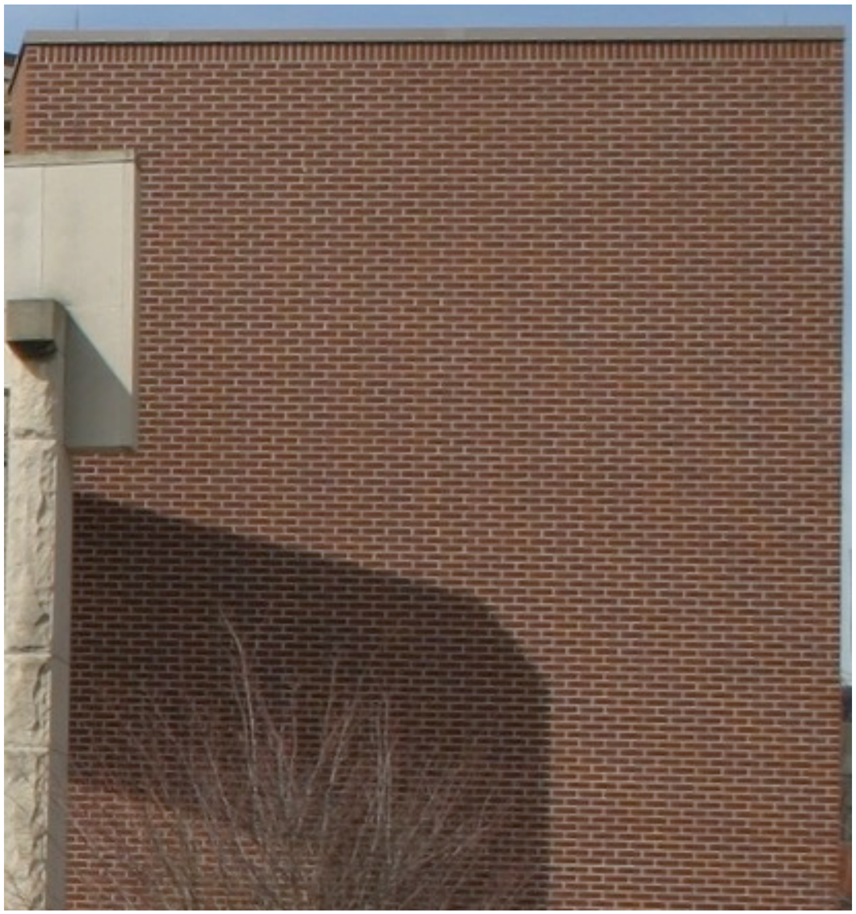
Computer Generated Imagery

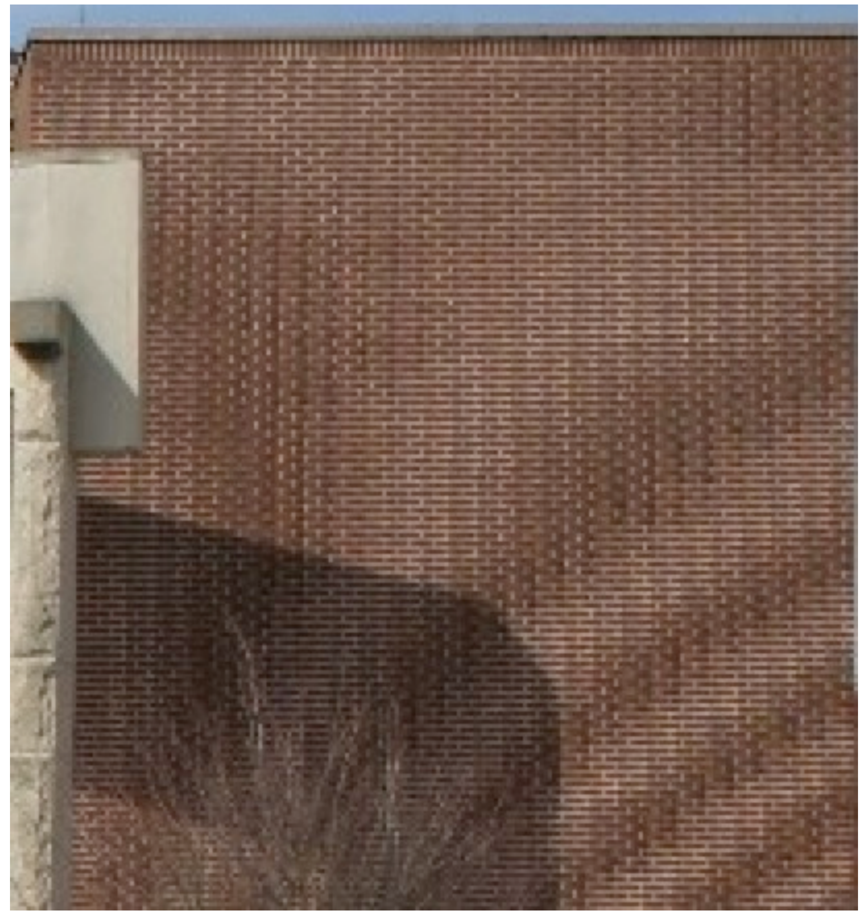
- Video games topped $90B in 2015.*
- Diagnostic imaging projected to top $30B by 2021.**
- Applications demand:
  - High-quality visuals.
  - Interactive performance.



* http://www.gamesindustry.biz/articles/2015-04-22-gaming-will-hit-usd91-5-billion-this-year-newzoo
** http://www.marketsandmarkets.com/PressReleases/diagnostic-imaging-market.asp

Aliasing Example

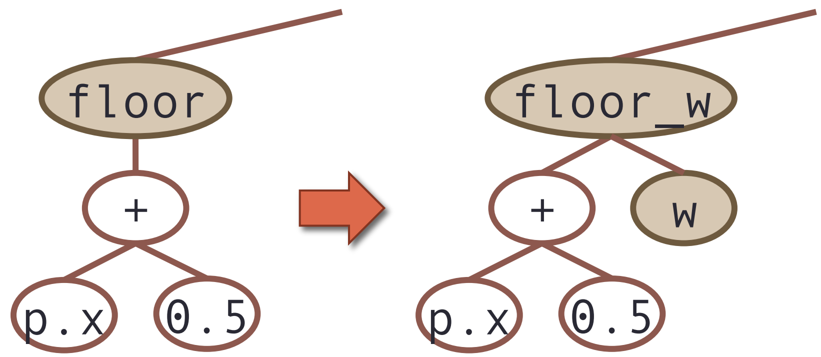




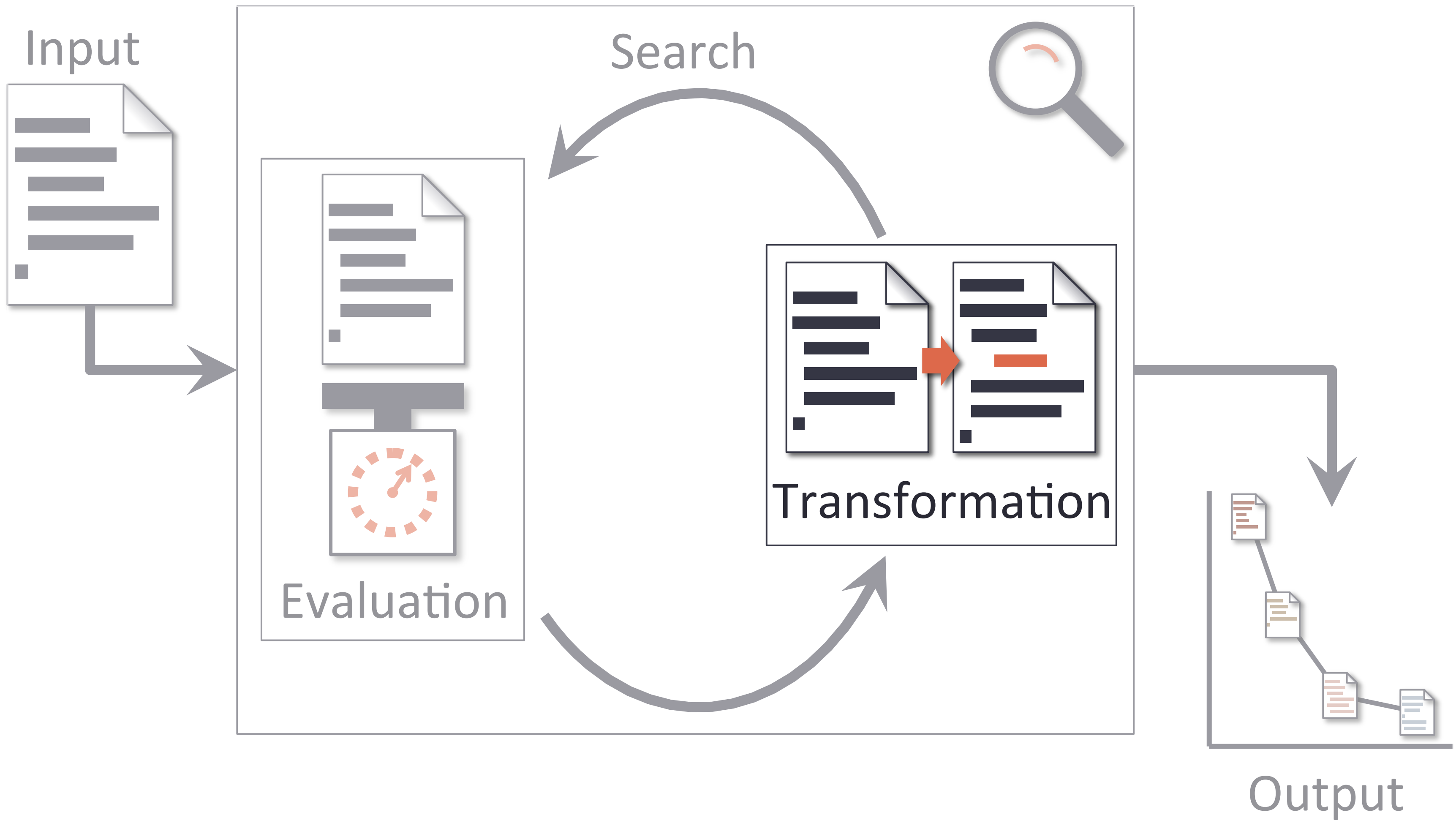
Credit “Moire pattern of bricks” by Colin M.L. Burnett, via Wikimedia Commons, licensed under CC BY-SA 3.0.

Project Overview

- Goal:
  - Reduce aliasing (= improve ***visual quality***) and retain interactive ***run times***.
- Approach:
  - Replace expressions that cause aliasing with non-aliasing expressions.

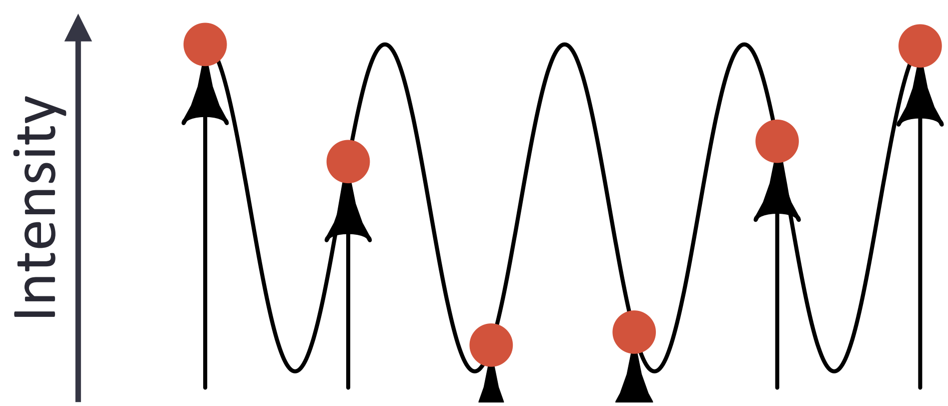


Search-Based Optimization Framework



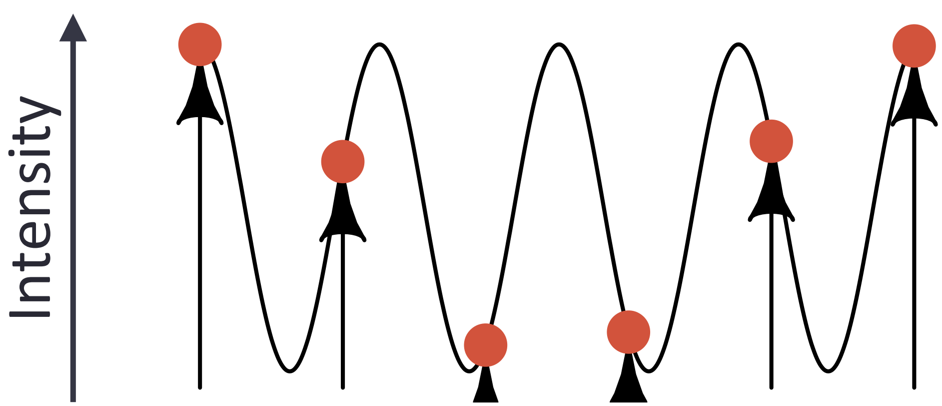
Aliasing

- Caused when samples (pixels) are ***widely spaced*** relative to details.



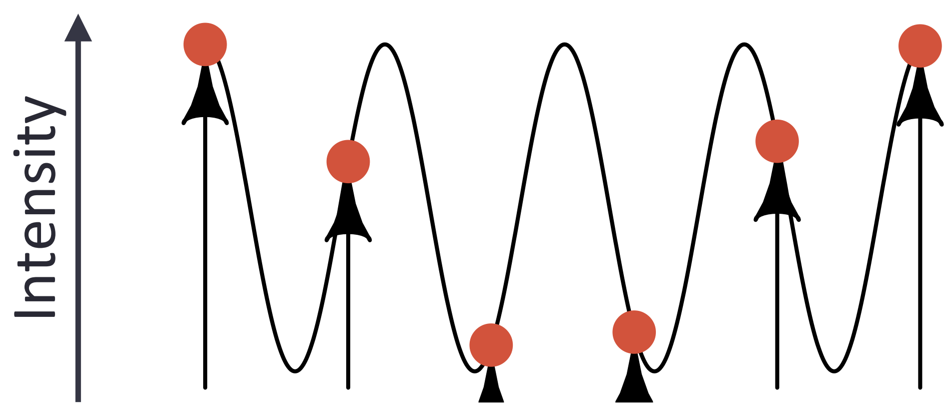
Aliasing

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



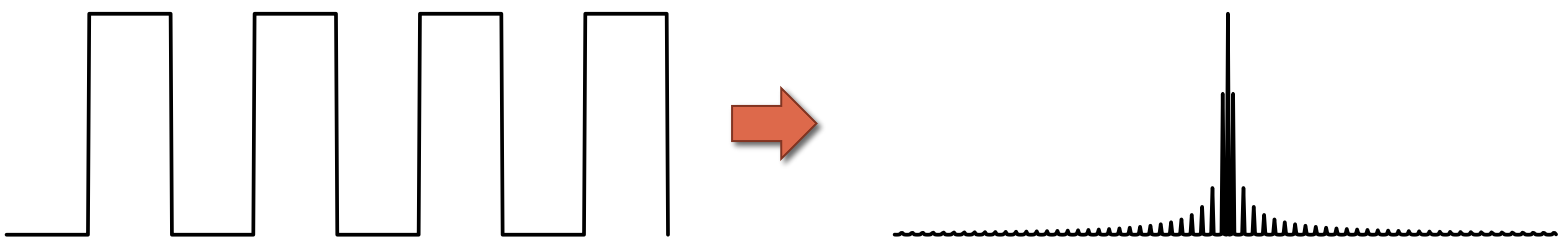
Aliasing

- Caused when samples (pixels) are ***widely spaced*** relative to details.
  - Reduce spacing (e.g., add more pixels = expensive!).
  - Remove details (e.g., smoothing or "***band-limiting***").



Nyquist Limit

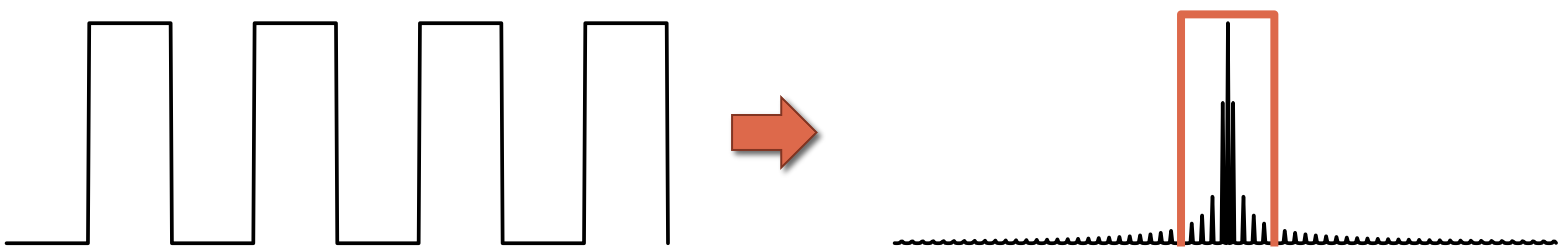
- Formally, aliasing is defined in terms of the ***Fourier transform*** of the image function.



- **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.



- **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:

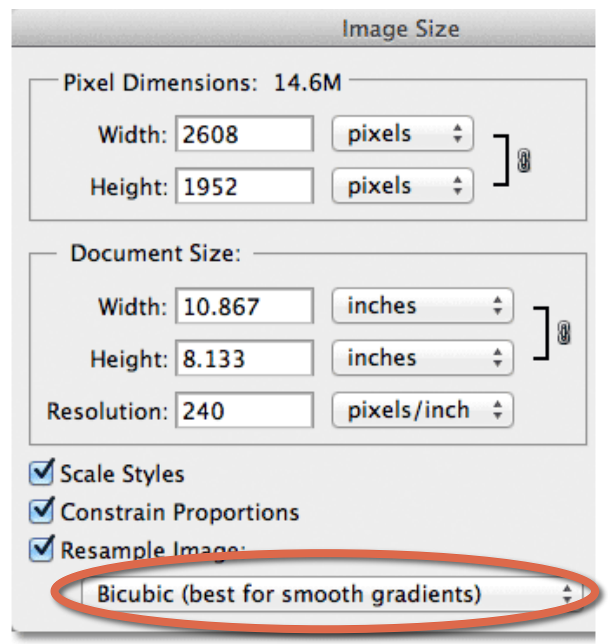
$$\mathcal{F}[f] \cdot \mathcal{F}[g] = \mathcal{F}[f * g]$$

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

Band-Limiting

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

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

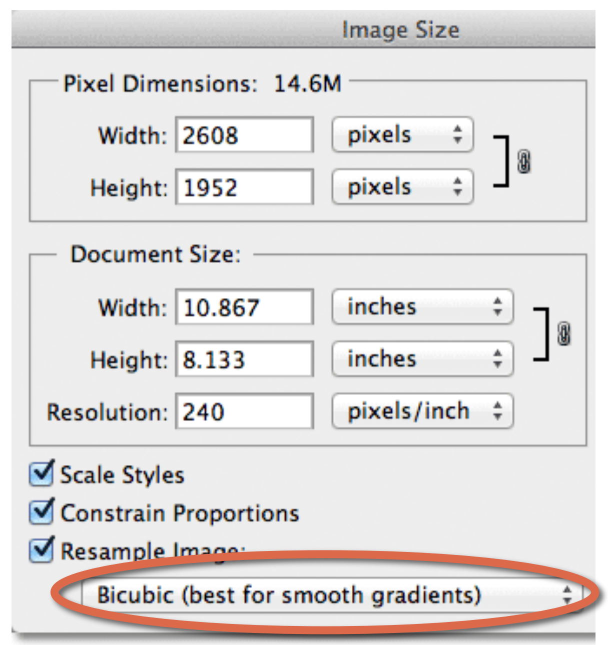


Band-Limiting

- ***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.

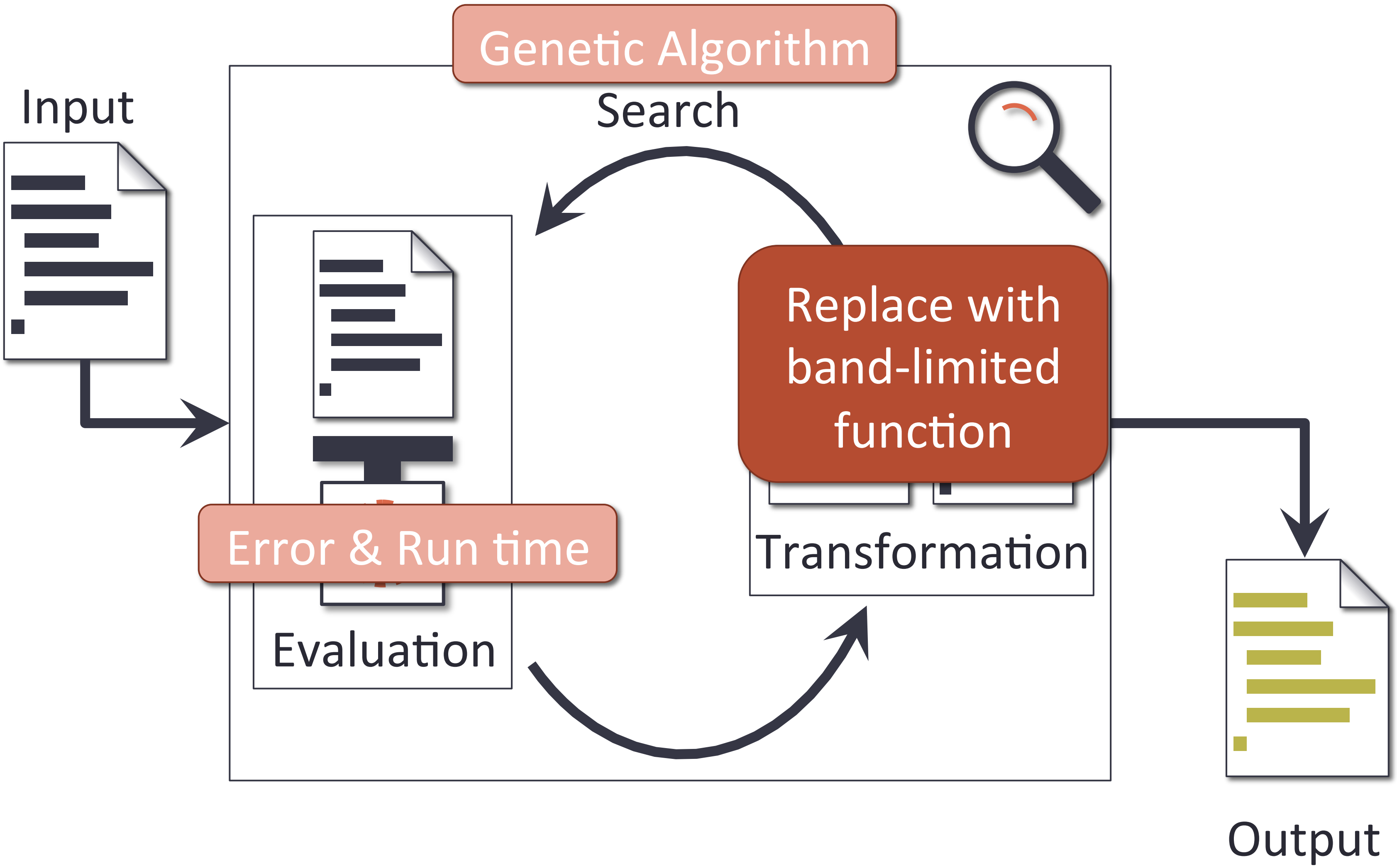


Our Band-Limiting Transformation

$f(x)$	$\hat{f}(x,w)$
x	x
x^2	$x^2 + w^2$
$fract_1(x)$	$\frac{1}{2} - \sum_{n=1}^{\infty} \frac{\sin(2\pi n x)}{\pi n} e^{-2w^2\pi^2 n^2}$
$fract_2(x)$	$\frac{1}{2w}\left(fract^2\left(x+\frac{w}{2}\right) + \left\lfloor x+\frac{w}{2} \right\rfloor - fract^2\left(x-\frac{w}{2}\right) - \left\lfloor x-\frac{w}{2} \right\rfloor\right)$
$fract_3(x)$	$\frac{1}{12w^2}\left(f'(x-w) + f'(x+w) - 2f'(x)\right)$ where $f'(t) = 3t^2 + 2fract^3(t) - 3fract^2(t) + fract(t) - t$
$\lvert x \rvert$	$x \operatorname{erf} \frac{x}{w\sqrt{2}} + w\sqrt{\frac{2}{\pi}} e^{-\frac{x^2}{2w^2}}$
$\lfloor x \rfloor$	$x - \widehat{fract}(x,w)$
$\lceil x \rceil$	$\widehat{floor}(x,w) + 1$
$\cos x$	$\cos x\, e^{-\frac{w^2}{2}}$
$saturate(x)$	$\frac{1}{2}\left(x \operatorname{erf} \frac{x}{w\sqrt{2}} - (x-1) \operatorname{erf} \frac{x-1}{w\sqrt{2}} + w\sqrt{\frac{2}{\pi}}\left(e^{-\frac{x^2}{2w^2}} - e^{-\frac{(x-1)^2}{2w^2}}\right) + 1\right)$
$\sin x$	$\sin x\, e^{-\frac{w^2}{2}}$
$step(a,x)$	$\frac{1}{2}\left(1 + \operatorname{erf} \frac{x-a}{w\sqrt{2}}\right)$
$trunc(x)$	$\widehat{floor}(x,w) - \widehat{step}(x,w) + 1$

- 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

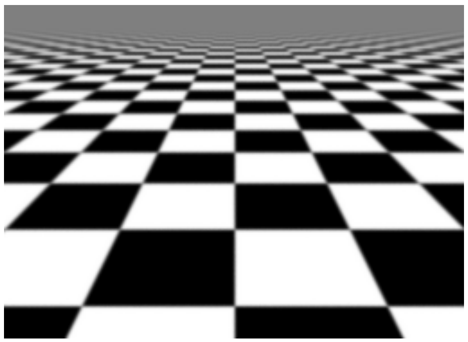


Evaluation

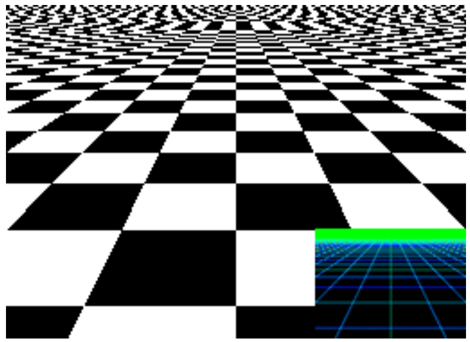
- **Benchmarks**: 11 programs used in previous work on antialiasing.
- Compare against ***16x supersampling***.
- Metrics:
  - ***Error*** relative to 2000x supersampling.
  - ***Run time***.

Results: Checkerboard

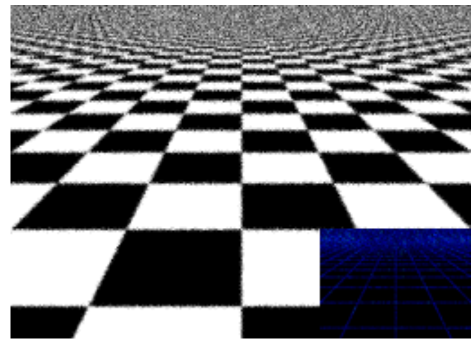
Target Image



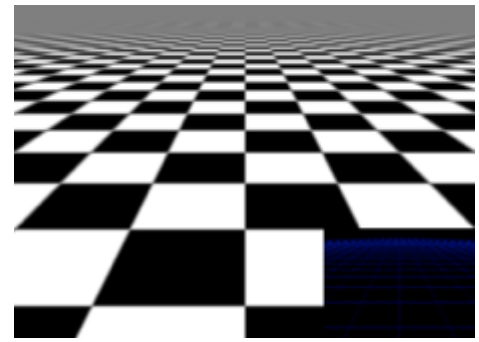
No Antialiasing



16x Supersampling

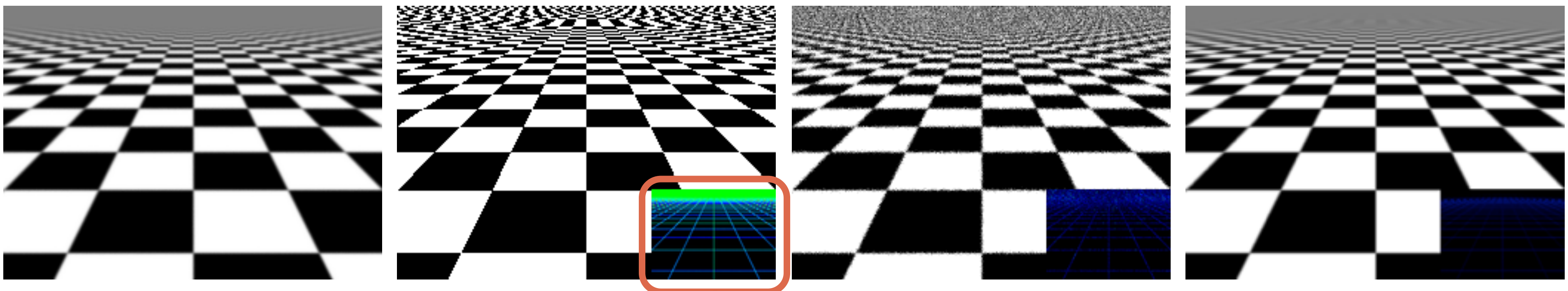


Our Approach



Results: Checkerboard

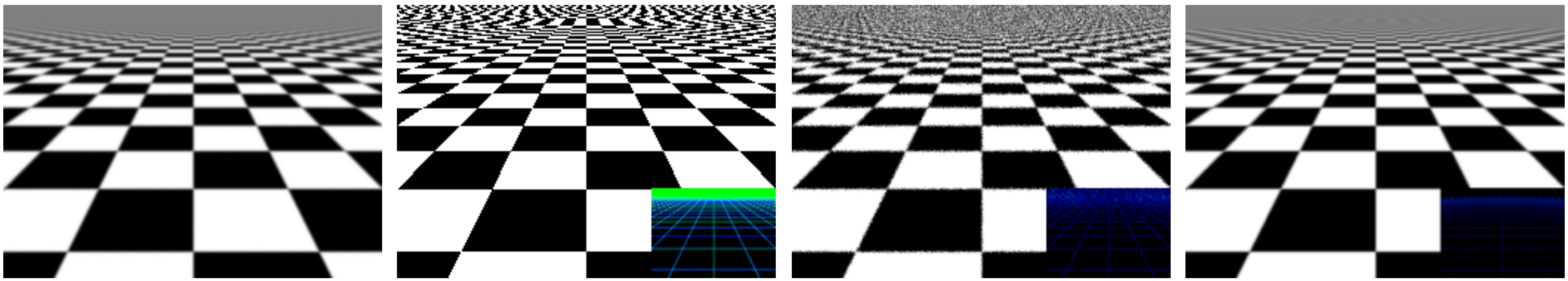
Target Image | No Antialiasing | 16x Supersampling | Our Approach



Error heatmap
L^2 in RGB

Results: Checkerboard

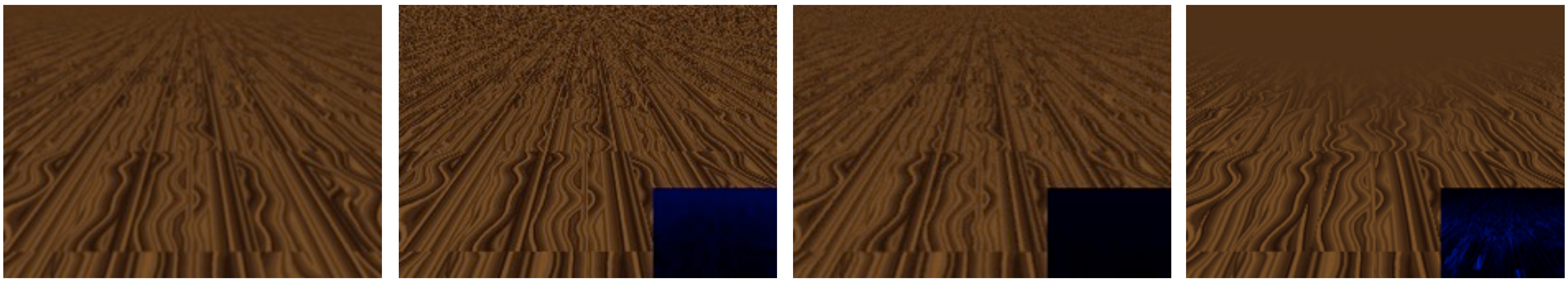
Target Image | No Antialiasing | 16x Supersampling | Our Approach



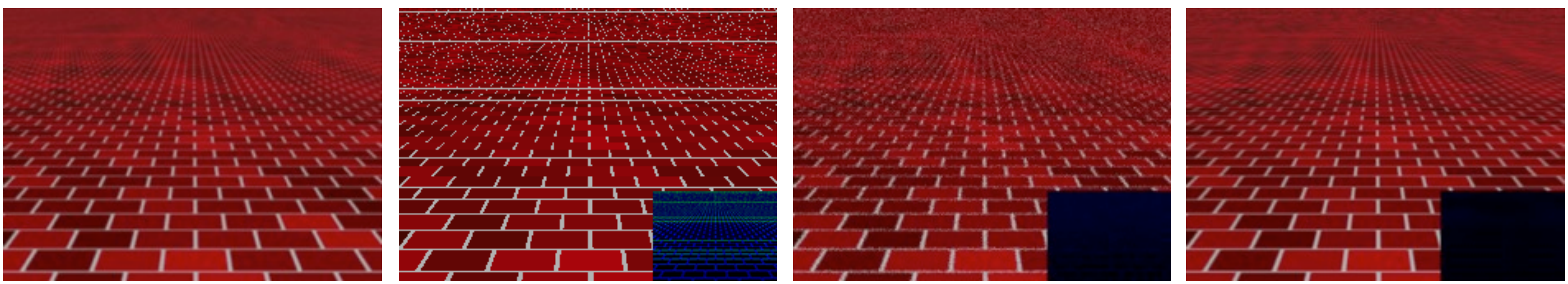
- **4x faster** than super-sampling.
- **2x less L^2 (RGB) error** than supersampling.

Results: Brick and Wood

Target Image	No Antialiasing	16x Supersampling	Our Approach

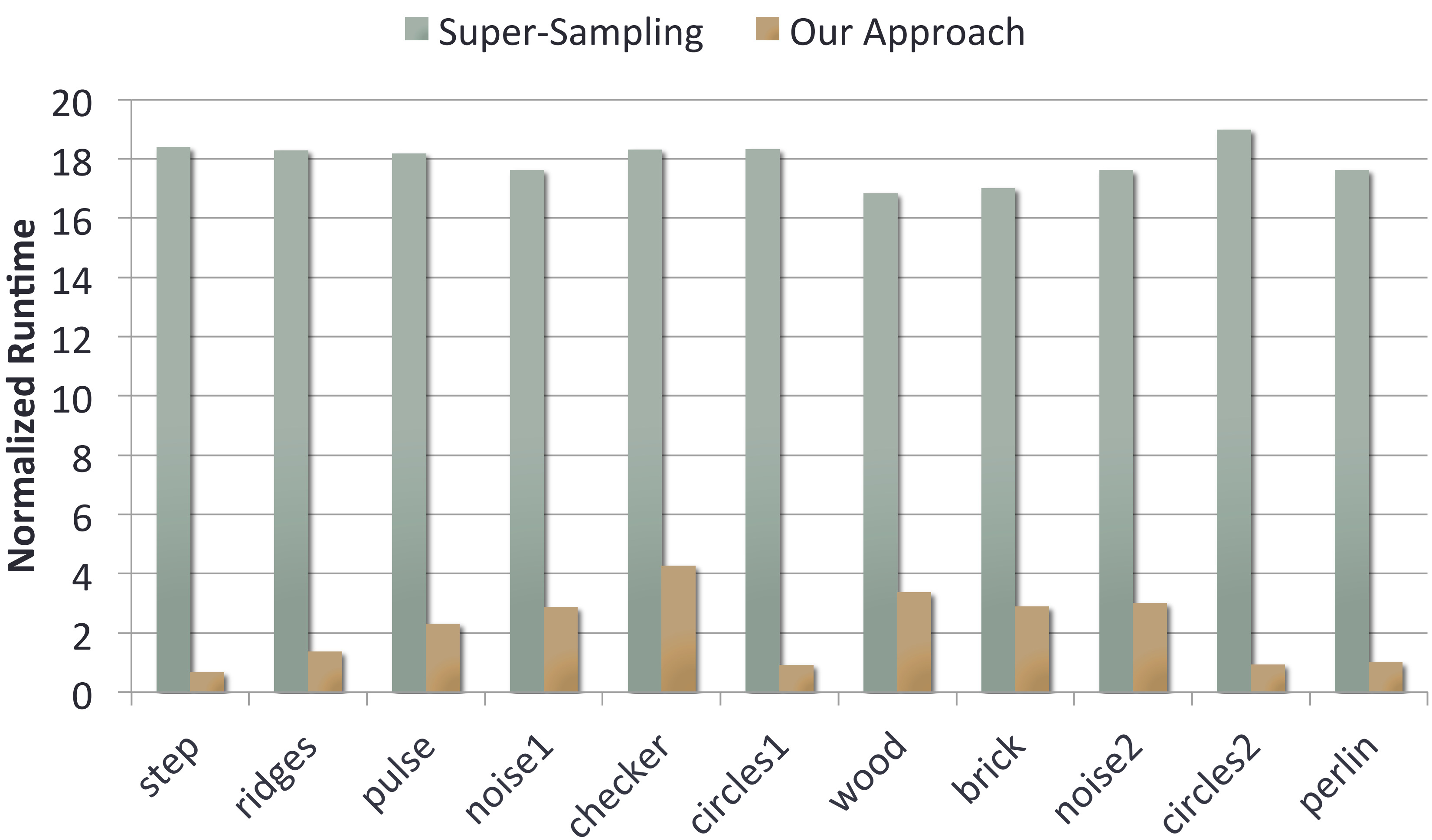


5x faster, 3x more L^2 error than supersampling.

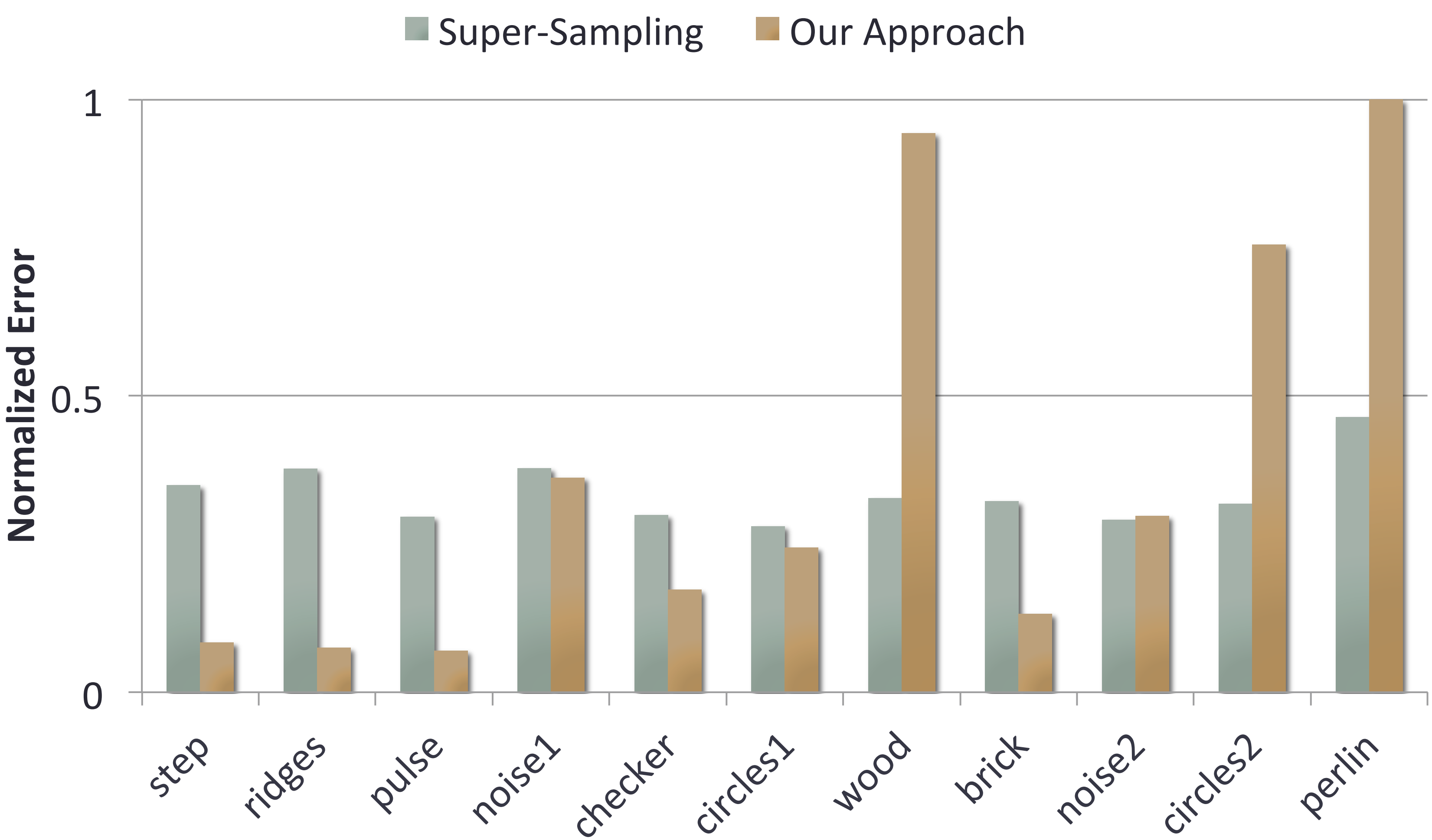


6x faster, 2x less L^2 error than supersampling.

Runtime Results



Error Results



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.
- Pacific Graphics 2015.

Outline

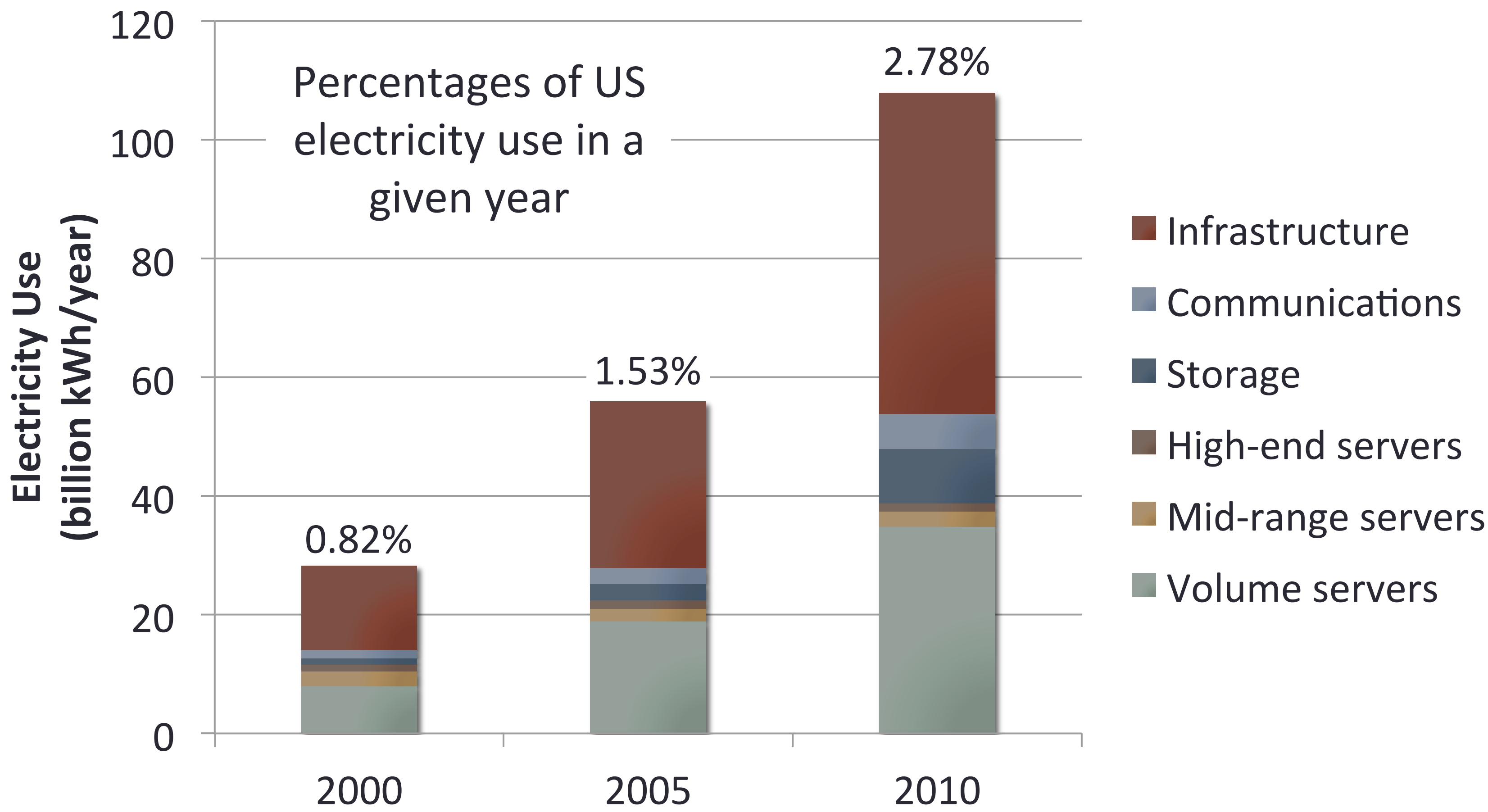
Overview

Application Domains

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

Concluding Thoughts

Data Center Energy Use



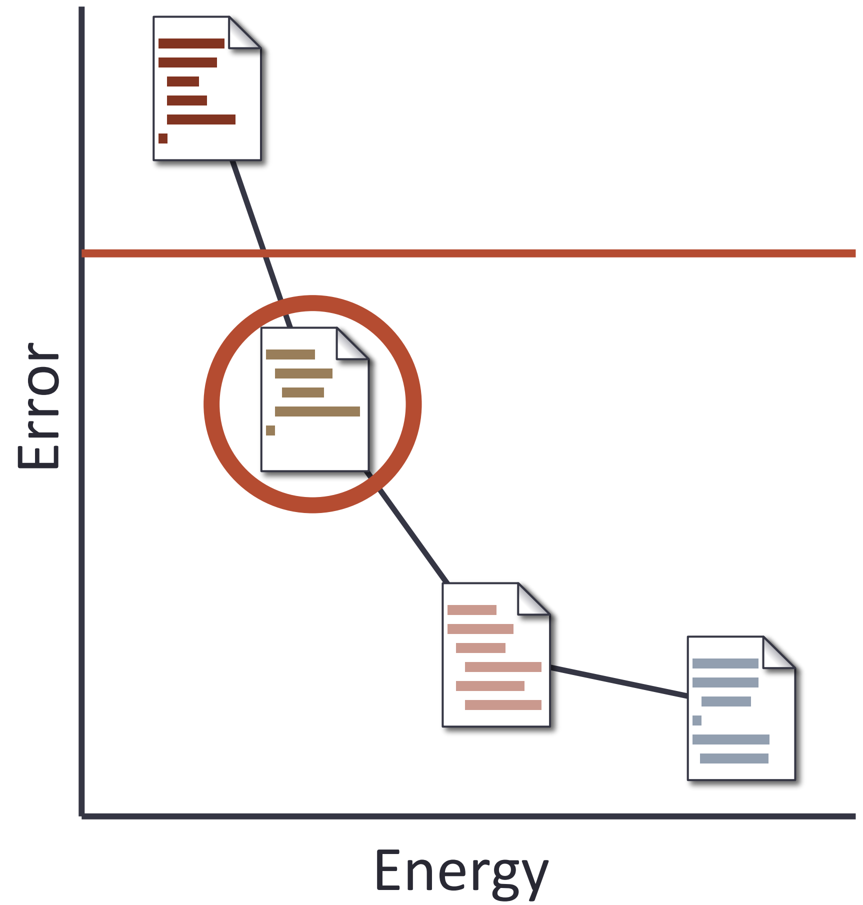
Reproduced from [Koomey 2011]

Approximate Computing Applications

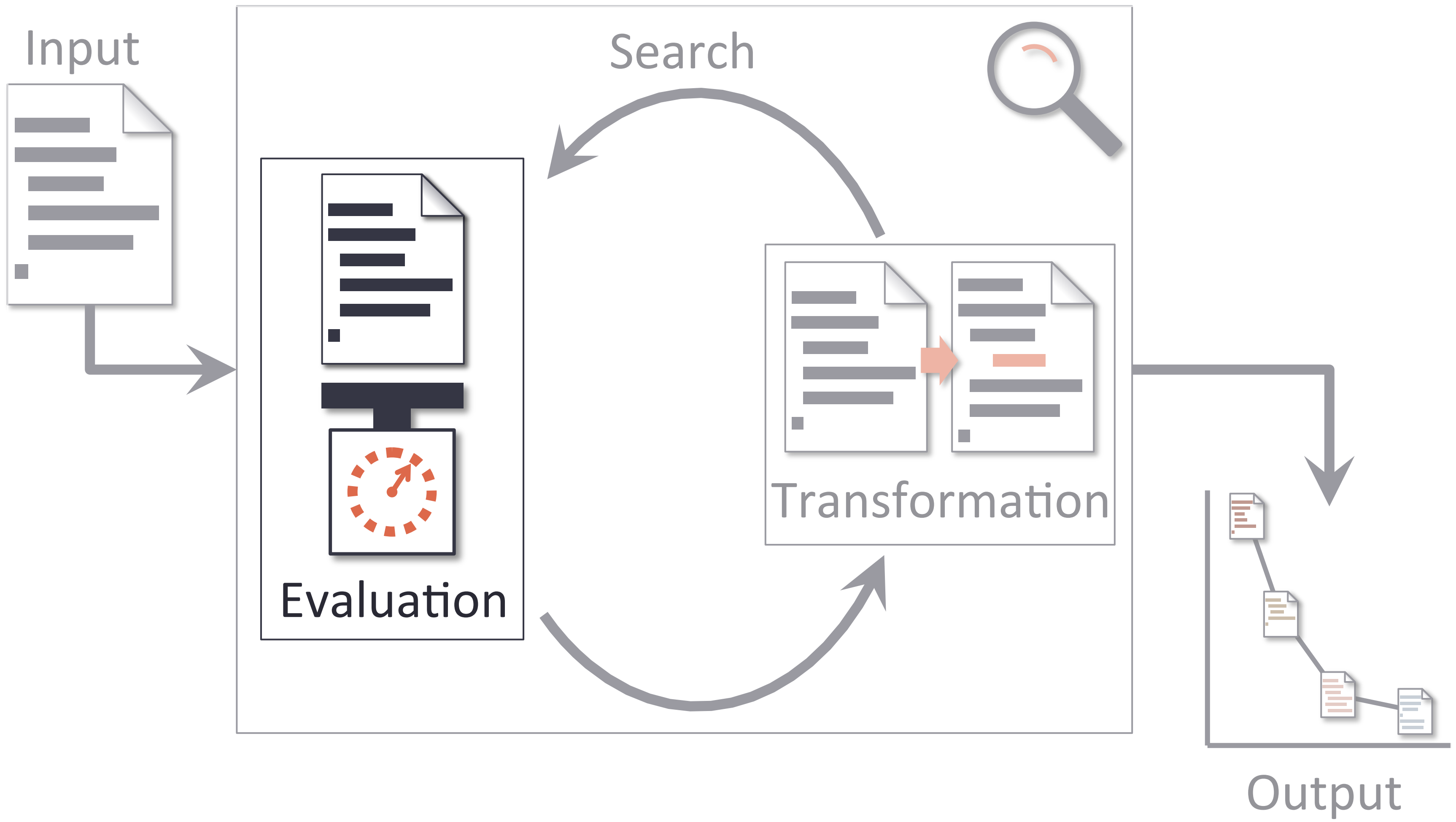
- “Correct” answer is unknown or not well defined.
  - Recommendation systems.
  - Search systems.
  - Prediction systems.

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



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

Slow

- ~~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 accu
- Disaggregation
  - Workload setup and cleanup
  - Daemon processes
- System configuration
  - Core allocation
  - Device sleep

Slow

Inaccurate

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 accu
- Disaggregation
  - Workload setup and cleanup
  - Daemon processes
- System config
  - Core allocation
  - Device sleep

Slow

Inaccurate

Coarse-grained

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 accu
- Disaggregation
  - Workload setup and cleanup
  - Daemon processes
- System config
  - Core allocation
  - Device sleep

Slow

Inaccurate

Coarse-grained

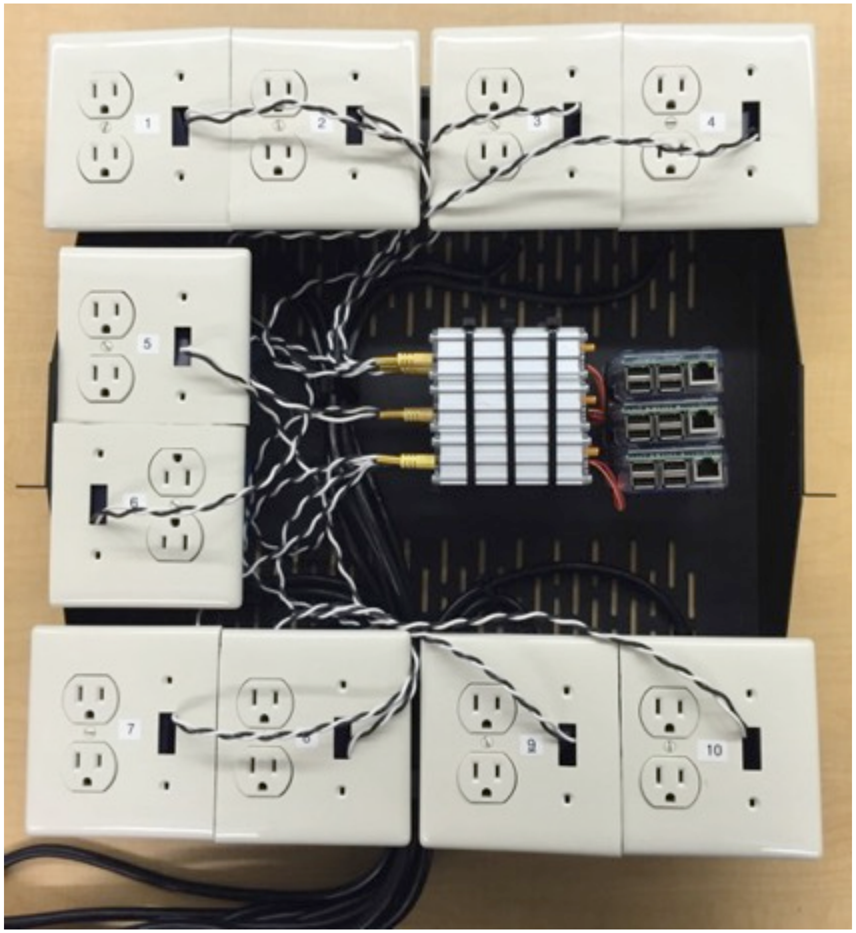
Cost prohibitive

Mechanisms

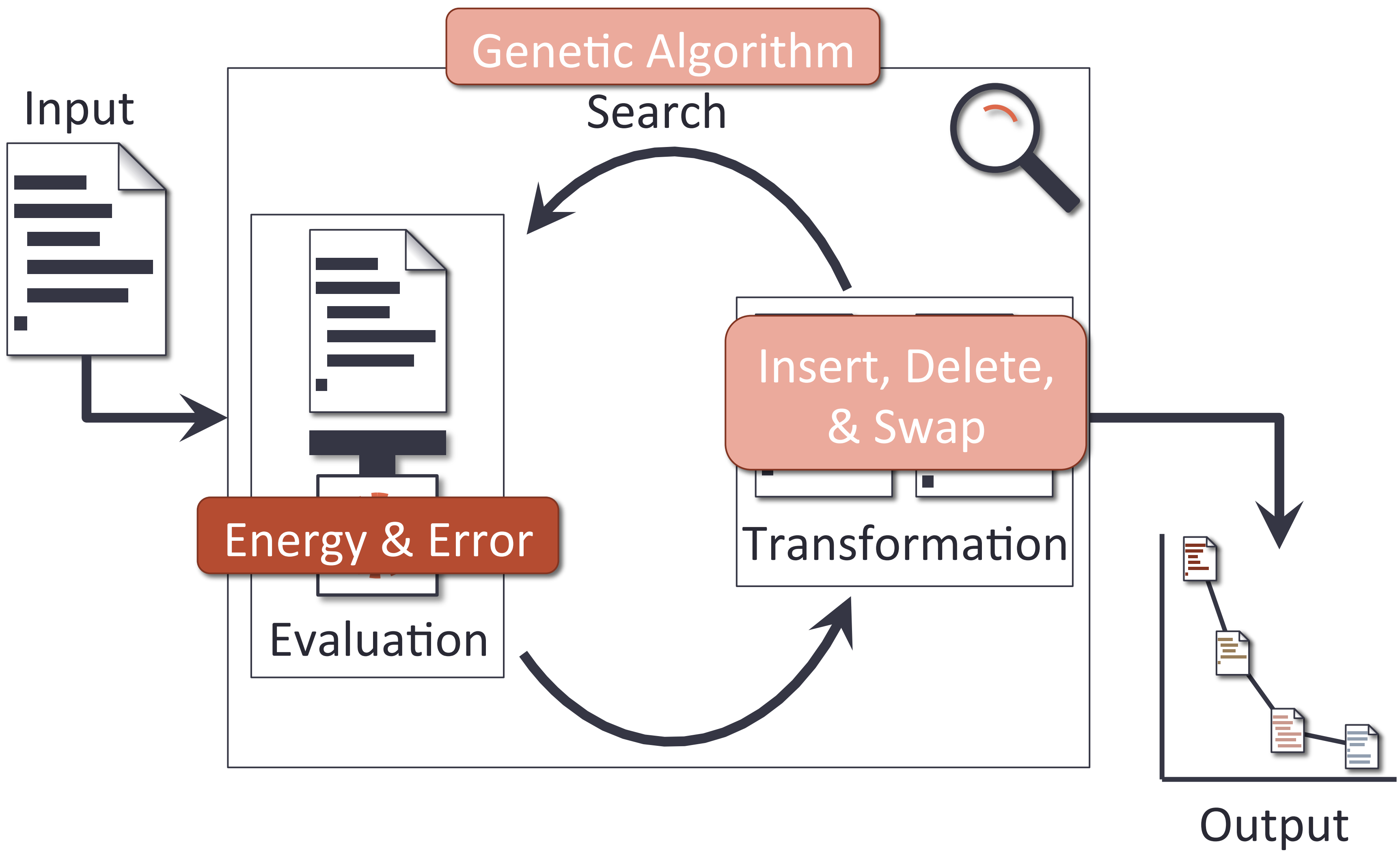
- ~~Simulation~~
  - ~~gem5~~
- ~~Power model~~
  - ~~Intel Power Gadget~~
  - ~~Mac Activity Monitor~~
- Physical
  - ~~Commodity energy meter~~
  - ~~Phasor Measurement Unit~~
  - Custom-built

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



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)

Benchmark	Application Domain	Error Metric
blackscholes	Financial analysis	RMSE
bodytrack	Computer vision	RMSE
ferret	Similarity search	Kendall's τ
fluidanimate	Animation	Hamming Distance
freqmine	Data mining	RMSE
swaptions	Financial analysis	RMSE
vips	Media processing	Image Similarity
x264	Media processing	Image Similarity

Data Center Benchmarks

Benchmark	Application Domain	Error Metric
blackscholes	Financial analysis	RMSE
bodytrack	Computer vision	RMSE
ferret	Similarity search	Kendall's τ
fluidanimate	Animation	Hamming Distance
freqmine	Data mining	RMSE
swaptions	Financial analysis	RMSE
vips	Media processing	Image Similarity
x264	Media processing	Image Similarity
blender	3D renderer	Image Similarity
libav	Media processing	Image Similarity

Data Center Benchmarks

Benchmark	Application Domain	Error Metric
blackscholes	Financial analysis	RMSE
bodytrack	Computer vision	RMSE
ferret	Similarity search	Kendall's τ
fluidanimate	Animation	Hamming Distance
freqmine	Data mining	RMSE
swaptions	Financial analysis	
vips	Media processing	
x264	Media processing	Image Similarity
blender	3D renderer	Image Similarity
libav	Media processing	Image Similarity

Order of magnitude larger.
Evaluate scalability.

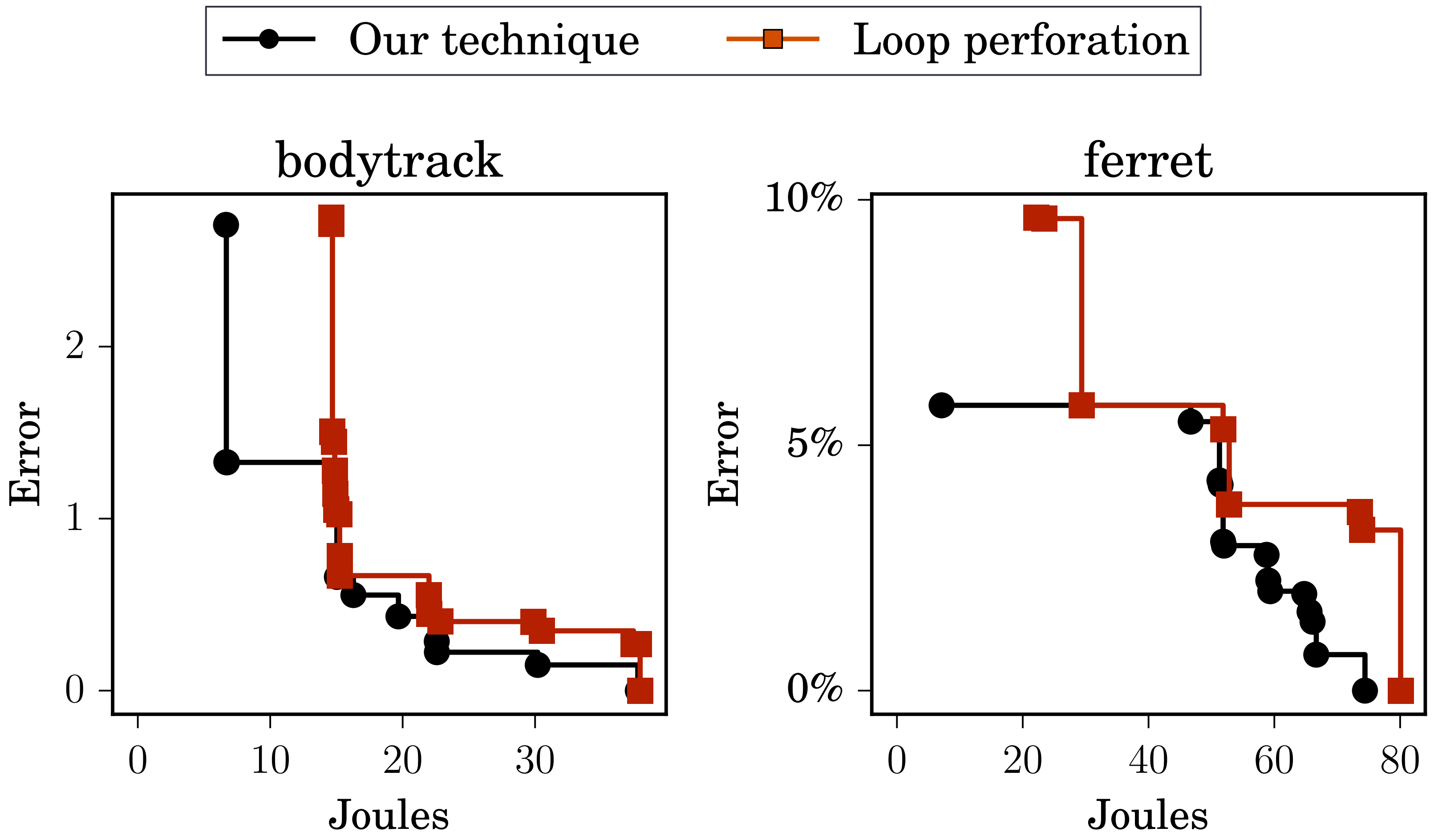
Acceptable Error

- 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).

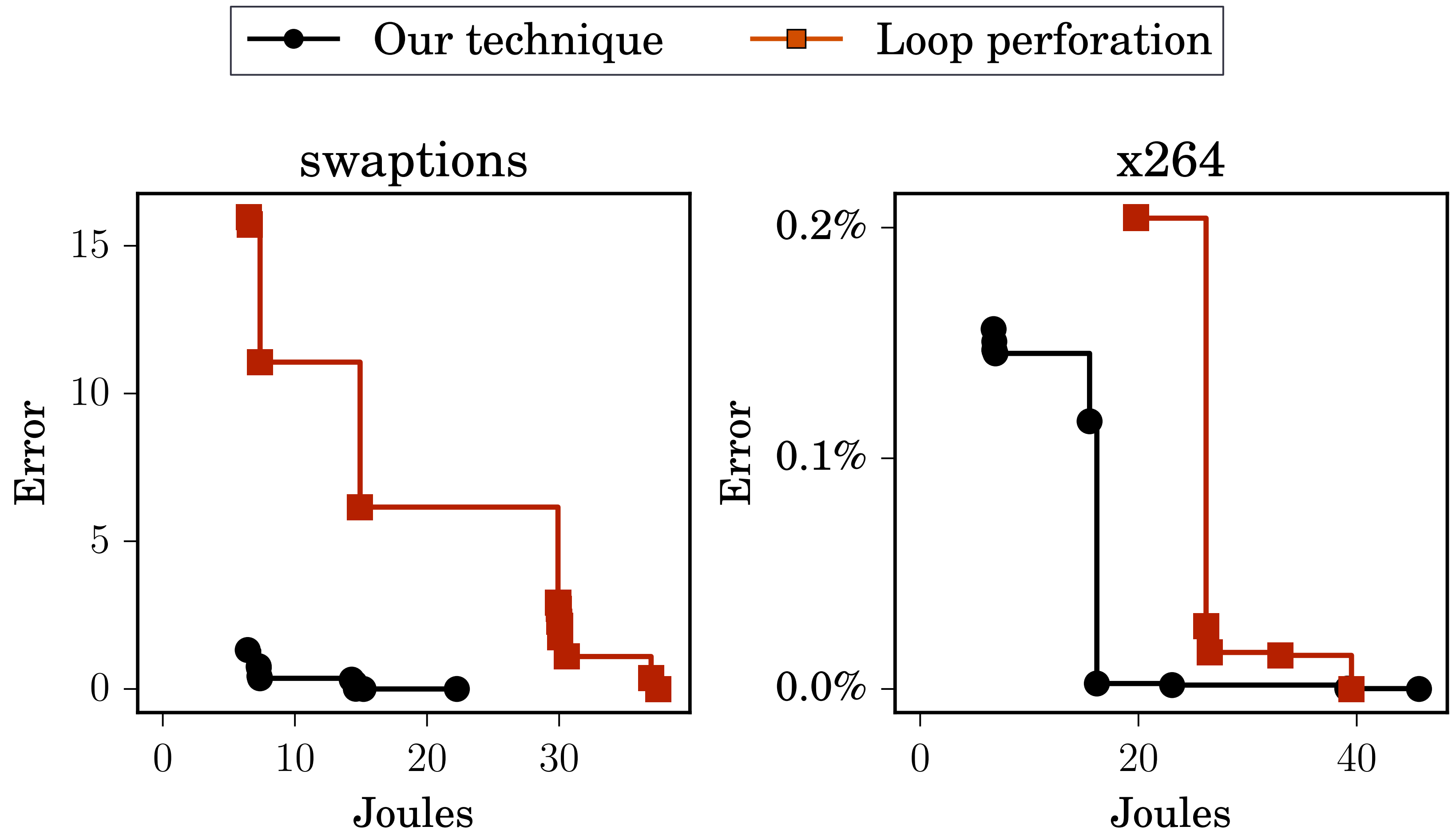
Energy Reduction Results (%)

Benchmark	No Error	Acceptable Error
blackscholes	**92**	92
bodytrack	0	**59**
ferret	0	**30**
fluidanimate	0	0
freqmine	**8**	8
swaptions	39	**68**
vips	21	**29**
x264	0	**65**
blender	1	**10**
libav	3	**92**

PARSEC Results

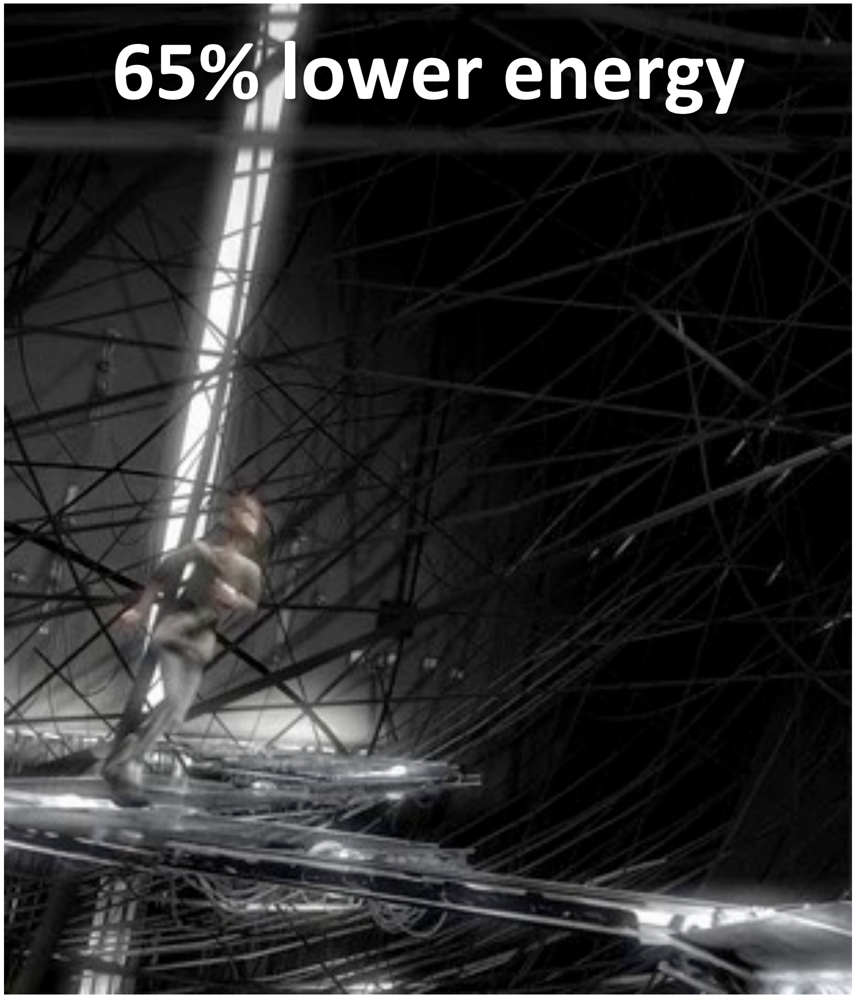


PARSEC Results



Can You Spot the Difference?

Can You Spot the Difference?



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

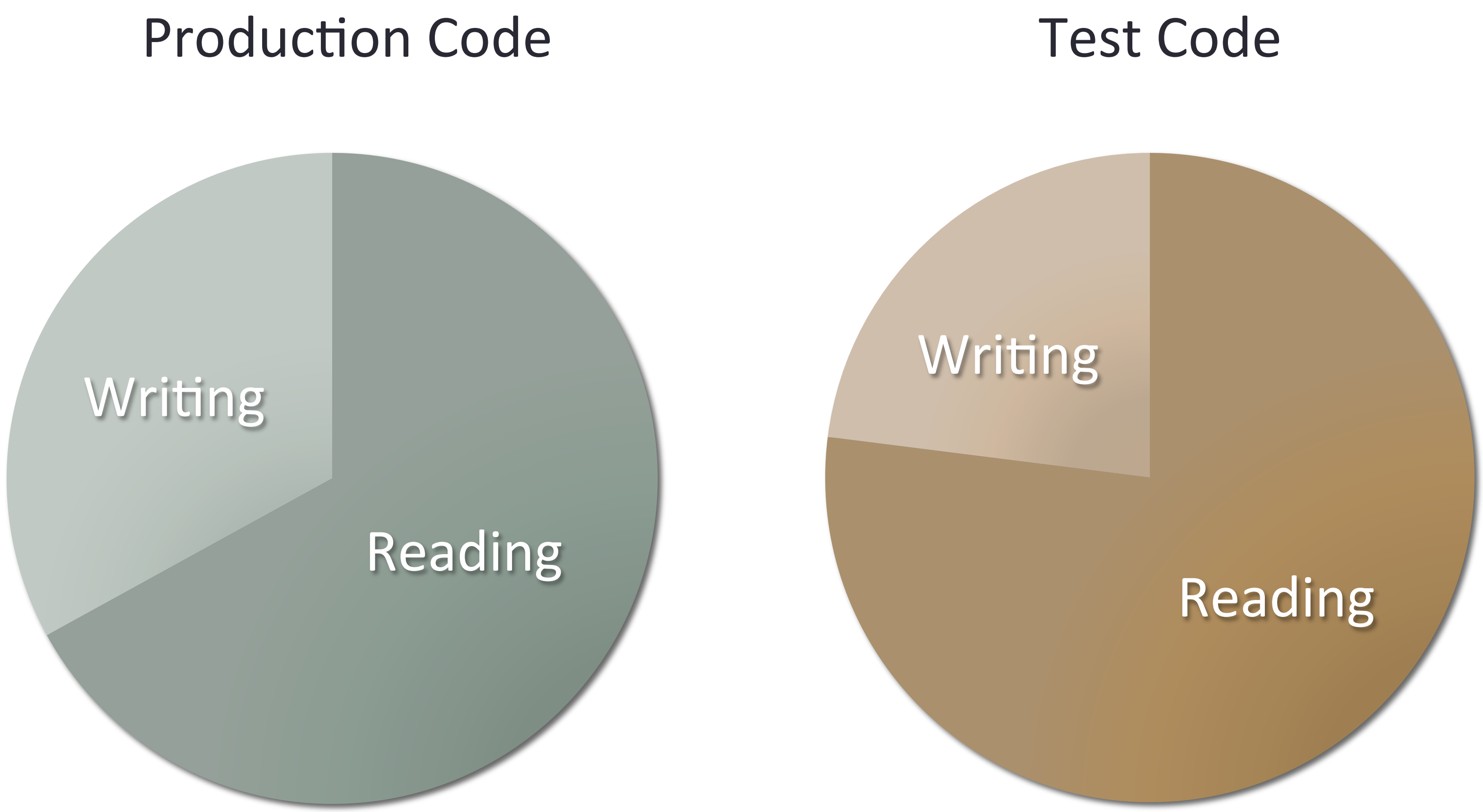
- Mars Spirit Rover ($1B).
  - Almost lost mission due to filesystem bug.*
- Knight Capital trading glitch ($440M).
  - Development software released into production.
- Inadequate testing costs the US over $60B.***

* Glenn Reeves and Tracy Neilson. "The mars rover spirit FLASH anomaly." IEEE Aerospace Conference, 2005.
** https://dealbook.nytimes.com/2012/08/02/knight-capital-says-trading-mishap-cost-it-440-million/
*** RTI Health, Social, and Economics Research. "The Economic Impacts of Inadequate Infrastructure for Software Testing." NIST, 2002.

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

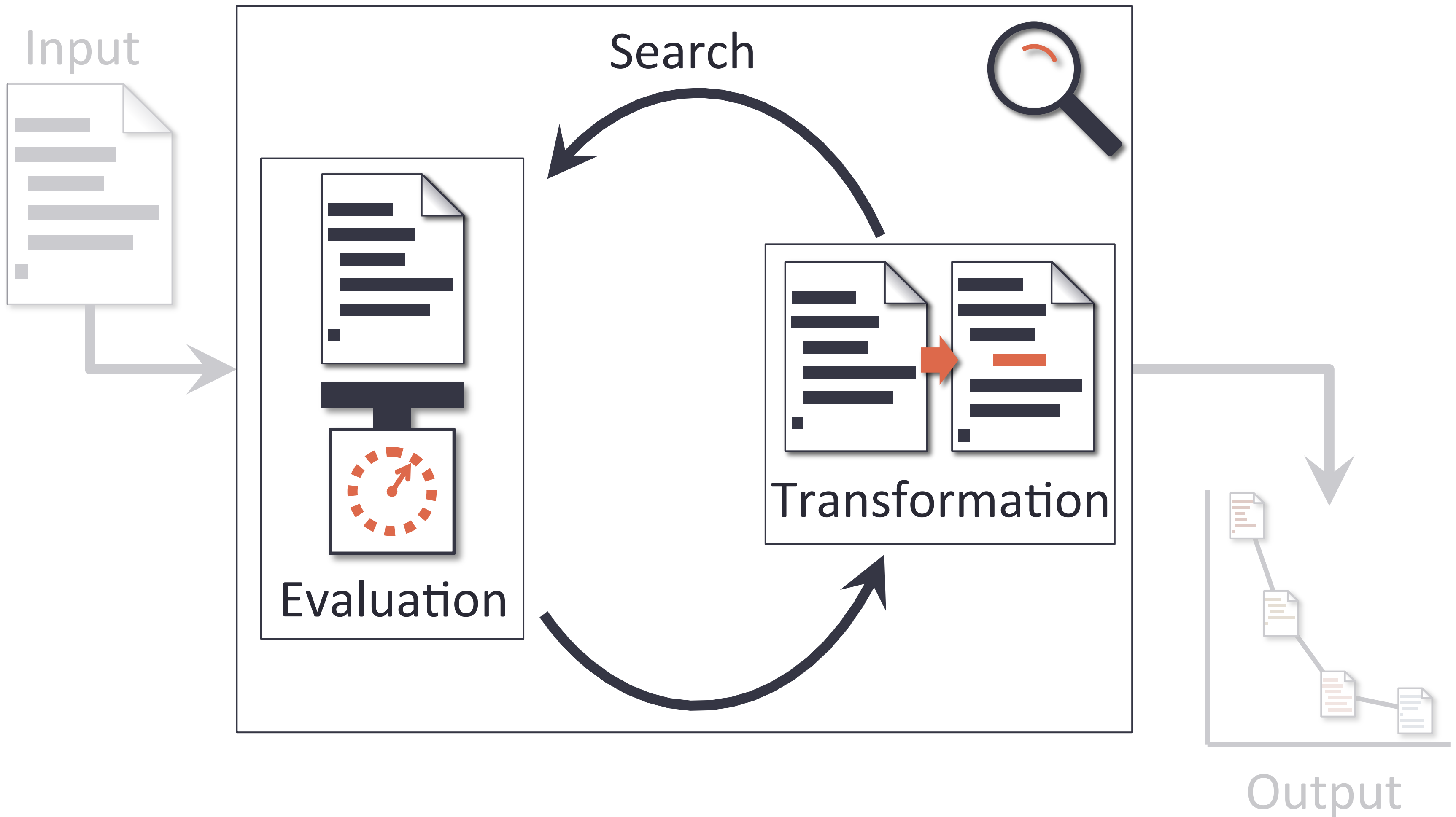


Adapted from [Beller, et al. 2015]

Project Overview

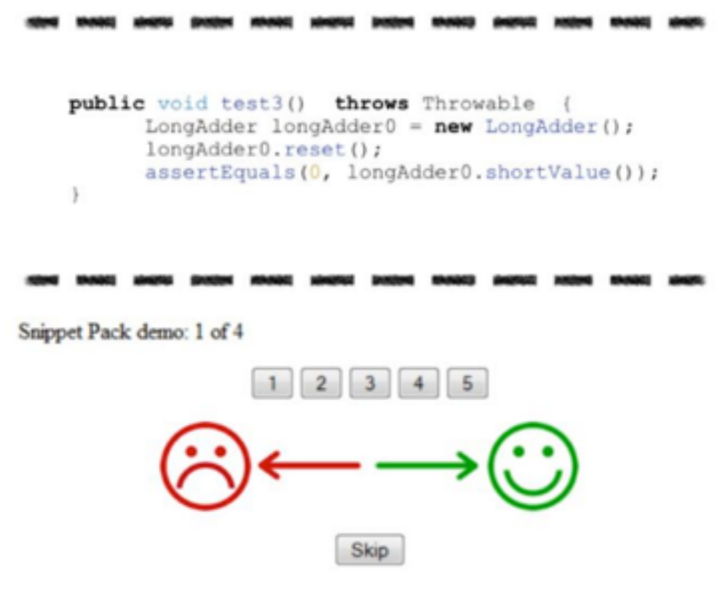
- Goal:
  - Generate ***readable***, ***high-coverage*** test suites.
- Approach:
  1. Model test readability.
  2. Optimize coverage and readability.
  3. Validate with human study.

Search-Based Optimization Framework



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 EvoSuite test suite generator for Java.
  - Optimizes coverage objectives via evolutionary search.

```
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 EvoSuite 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.

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

```
Foo foo = new Foo();
Bar bar = new Bar();
foo.setBar(bar);
assertTrue(foo.isBar());
```

Readability Transformation

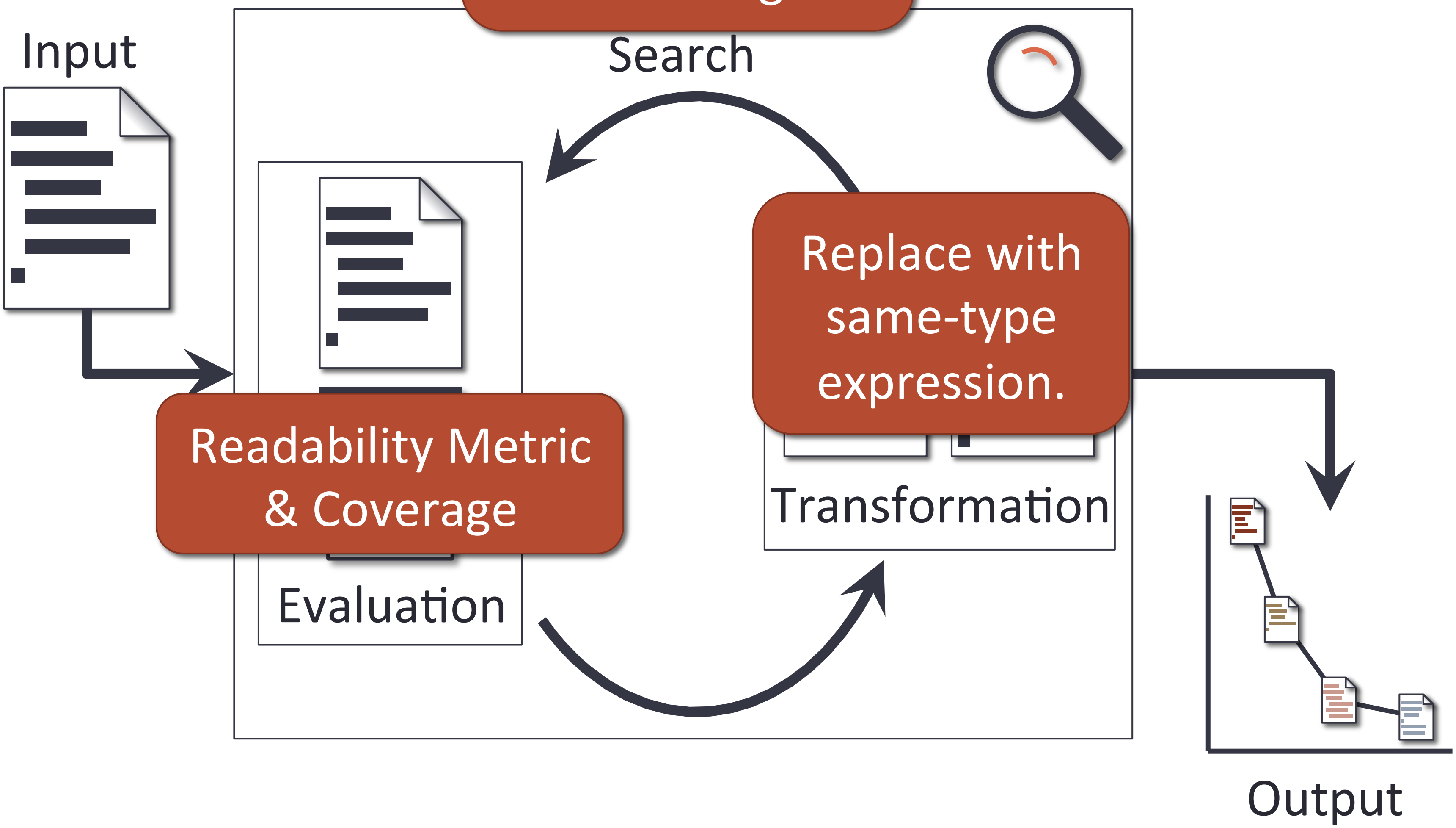
- Transformation:
  - Replace RHS of assignment with same-type expression.
  - Remove dead code.

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

```
Foo foo = new Foo();
Bar bar = new Bar(“Some parameter”, 17);
foo.setBar(null);
assertTrue(foo.isBar());
```

Search-Based Framework

2-stages: Genetic Algorithm & Hill Climbing



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

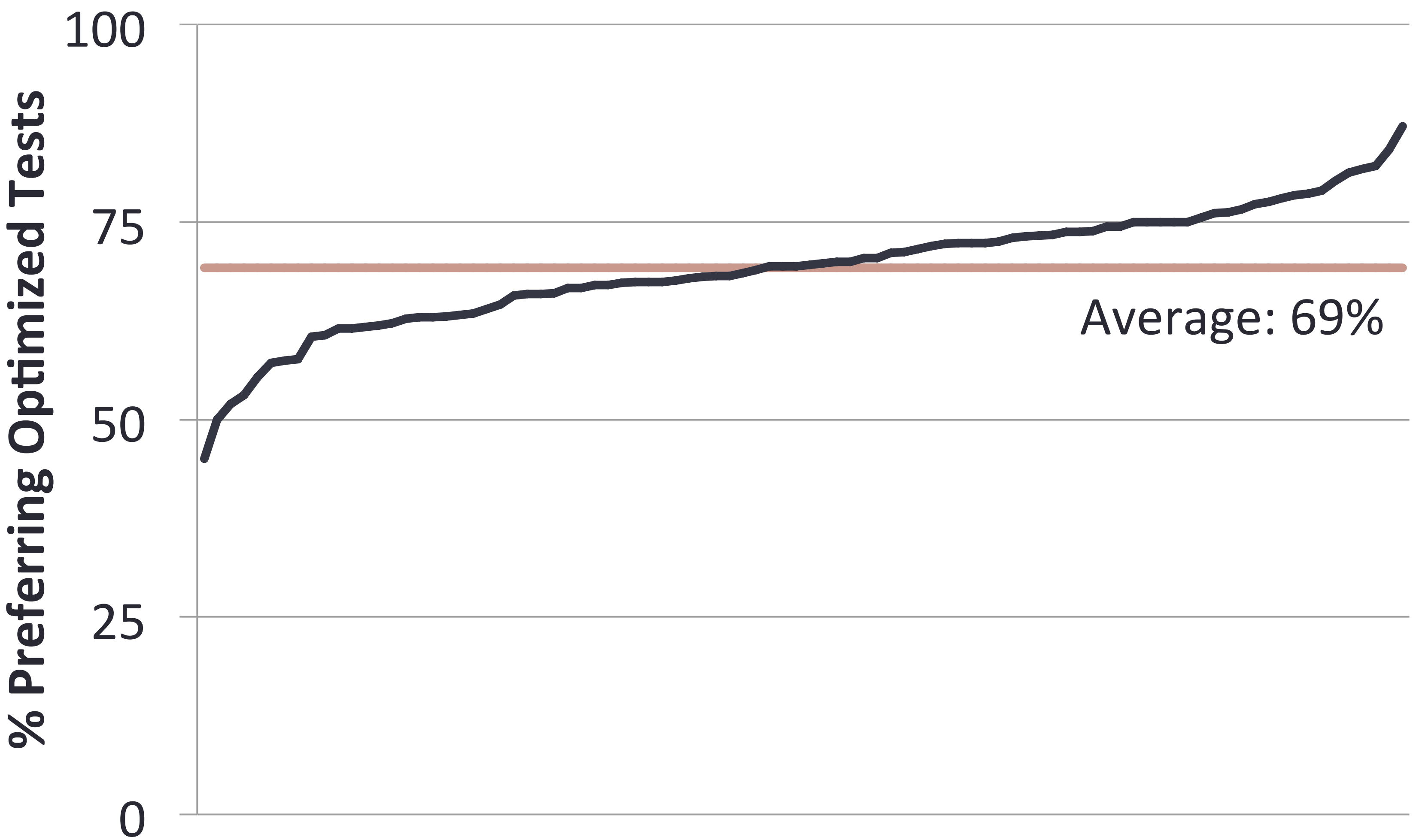
```
package org.apache.commons.cli;

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

public class CommandLine_ESTest {

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

Test Case B

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

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

public 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);
  }
}
```

Test A | Test B

Human Preference Results

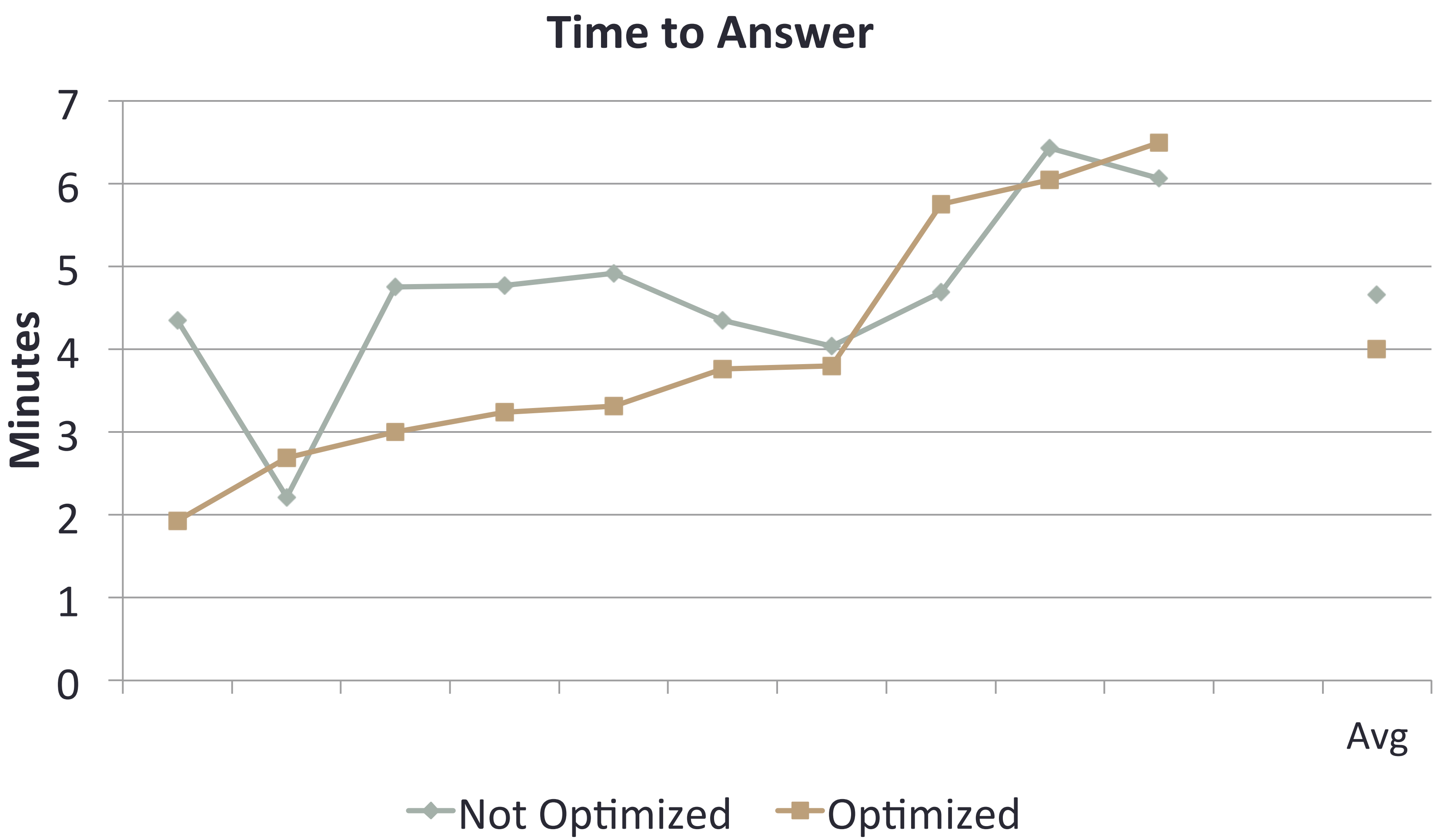


Test Understanding

```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
         //
      }
  }
}
```

Pass | Fail

Test Understanding Results



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.

Jonathan Dorn, Jeremy Lacomis, Westley Weimer, Stephanie Forrest. *Automatically Exploring Tradeoffs Between Software Output Fidelity and Energy Costs*. Transactions on Software Engineering. (*Reviewed and revised*)

Jonathan Dorn, Connelly Barnes, Jason Lawrence, Westley Weimer. *Towards Automatic Band-Limited Procedural Shaders*. Pacific Graphics. 2015.

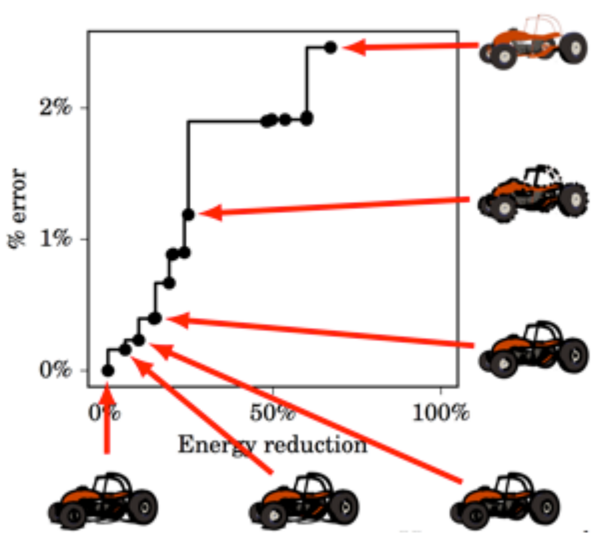
Ermira Daka, Jose Campos, Gordon Fraser, Jonathan Dorn, Westley Weimer. *Modeling Readability to Improve Unit Tests*. Foundations of Software Engineering. 2015. **ACM SIGSOFT Distinguished Paper Award**.

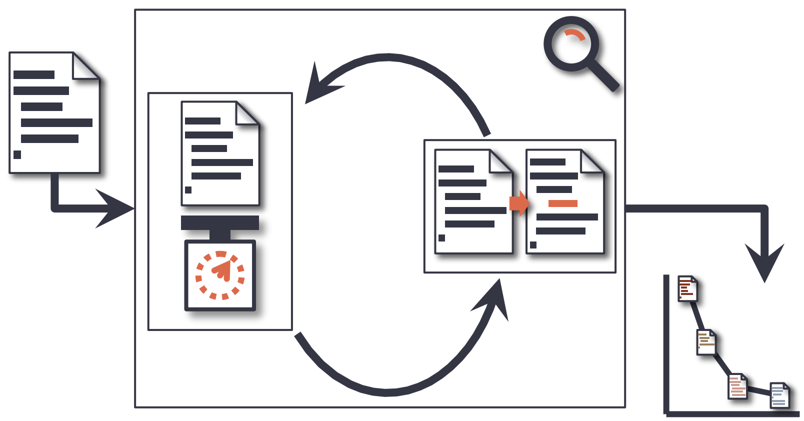
Ermira Daka, Jose Campos, Jonathan Dorn, Gordon Fraser, Westley Weimer. *Generating Readable Unit Tests for Guava. Symposium on Search Based Software Engineering*. 2015.

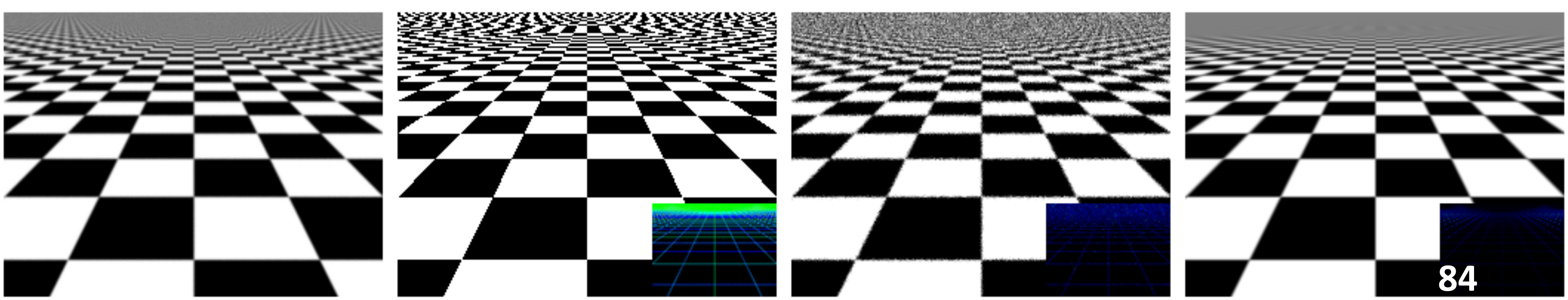
Eric Schulte, Jonathan Dorn, Stephen Harding, Stephanie Forrest, Westley Weimer. *Post-compiler Software Optimization for Reducing Energy*. Architectural Support for Programming Languages and Operating Systems. 2014.

Chris Gregg, Jonathan Dorn, Kim Hazelwood, Kevin Skadron. *Fine-Grained Resource Sharing for Concurrent GPGPU Kernels. 4th USENIX Workshop on Hot Topics in Parallelism*. 2012.

Optimizing Tradeoffs of Non-Functional Properties in Software



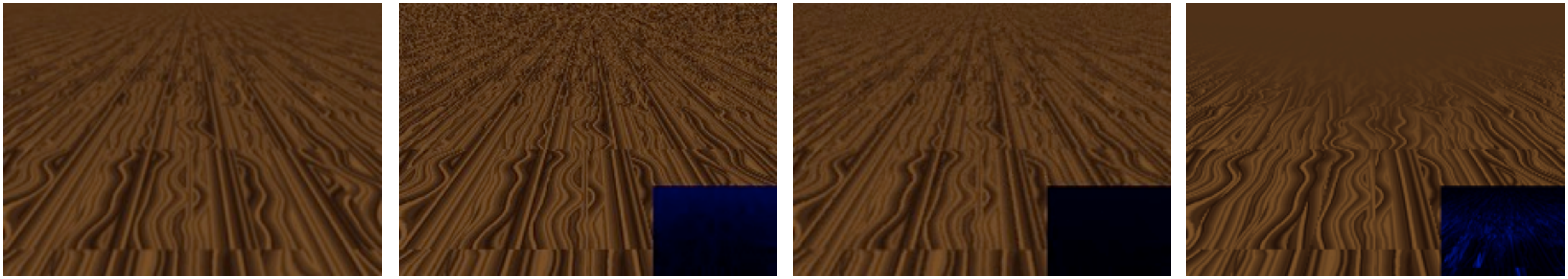




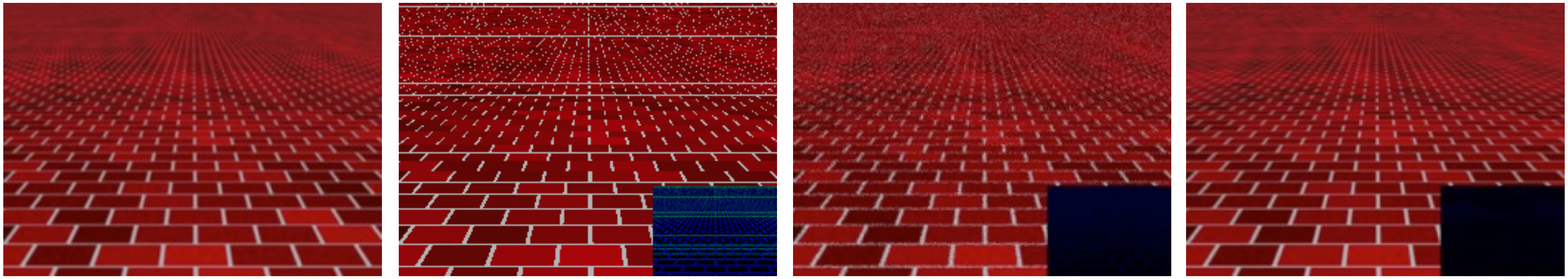
BACKUP

Results: Brick and Wood

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



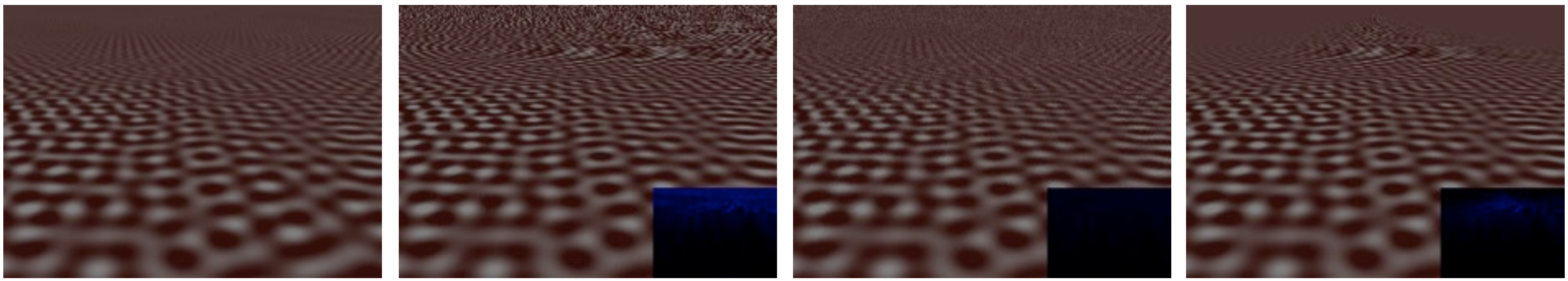
5x faster, 3x more L^2 error than supersampling.



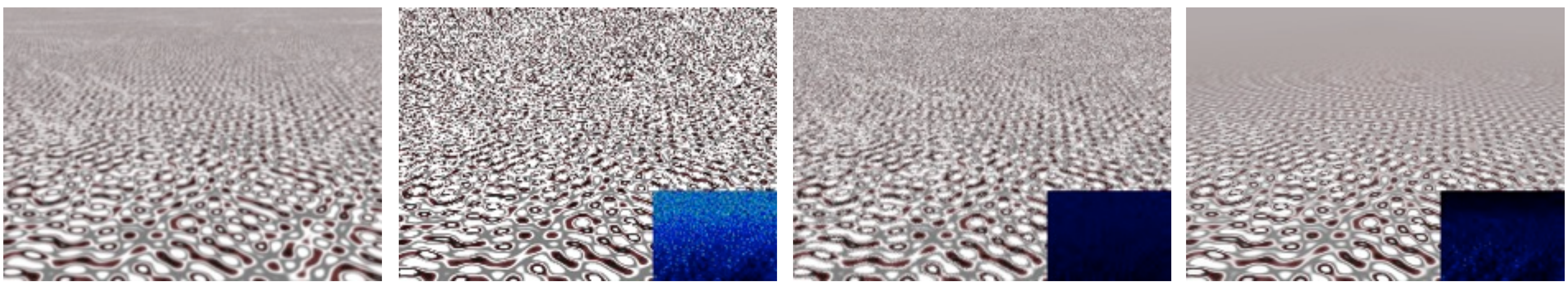
6x faster, 2x less L^2 error than supersampling.

Results: Noise1 and Noise2

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



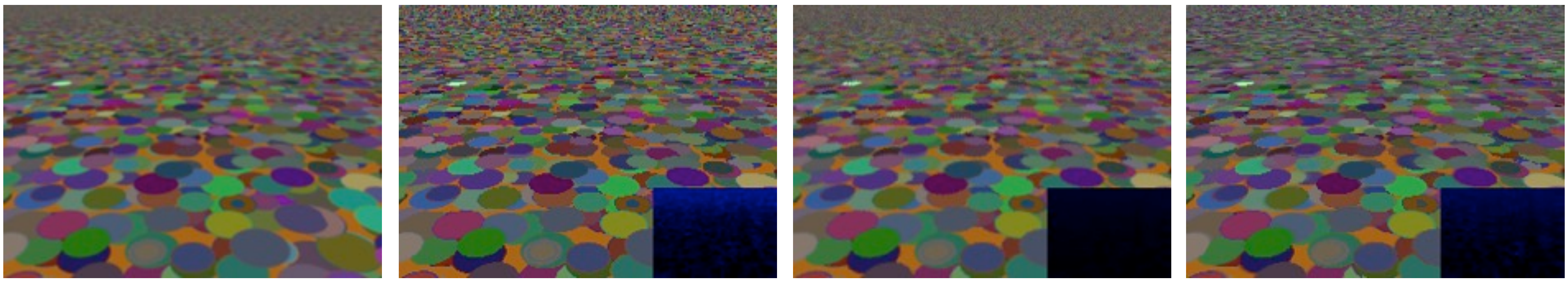
7x faster, same L^2 error as supersampling.



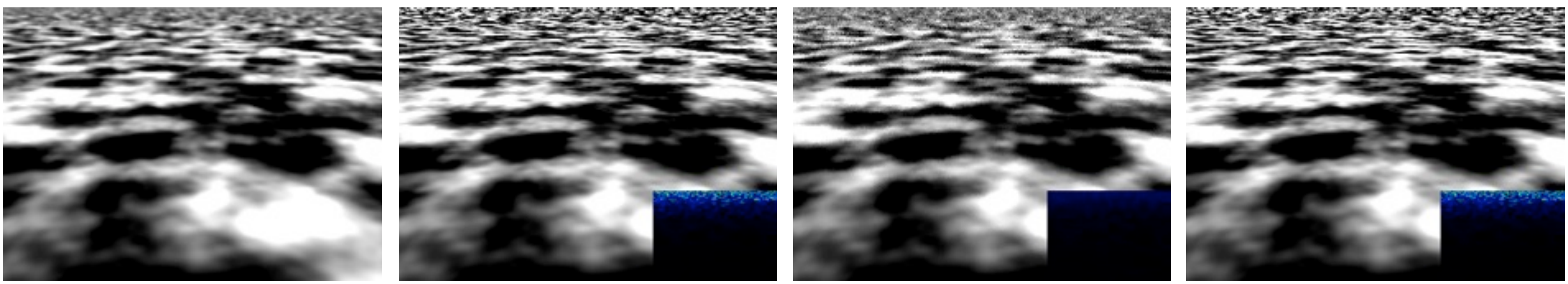
6x faster, sane L^2 error as supersampling.

Results: Circles2 and Perlin

Target Image	No Antialiasing	16x Supersampling	Our Approach



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



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
  …
  cmpl  %r13d, 40(%rsp)   ← Loop condition check

  movq  16(%rsp), %r9
  movsd %xmm0, (%r9)
  je    .L9           ← Jumps out of loop
  …
  call  _Z12CumNormalInvd
```

Assembly Optimization Example

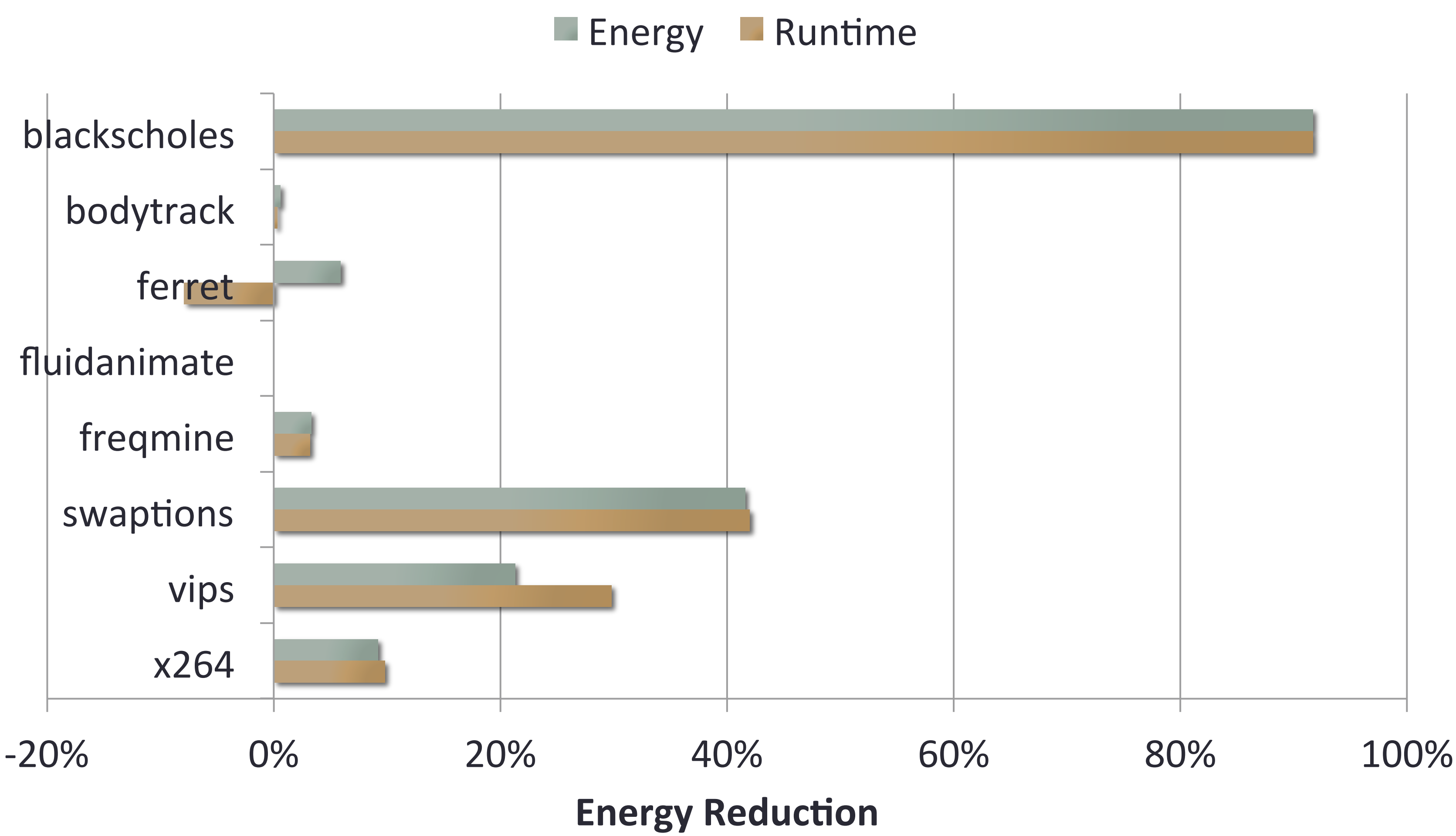
```
.L23:
  …
  cmpl  %r13d, 40(%rsp)
  xorl  %eax, %eax          ← Resets condition flags
  movq  16(%rsp), %r9
  movsd %xmm0, (%r9)
  je    .L9                 ← Always exits loop!
  …
  call  _Z12CumNormalInvd
```

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



Feature Predictive Power