Precision Selection for Energy-Efficient Pixel Shaders

Jeff Pool, Anselmo Lastra, and Montek Singh
Department of Computer Science
The University of North Carolina at Chapel Hill

Motivation

Energy is a primary concern in today's GPUs

- Mobile
- Desktop
- HPC

Power == Perf!

Don’t do Unnecessary Work

- Simplistic view:
  - 8-10 bit color buffers
  - FP32 => 24 bits of precision
  - **>60% wasted effort!**

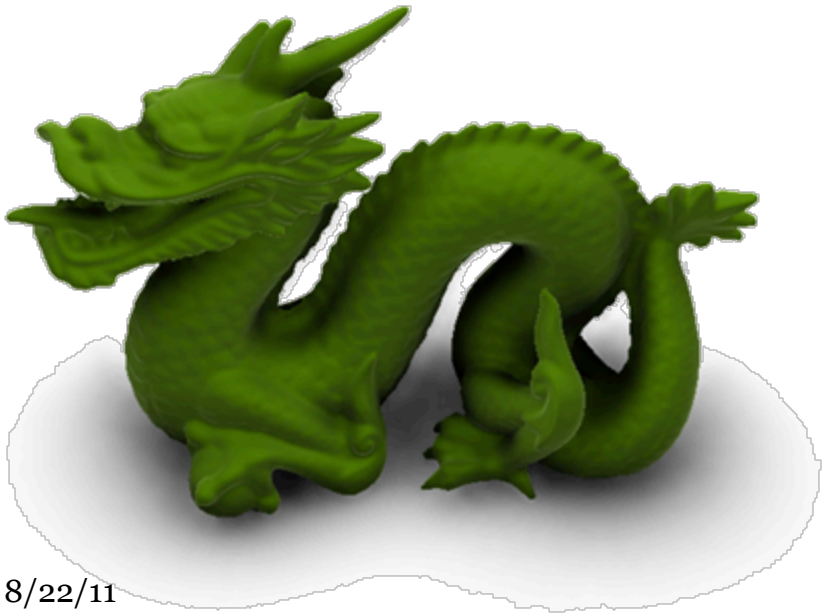
- It’s certainly more complicated, but worth exploring

Our Approach

Variable-precision pixel shaders

-Save energy in pixel shader arithmetic

Full Precision
(24 bits in mantissa, 100% energy)



Reduced Precision
(12.5 bits in mantissa, 29% energy)

We Can Target Energy in…

Computations

Moving data

Storing local data

Outline

- Prior work
- Precision-induced errors
- Precision selection and control
- Experimental results
- Conclusions

Outline

- Prior work
  - Variable-precision computation, rendering, and hardware
- Precision-induced errors
- Precision selection and control
- Experimental results
- Conclusions

Variable-Precision Computation

Speeding up iterative schemes on parallel machines

- Zubair et al., 1992

Low-power DSP

- Yoshizawa and Miyanaga, 2006

Precision in physics simulations <- floats!

- Yeh et al., 2007

Precision in Rendering

Variable-Precision fixed-function rendering
- Hao and Varshney, 2001

Depth buffer comparator
- Hensley, Singh, and Lastra, 2005

Triangle separation for correct occlusion
- Akeley and Su, 2006

Variable-precision vertex shaders
- Pool, Singh, and Lastra, 2008

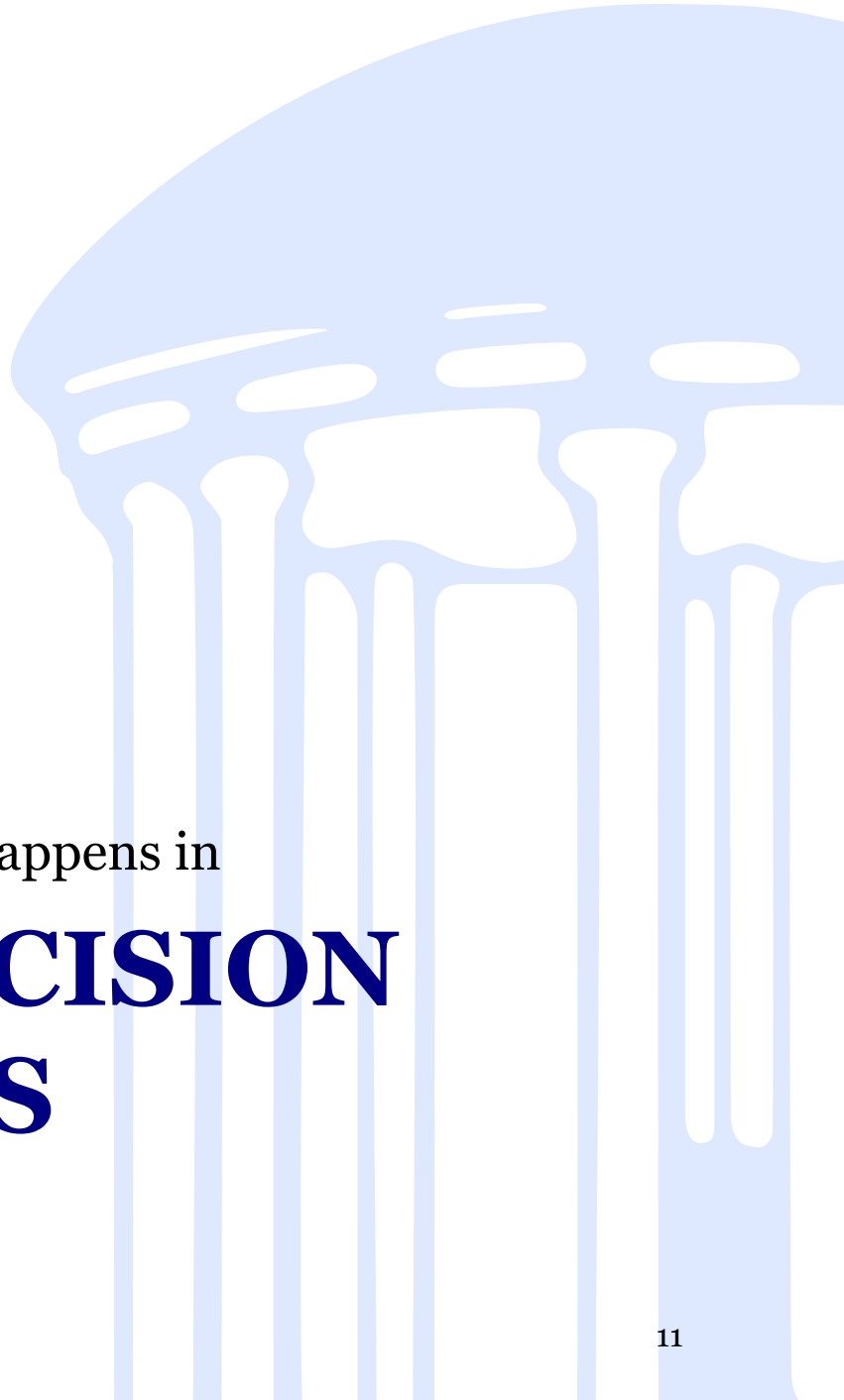
Variable Precision Hardware

Clock gating (Park et al., 2010)

Signal gating (Huang and Ercegovac, 2003)

Power gating

- Coarse (Usami et al., 2009, Sjalander et al., 2005)
- Fine (Pool et al., JOLPE 2011)



So, we have hardware, let's see what happens in

VARIABLE-PRECISION PIXEL SHADERS

Outline

- Prior work
- Precision-induced errors
  - Texture coordinates
  - Arithmetic
- Precision selection and control
- Experimental results
- Conclusions

A Pixel Shader

```
// Parameters:
//
//   sampler2D DIFFUSE_SAMPLER;
//   sampler2D OPACITY_SAMPLER;
//   float lod_blend_begin;
//
//
// Registers:
//
//   Name              Reg   Size
//   ----------------- ----- ----
//   lod_blend_begin c0        1
//   DIFFUSE_SAMPLER s0        1
//   OPACITY_SAMPLER s1        1
//


    ps_3_0
    def c1, 1, 0, 2, 0.5
    dcl_texcoord v0.xy  // IN<4,5>
    dcl_texcoord3 v1.xyz  // IN<7,8,9>
    dcl_texcoord5 v2.zw  // IN<17,18>
    dcl_2d s0
    dcl_2d s1

#line 205 "mw2_foliage.h"
    mad r0.x, v2.z, -r0.x, c1.x
    mov r0.y, c1.y
    dp2add_sat r0.x, r0.x, c0.x, r0.y
    add r0.x, -r0.x, c1.z
```

```
#line 201
    texld r1, v0, s1  // ComputeFoliageAlpha::alphaMask<0>

#line 205
    mul oC0.w, r0.x, r1.z  // ::PM_Inst_Off<3>

#line 221
    texld r0, v0, s0  // PixelMain::base<0,1,2>
#line 39 "gamma.h"
    mul r0.xyz, r0, r0  // ::GammaExpand<0,1,2>

#line 247 "mw2_foliage.h"
    mul r0.xyz, r0, v1  // PixelMain::OUT<0,1,2>

#line 215 "global_common.h"
    rsq r0.x, r0.x
    rcp r1.x, r0.x  // ::sqrt_fast<0>
    rsq r0.x, r0.y
    rsq r0.y, r0.z
    rcp r1.y, r0.x  // ::sqrt_fast<1>
    rcp r1.z, r0.y  // ::sqrt_fast<2>

#line 53 "global.h"
    mad r0.xyz, r1, -c1.w, c1.x
    mad oC0.xyz, r0, -r0, c1.x  // ::PM_Inst_Off<0,1,2>
```

Exaggerated Texture Coordinate Errors

Original frame
(24 mantissa bits)

Blocky textures
(8 mantissa bits)

Arithmetic Errors

Original frame
(24 mantissa bits)

… Different?
(8 mantissa bits)

Exaggerated Arithmetic Errors

Original frame
(24 mantissa bits)

Quantized and dark
(4 mantissa bits)

Different Errors, Different Tolerances

- Colors can be pushed far lower
- Only 13, 11, 9 bits needed to round to 12, 10, 8 bits for color components

Two types of instructions with different error characteristics…

SO WHAT?

So, Treat Them Separately

```
// Parameters:
//
//   sampler2D DIFFUSE_SAMPLER;
//   sampler2D OPACITY_SAMPLER;
//   float lod_blend_begin;
//
//
// Registers:
//
//   Name             Reg   Size
//   ---------------- ----- ----
//   lod_blend_begin c0       1
//   DIFFUSE_SAMPLER s0       1
//   OPACITY_SAMPLER s1       1
//

    ps_3_0
    def c1, 1, 0, 2, 0.5
    dcl_texcoord v0.xy  // IN<4,5>
    dcl_texcoord3 v1.xyz  // IN<7,8,9>
    dcl_texcoord5 v2.zw  // IN<17,18>
    dcl_2d s0
    dcl_2d s1

#line 205 "mw2_foliage.h"
    rcp r0.x, v2.w
    mad r0.x, v2.z, -r0.x, c1.x
    mov r0.y, c1.y
    dp2add_sat r0.x, r0.x, c0.x, r0.y
    add r0.x, -r0.x, c1.z
```

```
#line 201
    texld r1, v0, s1  // ComputeFoliageAlpha::alphaMask<0>

#line 205
    mul oC0.w, r0.x, r1.z  // ::PM_Inst_Off<3>

#line 221
    texld r0, v0, s0  // PixelMain::base<0,1,2>

#line 39 "gamma.h"
    mul r0.xyz, r0, r0  // ::GammaExpand<0,1,2>

#line 247 "mw2_foliage.h"
    mul r0.xyz, r0, v1  // PixelMain::OUT<0,1,2>

#line 215 "global_common.h"
    rsq r0.x, r0.x
    rcp r1.x, r0.x  // ::sqrt_fast<0>
    rsq r0.x, r0.y
    rsq r0.y, r0.z
    rcp r1.y, r0.x  // ::sqrt_fast<1>
    rcp r1.z, r0.y  // ::sqrt_fast<2>

#line 53 "global.h"
    mad r0.xyz, r1, -c1.w, c1.x
    mad oC0.xyz, r0, -r0, c1.x  // ::PM_Inst_Off<0,1,2>
```

So, Treat Them Separately

A

```
// Parameters:
//
//   sampler2D DIFFUSE_SAMPLER;
//   sampler2D OPACITY_SAMPLER;
//   float lod_blend_begin;
//
//
// Registers:
//
//   Name             Reg   Size
//   ---------------- ----- ----
//   lod_blend_begin c0       1
//   DIFFUSE_SAMPLER s0       1
//   OPACITY_SAMPLER s1       1
//


    ps_3_0
    def c1, 1, 0, 2, 0.5
    dcl_texcoord v0.xy  // IN<4,5>
    dcl_texcoord3 v1.xyz  // IN<7,8,9>
    dcl_texcoord5 v2.zw  // IN<17,18>
    dcl_2d s0
    dcl_2d s1

#line 205 "mw2_foliage.h"
    rcp r0.x, v2.w
    mad r0.x, v2.z, -r0.x, c1.x
    mov r0.y, c1.y
    dp2add_sat r0.x, r0.x, c0.x, r0.y
    add r0.x, -r0.x, c1.z
```

Could contribute to texture coordinates

```
#line 201
    texld r1, v0, s1  // ComputeFoliageAlpha::alphaMask<0>

#line 205
    mul oC0.w, r0.x, r1.z  // ::PM_Inst_Off<3>

#line 221
    texld r0, v0, s0  // PixelMain::base<0,1,2>
```

```
#line 39 "gamma.h"
    mul r0.xyz, r0, r0  // ::GammaExpand<0,1,2>

#line 247 "mw2_foliage.h"
    mul r0.xyz, r0, v1  // PixelMain::OUT<0,1,2>

#line 215 "global_common.h"
    rsq r0.x, r0.x
    rcp r1.x, r0.x  // ::sqrt_fast<0>
    rsq r0.x, r0.y
    rsq r0.y, r0.z
    rcp r1.y, r0.x  // ::sqrt_fast<1>
    rcp r1.z, r0.y  // ::sqrt_fast<2>

#line 53 "global.h"
    mad r0.xyz, r1, -c1.w, c1.x
    mad oC0.xyz, r0, -r0, c1.x  // ::PM_Inst_Off<0,1,2>
```

So, Treat Them Separately

A

```
// Parameters:
//
//   sampler2D DIFFUSE_SAMPLER;
//   sampler2D OPACITY_SAMPLER;
//   float lod_blend_begin;
//
//
// Registers:
//
//   Name             Reg   Size
//   ---------------- ----- ----
//   lod_blend_begin c0       1
//   DIFFUSE_SAMPLER s0       1
//   OPACITY_SAMPLER s1       1
//


    ps_3_0
    def c1, 1, 0, 2, 0.5
    dcl_texcoord v0.xy  // IN<4,5>
    dcl_texcoord3 v1.xyz  // IN<7,8,9>
    dcl_texcoord5 v2.zw  // IN<17,18>
    dcl_2d s0
    dcl_2d s1

#line 205 "mw2_foliage.h"
    rcp r0.x, v2.w
    mad r0.x, v2.z, -r0.x, c1.x
    mov r0.y, c1.y
    dp2add_sat r0.x, r0.x, c0.x, r0.y
    add r0.x, -r0.x, c1.z
```

Could contribute to texture coordinates

```
#line 201
    texld r1, v0, s1  // ComputeFoliageAlpha::alphaMask<0>

#line 205
    mul oC0.w, r0.x, r1.z  // ::PM_Inst_Off<3>

#line 221
    texld r0, v0, s0  // PixelMain::base<0,1,2>
```

```
#line 39 "gamma.h"
    mul r0.xyz, r0, r0  // ::GammaExpand<0,1,2>

#line 247 "mw2_foliage.h"
    mul r0.xyz, r0, v1  // PixelMain::OUT<0,1,2>

#line 215 "global_common.h"
    rsq r0.x, r0.x
    rcp r1.x, r0.x  // ::sqrt_fast<0>
    rsq r0.x, r0.y
    rsq r0.y, r0.z
    rcp r1.y, r0.x  // ::sqrt_fast<1>
    rcp r1.z, r0.y  // ::sqrt_fast<2>

#line 53 "global.h"
    mad r0.xyz, r1, -c1.w, c1.x
    mad oC0.xyz, r0, -r0, c1.x  // ::PM_Inst_Off<0,1,2>
```

B

Will *NOT* contribute to texture coordinates

Outline

- Prior work
- Precision-induced errors
- Precision selection and control
  1. Static analysis
  2. Programmer/Artist-directed
  3. Automatic closed-loop
- Experimental results
- Conclusions

Static Program Analysis

```
// Parameters:
//
//   sampler2D DIFFUSE_SAMPLER;
//   sampler2D OPACITY_SAMPLER;
//   float lod_blend_begin;
//
//
// Registers:
//
//   Name              Reg   Size
//   ----------------- ----- ----
//   lod_blend_begin c0        1
//   DIFFUSE_SAMPLER s0        1
//   OPACITY_SAMPLER s1        1
//
    ps_3_0
    def c1, -1, 0, 2, 0.5
    dcl_texcoord v0.xy  // IN<4,5>
    dcl_texcoord3 v2.xyz  // IN<7,8,9>
    dcl_texcoord5 v2.zw  // IN<17,18>
    dcl_2d s0
    dcl_2d s1

#line 205 "mw2_foliage.h"
    rcp r0.x, v2.w
    mad r0.x, v2.z, -r0.x, c1.x
    mov r0.y, c1.y
    dp2add_sat r0.x, r0.x, c0.x, r0.y
    add r0.x, -r0.x, c1.z

#line 201
    texld r1, v0, s1  // ComputeFoliageAlpha::alphaMask<0>

#line 205
    mul oC0.w, r0.x, r1.z  // ::PM_Inst_Off<3>

#line 221
    texld r0, v0, s0  // PixelMain::base<0,1,2>

#line 39 "gamma.h"
    mul r0.xyz, r0, r0  // ::GammaExpand<0,1,2>
#line 247 "mw2_foliage.h"
    mul r0.xyz, r0, v1  // PixelMain::OUT<0,1,2>

#line 215 "global_common.h"
    rsq r0.x, r0.x
    rcp r1.x, r0.x  // ::sqrt_fast<0>
    rsq r0.x, r0.y
    rsq r0.y, r0.z
    rcp r1.y, r0.x  // ::sqrt_fast<1>
    rcp r1.z, r0.y  // ::sqrt_fast<2>

#line 53 "global.h"
    mad r0.xyz, r1, -c1.w, c1.x
    mad oC0.xyz, r0, -r0, c1.x  // ::PM_Inst_Off<0,1,2>
```

10 bits

12 bits

11 bits

10 bits

9 bits

And so on…

Static Program Analysis

The Good

- Compile time process
- Amortized time/energy
- No user effort
- Lowest HW cost

The Bad

- Must assume worst-case (unknown data)
- Worst case for texture coordinates is 0 bits
- Very low energy savings
- Loops

Programmer-Directed Precisions

Precisions are chosen as the effect is designed

[DoF.mkv]

Programmer-Directed Precisions

The Good

- High energy savings
- No runtime overheads
- Low HW cost
- No artificial metrics or approximations of the HVS

The Bad

- User effort necessary

Closed-Loop Precision Selection

Run time – nothing static, here

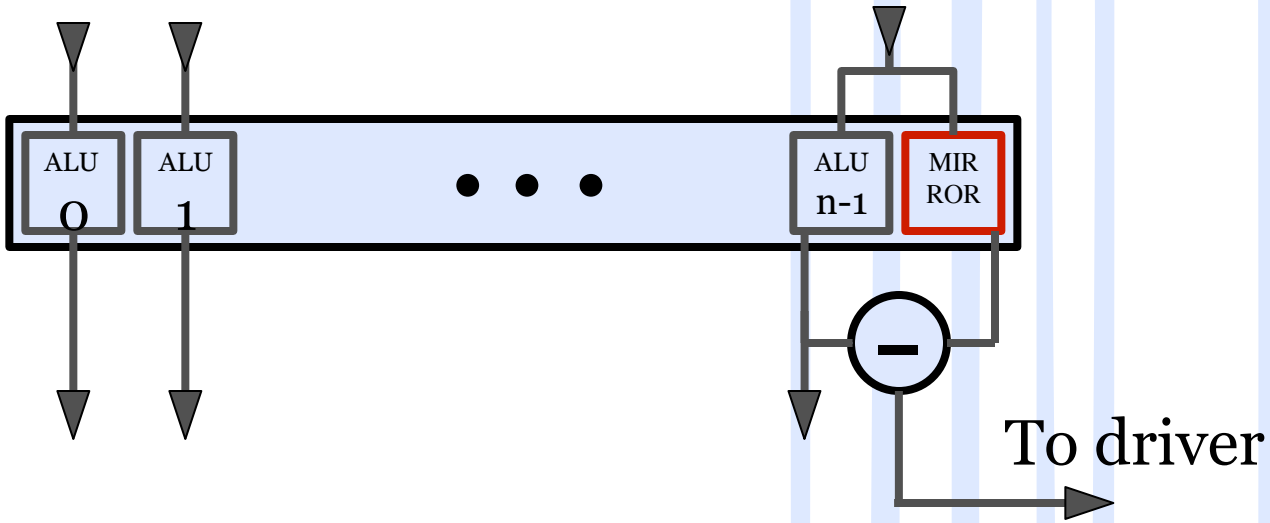
Several facets

- Monitoring
- Sampling
- Control

Closed-Loop Monitoring

"Is the current precision too low?"

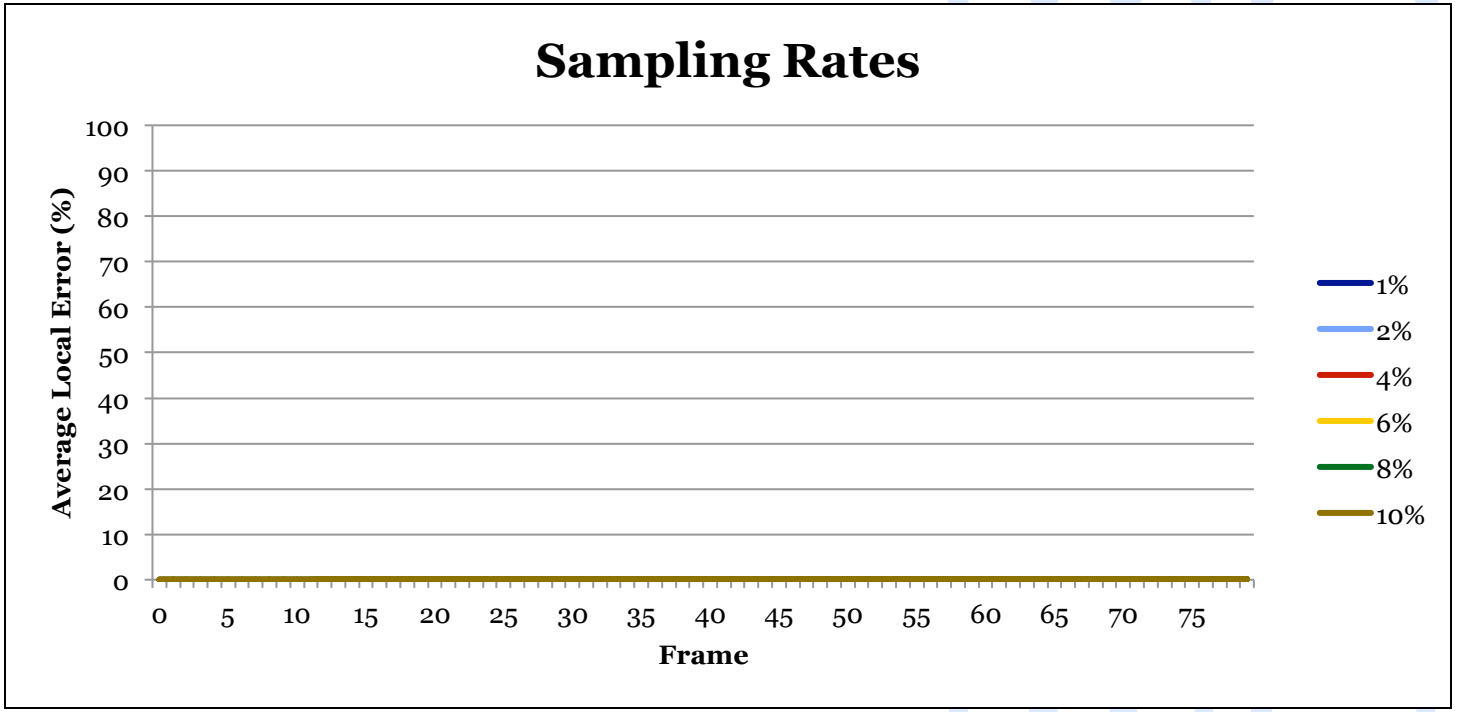
- Compute difference between full and reduced pixel values
- Continuously throughout frame
- Possible implementation



Closed-Loop Sampling

Redundant ALU can't sample every pixel

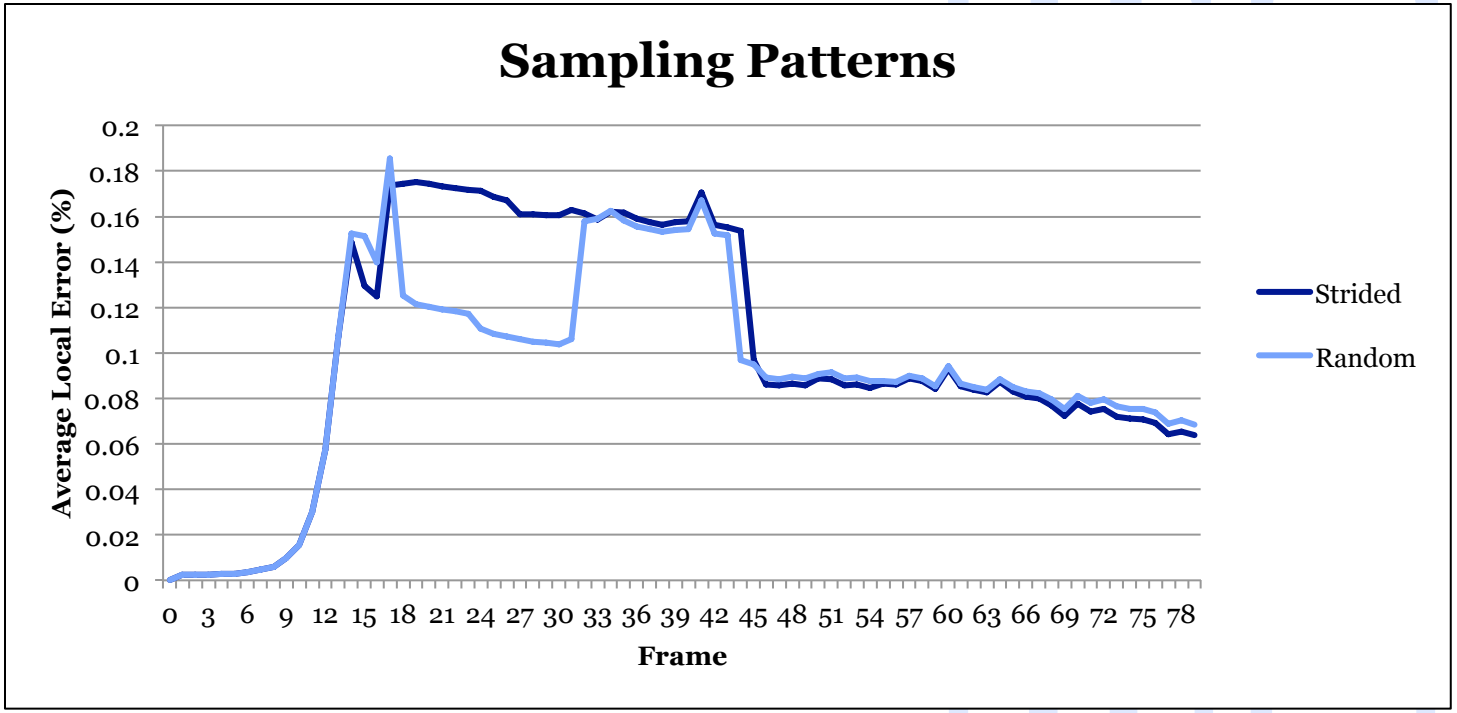
- Would negate energy savings, anyway
- We can get away with much sparser sampling …



Closed-Loop Sampling

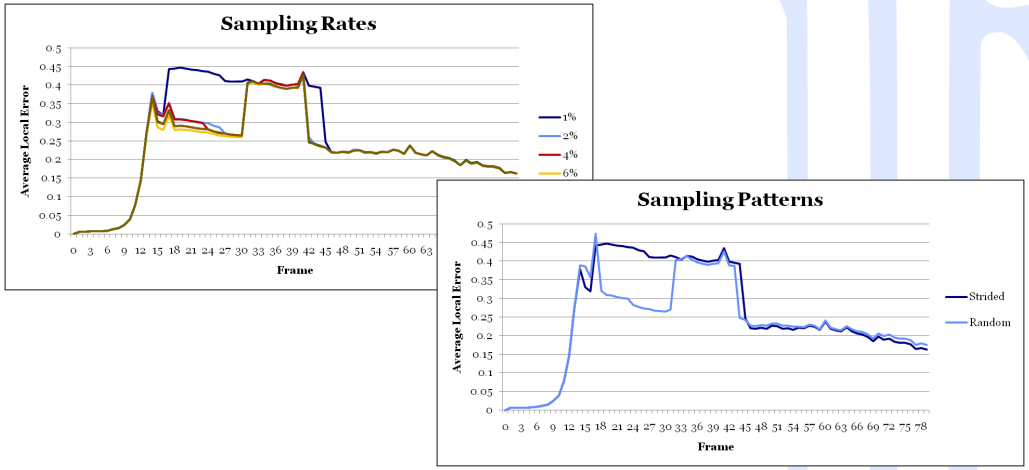
Redundant ALU can't sample every pixel

- We can get away with much sparser sampling
- Without needing a random pattern…



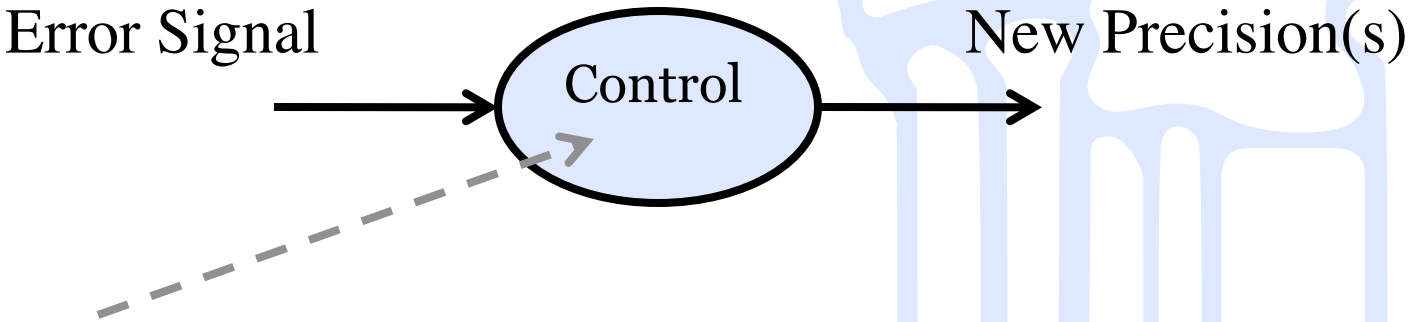
Closed-Loop Sampling

Sampling 1% of generated pixels in a strided pattern is good enough



… but how do we use the sampled data to control two precisions?

Control Systems



5 Schemes:

- Simple
- Simple with delay
- Texture fetch priority
- Dual test
- Dual test with gradient climb

Caveat: Local vs. Global Errors

- Dynamic control schemes see “local,” single-shader errors
- These errors can propagate to later shader passes, which incur their own errors
- Per-shader errors may not correspond to errors in the final rendered frame
- Not a problem we’ve seen in practice

Closed-Loop Control

The Good

- Medium energy savings
- No assumptions about data stream
- No user effort

The Bad

- Complicates shader design
- Control overheads
  - Energy
  - HW
- Transient errors
- Local != Global errors

Outline

- Prior work
- Precision-induced errors
- Precision selection and control
- Experimental results
  - Precisions
  - Errors
  - Energy savings
- Conclusions

Experimental Setup

Static analysis

- Analyze shaders to find minimum safe operating precision

Programmer-directed

- Modify several demo applications
- Allow the user to choose precisions

Automatic closed-loop

- Modify the ATTILA simulator
- Apply our 5 control schemes to several test scenes

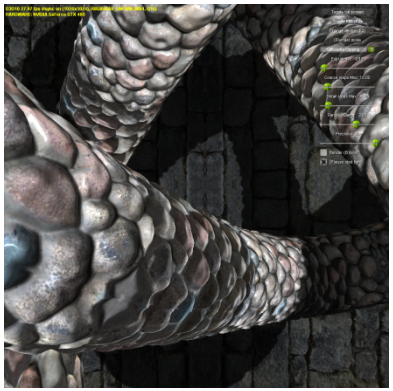
Data Sets

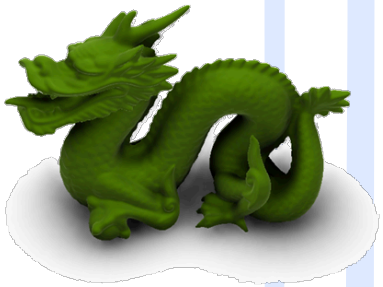
Data Set	Static	Directed	Dynamic
Depth of Field	✓	✓	
Parallax Mapping	✓	✓	
SSAO	✓	✓	
Half Life 2: Lost Coast	✓		✓
Doom 3	✓		✓
Need for Speed: Undercover	✓		✓
Metaballs	✓		✓

Data Sets

HEALTH 67
SUIT 3
AMMO 30 27 0

Races
Select The Closest Event Below With Race Now
Aubrey Street
Sprint





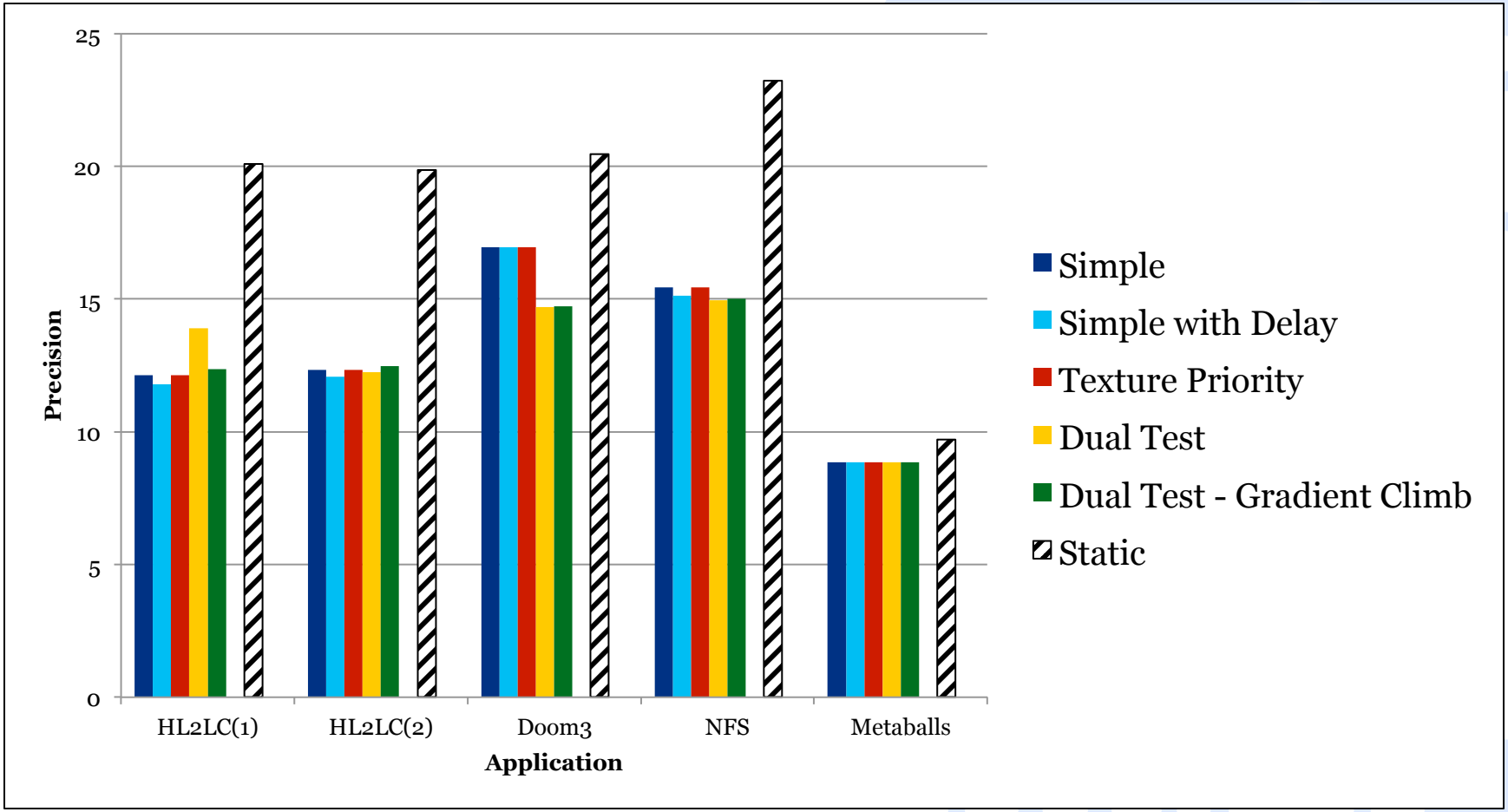
2 f p s
91
8

Metaballs?

Static Precisions

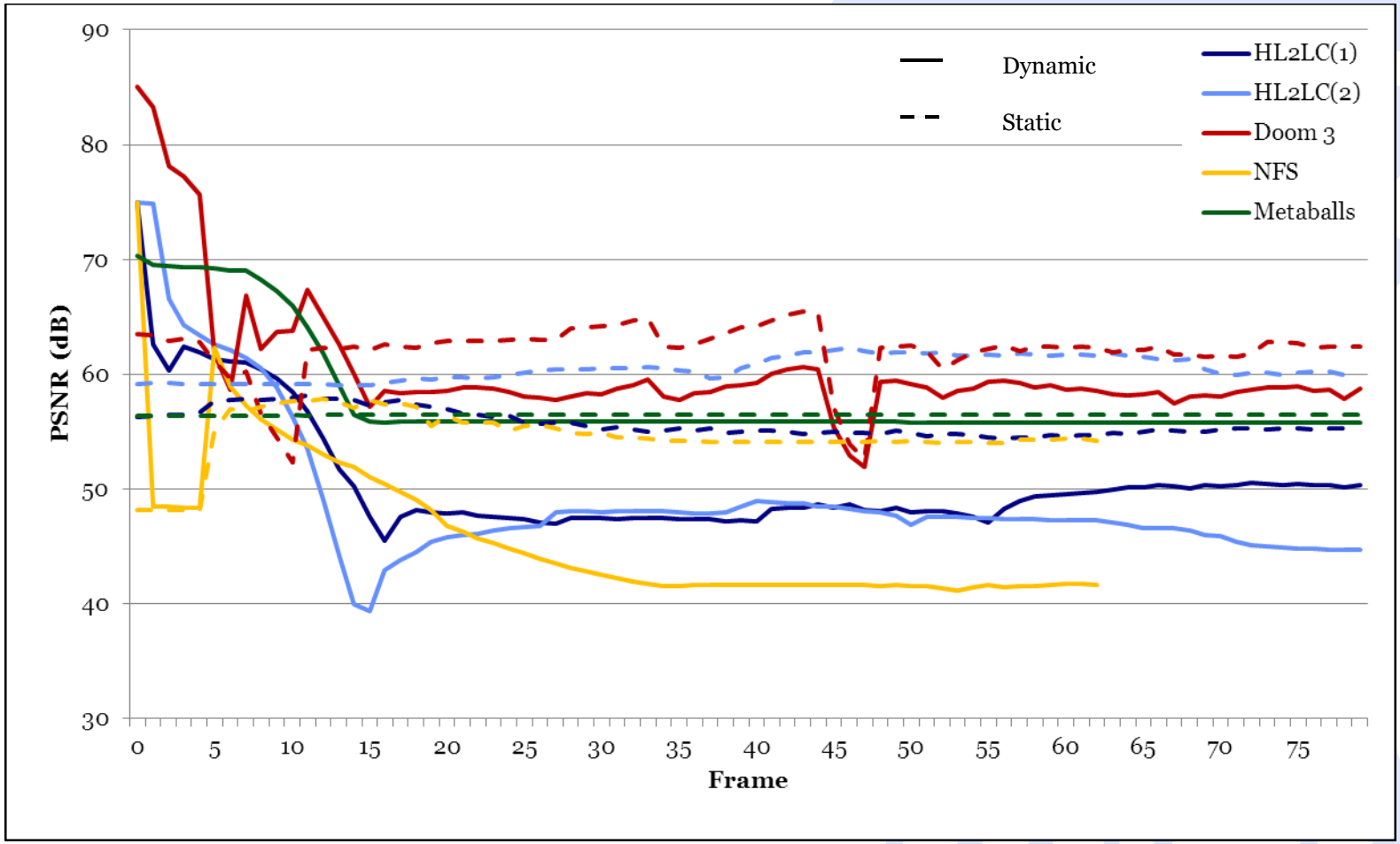
Scene	Precision
HL2LC(1)	19.2
HL2LC(2)	19.0
Doom 3	19.7
NFS	21.8
Metaballs	9.7
SSAO	20.1
Parallax	23.3
DoF	18.5

Closed-Loop Control Schemes



Closed-Loop Precisions

Closed-Loop Errors

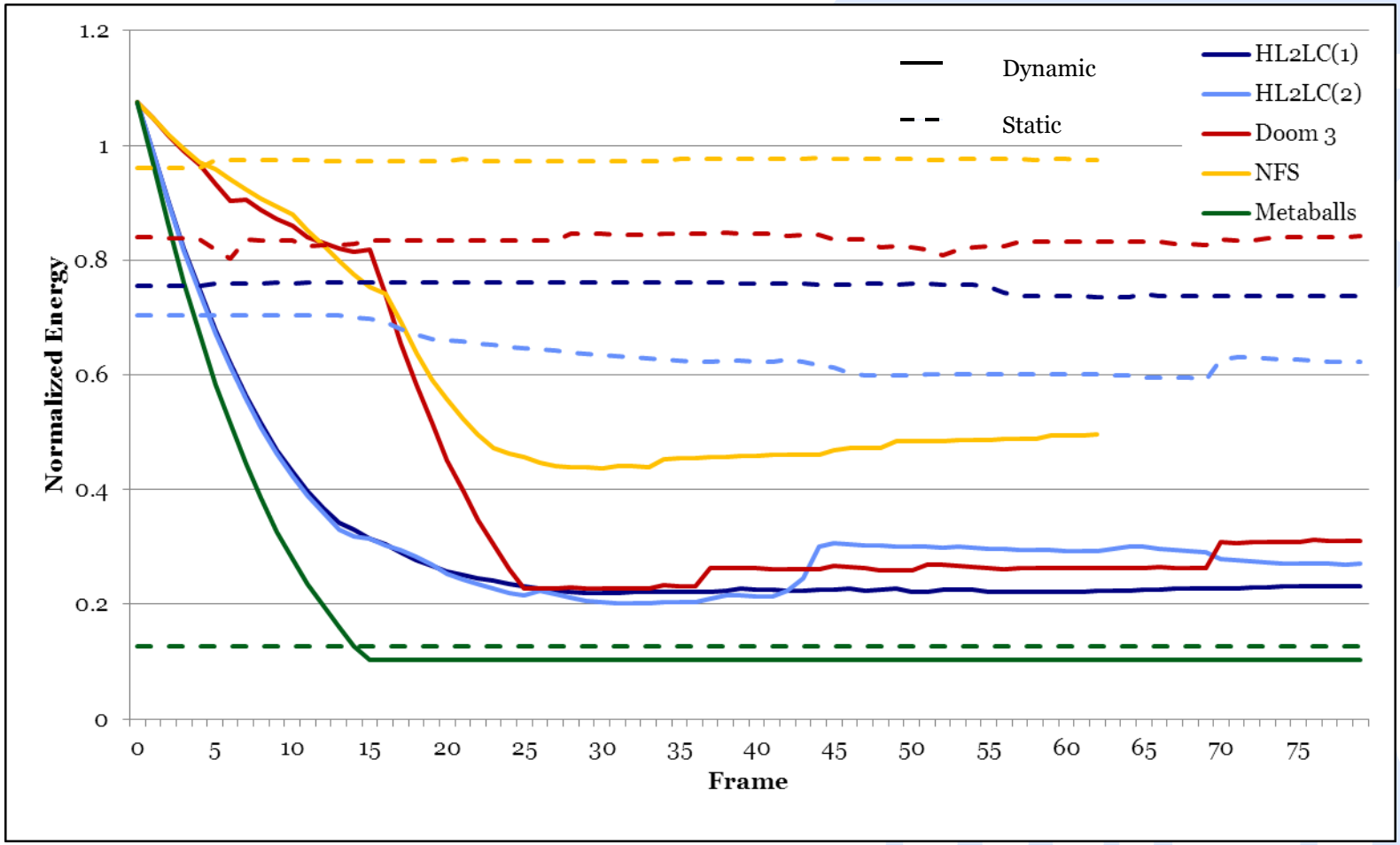


Closed-Loop Errors

These errors are unnoticeable in practice

[HL2LC(1)split.mp4]

Closed-Loop Energy Savings



Programmer-Directed

	Directed			Static	
Scene	Precision	PSNR	Savings	Precision	Savings
SSAO	13.0	53.4	71%	20.1	49%
Parallax	15.2	39.7	61%	23.3	-2%
DoF	12.0	45.6	79%	18.5	33%

Outline

- Prior work
- Precision-induced errors
- Precision selection and control
- Experimental results
- Conclusions

Which Precision Selection Method?

Static

- Not enough savings

Programmer-directed

- High energy savings
- No runtime overheads (energy, HW)
- Reasonable human effort

Closed-loop automatic

- Too invasive to be practical
- Runtime overheads
- Possible to see unexpected errors

Directed Approach

- High savings
  - 70-80% in arithmetic
  - 10-20% overall
- Low errors
  - Acceptable by design
  - Quantitatively low (PSNR, % error)

Future Work

- Pixel shader arithmetic = 20-30% overall GPU energy
- 10-20% total savings is nice, but further savings are possible
  - Data transmission and storage
  - No need to handle unused bits
- FP16 – is it reasonable again?

Thanks to:

- Steve Molnar, Dinesh Manocha, and John Poulton
- Justin Hensley
- NSF
  - CCF-0702712
  - RI #0303590

QUESTIONS?