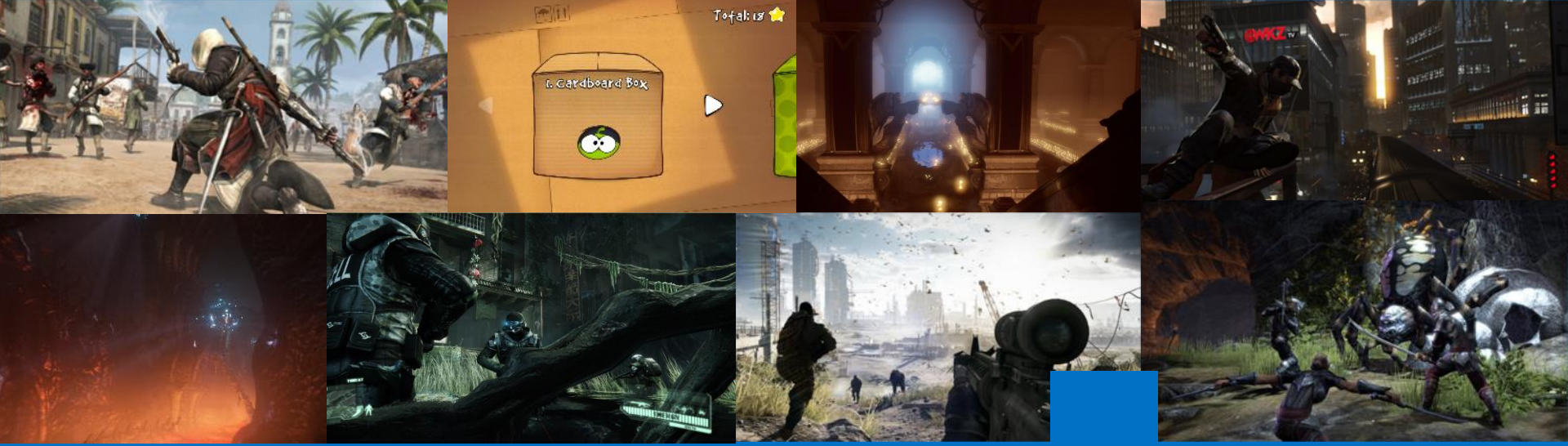
Achieving the Best Performance with Intel Graphics Tips, Tricks, and Clever Bits

Blake Taylor, Graphics Software Development and Validation (GSDV)

GDC 2014

Agenda

- Application
- Platform
- IA Graphics
  - Sampler
  - Fillrate
  - Arithmetic Logic
  - Geometry
- Scaling Your Game
- Conclusion



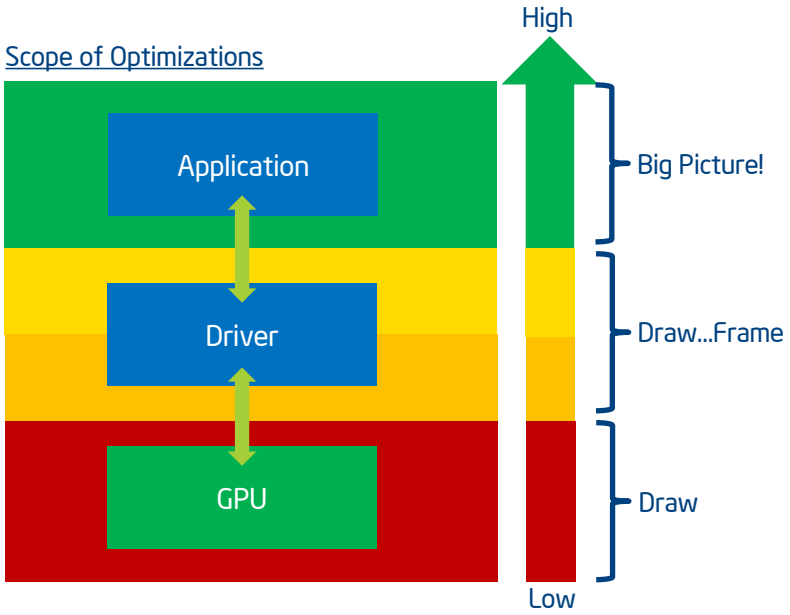
Application

Performance starts at the top

Efficient GPU Programming

Making the most of the pipeline!

- Optimizations within the IA software stack
  - Application specific
  - Generic
- Greatest impact from application optimization
  - Meet your friendly AE!



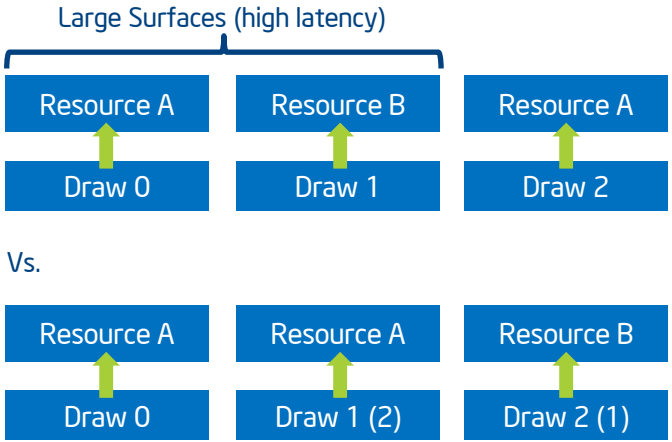
Tooltip!

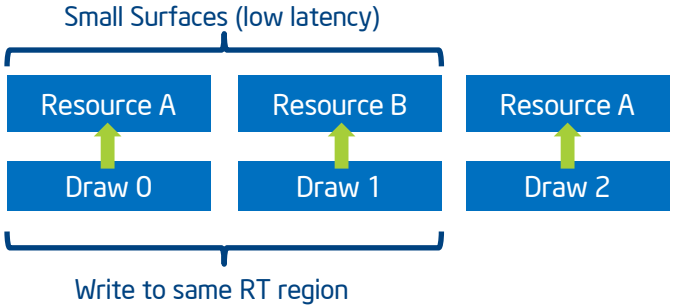
- Use GPA™ to find and optimize GPU hotspots

Draw Dispatching and Resource Update

Be conscious of memory access patterns of dispatched operations

- 3D / 2D operation scheduling
- State / shader changes
- Resource locality





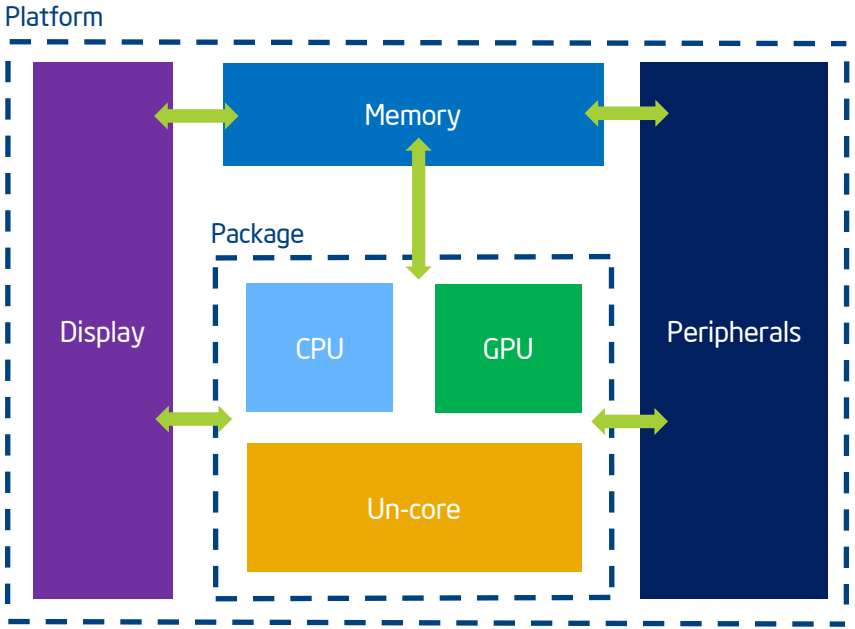
Platform

More than just the sum of it's parts...

Platform

Graphics is only part of the puzzle

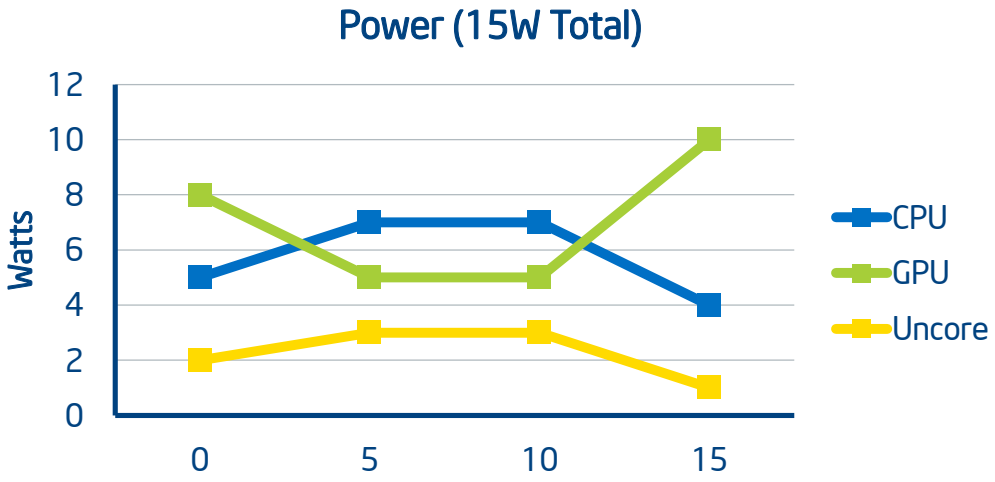
- Unique architecture characteristics
  - Power & performance
  - Memory hierarchy
- Paired platform
  - CPU
  - System memory
- Other constraints
  - Thermal
  - Power

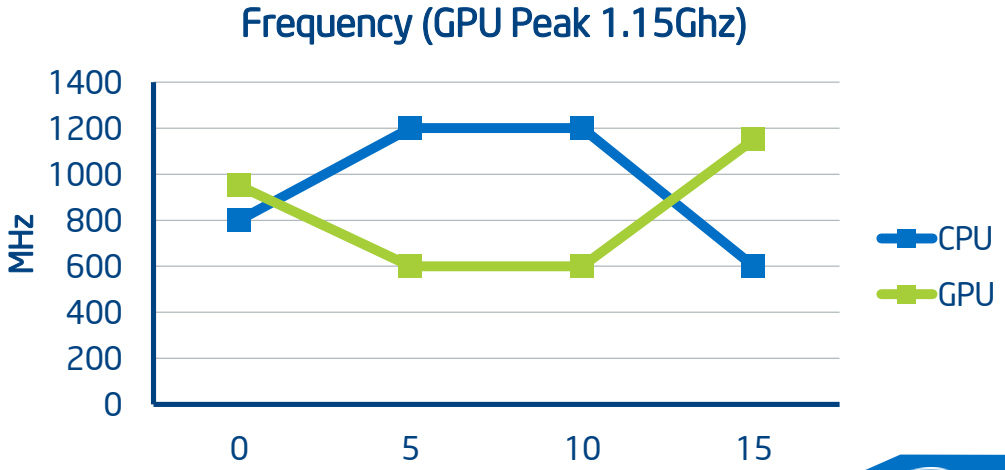


CPU Optimization

Relationship between CPU / GPU

- CPU or GPU bottleneck
- CPU can limit GPU
  - Whaaa?....





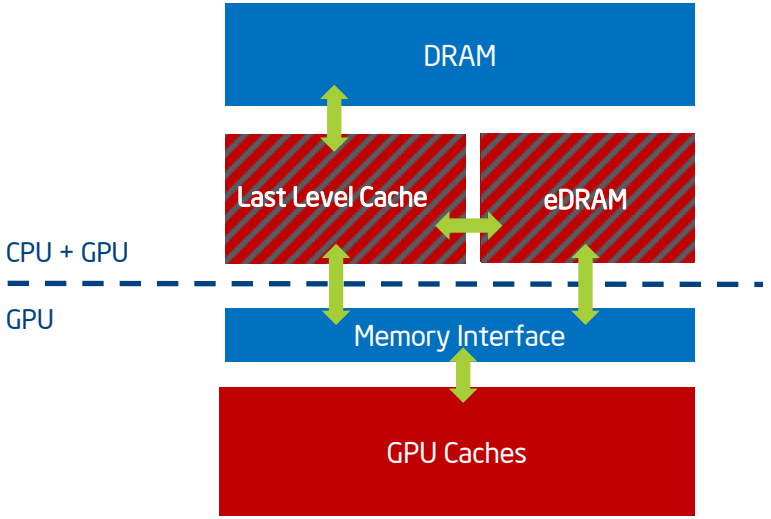
Tooltip!

- Use VTune™ to find and optimize CPU hotspots

Cache Locality Is King

Optimize memory accesses for both CPU and GPU

- Memory bandwidth bound
- Hierarchy varies with platform
  - Optional CPU + GPU Caches
    - Last Level Cache (LLC)
    - Embedded DRAM (eDRAM)
  - GPU



Varying availability

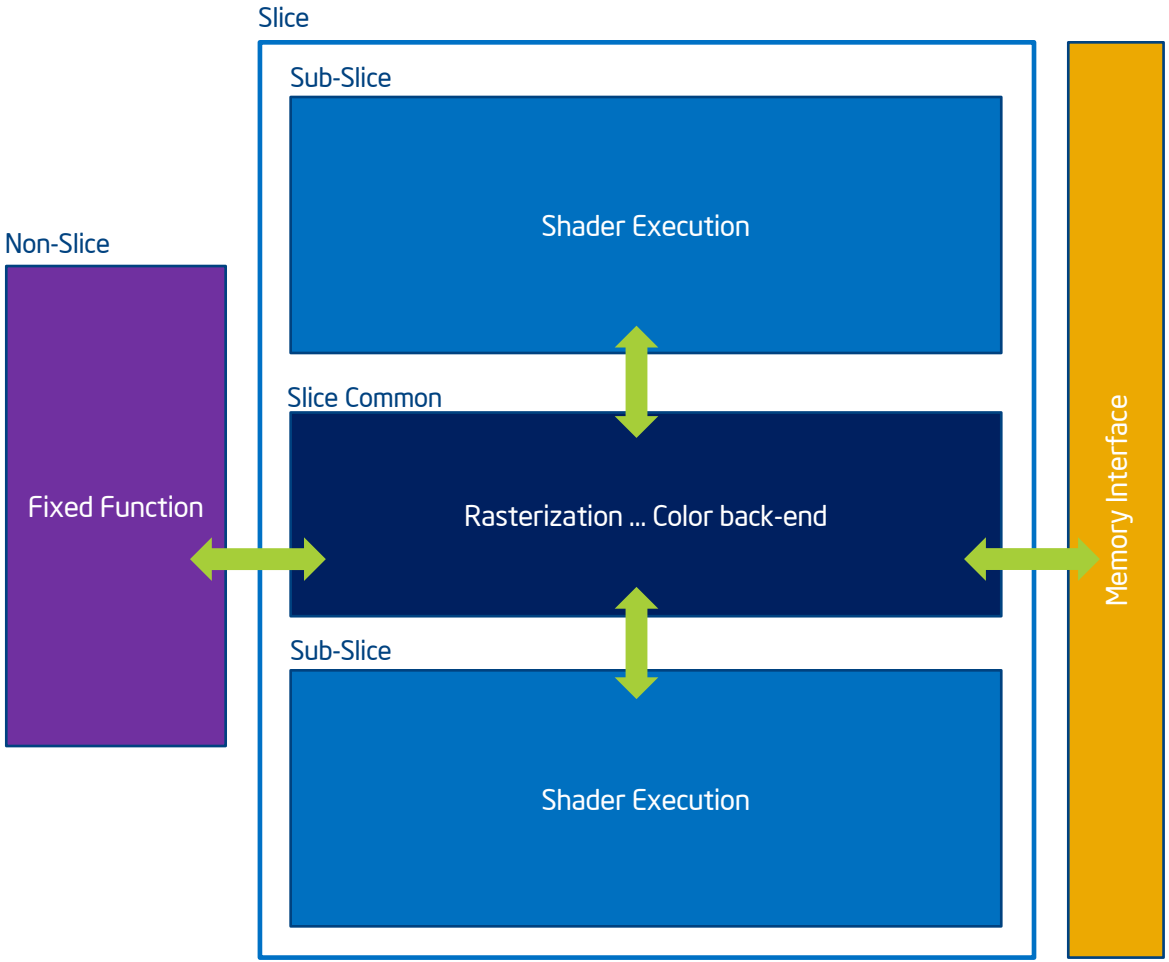
IA Graphics

This is what you came for right?

Architecture

Architectural components

- Non-Slice
  - Fixed function
    - Transformation
    - Clipping
- Slice
  - Slice common
    - Rasterization
    - Shader dispatch
    - Color back-end
  - Sub-slice(s)
    - Shader execution

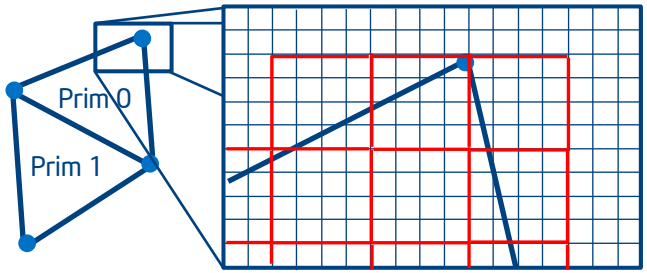


Architecture Scaling

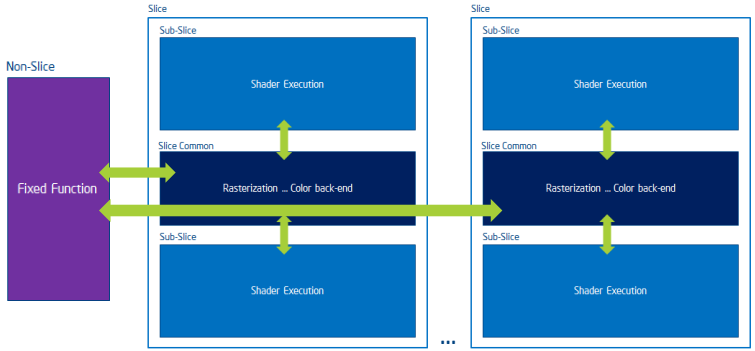
Scaling Components

- Slice
  - Parallel primitive processing
- Sub-slice
  - Parallel span processing

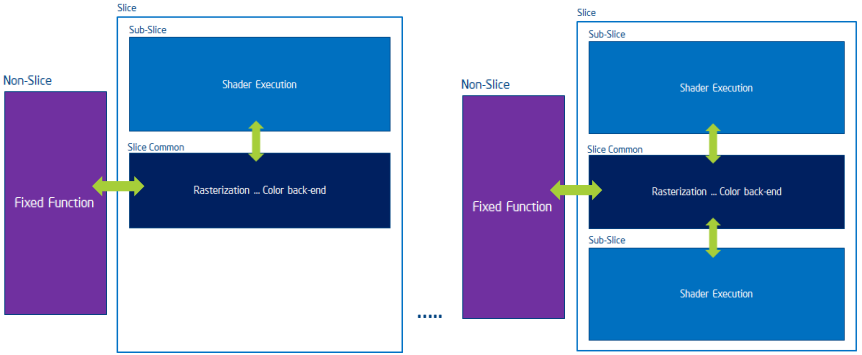
Ex. Post Clip Primitive Processing (4x4 pixel spans)



Slice Scaling (1 – N Slices)



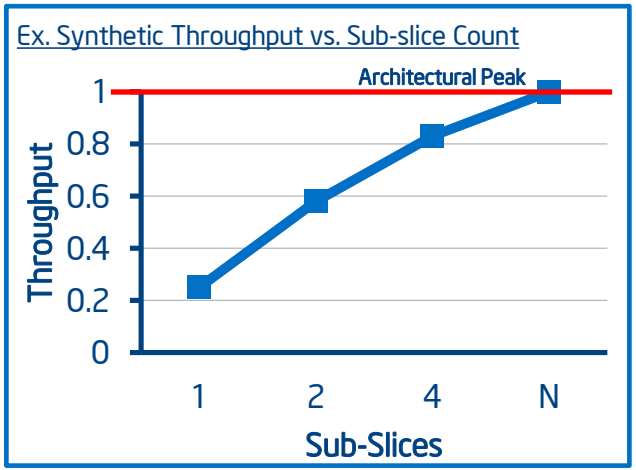
Sub-Slice Scaling (1 – N Sub-Slices)

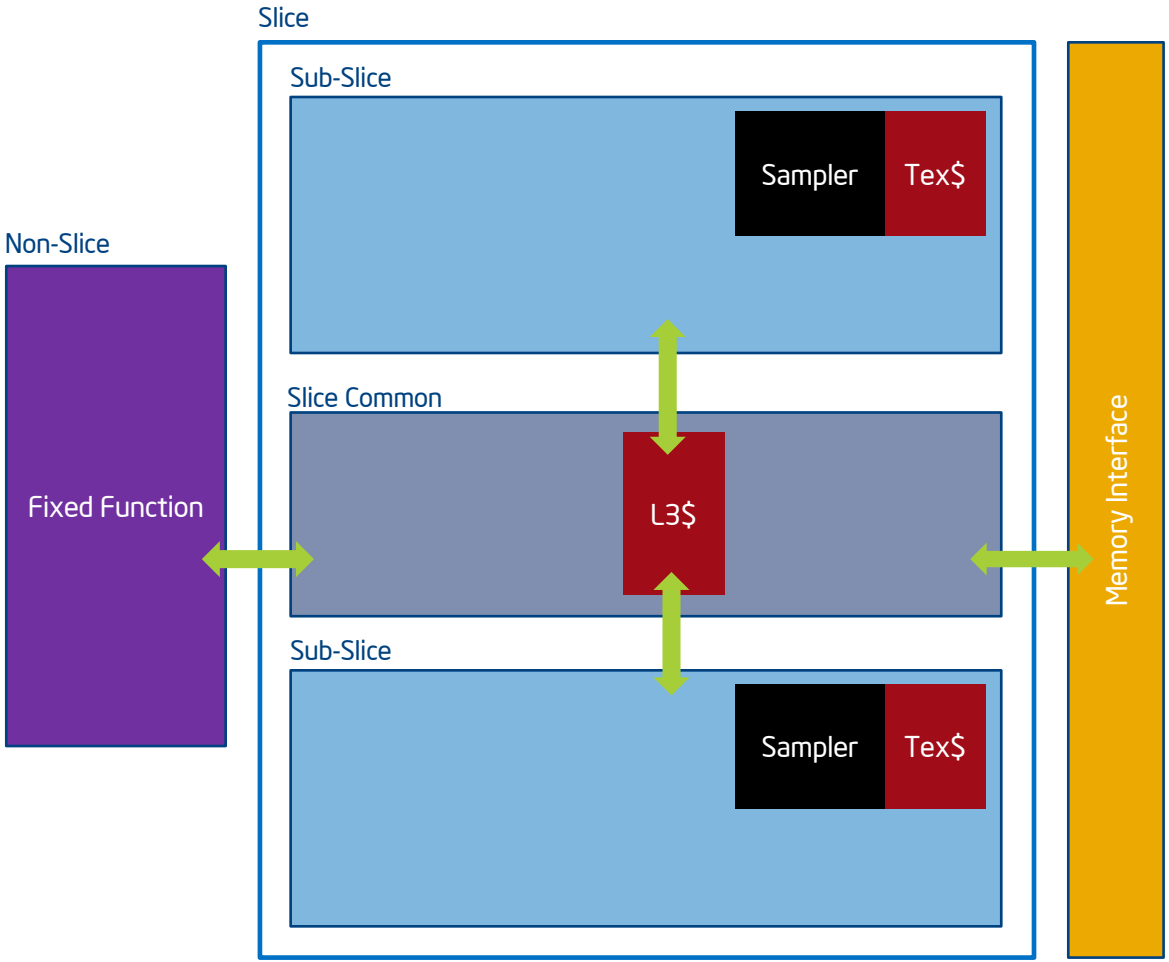


Sampler

1 Sampler Per Sub-Slice

- Local texture cache (Tex$)
- Backed by common L3$

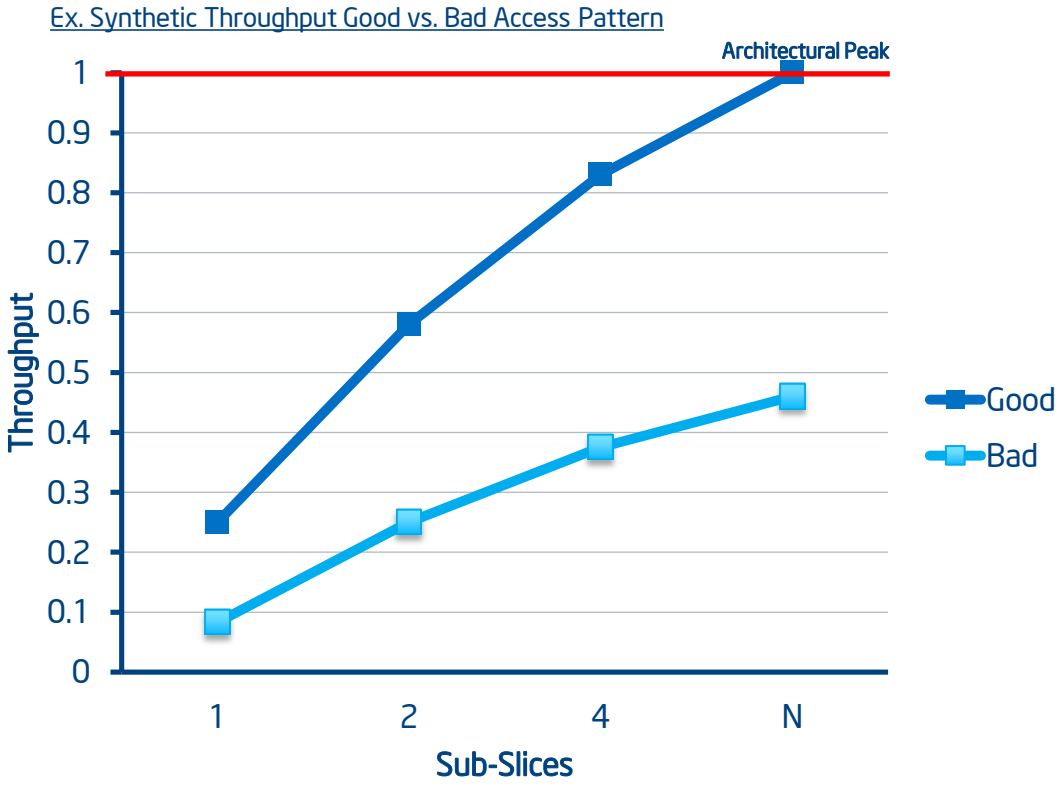




Sampler Performance

Remember Cache Locality? ☺

- Throughput
  - Format
  - Sampling pattern
- Poor access pattern
  - Increased memory b/w
  - Increased latency



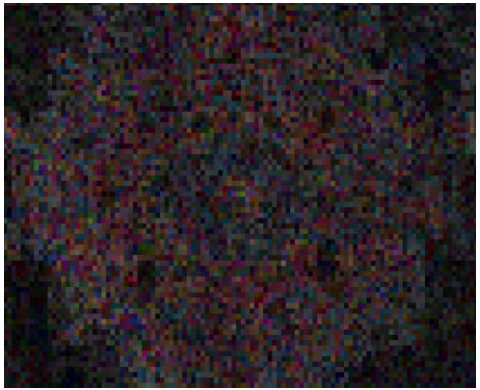
Texture Compression

Utilize as much as possible!

- Offline compression
- Dynamic compression

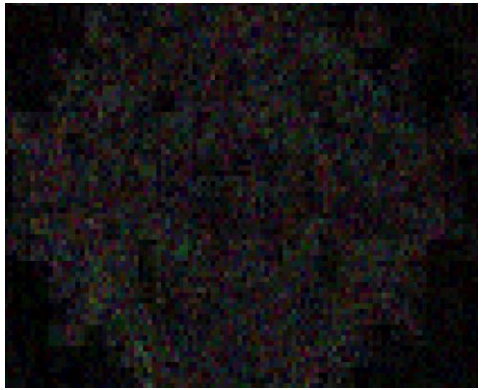
Original Surface

BC1



Error with BC1

BC7

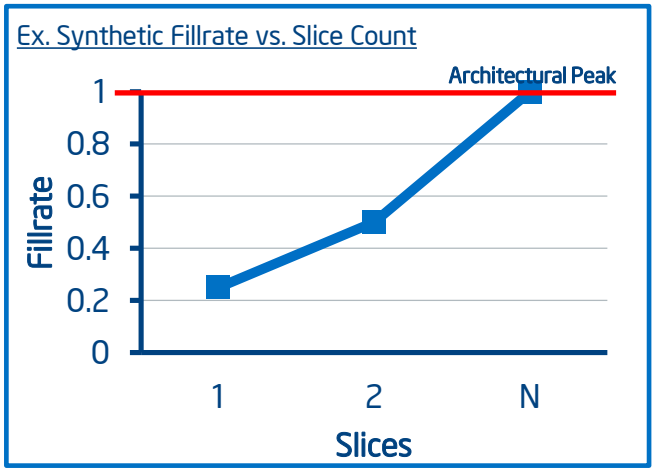


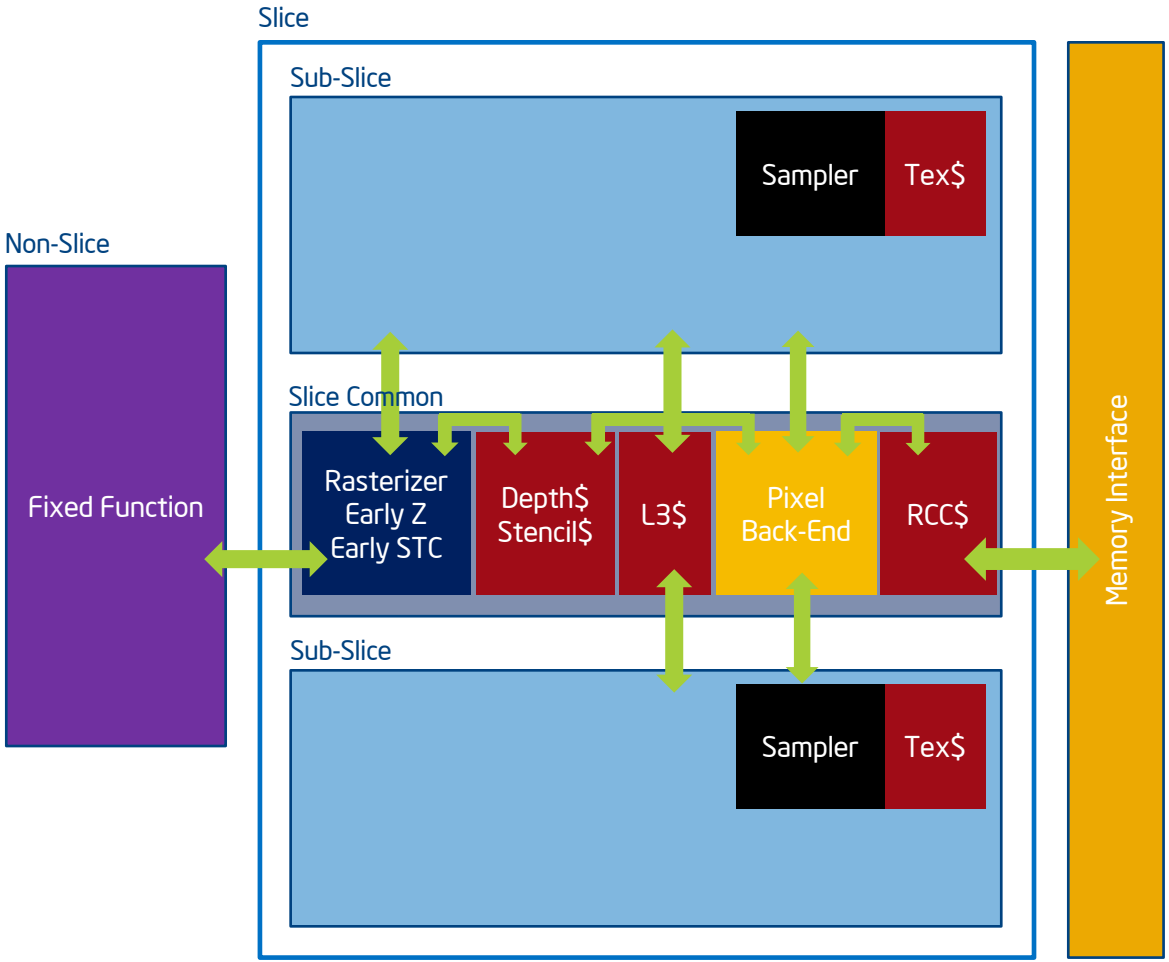
Error with BC7

Fillrate

Per Slice-Common

- Pixel Back-End
- Color Cache (RCC$)

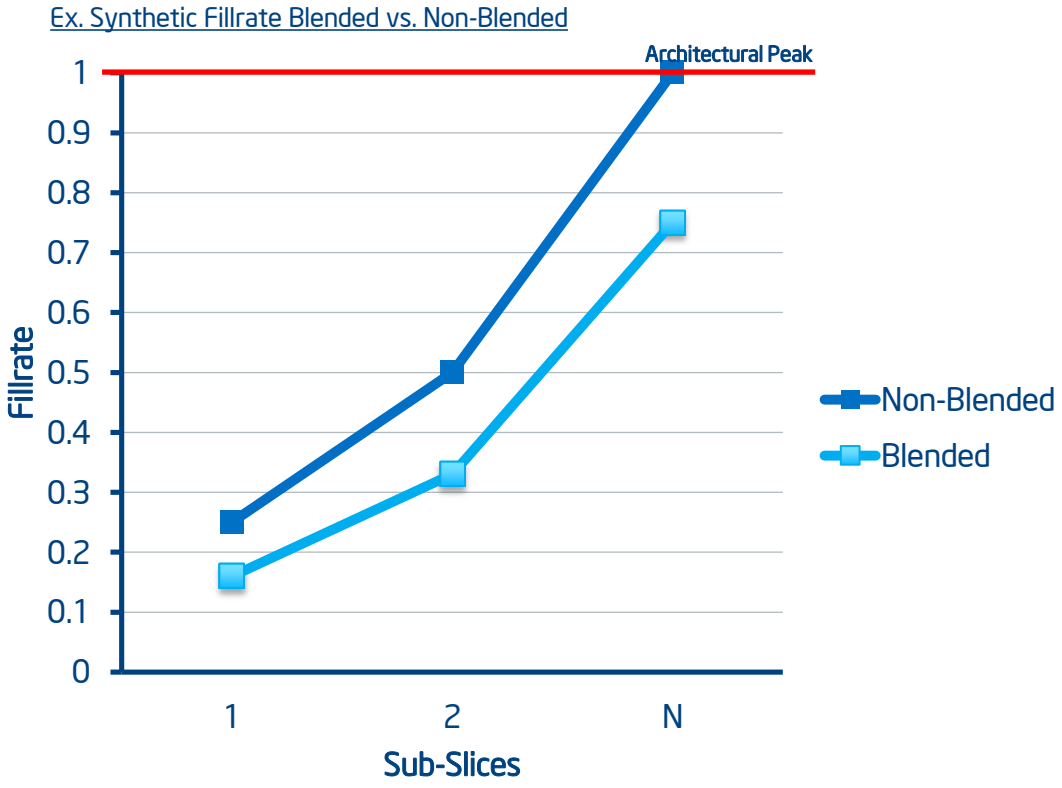




Fillrate Performance

Pumping out color

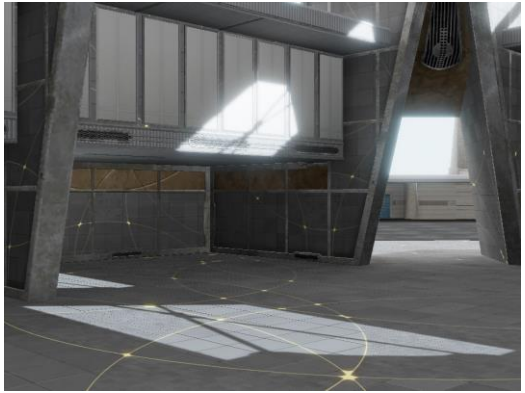
- Throughput
  - Format
  - Dimension + region
- Other factors
  - Rasterization
  - Early Z/STC
  - Pixel Shader Execution
  - Late Z/STC
  - Blend function + mode



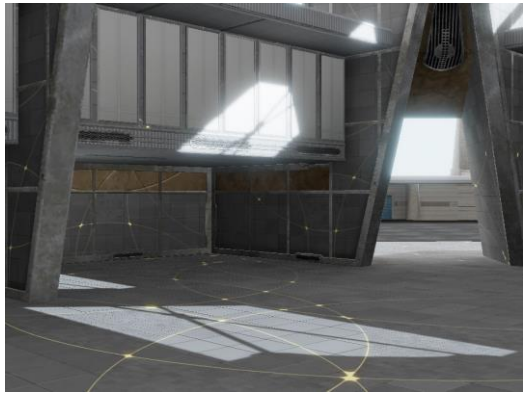
Surface Format

Select the appropriate format for color range

- Intermediate / final render targets
- Cause
  - Higher precision format chosen un-necessarily
- Effect
  - Reduced fill rate
  - Increased memory bandwidth



HDR (R16G16B16A16)

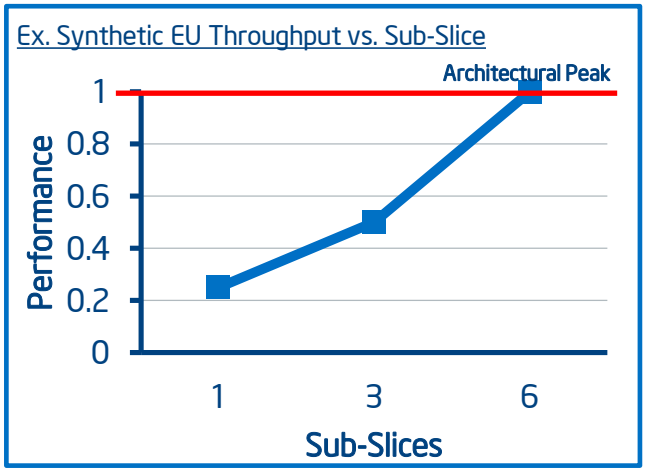


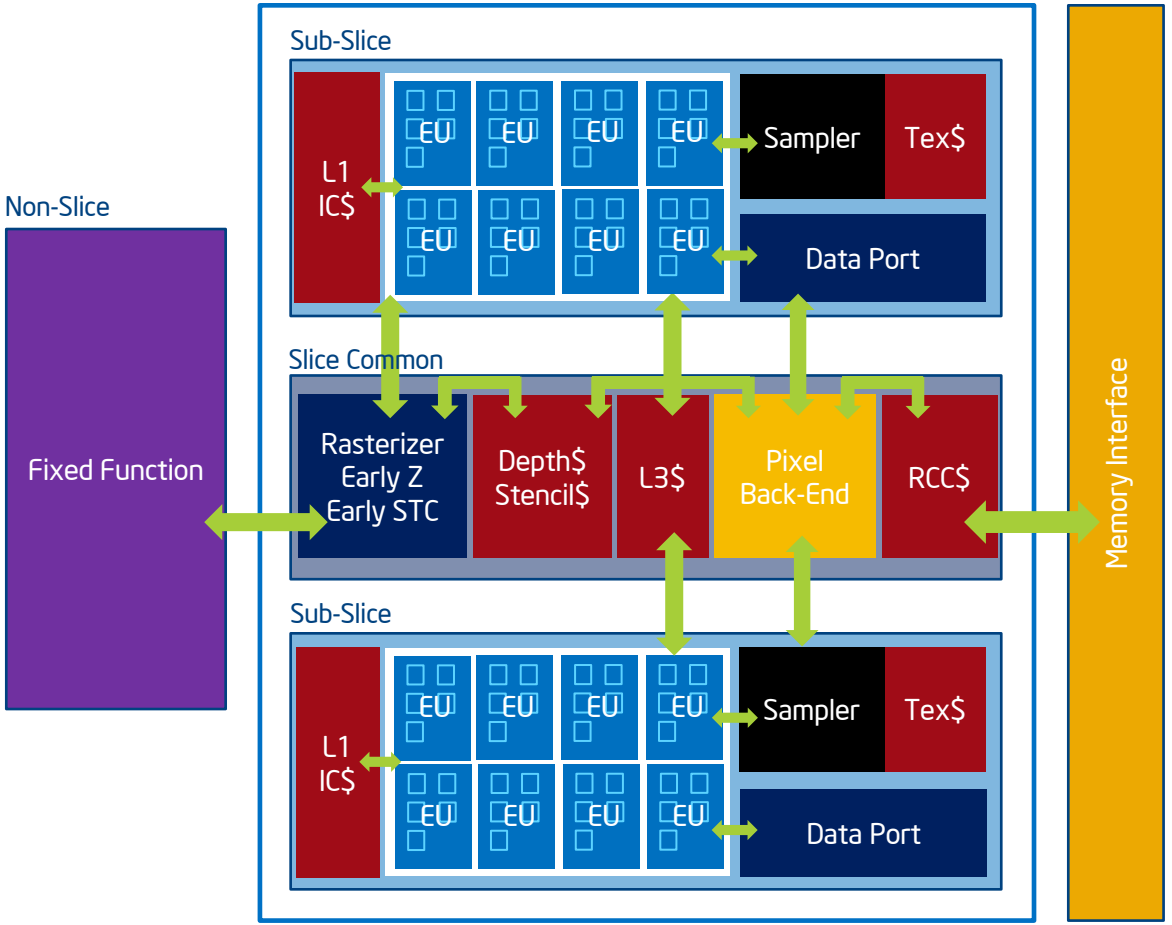
HDR (R10G10B10A2)

Arithmetic Logic

Block Per Sub-Slice

- Execution Units (EUs)
- Instruction Cache (IC$)

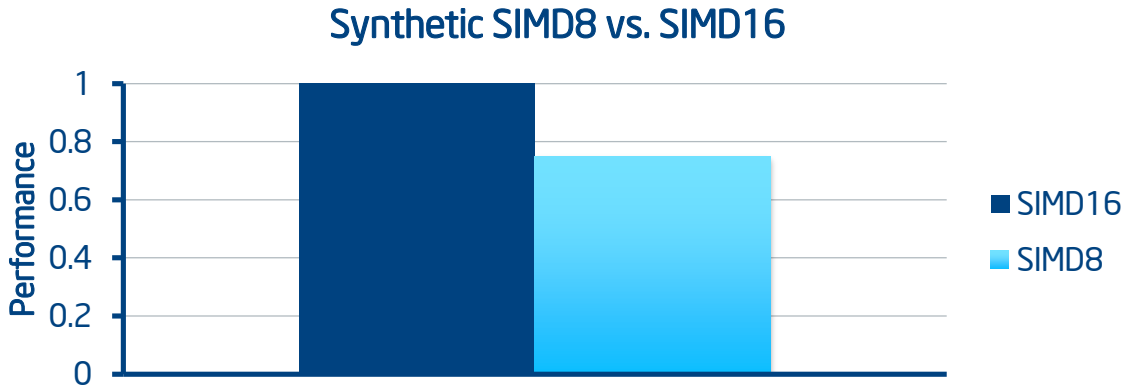


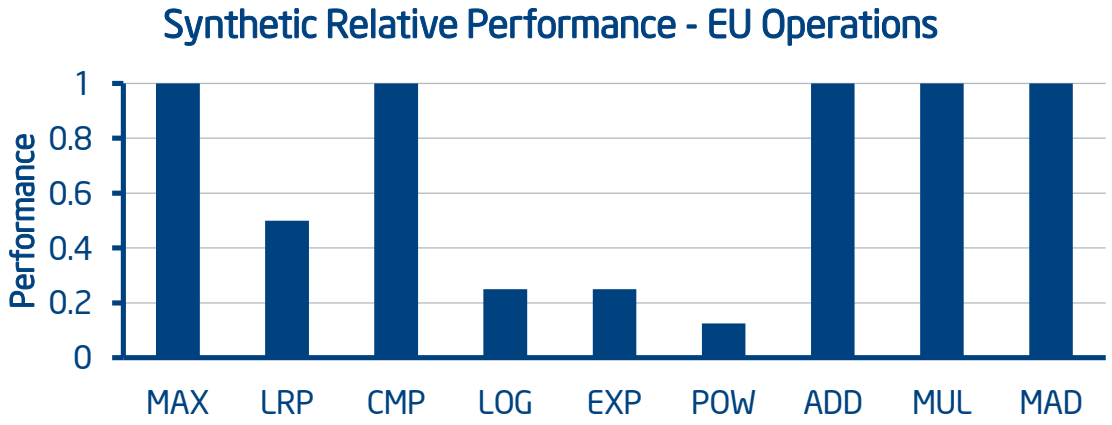


Arithmetic Logic Performance

Algorithmic Complexity

- Control flow
- Math
- Extended math
- Max concurrent registers





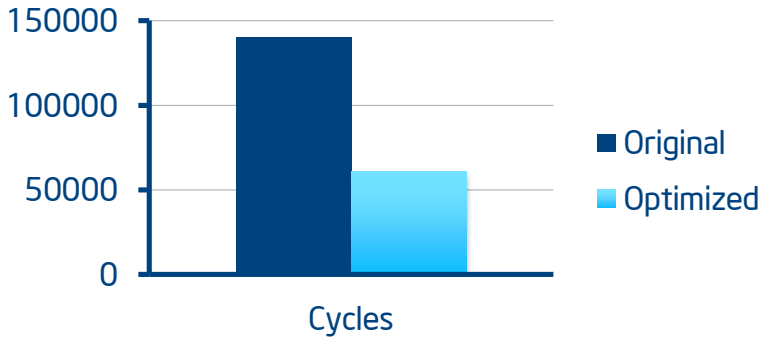
Shader Optimization

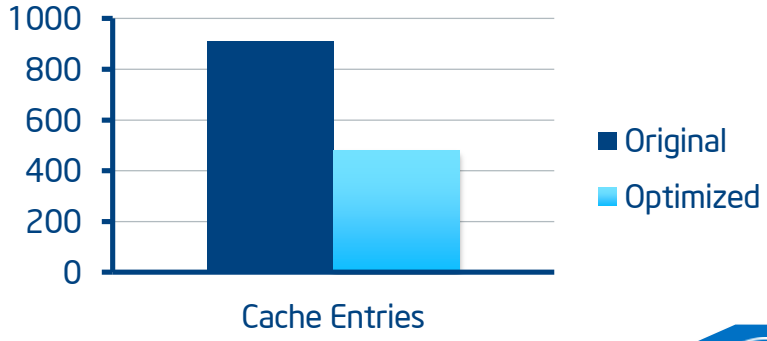
Optimal code based on purpose

- Shader scaling
- The case of the generic shader
  - Generation of un-used outputs ☹

```
vs_3_0
def c17, 2, -1, 0, 1
def c18, 1.44269502, 0.00999999978, -1.44269502, 0
dcl_position v0
dcl_normal v1
dcl_color v2
dcl_position o0
dcl_texcoord o1
dcl_texcoord1 o2
dcl_texcoord2 o3.xyz
dcl_texcoord3 o4.xyz
dcl_color o5
dcl_texcoord4 o6
dcl_texcoord5 o7
dcl_texcoord6 o8.xy
mul r0, c5, v0.y
mad r0, v0.x, c4, r0
mad r0, v0.z, c6, r0
mad o0, v0.w, c7, r0
.. 76 instructions...
mov o7, v2
```

```
vs_3_0
dcl_position v0
dcl_position o0
mul r0, c5, v0.y
mad r0, v0.x, c4, r0
mad r0, v0.z, c6, r0
mad o0, v0.w, c7, r0
```

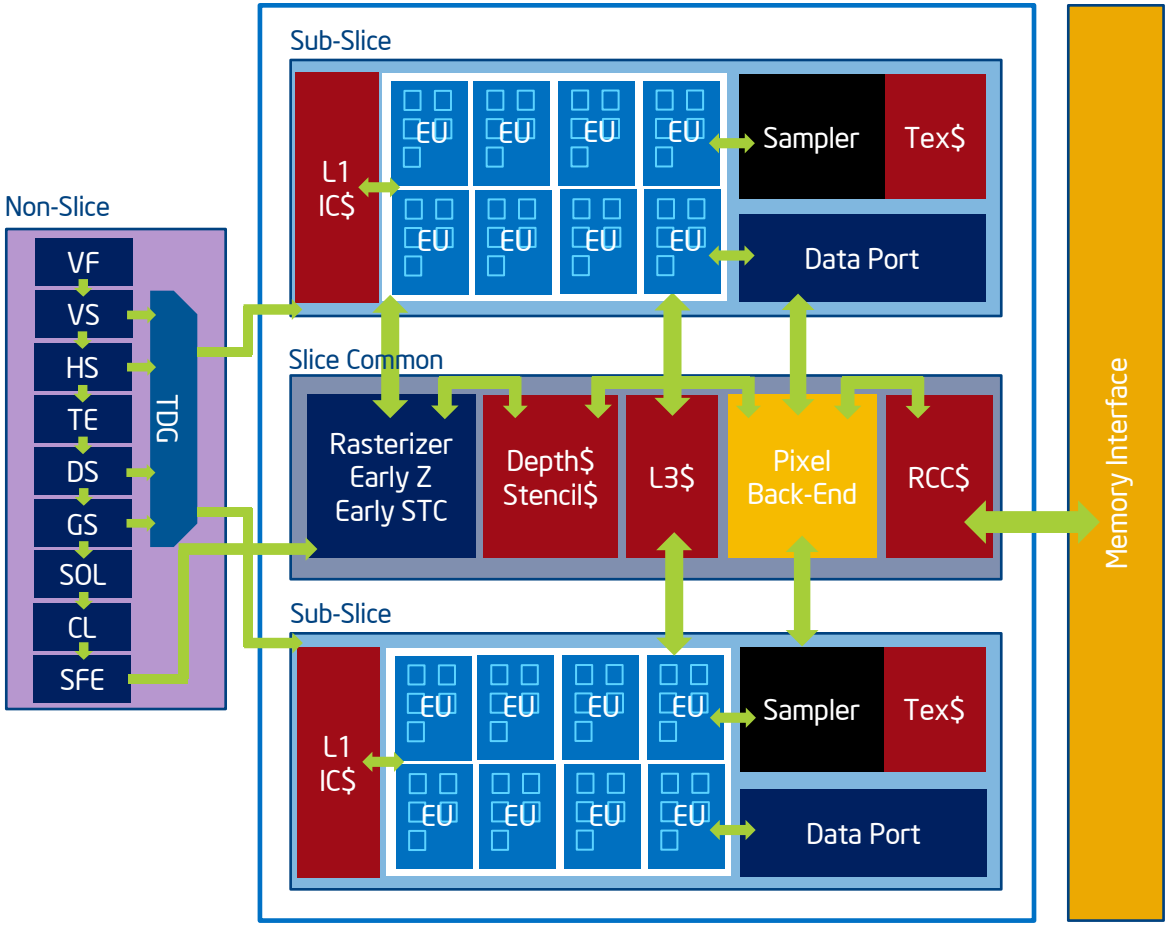




Geometry

Single Non-Slice

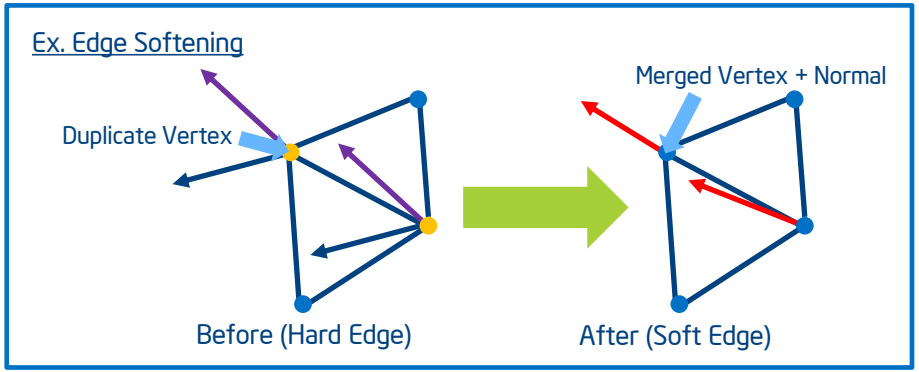
- Fixed Function
  - VS
  - HS
  - TE
  - DS
  - GS
  - SOL
- Clipper
- Setup Front-End



Optimizing Geometry for Algorithmic Complexity

Optimal definitions for a single piece of geometry

- Quality scaling with platform
- Purpose
  - Lighting, depth, animation…



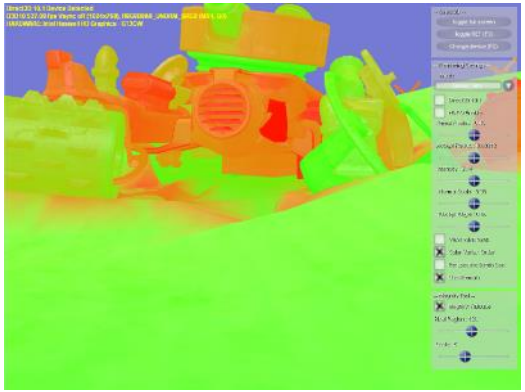
Model with Hard Edges

Model with Soft Edges

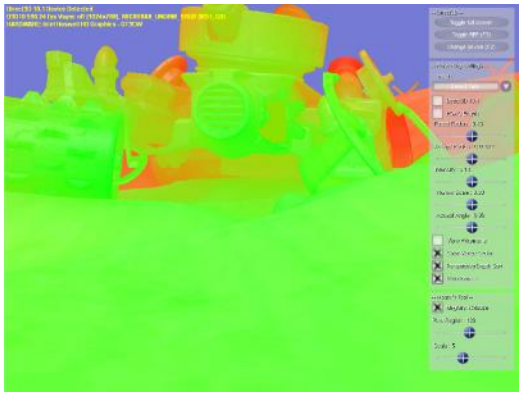
Optimizing Primitive Ordering

Primitive scheduling within a single draw

- Ordering primitives for both locality and latency
- Two cases
  - View dependent
  - View independent
- Sample example (HDAO10.1)
  - Primitive dispatch color coded (green -> red)
  - 2%-13% performance gain



Original Ordering



View Dependent Ordering



Scaling Your Game

Burn baby burn, heat inferno…

Lost Planet 2 : Images courtesy of Capcom

Why do you care?

Wide Range of Platforms + CPU + GPU

- Each with unique performance characteristics
- All of which the user hopes to run your game
  - And run it well ☺

Better selling point? more platforms + happy users == more money? $$$ ☺

How Well Does Your Game Scale?

- Created a game
- Quality settings

Memory Bandwidth

It's all about the memory.. baby

- Will vary greatly with platform
- Why do you care?
  - Read from memory
  - Write to memory

Goal

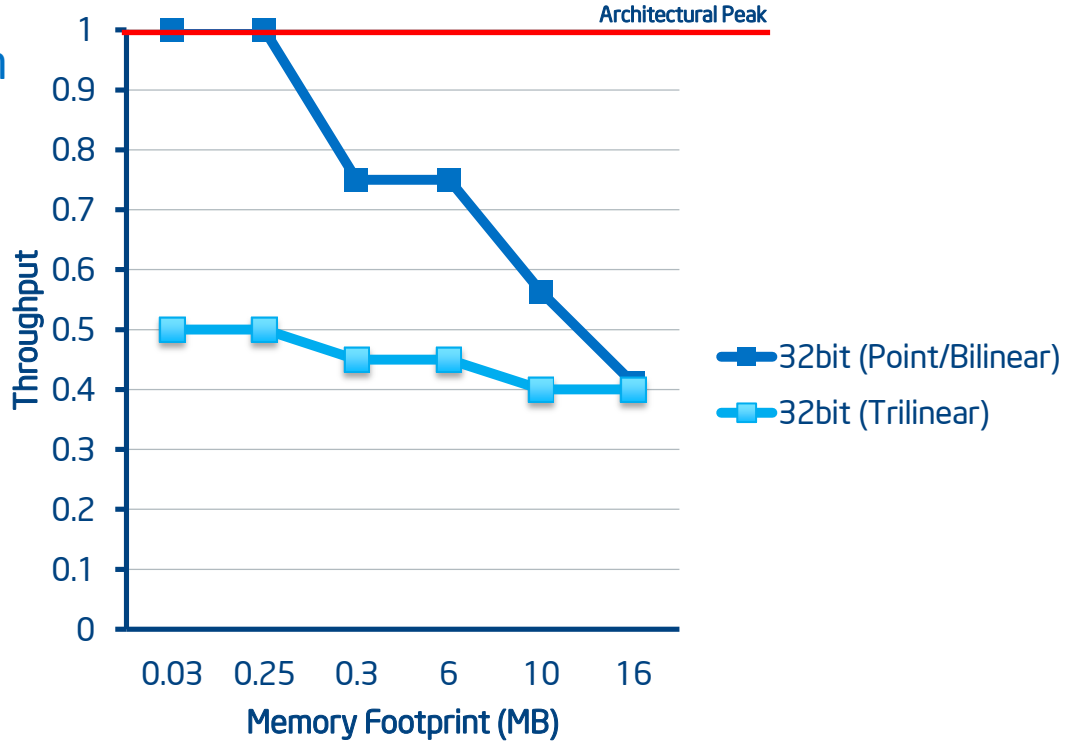
- Establish memory ceiling (budget)

Sampler Throughput

Varies with architecture and platform

- Measure all use cases
  - Dimension
  - Format
  - Filtering mode

Ex. Synthetic Sampler Throughput 32bit Use Cases



Goal

- Select optimal format & dimension

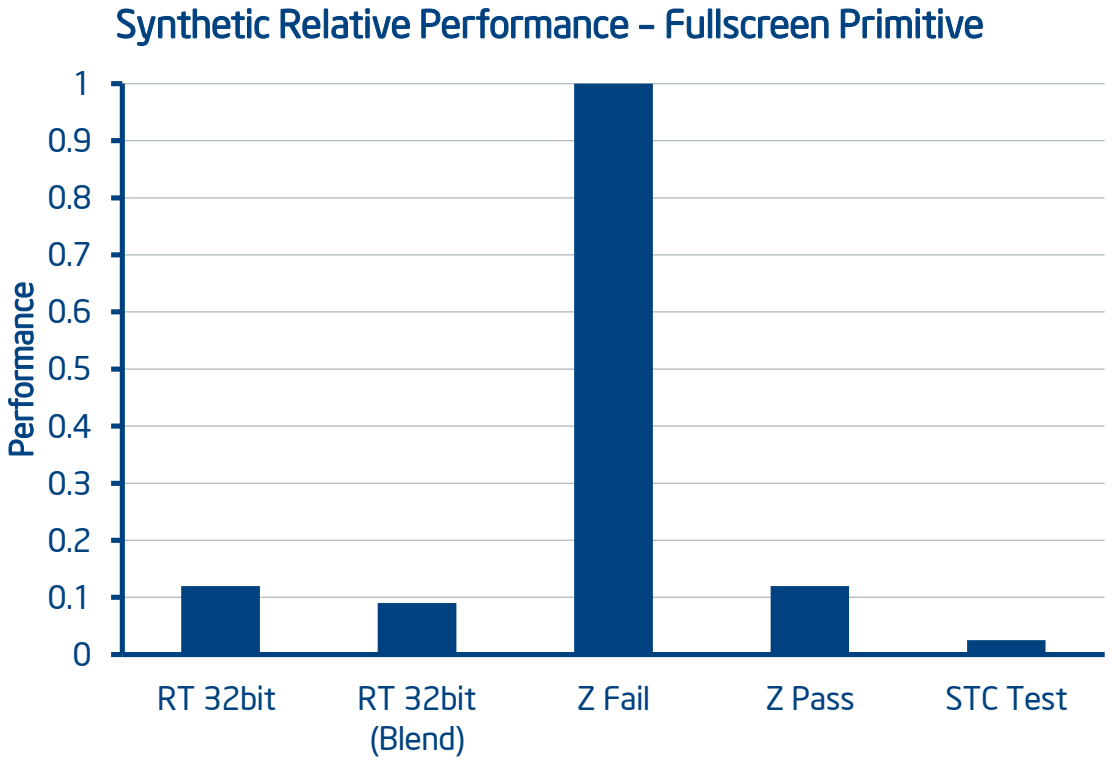
Fill Rate

Multiple surface types

- Render target
  - Format
  - Dimension
  - Blended / Non-blended
- Depth
  - Read +/ Write
- Stencil
  - Read +/ Write

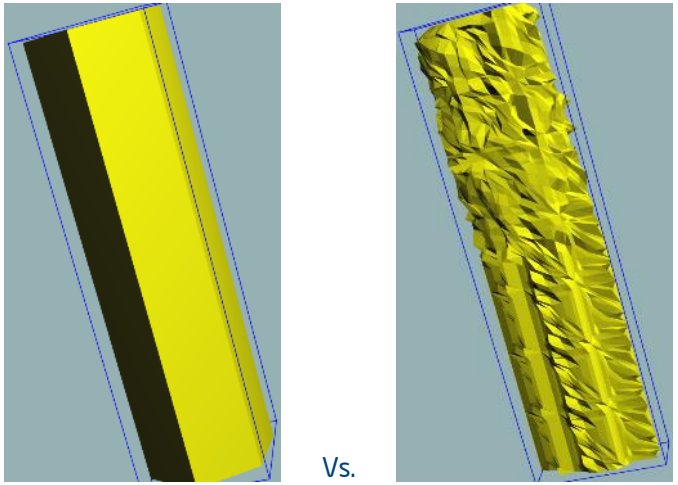
Goal

- Understand relative performance
- Optimal format, dimension, and algorithm



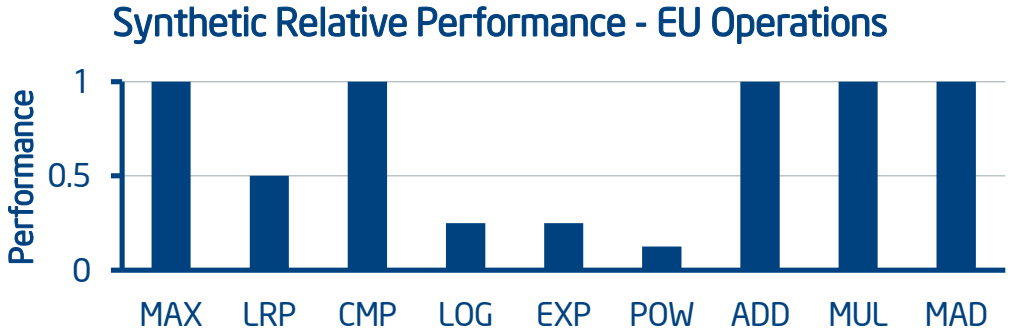
Geometry Throughput

Ex. Geometry Selection Low vs. High Throughput



Fixed function bandwidth and Arithmetic Logic

- Fixed function
  - Clip / Cull
  - Rasterization
- Geometry transformation
  - ALU



Goal

- Optimal geometry and algorithm

THE END

Conclusion

Wrapping it all up in a bow..

Looking Forward

Same game for desktop to phone

- Wide array of platforms
- Adaptable quality settings
- Scaling algorithms
- Optimization

Thanks for attending!

Questions?

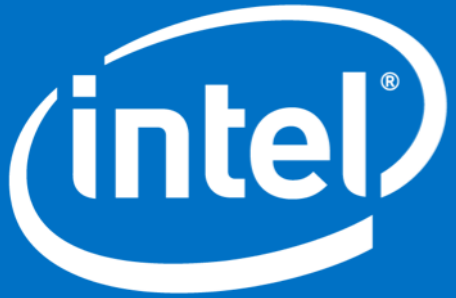
Contact Information

- E-mail : robert.b.taylor@intel.com

Ready for More? Look Inside™.

Keep in touch with us at GDC and beyond:

- Game Developer Conference
  Visit our Intel® booth #1016 in Moscone South
- Intel University Games Showcase
  Marriott Marquis Salon 7, Thursday 5:30pm
  RSVP at **bit.ly/intelgame**
- Intel Developer Forum, San Francisco
  September 9-11, 2014
  intel.com/idf14
- Intel Software Adrenaline
  @inteladrenaline
- Intel Developer Zone
  software.intel.com
  @intelsoftware


intel

Up Next...

12:30 – 1:30

Realistic Cloud Rendering using Pixel Synchronization

Presented by:

Egor Yusov - Intel