intel GAME DEVELOPERS CONFERENCE

THE ARCHITECTURE OF INTEL® PROCESSOR GRAPHICS: GEN11

Antoine Cohade
Sr. Dev. Rel. Engineer
@acohade1

Michael Apodaca
Principal Engineer, 3D SW architect
@gatorfax

Special thanks to contributors & reviewers: A. Lake, S. Junkins, S. McCalla, T. Schluessler

Legal Notices and Disclaimers

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

You may not use or facilitate the use of this document in connection with any infringement or other legal analysis concerning Intel products described herein. You agree to grant Intel a non-exclusive, royalty-free license to any patent claim thereafter drafted which includes subject matter disclosed herein.

The products and services described may contain defects or errors known as errata which may cause deviations from published specifications. Current characterized errata are available on request.

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Performance varies depending on system configuration. No computer system can be absolutely secure. Check with your system manufacturer or retailer or learn more at [intel.com].

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more complete information visit www.intel.com/benchmarks.

Optimization Notice: Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Results have been estimated or simulated using internal Intel analysis or architecture simulation or modeling, and provided to you for informational purposes. Any differences in your system hardware, software or configuration may affect your actual performance.

Intel, Core and the Intel logo are trademarks of Intel Corporation in the U.S. and/or other countries.

*Other names and brands may be claimed as the property of others

Agenda

- Graphics Pipeline Basics
- *for_each(hardwareBlock in Gen11 {*

  describe(hardwareBlock);

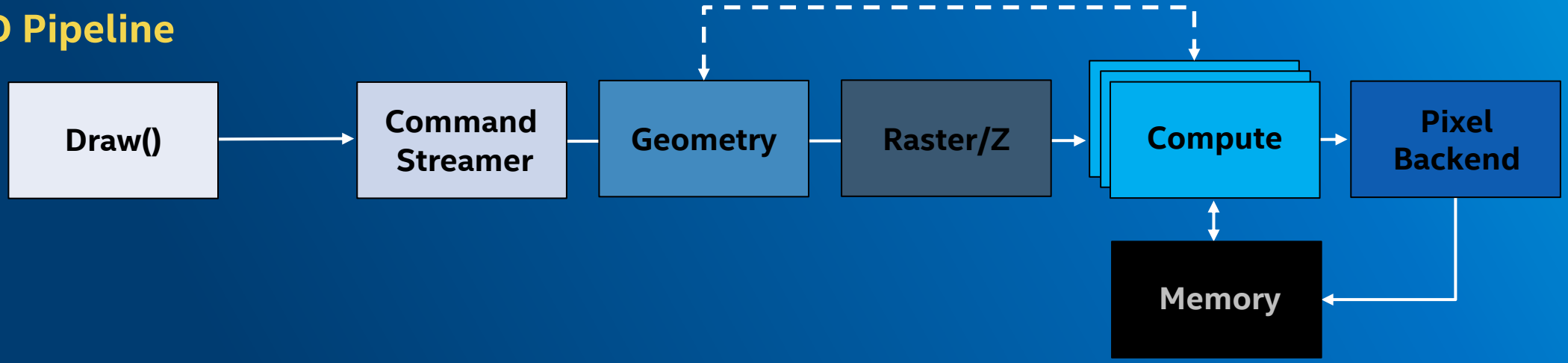
  *give(performanceTip**);*

  });
- Tiled-Based Rendering
- Coarse Pixel Shading

**Most Performance tips also apply to Gen9

Graphics Pipeline 101

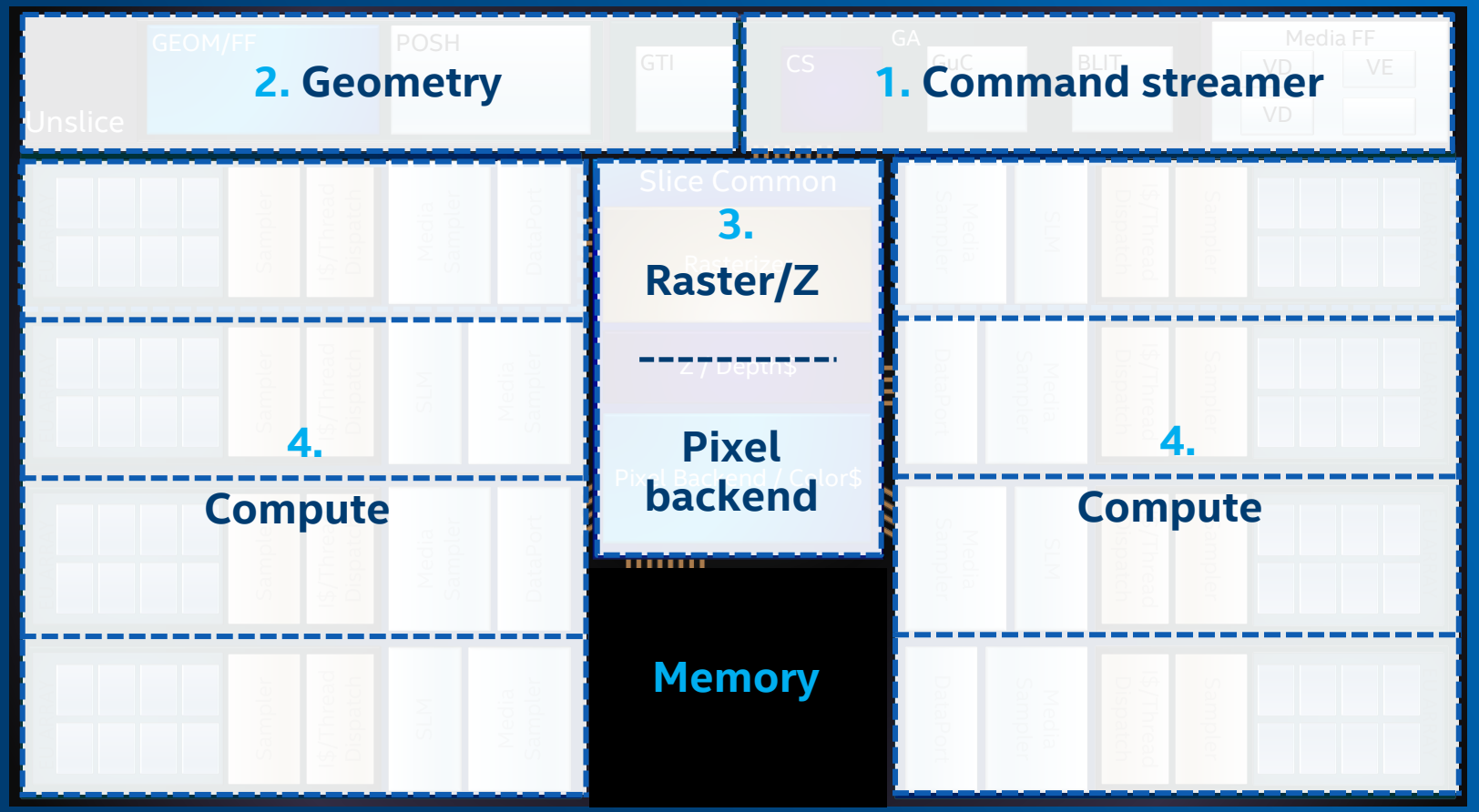
3D Pipeline



Compute Pipeline

How does that match to Gen?

How does that match to Gen?



Un-Slice (1,2)

1. Global Assets: Command Streamer, Ring interface, *pwr mgmt*, *Blitter*, *Thread Dispatch*
2. Geometry Fixed Functions: 3D HW pipeline (geometry)

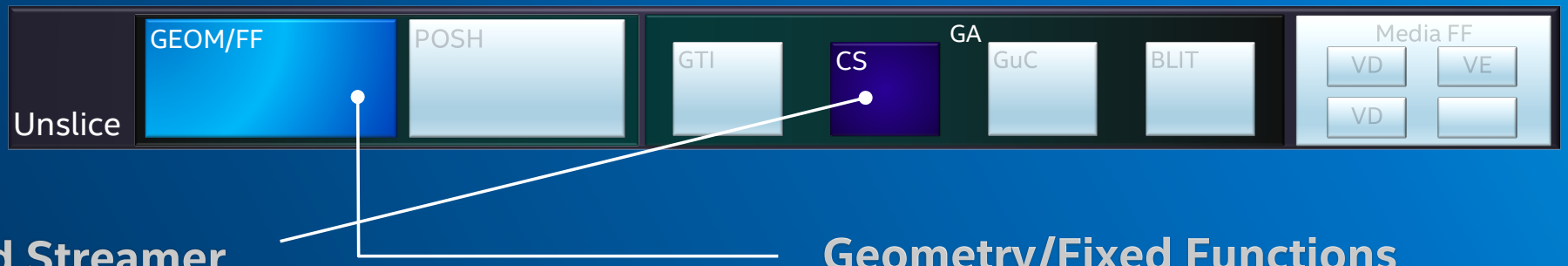
And also:

Media FF: Low power HW encode, decode, & video processing

Slice (3,4)

3. Slice-Common/L3: Setup, Rasterizer, Z, L3 Cache, and Pixel processing
4. Sub-Slices: Array of Execution Units, Inst Caches (IC$), Texture Samplers w/ Caches, Load/Store Units

Unslice architecture



Command Streamer

- Reads and dispatches command buffer
  - 3D commands to Geometry/Fixed Functions
  - Compute commands to Thread Dispatcher
- Builds indirect command buffers if needed

Performance tips:

- Avoid small empty drawIndirects – use ExecuteIndirect or MultiDrawIndirect Extensions!
- Separate and regroup 3D and Compute workloads together to get the most of the hardware

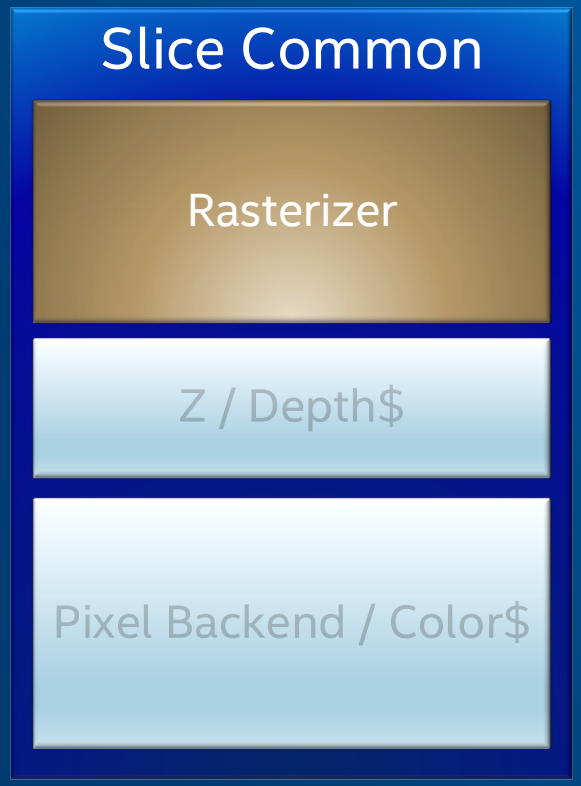
Geometry/Fixed Functions

- Vertex fetch/transform/write in order into Memory
- 1.5x over Gen9 (from 4 to 6 attributes (float4) /clk)

Performance tips:

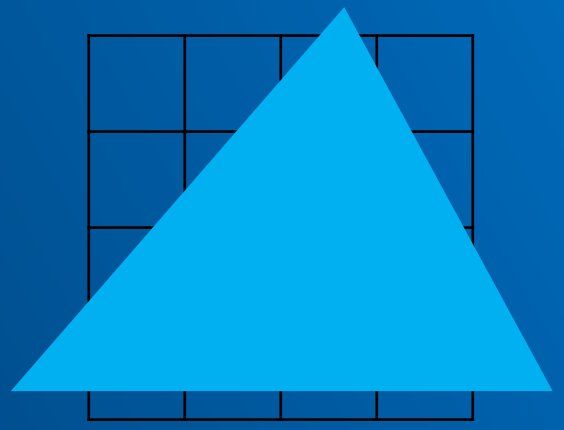
- Pack attributes into float4 when possible
- Hold your Prim. count on a leash
  - CPU occlusion and size culling
  - Level Of Details (LODs)
  - Frustum culling
  - Backface culling

Slice Common Architecture: Rasterizer

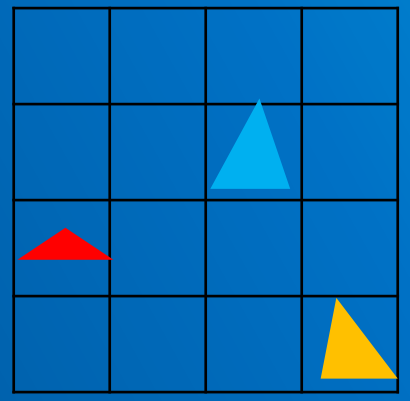


- Handles Clipping, Viewport Scaling, orientation/thin triangles culling, and of course rasterization!
- Hierarchical Rasterization up to 256 pix/clk (16 * 4x4)
  - Determines if each 4x4(span) is fully covered, partially, or void
  - If partially, fine grain raster at 4x4 px/clk

Performance tip: Beware of small triangles - use LODs and impostors:

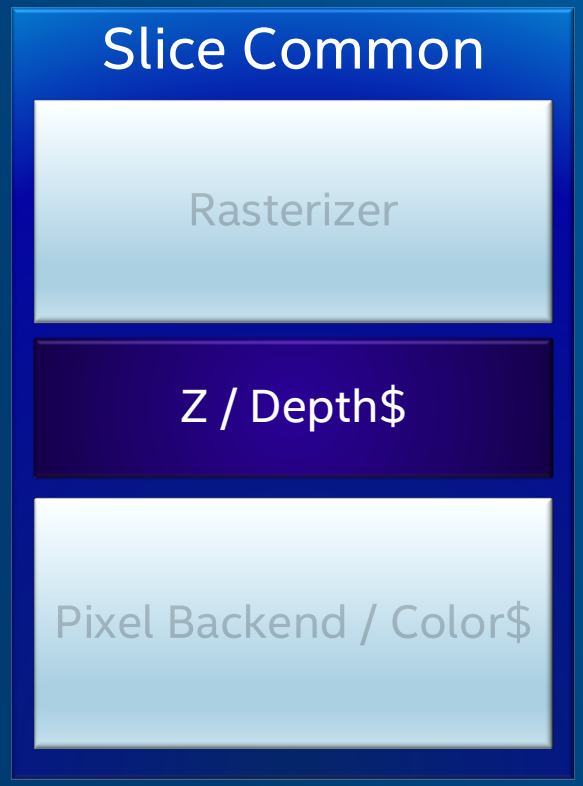


15 px in 1 clk ☺



3 px in 3 clk ☹

Slice Common Architecture: 2 Z/Depth$ Pipes



Handles Hierarchical-Z …

- 2 modes: PlaneZ or Min/Max
- up to 2 x 64 px/clk
- 2 x 12kB cache

… and Intermediate-Z (per px)

- up to 2 x 16 px/clk
- 2 x 32kB cache

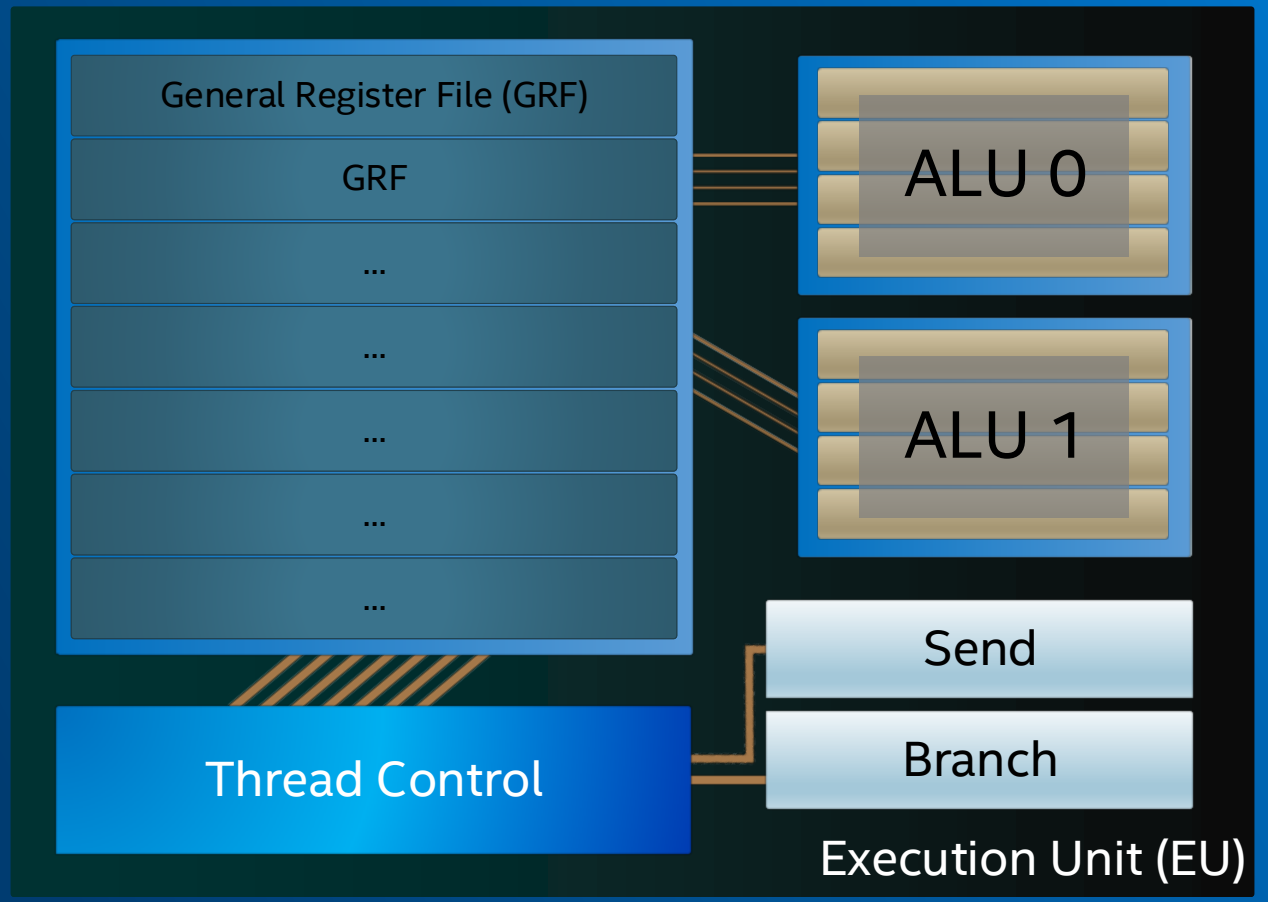
Early-Z test can happen under 1 major condition:

- We don't need the Pixel Shader to know the final depth

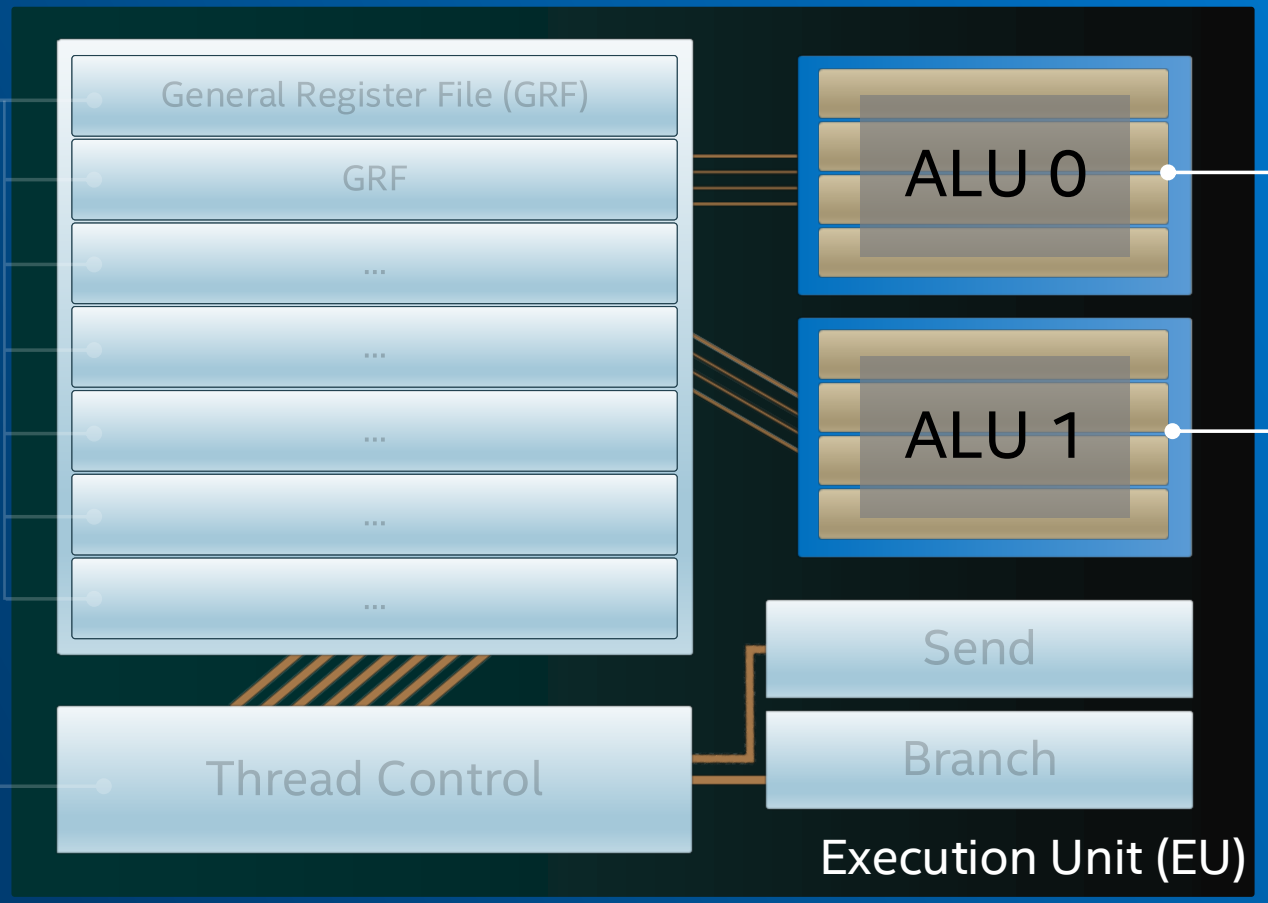
Performance tip: Avoid when possible:

- Discard in Pixel Shader (PS) -> test can happen before but late-Z write
- Writing depth manually in the PS -> both test and write happen late

Gen Basic Building Block: Execution Unit (EU)



Gen Basic Building Block: Execution Unit (EU)



Natively 2 x SIMD4

- Each ALU supports fp. & int math
- 8 x 32-bit or 16 x 16-bit ops/cycle
- ALU1 capable of high throughput transcendental Math
- Min FPU instruction latency is 2 clocks
- FPUs are fully pipelined across threads
- instructions complete every cycle

Full rate: MUL, ADD, MAD, MIN/MAX, CMP

1/4 rate: SQRT/RSQRT, LOG/EXP, TRIG

1/8 rate: POW, DIV

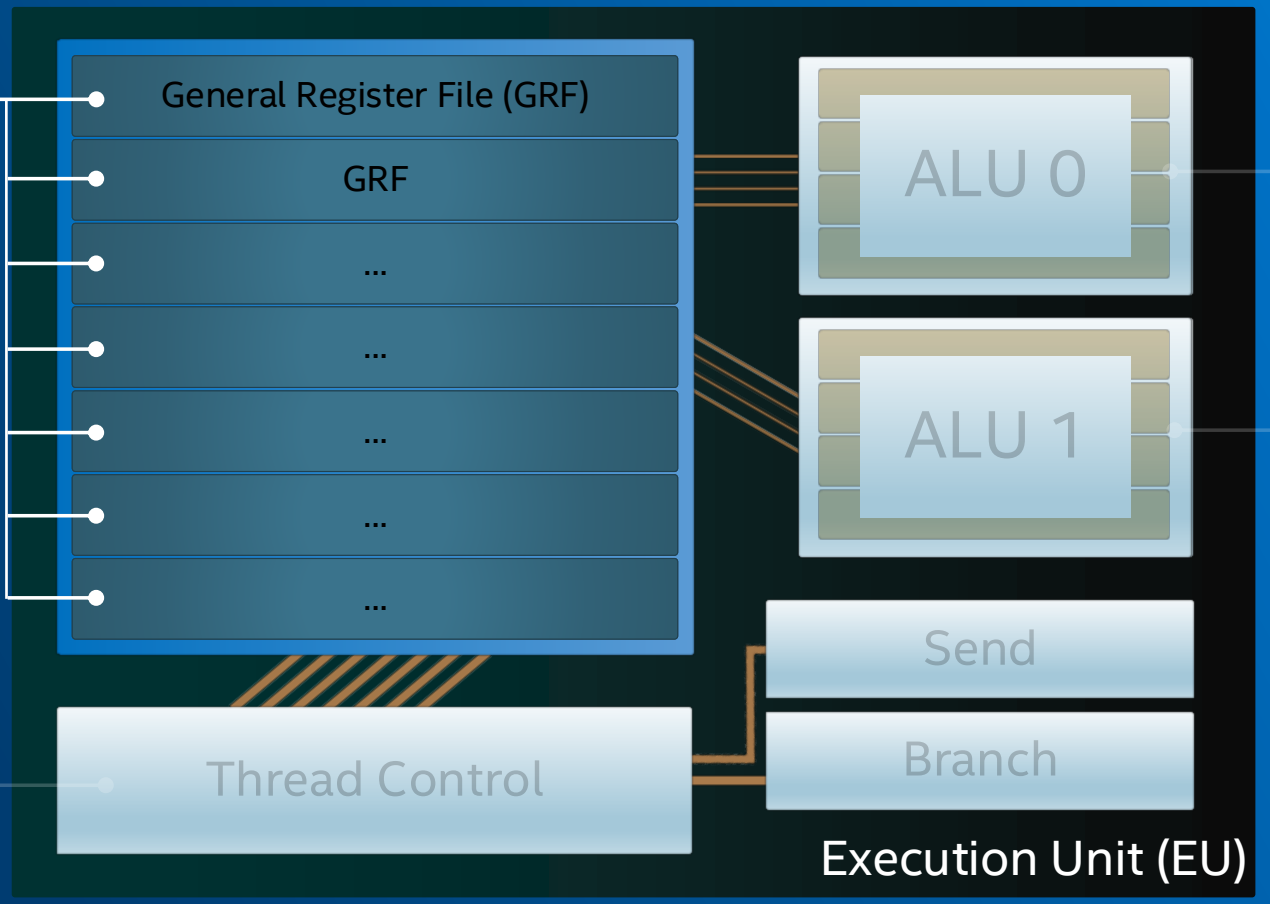
Gen Basic Building Block: Execution Unit (EU)

7 Threads

- Cover latency for Load, LDS, etc.
- Each can run a diff. shader

28kB register space

- 7 x 4 kB
- 128 registers (grf.)/ thread
- grf. are 8 x 32-bit (32B)



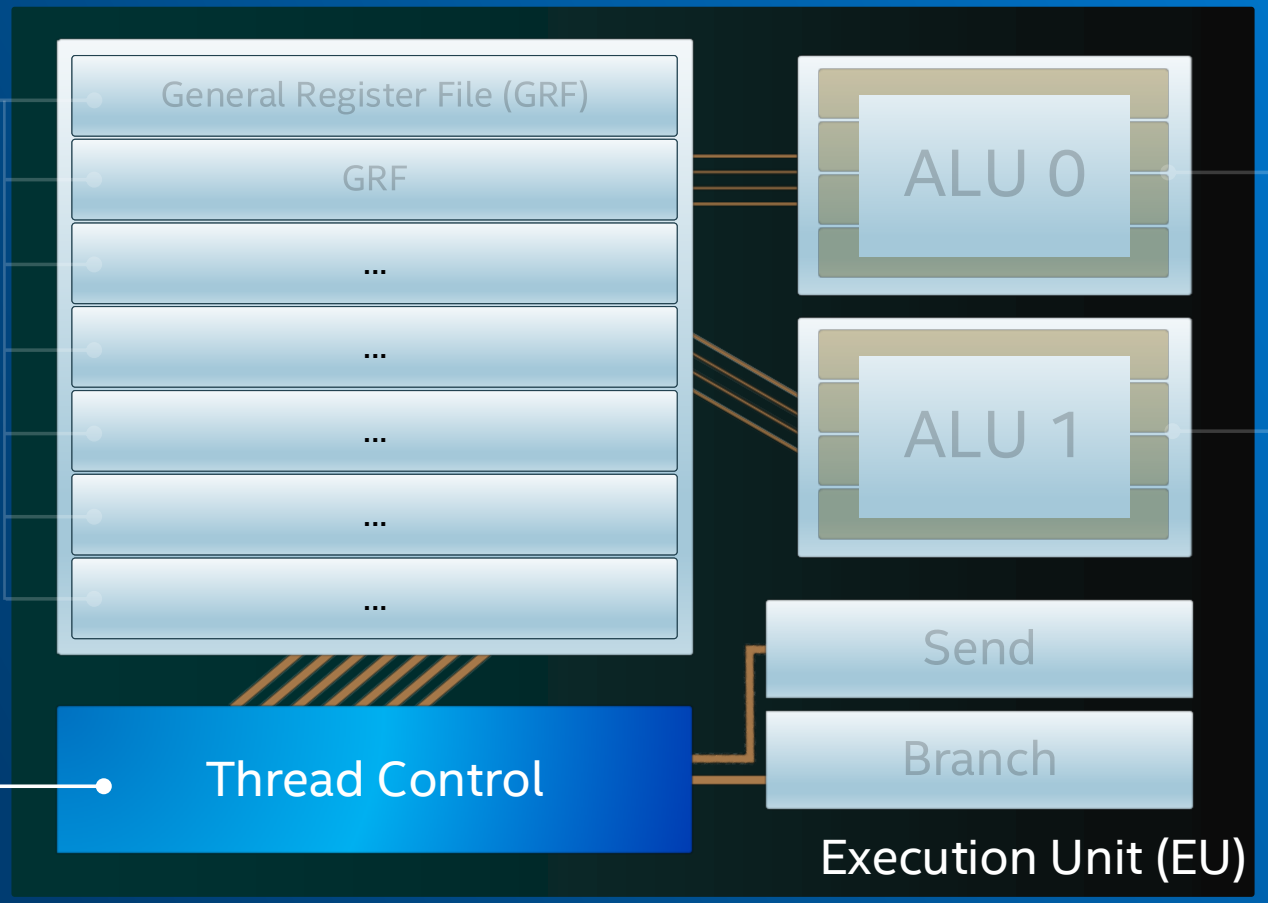
Natively 2 x SIMD4

- Each ALU supports fp. & int math
- 8 x 32-bit or 16 x 16-bit ops/cycle
- ALU1 capable of high throughput transcendental Math
- Min FPU instruction latency is 2 clocks
- FPUs are fully pipelined across threads
- Instructions complete every cycle

Thread Control

- Thread Arbiter picks instructions to run from runnable thread(s)
- Dispatches instructions to functional units
- co-issues up to 4 instructions per cycle

Gen Basic Building Block: Execution Unit (EU)



Thread Control

- Thread Arbiter picks instructions to run from runnable thread(s)
- Dispatches instructions to functional units
- co-issues up to 4 instructions per cycle

EU SIMD Explained

Physically SIMD4x2 but logically support 1/2/4/8/16 or 32-wide instructions

- > SIMD8 instructions pair adjacent GRF.
- SIMD16 would pair 2 physical GRF. to a single logical 64B grf.
- Compiler makes the decision: for 3D workload using either 8 or 16-wide

Performance tip: Reduce register pressure:

- High SIMD/reduced spills
- Better codegen / scheduling

GRF Usage:	<64	64-128	>128
Instruction used	SIMD16 ☺	SIMD8 😐	SIMD8 with Spills ☹

Intel GPA has metrics to help and we now have a standalone Shader Compiler !!

How To Reduce Register Pressure

Don't:

- Branch on constant buffer conditions
- Non uniform access to buffer data
- Excessive variable decl. (esp. arrays)

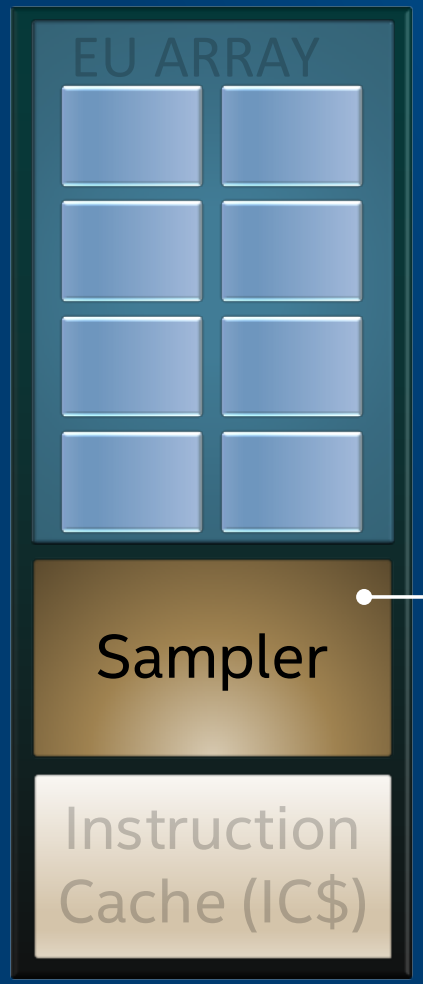
Do's:

- Use Partial precision (min16float)
- Move common code outside branches

GRF Usage:	<64	64-128	>128
Instruction used	SIMD16 ☺	SIMD8 😐	SIMD8 with Spills ☹

A lot more details in the Gen Graphics performance guide:
https://software.intel.com/en-us/documentation/graphics-api-performance-guide-for-intel-processor-graphics-gen9

Subslice: An Array Of 8 EUs With a Sampler



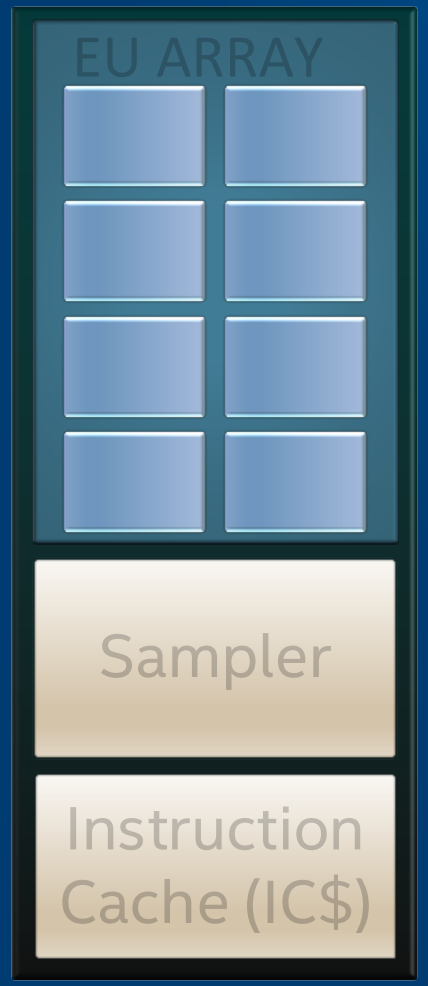
Sampler takes request from the EUs, fetches texels from memory, and returns them to the EUs:

- Handles Texture Decompression
- 4tx/clk bilinear and point < 64b and aniso. 2x 32b
  - >64 formats and aniso. 4x runs at ½ speed
- Multiple levels of cache

Sampler performance tips:

- Increase spatial locality
- Reduce texture state changes
- Use simple filtering modes (aniso. 16x is 1/16th bilinear rate!!)

Subslice: An Array Of 8 EUs



CS thread groups are guaranteed running on a single subslice:

- Gen hardware is able to run 56 hardware threads/subslice
- thread group size == *1024 are an issue*

If we run in SIMD16, we'd have 1024/16 = 64 hardware threads, which is bigger than the 56 available. To compensate, the shader compiler has to use SIMD32.

- This causes 2 issues:
  - Occupancy is low (32 HW threads/56 available = 57%)
  - Potential spill fills, as this doubles the register count required.

Performance tip: 8x8 thread groups is great on all hardware!

2 Subslices grouped together with…

Shared Local Memory (SLM)

- 128kB
- 128B/clk
- ~1/4 latency vs. Gen9

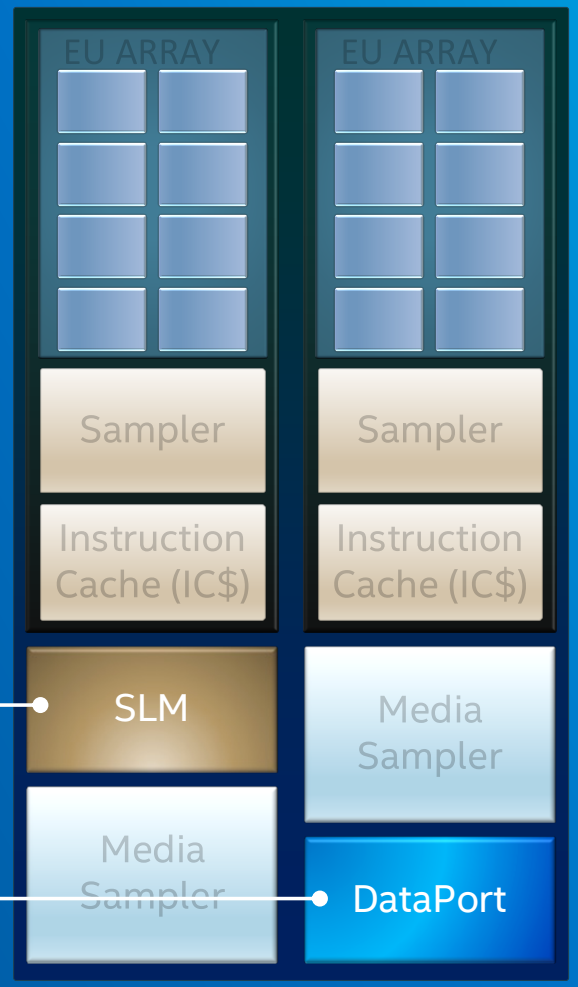
Performance tips:

- Using too much SLM can hurt occupancy
- Regroup memory access to ensure full cache line access (64B)

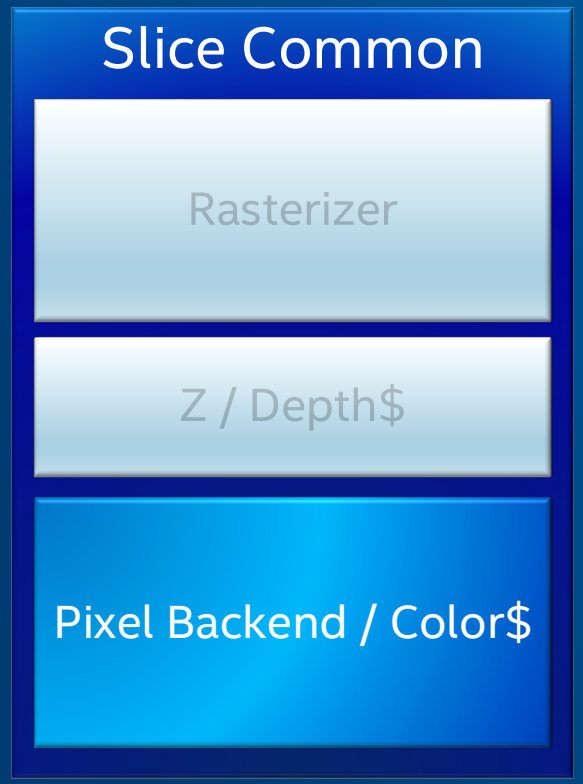
Data Port (LD/ST)

- Handles communication with L3: 64B/clk
- UAV read/writes

Performance tip: Only create UAVs when necessary



Slice Common Architecture: Pixel Backend



Handles color writes and blending to render target

- Writes and blend at 16 px/clk
- Has its own 2 x 32k color cache

2:1 Lossless compression

- Saves bandwidth when writing RT data
- Saves bandwidth when display engine reads displayable RT data
- Saves bandwidth when texture sampler reads intermediate RT data
- MSAA textures are not compressed

Summary – 8 Subslices = 64 EUs

Tile-Based Deferred Rendering (TBDR)

Problem statement: in many cases, bandwidth becomes the limiting factor for low-thermal design point (TDP) platforms.

Solution(s):

- Tile-based rendering is a well-known industry solution for low-power platforms; especially when discarding intermediate data
- Significant amount of bandwidth and execution cycles can be saved for discarded triangles; e.g. frustum or backface culled
- Enable opportunistically for scenarios the benefit the most

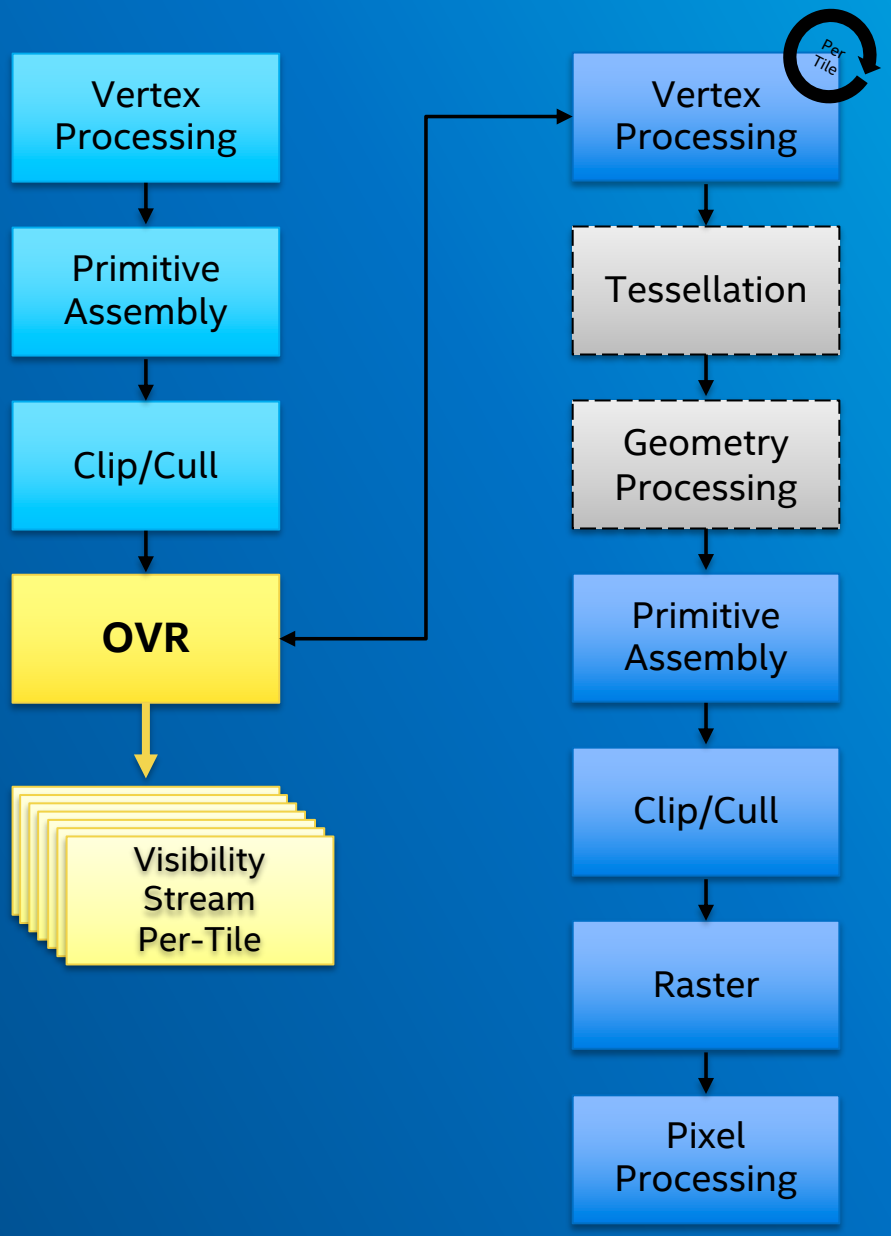
Position-Only Shading Tiled-Based Rendering (PTBR)

Position-Only Shading pipeline generates per-tile visibility stream; using (driver-generated) position-only vertex shader

Render pipeline replays full vertex processing and per-tile pixel processing only for visible primitives

Object Visibility Recorder (OVR) ensures consistency between pipelines

Dynamic enabling of PTBR pipeline allows driver to choose based on optimal conditions; e.g., bandwidth-limited render passes



PTBR Performance Guidelines

Use explicit API render passes and discard:

- VkRenderPass/VkSubpass with VK_ATTACHMENT_STORE_OP_DONT_CARE
- (RS5) ID3D12GraphicsCommandList4::EndRenderPass with D3D12_RENDER_PASS_ENDING_ACCESS_TYPE_DISCARD

During high-bandwidth render passes (e.g., MSAA, blending) avoid:

- Tessellation, Geometry Shading and/or Stream Output, and Compute Shading
- Use per-attribute vertex buffers - position only

GPA: *per-Draw* performance analysis will disable PTBR

Coarse Pixel Shading (CPS): Motivation

- Pixel density increasing
- Performance not scaling at same density as pixel resolution
- Need mechanisms to hit high frame rate and not waste pixel throughput
- Software solutions:
  - Dynamic Resolution Rendering
  - Checkerboard rendering
  - …
- Can also explore hardware solutions…

Intel has highly optimized open source implementations of these available

Topic of these slides

Coarse Pixel Shading Overview

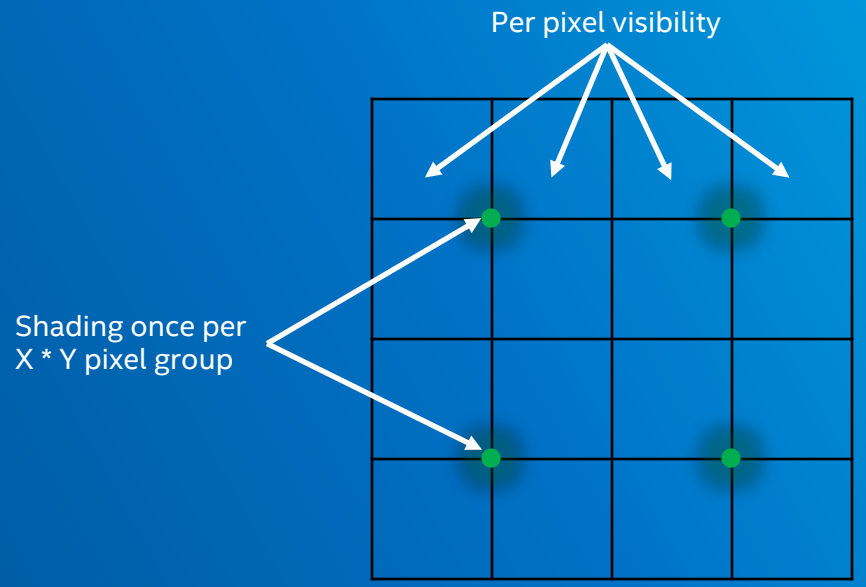
Set Pixel Shading Rate to be:

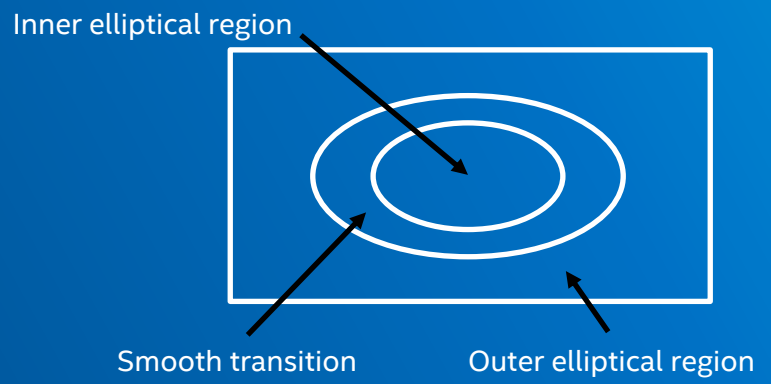
- Floating point value between 1 and 4 in X and Y dimension
- Rounded to integer value to determine actual shading rate

HW support for 2 modes of operation:

- Foveated:
  - Specify a CPS rate for inner and outer elliptical region
  - Specify an inner and outer ellipse, smooth transition between them
- Draw call:
  - Set a state for all future Draw() calls
  - Set multiple times per frame

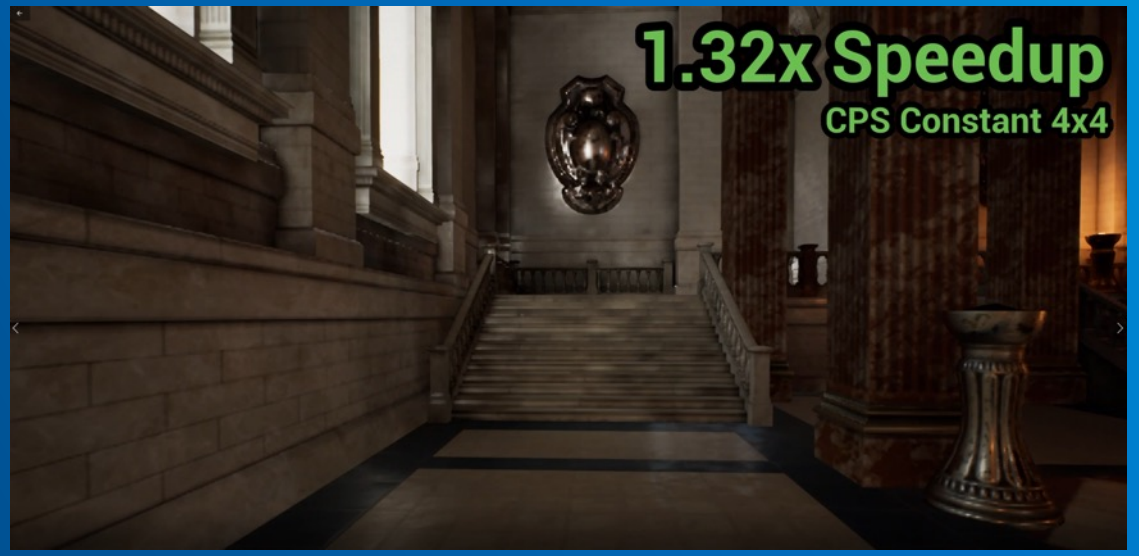
Will be exposed in D3D12 (aka Variable Rate Shading) as well as future Vulkan extensions.





Coarse Pixel Shading (CPS)

- Proof of Concept integration into Unreal Engine SunTemple demo
- CPS rate applied to all geometry when under motion at 2x2 or 4x4 rates
- Early perf gains between ~20-40%
- *Microsoft Variable Rate Shading Talk happening now !*
- *Come see the demo at the Intel Booth!*

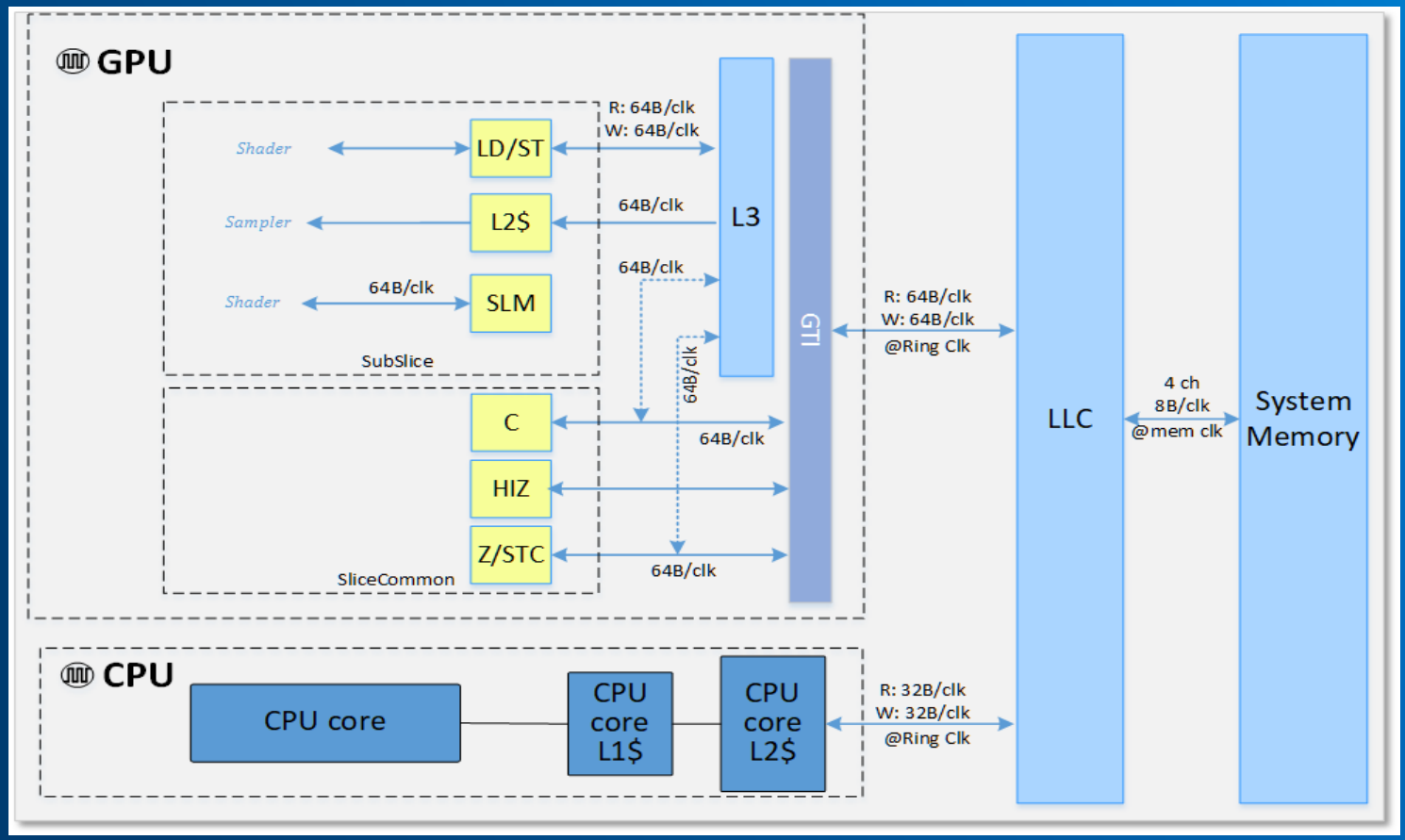


Conclusion

- 64EUs * SIMD4x2 * 2ops/clk = 1024 ops/clk
  - Up to 1 Tflops !!
  - And you now have the keys to make the most of it
- New performance features allowing to get even more from the hardware
  - CPS
  - PTBR
- Biggest gaming announcement in 2019 – Raising the mainstream gaming bar
- A lot more details in the Gen11 whitepaper : https://software.intel.com/en-us/download/graphics-api-architecture-guide-for-intel-processor-graphics-gen11

QUESTIONS?

Gen11 Memory Hierarchy



Backup

Key Peak Metrics Slice Attribute	Gen9 GT2	Gen11 GT2
# of Slices	1	1
# of Sub-Slices	3	8
# of Cores (Eus)	24 (3x8)	64 (8x8)
Single Precision FLOPs per ~~Clk~~Clock (MAD)	384	1024
Half Precision FLOPs per ~~Clk~~Clock (MAD)	768	2048
Int8 IOPs per ~~Clk~~Clock	768	1024
Register File Per Sub-Slice	224KB	224KB
# of Samplers	3	8
Point/Bilinear Texels/~~Clk~~Clock (32bpt)	12	32
Point/Bilinear Texels/~~Clk~~Clock (64bpt)	12	32
Trilinear Texels/~~Clk~~Clock (32bpt)	6	16
Shared Local Memory Total	192KB(=3 x 64KB)*	512KB(=8 x 64KB)
Slice-Common Attributes		
Pixels/~~Clk~~Clock (RGBA8) wo. Alpha Blend	8	16
Pixels/~~Clk~~Clock (RGBA8) w. Alpha Blend	8	16
HiZ Zixels/~~Clk~~Clock	64	128
L3$ Cache	768 KB	3072 KB
Geometry Attributes		
Vertex Attributes Per ~~Clk~~Clock	4	6
Primitive/~~Clk~~Clock (backface Cull – strips)	1	1
Primitive/~~Clk~~Clock (backface Cull – lists)	0.67	0.67
Global Attributes		
GTI Bandwidth (Bytes/~~Clk~~Clock)	R: 64 W: 32	R: 64 W: 64
LLC Configuration	2-8MB	TBD
DRAM Configuration	2x64 LPDDR3/DDR4	4x32 LPDDR4/DDR4
Peak DRAM Data Rate (Mbps)	Up to 2Ch 2400	Up to 2Ch 3733
Max Peak DRAM BW (GBps)	Up to 38.4 GBps	Up to 59.7 GBps
*Note - Gen9 L3$ includes SLM, Data and Fixed Function Data