

# Intel<sup>®</sup> Processor Graphics Gen11 Architecture

Version 1.0

# CONTENTS

---

1	Introduction.....	3
2	Architecture Highlights.....	4
3	System on a Chip (SoC) Architecture.....	6
3.1	Ring Interconnect.....	7
4	The Gen11 Processor Graphics Architecture.....	8
4.1	Global Assets, Media FF and GTI.....	9
4.2	Architecture Configurations, Speeds, and Feeds.....	10
4.3	Slice Architecture.....	11
4.3.1	Geometry.....	12
4.3.2	Subslice Architecture.....	12
4.3.3	Shared Local Memory.....	14
4.3.4	Texture Sampler.....	14
4.3.5	Dataport.....	15
4.4	Slice Common.....	15
4.4.1	Raster.....	15
4.4.2	Depth.....	15
4.4.3	Pixel Dispatch.....	16
4.4.4	Pixel Backend/Blend.....	16
4.4.5	Level-3 Data Cache.....	17
4.5	Memory.....	17
4.5.1	Memory Efficiency Improvements.....	17
4.5.2	Unified Memory Architecture.....	18
4.6	Display Controller.....	19
5	Key Gen11 Technologies.....	21
5.1	Coarse Pixel Shading.....	21
5.2	Position only shading tile based rendering.....	25
5.3	Intel Adaptive Sync.....	27
6	More Information.....	31
7	Notices.....	31

FIGURE 1: INTEL® CORE PROCESSOR, SOC AND ITS RING INTERCONNECT ARCHITECTURE .....	6
FIGURE 2: GEN11 SLICE ARCHITECTURE .....	8
FIGURE 3: KEY PEAK METRICS GEN9 AND GEN11 .....	10
FIGURE 4: THE EXECUTION UNIT (EU).....	13
FIGURE 5: EU LATENCY AND ATOMIC EFFICIENCY .....	14
FIGURE 6: BACK ANNOTATION DEPTH TEST .....	16
FIGURE 7: SOC CHIP LEVEL MEMORY HIERARCHY AND ITS THEORETICAL PEAK BANDWIDTHS.....	19
FIGURE 8: THE CANONICAL CITADEL 1 CPS IMAGE RENDERED AT 2560X1440. ....	21
FIGURE 9: CPS RATES CAN TAKE ON THESE VALUES IN THE X AND Y DIMENSION. ....	23
FIGURE 10: IN THIS IMAGE, THE GEOMETRY WITH RED BOXES ARE IDENTIFIED .....	23
FIGURE 11: BLOCK DIAGRAM OF POSITION ONLY TILE BASED RENDERING (PTBR) .....	25
FIGURE 12: SCREEN TEARING EXAMPLE 1 .....	27
FIGURE 13: ADAPTIVE SYNC SOLUTION FOR FRAME TEARING.....	29

# 1 INTRODUCTION

---

Intel® processor graphics is a proprietary Intel technology that provides graphics, compute, media, and display capabilities for many of Intel's processor system-on-a-chip (SoC) products. This whitepaper focuses on the components of Gen11 architecture. For short hand, in this paper we may use the term Gen11 to refer to just those components. Processor products derived from Intel processor graphics Gen11 will be released in the near future.

## 2 ARCHITECTURE HIGHLIGHTS

---

Intel's on-die integrated processor graphics architecture offers outstanding real time 3D rendering and media performance. In addition, its underlying compute architecture also offers general purpose compute capabilities that delivers up to a teraFLOP performance. The architecture of Intel processor graphics delivers a full complement of high-throughput floating-point and integer compute capabilities, a layered high bandwidth memory hierarchy, and deep integration with on-die CPUs and other on-die SoC devices. While Gen11 will typically ship with 64EUs, there may be different configurations.

### Up to a TERAFL0P Performance

- Gen11 processor graphics is based on Intel's 10nm process utilizing the 3<sup>rd</sup> generation FinFET technology. Additional refinements have been implemented throughout the micro architecture to provide significant performance per watt improvements. Gen11 supports all the major APIs DirectX™\*, OpenGL™\*, Vulkan™\*, OpenCL™\* and Metal™\*.
- Gen11 consists of 64 execution units (EUs) which increases the core compute capability by 2.67x<sup>1</sup> over Gen9. Gen11 addresses the corresponding bandwidth needs by improving compression, increasing L3 cache as well as increasing peak memory bandwidth.

In addition to the raw improvements in compute and memory bandwidth increases, Gen11 introduces key new features that enable higher performance by reducing the amount of redundant work.

### Coarse pixel shading (CPS)

- Coarse pixel shading is a technique that Intel® first described in the seminal [2014 High Performance Graphics Paper "Coarse Pixel Shading."](#) Games today typically provide mechanism to render at lower resolution and then upscale to selected screen resolution to enable playable frame rates with high DPI screens. This may result in artifacts such as aliasing or jaggies resulting in markedly diminished visual quality.
- Coarse pixel shading enables application developers with a new rate control on pixel shading invocations. CPS is better than upscaling because it allows developers to preserve the visibility sampling at the render target resolution while sampling the more slowly varying color values at the coarse pixel rate. By removing the upsampling stage, CPS can improve the overall performance.

### Position Only Shading Tile Based Rendering (PTBR)

- The motivation of tile-based rendering is to reduce memory bandwidth by efficiently managing multiple render passes to data per tile on die.

---

<sup>1</sup> Assumes Gen11 GT2 to Gen9 GT2 at ISO-Frequency

- In order to support tile based rendering, Gen11 adds a parallel geometry pipeline that acts as a tile binning engine. It is used ahead of the render pipeline for visibility binning pre-pass per tile. It loops over geometry per tile and consumes visibility stream for that tile.
- PTBR accomplishes its goal to keep data per tile on die by utilizing the L3 cache which has been enhanced to support color and Z formats. This collapses all the memory reads and writes within the L3 cache thereby reducing the external bandwidth.

### 3 SYSTEM ON A CHIP (SoC) ARCHITECTURE

Intel's® processors are complex SoCs integrating multiple CPU cores, Intel® Gen11 processor graphics and additional fixed functions all on a single shared silicon die. The architecture implements multiple unique clock domains, which have been partitioned as a per-CPU core clock domain, a processor graphics clock domain, and a ring interconnect clock domain. The SoC architecture is designed to be extensible for a range of products and enable efficient wire routing between components within the SoC.

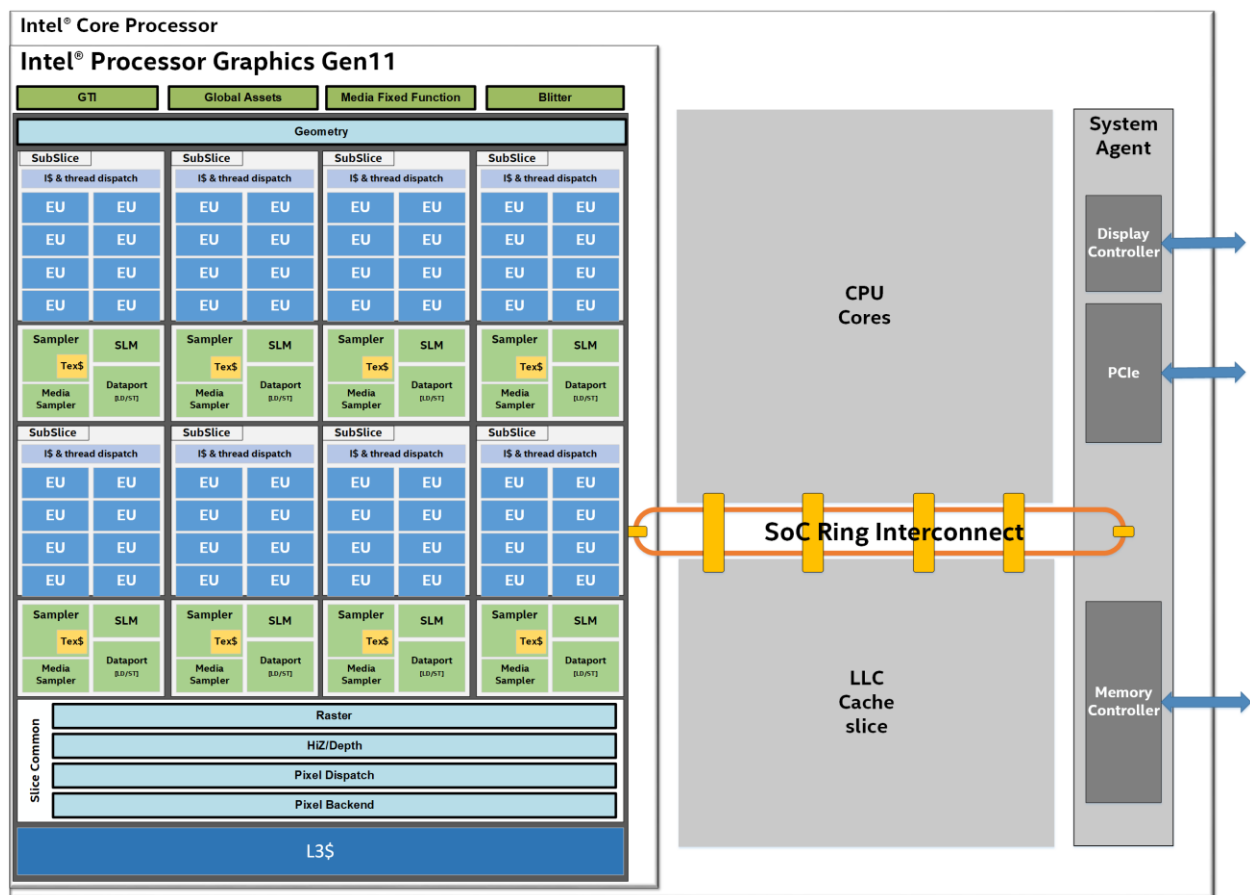


Figure 1: Intel® core processor, SoC and its ring interconnect architecture.

### **3.1 RING INTERCONNECT**

The on-die bus between CPU cores, caches, and Intel® processor graphics is a ring based topology with dedicated local interfaces for each connected “agent” including the Intel processor graphics. A system agent is also connected to the ring which facilitates all off-chip system memory transactions to/from CPU cores and to/from Intel processor graphics. Intel® processors include a shared Last Level Cache (LLC) that is also connected to the ring. In addition, the LLC is also shared with Intel processor graphics.



## 4 THE GEN11 PROCESSOR GRAPHICS ARCHITECTURE

Gen11 architecture brings the next generation of processor graphics based on Intel's latest 10nm process for Core® Processors. Gen11 represents a monolithic design adding significant micro-architectural effort to improve performance per watt efficiency. Gen11 architecture is targeted for modern thin and light mainstream and premium designs.

Gen11 architecture is an evolution over Gen9 with enhancements throughout the architecture to improve performance per flop by removing bottlenecks and improving the efficiency of the pipeline.

Gen11 is a 64EU Architecture supporting 3D rendering, compute, programmable and fixed function media capabilities. Gen11 architecture is split into: Global Assets which contains some fixed function blocks that interface to the rest of the SoC, the Media FF, the 2D Blitter and the Slice. The Slice houses the 3D Fixed Function Geometry, 8 sub-slices containing the EUs, and a slice common that contains the rest of the fixed function blocks supporting the render pipeline and L3 cache. Figure 2 reflects this categorization.

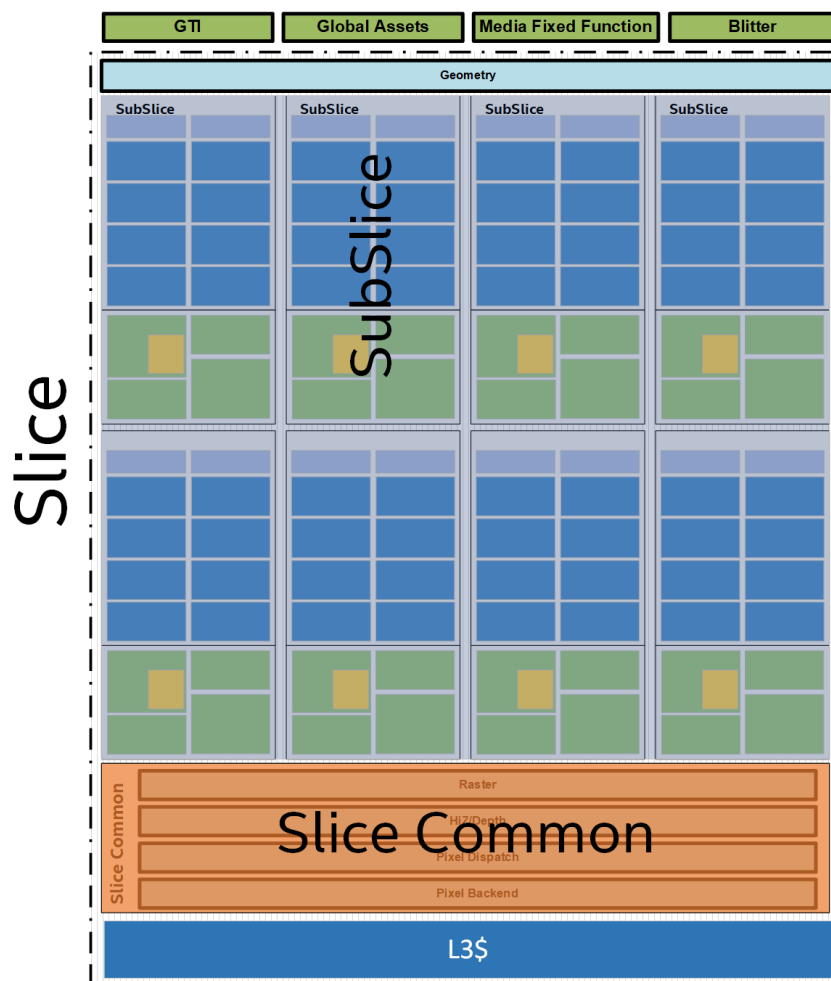


Figure 2: Gen11 Slice Architecture

## 4.1 GLOBAL ASSETS, MEDIA FF AND GTI

Global Assets presents a hardware and software interface from GPU to the rest of the SoC including Power Management.

Graphics Technology Interface (GTI) is the gateway between GPU and the rest of the SoC. The rest of the SoC includes memory hierarchy elements such as the shared LLC memory and system DRAM.

The GTI in Gen11 has been improved to provide higher bandwidth, going from 32B/clock to a 64B/clock interface for write operations. Additionally, the internal queues are sized to handle latency and higher bandwidth in the SoC.

As shown in Figure 2, Media Fixed Functions (Media FF) are located outside of the Slice and the Media Sampler is placed alongside the Texture Sampler inside the slice. The Media Sampler is a co-processor to the EU to execute select media primitives while the Media FF's perform larger tasks (e.g. frame or slice boundary) with independent front-ends including the Multi-format Codec (MFX), Visual Quality Enhancement (VQE) and Scaler and Format Conversion (SFC) units. MFX performs decode and encode of various video codec standards, VQE is a pixel processing pipeline for pre/post-processing, and SFC is an in-line scaler with memory format conversion capabilities.

The Media FF baseline configuration increases in Gen11 to include 2 MFX (up from 1 in Gen9) along with 1 VQE and 1 SFC. The 2 MFX units can be used for better concurrency such as video playback and video encoding simultaneously. Alternatively, workloads can utilize both engines concurrently to increase performance or reduce clock frequencies to increase battery life.

With Gen11, MFX VP9 decode bit depth support is increased up to 10bits which is required for HDR video scenarios. Additionally, both HEVC and VP9 have been improved to support higher quality chroma subsampling of 4:2:2 and 4:4:4 for both HEVC and VP9. 4:2:2 benefits users in high-end video production while 4:4:4 is optimal for screen content such as text/document screen recording and sharing. MFX encoding on Gen11 introduces VP9 support and significantly improves HEVC encoding compression efficiency over Gen9.

Gen11 VQE expands hardware denoise bit depth to 10bits. High dynamic range (HDR) workloads will benefit from new software programmable 3DLUT tables such as HDR2HDR and HDR2SDR tone mapping.

## 4.2 ARCHITECTURE CONFIGURATIONS, SPEEDS, AND FEEDS

The following table presents the theoretical peak throughput of the compute architecture of Intel processor graphics, *aggregated* across the entire graphics product architecture. Values are stated as “per clock cycle”, as final product clock rates were not available at time of this writing. It also shows a comparison to Gen9 GT2

Key Peak Metrics	Gen9 GT2	Gen11 GT2
<b>Slice Attribute</b>		
# of Slices	1	1
# of Sub-Slices	3	8
# of Cores (EUs)	24 (3x8)	64 (8x8)
Single Precision FLOPs per Clock (MAD)	384	1024
Half Precision FLOPs per Clock (MAD)	768	2048
Register File Total	672KB(=3x224KB)	1792KB(=8x224KB)
# of Samplers	3	8
Point/Bilinear Texel's/Clock (32bpt)	12	32
Point/Bilinear Texel's/Clock (64bpt)	12	32
Shared Local Memory Total	192KB(=3 x 64KB)*	512KB(=8 x 64KB)
<b>Slice-Common Attributes</b>		
Pixels/Clock (RGBA8) wo. Alpha Blend	8	16
Pixels/Clock (RGBA8) w. Alpha Blend	8	16
HiZ Zixel's/Clock	64	128
L3\$ Cache	768 KB	3072 KB
<b>Geometry Attributes</b>		
Primitive / Clock (backface Cull – strips)	1	1
Primitive / Clock (backface Cull – lists)	0.67	0.67
<b>Global Attributes</b>		
GTI Bandwidth (Bytes/Clock)	R: 64 W: 32	R: 64 W: 64
LLC Configuration	2-8MB	TBD
DRAM Configuration	2x64 LPDDR3/DDR4	4x32 LPDDR4/DDR4
<b>* Note - Gen9 L3\$ includes SLM</b>		

Figure 3: Key Peak Metrics Gen9 and Gen11

### 4.3 SLICE ARCHITECTURE

For Gen11-based products, 8 SubSlices are aggregated into 1 slice. Thus a single slice aggregates a total of 64 EU. Aside from grouping SubSlices, the Slice integrates additional logic for the geometry, L3 cache, and the Slice Common.

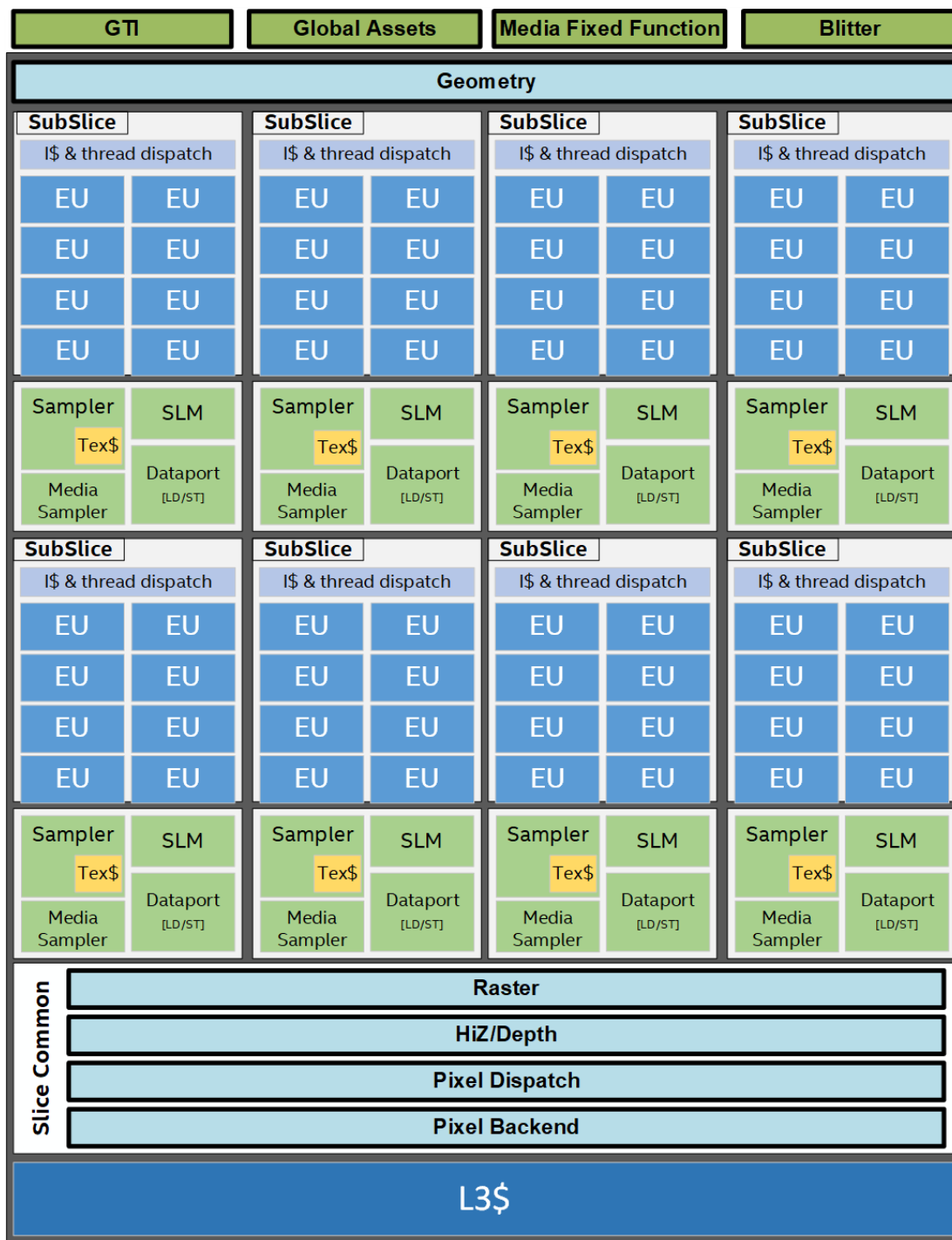


Figure 4: Gen11 detailed block diagram.

### 4.3.1 Geometry

Gen11 3D Geometry Fixed Function contains the typical render front-end that maps to the logical pipeline in DirectX™, Vulkan™, OpenGL™ or Metal™ APIs. Additionally, it includes a Position Only Shading pipeline, or POSh pipeline used to implement Tile-Based Rendering (PTBR). This Section describes the traditional geometry pipeline while section 5.2 describes the POSh pipeline used in tile based rendering.

Vertex fetch (VF), one of the initial stages in the geometry pipe is responsible for fetching vertex data from memory for use in subsequent vertices, reformatting it, and writing the results into an internal buffer. Typically, a vertex consists of more than one attribute, e.g. position, color, normal, texture coordinates, etc. Usage of more vertex attributes has grown with increases in workload complexity. To that end, Gen11 increases the VF input rate from 4 attributes/clock to 6 attributes/clock as well as improves the input data cache efficiency. Another important VF change in Gen11 is the increase in number of draw calls handled concurrently to enable streaming of back to back draw calls. Newer APIs like DX12™\* and Vulkan™\* have significantly reduced the overheads for draw calls, enabling workloads to improve visuals by increasing the number of draw calls per frame.

Gen11 has also made tessellation improvements. It provides up to a 2X increase in the Hull Shader thread dispatch rate as well as further increases the output topology efficiency, especially for patch primitives subject to low tessellation factors.

### 4.3.2 Subslice Architecture

In Gen11 architecture, arrays of EUs are instantiated into a group called a Subslice. For scalability, product architects can choose the number of EUs per subslice. For most Gen11-based products, each subslice contains 8 EUs. Each subslice contains its own local thread dispatcher unit and its own supporting instruction caches. Each Subslice also includes a 3D texture sampler unit, a Media Sampler Unit and a dataport unit.

#### 4.3.2.1 Execution Unit (EU) Architecture

The foundational building block of Gen11 architecture is the execution unit, commonly abbreviated as EU. The architecture of an EU is a combination of simultaneous multi-threading (SMT) and fine-grained interleaved multi-threading (IMT). These EUs are compute processors that drive multiple issue, single instruction, multiple data arithmetic logic units (SIMD ALUs) pipelined across multiple threads, for high-throughput floating-point and integer compute. The fine-grain threaded nature of the EUs ensures continuous streams of ready to execute instructions, while also enabling latency hiding of longer operations such as memory scatter/gather, sampler requests, or other system communication. Depending on the software workload, the hardware threads within an EU may all be executing the same compute kernel code, or each EU thread could be executing code from a completely different compute kernel.

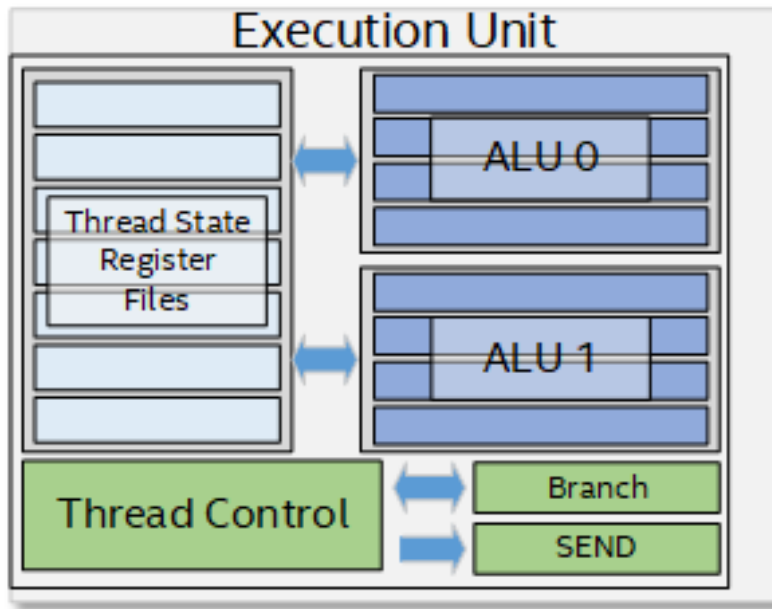


Figure 4: The Execution Unit (EU)

#### 4.3.2.2 SIMD ALUs

In each EU, the primary computation units are a pair of SIMD floating-point units (ALUs). Although called ALUs, they support both floating-point and integer computation. These units can execute up to four 32-bit floating-point (or integer) operations, or up to eight 16-bit floating-point operations. Effectively, each EU can execute 16 FP32 floating point operations per clock [2 ALUs x SIMD-4 x 2 Ops (Add + Mul)] and 32 FP16 floating point operations per clock [2 ALUs x SIMD-8 x 2 Ops (Add + Mul)].

Each EU is multi-threaded to enable latency hiding for long sampler or memory operations. Associated with each EU is 28KB register file (GRF) with 32bytes/register.

As depicted in Figure 4, one of the ALUs support 16-bit and 32-bit integer operations and the other ALUs provides extended math capability to support high-throughput transcendental math functions.

#### 4.3.2.3 SIMD Code Generation for SPMD Programming Models

Compilers for single program multiple data (SPMD) programming models, such as OpenCL™\*, Microsoft DirectX\*\* Compute Shader, OpenGL\*\* Compute, and C++AMP™\*, generate SIMD code to map multiple kernel instances<sup>2</sup> to be executed simultaneously within a given hardware thread. The exact number of kernel instances per-thread is a heuristic driven compiler choice. We refer to this compiler choice as the dominant SIMD-width of the kernel. In OpenCL™\* and

DirectX™ Compute Shader, SIMD-8, SIMD-16, and SIMD-32 are the most common SIMD-width targets.

### 4.3.3 Shared Local Memory

The SLM is a 64KB highly banked data structure accessible from the 8 EUs in the Subslice. The change in architecture is depicted Figure 5. In Gen11 architecture, the SLM and memory access are split such that the one is through the dataport function while the other is accessed directly from the EUs.

The proximity to the EUs provides low latency and higher efficiency since SLM traffic does not interfere with L3/memory access through the dataport or sampler. The SLM is banked to byte granularity allowing high degree of access flexibility from the EUs. This change provides an increase in the overall effective rate of local and global atomics.

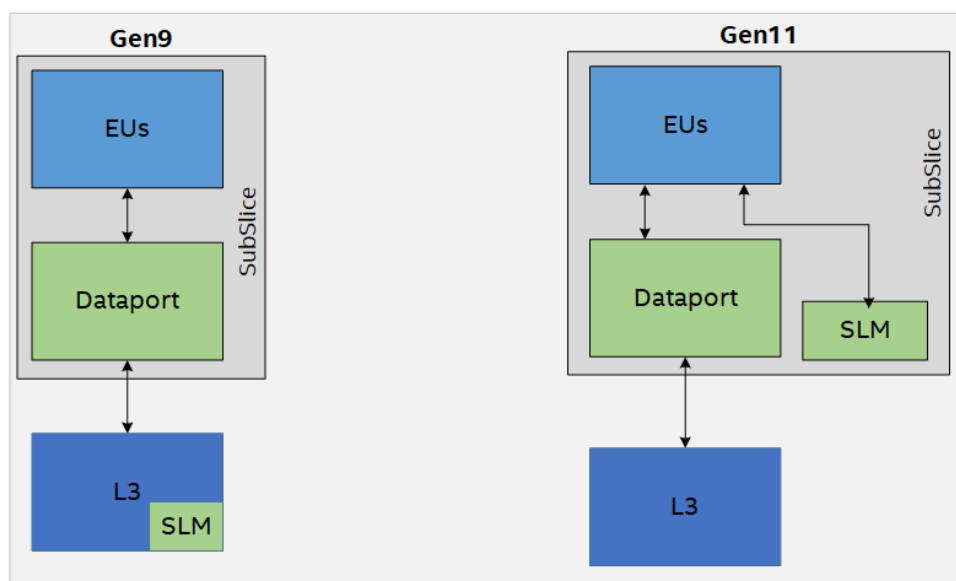


Figure 5: EU latency and atomic efficiency

SPMD programming model constructs such as OpenCL's™ *local* memory space or DirectX™ Compute Shader's *shared* memory space are shared across a single work-group (thread-group). For software kernel instances that use shared local memory, driver runtimes typically map all instances within a given OpenCL™ work-group (or a DirectX™ thread group) to EU threads within a single subslice. Thus all kernel instances within a work-group will share access to the same 64 Kbyte shared local memory partition. Because of this property, an application's accesses to shared local memory should scale with the number of subslices.

### 4.3.4 Texture Sampler

The Texture Sampler is a read-only memory fetch unit that may be used for sampling of texture and image surface types 1D, 2D, 3D, cube, and buffers. The sampler includes a cache, a decompressor, and a filter block. The Texture Sampler supports dynamic decompression of many block compression texture formats such as DirectX™ BC1-BC7, and OpenGL™ ETC,

ETC2, and EAC. Additionally, the texture sampler supports lossless compressed surfaces. The sampler is also compliant with the latest Compute and 3D API's for capability and quality.

Improvements on Gen11 include:

- Sampling rate for anisotropic filtering of 32bit surface formats is increased by 2X for all depths of anisotropy (2X anisotropic filtering is now the same rate as trilinear filtering).
- Sampling rate on volumetric surfaces has been increased by 2X on point sampled 32bit formats as well as bilinear filtered 64bit formats (point-sample is now full-rate of 4ppc for most surface formats).
- Sampling rate for trilinear filtering of 2D surfaces with 64bit surface formats is increased by 2X.

#### **4.3.5 Dataport**

Each Subslice also contains a memory interface unit called the Dataport. The Dataport supports efficient un-typed/typed read/write operations to L3 cache, render cache and other buffers through flexible SIMD scatter/gather operations. To maximize memory bandwidth, the unit dynamically coalesces scattered memory operations into fewer operations over non-duplicated 64-byte cache line requests. For example, a SIMD-16 gather operation against 16 unique offset addresses for 16 32-bit floating-point values, might be coalesced to a single 64-byte read operation if all the addresses fall within a single cache line.

Gen11 reduces L3 cache and memory BW for blend operations that do not require read access to target surfaces.

### **4.4 SLICE COMMON**

#### **4.4.1 Raster**

The Raster block converts polygons to a block of pixels called subspans. Gen11 significantly increases the conversion rate by 16x for 1xAA and by 4X for 4xMSAA.

In addition to normal rasterization, Gen9 supports conservative rasterization which tests pixel for partial coverage and marks it as covered for rasterization. This implementation meets the requirements of tier3 hardware per D3D12 specification and enables advanced rendering algorithms for collision detection, occlusion culling, shadows or visibility detection. Gen11 improves conservative rasterization throughput by about 8x.

Besides supporting rendering primitives, Gen devices also support line rendering which are typically important in workstation applications.

#### **4.4.2 Depth**

The depth test function is used to perform the "Z Buffer" hidden surface removal. The depth test function can be used to discard pixels based on a comparison between the incoming pixel's depth value and the current depth buffer value associated with the pixel.



Depth tests are performed at two levels of granularity, coarse and fine. The coarse tests are performed by HiZ where testing is done on 8x4 pixel block granularity. In addition, the HiZ block supports Fast Clear which allows clearing depth without writing the depth buffer. The test performed at a finer granularity (per pixel, per sample) are done by the Z block.

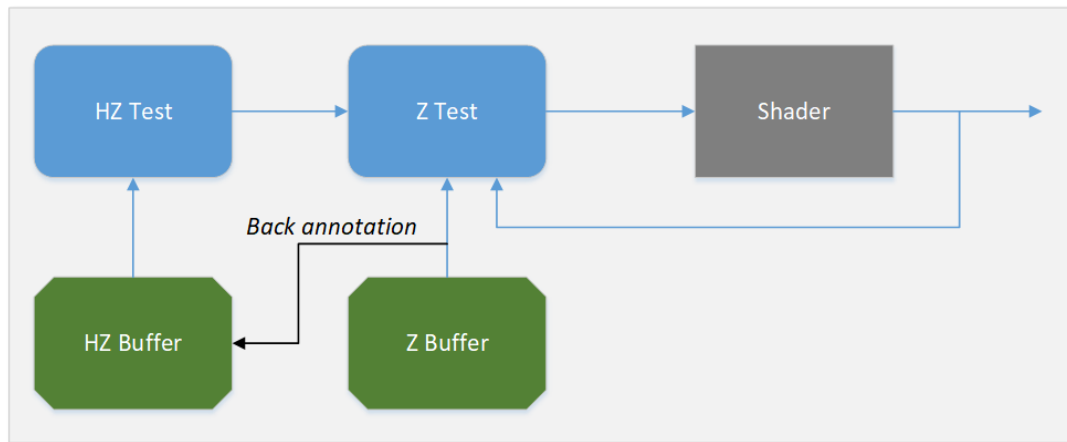


Figure 6: Back annotation depth test

In Gen11, the Z buffer min/max is *back annotated* into HiZ buffer reducing future *nondeterministic* or *ambiguous* tests. When HiZ buffer does not have visibility data till post shader, the resulting tests are nondeterministic in HiZ resulting in Z to per pixel testing. Back annotation allows updating the HiZ buffer with results from Z buffer as shown in figure 6. HiZ test range is narrowed, resulting in coarse testing instead of pixel level for normal rendering or per sample level when MSAA is enabled. Thus, the overall depth test throughput is increased while the corresponding Z memory BW is simultaneously decreased.

#### 4.4.3 Pixel Dispatch

The Pixel Dispatch block accumulates subspans/pixel information and dispatches threads to the execution units. The pixel dispatcher, decides the SIMD width of the thread to be executed, choosing between SIMD8, SIMD16 and SIMD32. Pixel Dispatch chooses this to maximize execution efficiency and utilization of the register file. The block load balances across the shader units and ensures order in which pixels retire from the shader units.

In Gen11, pixel dispatch includes the function of “coarse pixel shader” which is described in detail in Sections 5.1. When CPS is enabled, the coarse pixels generated are packed which reduces the number of pixel shading invocations. The reference or the mapping of a coarse pixel to pixel is maintained until the pixel shader is executed.

#### 4.4.4 Pixel Backend/Blend

The Pixel Backend (PBE) is the last stage of the rendering pipeline which includes the cache to hold the color values. This pipeline stage also handles the color blend functions across several source and destination surface formats. Lossless color compression is handled here as well.

Gen11 exploits use of lower precision in render target formats to reduce power for blending operations.

#### **4.4.5 Level-3 Data Cache**

In Gen11, the L3 data cache capacity has been increased to 3MB. Each application context has flexibility as to how much of the L3 memory structure is allocated in:

- Application L3 data cache
- System buffers for fixed-function pipelines.

For example, 3D rendering contexts often allocate more L3 as system buffers to support their fixed-function pipelines.

All sampler caches and instruction caches are backed by L3 cache. The interface between each Dataport and the L3 data cache enables both read and write of 64 bytes per cycle.

Z, HiZ, Stencil and color buffers may also be backed in L3 specifically when tiling is enabled.

In typical 3D/Compute workloads, partial access is common and occurs in batches and makes ineffective use of memory bandwidth. In Gen11, when accessing memory, L3 cache opportunistically combines partial access of a pair of 32B to a single 64B thereby improving efficiency.

### **4.5 MEMORY**

#### **4.5.1 Memory Efficiency Improvements**

Intel® processor graphics architecture continuously invests in technologies which improve graphic memory efficiency besides improving raw unified memory bandwidth.

Gen9 architecture introduced lossless compression of both render targets and dynamic textures. Games tend to have a lot of render to texture cases where the intermediate rendered buffer is used as a texture in subsequent drawcalls within a frame. As games target higher quality visuals, the bandwidth used by dynamic textures as well as higher resolution becomes increasingly important. Lossless compression aims to mitigate this by taking advantage of the fact that adjacent pixel blocks within a render target vary slowly or are similar which exposes opportunity for compression. Compression yields write bandwidth savings when the data is evicted from L3 cache to memory as well as for read bandwidth savings in case of dynamic textures or alpha blending of surfaces. These improvements results in additional power savings.

Gen11 enables two new optimizations to lossless color compression:

- Support for sRGB surface formats for dynamic textures. Use of gamma corrected color space is important especially as the usage of high dynamic range is increasing.

- The compression algorithm exploits the property that a group of pixels can have the same color when shaded using coarse pixel shading as discussed in section 5.1.

Additionally, memory efficiency is further improved by tile based rendering technology (PTBR) discussed in section 5.2. Fundamentally, it makes the render target and depth buffer stay on chip memory during the render pass while overdraws are collapsed. There are opportunities to discard temporary surfaces by not writing back to memory. PTBR additionally improves sampler access locality and makes on chip cache hierarchy more efficient.

#### **4.5.2 Unified Memory Architecture**

Intel® processor graphics architecture has long pioneered sharing DRAM physical memory with the CPU. This unified memory architecture offers a number of system design, power efficiency, and programmability advantages over PCI Express-hosted discrete memory systems.

The obvious advantage is that shared physical memory enables zero copy buffer transfers between CPUs and Gen11 compute architecture. By zero copy, we mean that no buffer copy is necessary since the physical memory is shared. Moreover, the architecture further augments the performance of such memory sharing with a shared LLC cache. The net effect of this architecture benefits performance, conserves memory footprint, and indirectly conserves system power not spent needlessly copying data. Shared physical memory and zero copy buffer transfers are programmable through the buffer allocation mechanisms in APIs such as Vulkan™\*, OpenCL2™\* and DirectX12™\*.

Gen11 supports LPDDR4 memory technology capable of delivering much higher bandwidth than previous generations. The entire memory sub-system is optimized for low latency and high bandwidth. Gen11 memory sub-system features several optimizations including fabric routing policies, and enhanced memory controller scheduling algorithms which increases overall memory bandwidth efficiency. The memory sub-system also includes QOS features that help balance bandwidth demands from multiple high-bandwidth agents. Figure 7 shows the SoC chip and memory hierarchy.

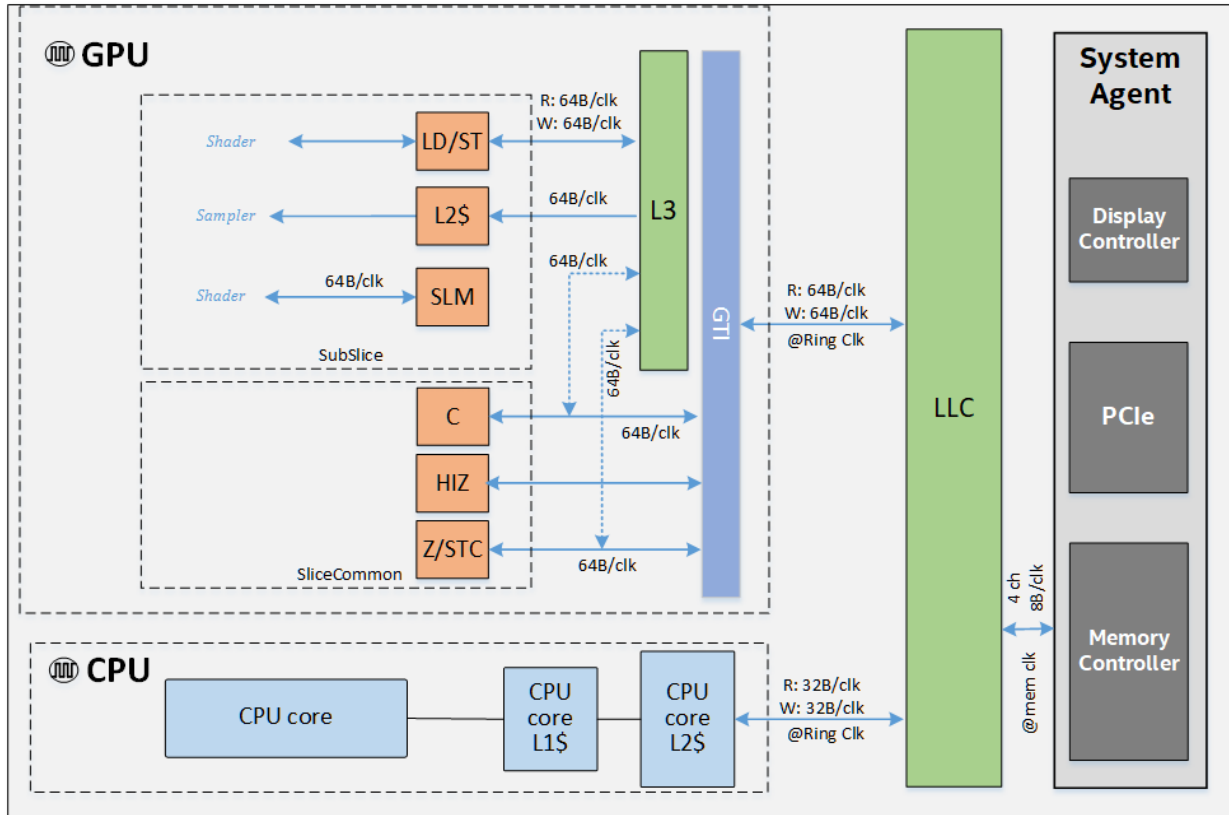


Figure 7: SoC chip level memory hierarchy and its theoretical peak bandwidths

## 4.6 DISPLAY CONTROLLER

The graphics story is not complete without describing the display controller which “paints” the images on the screen. Like Gen 9, the display controller in Gen11 is also integrated in the system agent largely because of the display’s affinity with memory. Over the life of a device, the display controller can consume far more memory bandwidth than any other client. This means that display controller is also one of the most active participants in power management of the SoC. Gen11 display controller has several new features which focus specifically on power management. Panel Self Refresh and Display Context Save and Restore are two of the most prominent features.

Panel Self Refresh is an embedded panel feature available on eDP™\*. PSR panels have a memory copy of the most recently displayed frame which, when the display controller indicates, can be swapped for the display stream. This allows the display controller to stop clocks and go to a low power state. When combined with the Display Controller Save and Restore feature, it can save significant power by additionally shutting off power to the display pipelines. Display Save and Restore saves the context of the display controller into a sustain power SRAM, and then restores the state after the power returns. This feature not only saves

power in the display controller, but also in the entire device as it allows the SoC to reach very deep power states while it is running.

Another key feature of Gen11 platform is the integration of the USB Type-C subsystem. The display controller has dedicated outputs for USB Type-C and DisplayPort™\* alt mode is supported on all USB Type-C outputs. Additionally, output of the display controller can target the Thunderbolt controller, which can also tunnel DisplayPort™\*.

The display controller also has made internal structural changes to improve performance. Chief among these is a shift to a wider pixel processing path in response to the increasing resolutions of monitors. As the number of pixels increases the rate at which the display controller must process them increases as well, so processing pixels two at a time reduces the internal frequency required for the display controller by 50%. Even though this creates a larger display controller, it still comes out ahead as power is more directly impacted by frequency.

Another power feature is display streaming “race to halt”. This feature adds a large buffer on die for the display controller to fetch in to. This allows the display controller to collect pixels very quickly for a large part of the screen, and then shut down the fabric and memory controller while just using the streaming buffer. With the large buffer included in Gen11, the display engine can concentrate its memory accesses into a burst which allows the memory controller to go into power saving modes for longer periods of time than previous designs.

The display controller also supports a compressed memory format generated by the graphics engine to reduce memory bandwidth.

Display upgrades were not limited to power features. The display scaler, which is used for both plane scaling and pipe scaling also got an upgrade going from 7x5 tap filters to 7x7 filters which provides a notable increase in quality. The entire display pixel pipe has gained precision in response to the forthcoming deep color and High Dynamic Range displays. Deep Color refers to those displays that support a color gamut that is larger (sometimes *much larger*) than the standard sRGB color space. High Dynamic Range device additionally increase the range of brightness levels available.

Finally, support for DisplayPort™\* Adaptive Sync has been added for the embedded display. This feature allows the display in combination with a supported monitor to adjust the refresh rate based on the workload. This is usually used to adjust the refresh rate to match the frame rate of the renderer or media stream. This provides a significantly improved user experience over devices that do not support adaptive sync. Note, asynchronous mode typically referred to as Vsync off.

## 5 KEY GEN11 TECHNOLOGIES

### 5.1 COARSE PIXEL SHADING

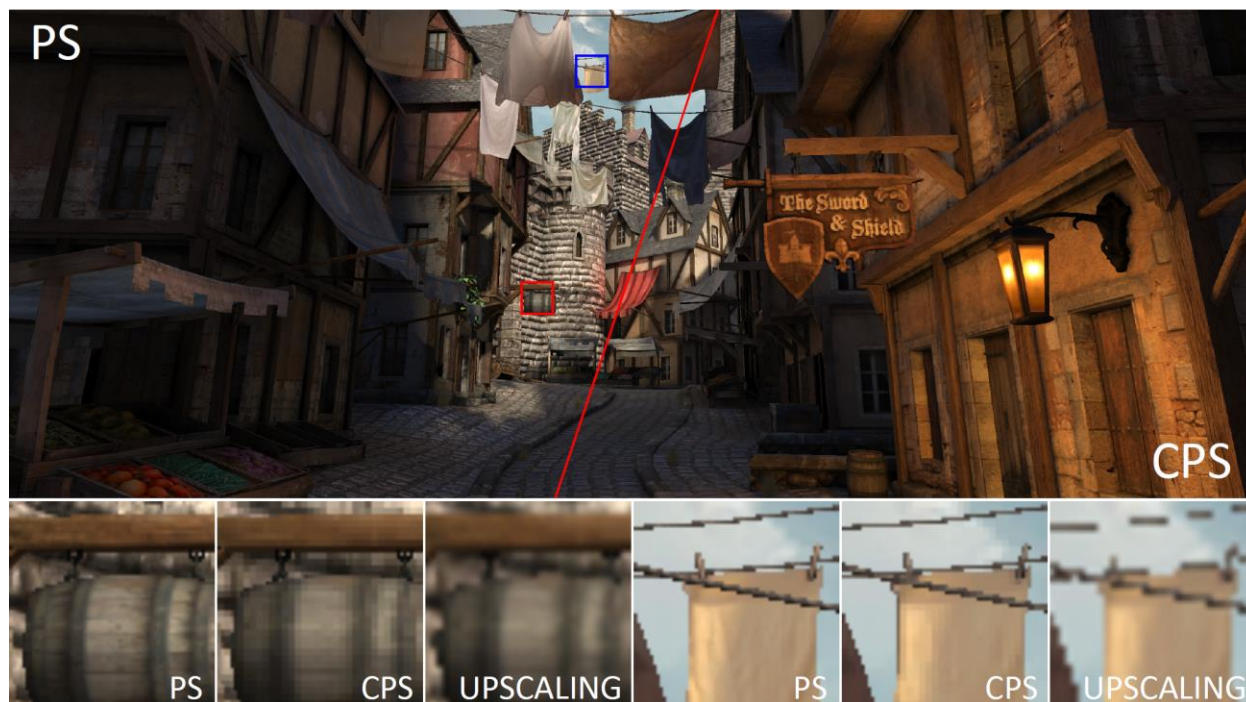


Figure 8: The canonical Citadel 1 CPS image rendered at 2560x1440 with a 1x1 pixel rate on the left and 2x2 CPS shading on the right. While CPS halves the number of shader invocations, there is almost no perceivable difference on a high pixel density display. An up-scaled image with no anti-aliasing applied is also shown for comparison rendered at 1280x720. Reprinted with permission from [Vaidyanathan et al. 2014].

Coarse Pixel Shading (CPS) was pioneered by Intel® in a High-Performance Graphics 2014 research paper [Vaidyanathan et al. 2014]. At that time, we observed that screen resolution was scaling faster than raw pixel throughput in the PC gaming ecosystem and worked to develop techniques to address the problem. Gen11 is Intel's first generation of hardware to support this capability.

CPS allows us to decrease the total amount of work done when rendering portions of the scene where the decrease in shading rate will not be noticed. We can also use this technique to lower the total overall power requirements or hit specific frame rate targets by decreasing the shading resolution while preserving the fidelity of the edges of geometry in the scene.

A common technique to handle pixel bound workloads is to render at a lower resolution then up scaled to the target framebuffer resolution. Coarse pixel shading is a more advanced technique designed to preserve the visibility information at the higher render target resolution

while sampling the more slowly varying color values at the coarse pixel rate. Performance is further improved as there is no up-scaling stage involved in CPS-all coarse pixel resolutions are rendered in a single pass!

We envision the primary use cases for coarse pixel shading include the following scenarios:

1. Objects at a distance from the camera: Objects far away from the camera are less likely to need as much detail as objects nearer to the camera as their details are not as noticeable, particularly in high density scenes common in today's game engines and when the object takes up a small amount of screen real estate.
2. Objects under motion: Objects undergoing motion make it harder to make out details in the shading complexity, therefore it doesn't make sense to spend as much time to render fidelity the user isn't going to see. While this manifests as a blurring of content to the human visual system, the reason is due to object movement and not a post processing effect.
3. Objects undergoing blur or obfuscation: Engines today use any number of post-processing techniques to realize the artistic intent of the game experience, including but not limited to motion blur, bokeh effects, heat caustics, fog and other atmospherics. These effects are going to blur or obscure rendered content.
4. Objects in user's visual periphery: It is known that the foveal region is a region around the center of the eye of about 5 degrees [Guenter et al. 2012]. The human visual system does not make out the details outside of this region, it therefore doesn't make sense to spend time rendering high detail in this region. If we can track the eye we can know where the user is looking and set up our rendering pipeline to spend most of the rendering time in these "foveal" regions, and less in the periphery. Even without a foveated rendering scenario, it can be valuable to concentrate rendering in specific regions of the screen by creating a center point of the user foci and gradually decrease the sampling rate based on the distance from this focus point.
5. Objects known to have slow varying lighting parameters: Some content, for example a stylistic cartoon theme, contains objects that have parameters that vary smoothly across many pixels. These scenarios are perfect candidates for coarse pixel shading.
6. Power savings: When computing fewer color values relative to screen size over a naïve per pixel approach we can save power. This can be important in mobile scenarios where battery life is a factor.
7. Cloud Rendering: Rendering in the cloud is likely to introduce encoding artifacts that dominate the degradation in visual quality at the end user display. Therefore, we can better utilize the server capacity by decreasing the pixel shading rate via CPS but maintain higher quality than if we just reduced the rendering resolution, resulting in an overall higher quality end user experience.

## **How it works**



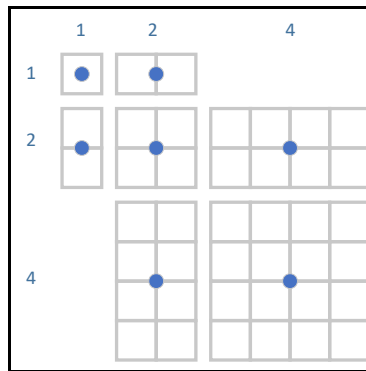


Figure 9: CPS rates can take on these values in the X and Y dimension. The blue dot is the default position for where the pixel shader is evaluated. This value is then propagated to all the pixels that are members of this coarse pixel.

Coarse Pixel Shading works by reducing the number of times the pixel shader executes, thus saving valuable rendering time. To preserve detail along edges, sample coverage and depth continue to be sampled at the render target resolution, only the color value is determined at a reduced frequency then propagated to each pixel that makes up the coarse pixel. A key advantage of coarse pixel shading is that the coarse pixel shading rates (2x2, 1x2, 4x4, etc.) can be processed in a single pass.

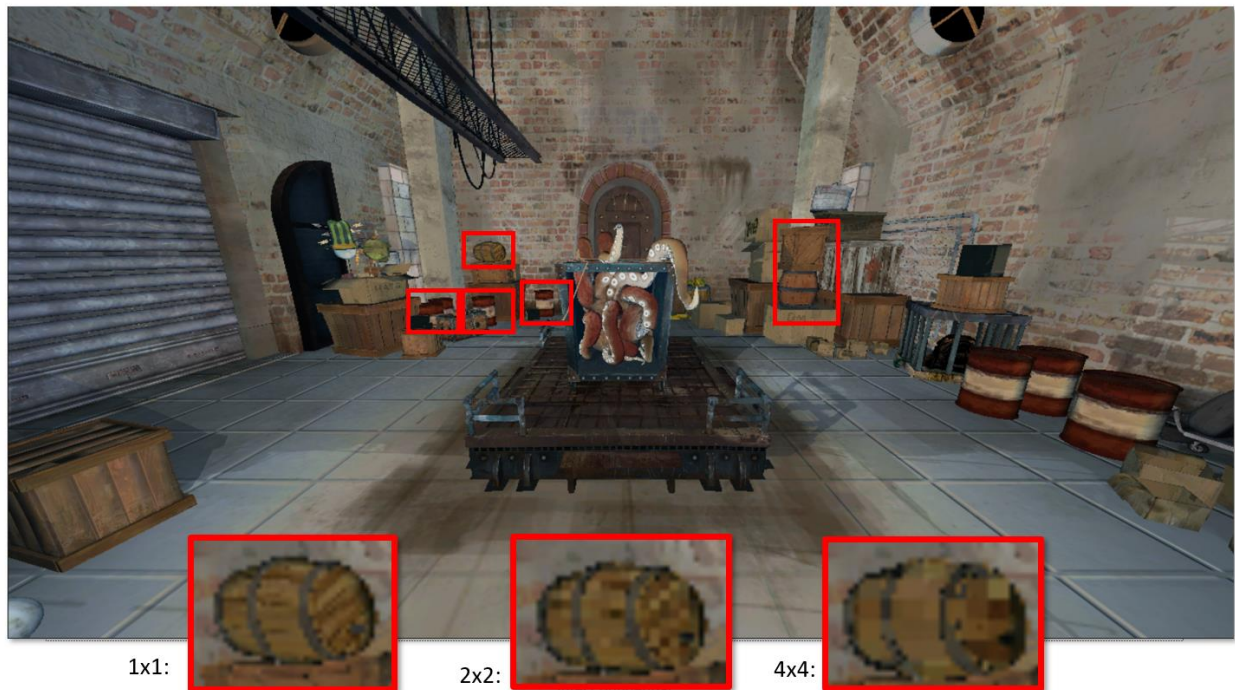


Figure 10: In this image, the geometry with red boxes are identified as being sufficiently far away from the camera, and of minor significance to the overall image quality, so the color shading frequency can be reduced with no discernable impact to the visual quality on a per frame basis and improved overall frame rate over time.



## **Using Coarse Pixel Shading**

Coarse Pixel Shading is easy to integrate into an application and requires no re-authoring or re-export of the content in the application. After initialization of CPS and enabling via the pipeline state description the CPS rate only needs to be specified before a draw call is executed. In addition, we support a foveated scheme described in more detail in the Coarse Pixel Shading Whitepaper available at the Intel Developer Zone [Lake, et al. 2019].

## **Integrating CPS into an application**

Coarse Pixel Shading is a key feature and the best way to take advantage of it is leveraging the integration of the use cases described above into your application. For integration details and sample code see the CPS programming guide available on the Intel Developer Zone.

## **CPS and MSAA**

Coarse Pixel shading and MSAA work together but remain decoupled in Gen11. For example, 4xMSAA with 2x2 coarse pixels means that there are 4 samples within a single regular pixel with 4 regularly sized pixels making up the single coarse pixel. Each pixel continues to have 4 sample points for coverage but will obtain the color value for the coarse pixel from the center (or centroid) of the coarse pixel.

## 5.2 POSITION ONLY SHADING TILE BASED RENDERING

Tile-based rendering technology has been employed to reduce the enormous bandwidth demands of contemporary GPUs. In a tile-based approach, a render target is divided into  $n$  number of rectangular regions – i.e. tiles – that in turn are rendered one at a time. At the most basic level, only the triangles that affect a tile are rendered to limit the working set.

One can assume that these tile-based rendering engines move the color and depth buffer out of main memory to the on-chip tile cache. Since this cache is much closer to the compute engines, far less power is required to access it. The bandwidth advantage comes by implementing the depth/stencil testing and blending entirely via on-chip tile cache.

Moreover, tiling also helps in suppressing the write bandwidth associated with overdrawn pixels. Finally, there are other advantages where the tiling engines have to write only a minimum set of results to memory – i.e. no depth/stencil values, and no multi-sample render target data.

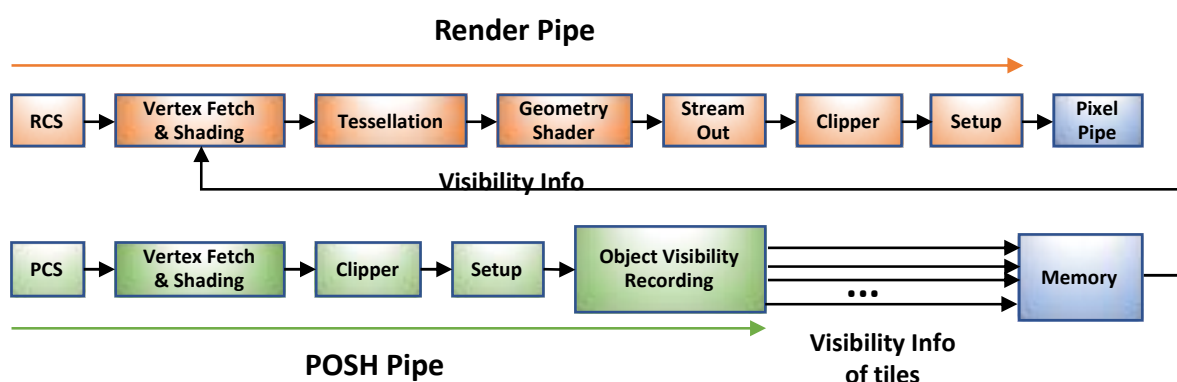


Figure 11: Block diagram of Position only tile based rendering (PTBR)

As depicted in Figure 11, Gen11 adopts position only shading tile-based rendering (PTBR). The PTBR paradigm includes the utilization of two distinct geometry pipes: a new position only shading (POSH) pipe and a typical render pipe.

The POSH pipe executes the position shader in parallel with the main application, but typically generates results much faster as it only shades position attributes and avoids rendering of pixels. The POSH pipe runs ahead and uses the shaded position attribute to compute visibility information for triangles to gauge whether they are culled or not. Object visibility recording unit of the POSH pipe calculates the visibility, compresses the information and records it in memory. Note that the object visibility recording unit can be programmed to record visibility of multiple tiles simultaneously.

The POSH pipe is programmed via the driver to get the visibility for multiple streams in parallel where each stream represents a tile of the whole render target. Next, the POSH pipe loops over

geometry per tile and consumes per tile visibility for deciding whether the geometry is included or excluded which is then subsequently sent to the render pipe. The render pipe is responsible for rendering each of the tiles.

Additionally, the size and the number of tiles are calculated via the driver where it uses information like number of attached render targets and surface formats of the render-targets to generate additional information of bytes/pixel. Tile cache size and bytes/pixel determines the dimensions as well as the number of tiles.

Moreover, tile-based rendering assists Multisample Anti-Aliasing (MSAA) to resolve bandwidth performance. Furthermore, PTBR can also be used to support tiling extensions like discarding depth/multi-sample render-targets, pixel local storage etc. Note that PTBR does not require ISV intervention unlike CPS, however, PTBR can provide benefit to applications which utilize 3D graphics APIs designed for tile-based rendering architectures.

## 5.3 INTEL ADAPTIVE SYNC

Modern day personal computer usages can typically include gaming, video playback, browsing, and office usages such as PowerPoints™\* and Spreadsheets™\*. As a consequence, the various content is created at different frame rates. Traditionally, display panels function at a constant refresh rate (e.g.60Hz), which in most cases, doesn't exactly match with the content frame rates that are unique and varying.

For example, the rendered frame rate for gaming could vary from 30 frames per second (fps) to 120 fps or more. The usual frame rates for video playback are 24 fps, 30 fps, and 60 fps while the frame rates for the other desktop applications might be much lower.

With the V-Sync mode of operation, the display buffer is only refreshed during the vertical blanking interval between frames. If the usage's render framerate is lower than the refresh rate of the display, there will be repeat frames rendered on the display from time to time. This effect degrades the user experience by manifesting as stutter and lag. If the render framerate is higher than the refresh rate of the display, the updated frame must wait until the vertical blanking period to be displayed on the panel. Although this is technically undesirable, it may not be perceivable to the user.

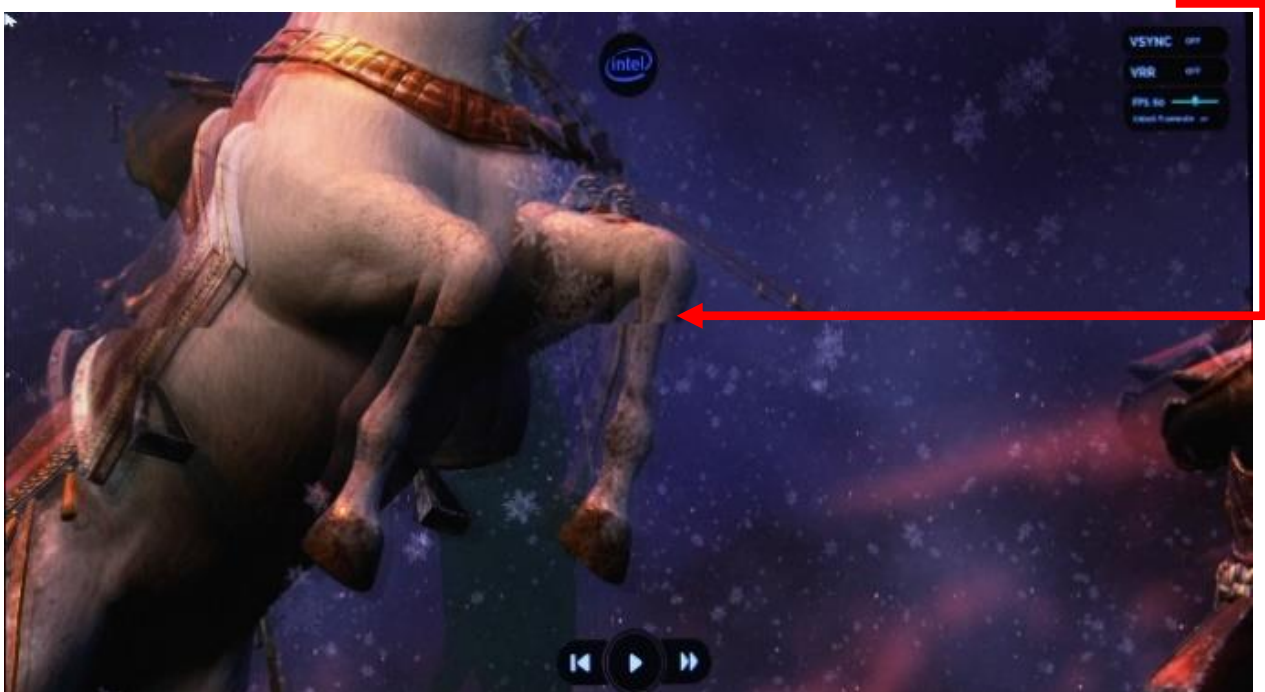


Figure 12: Screen tearing example 1

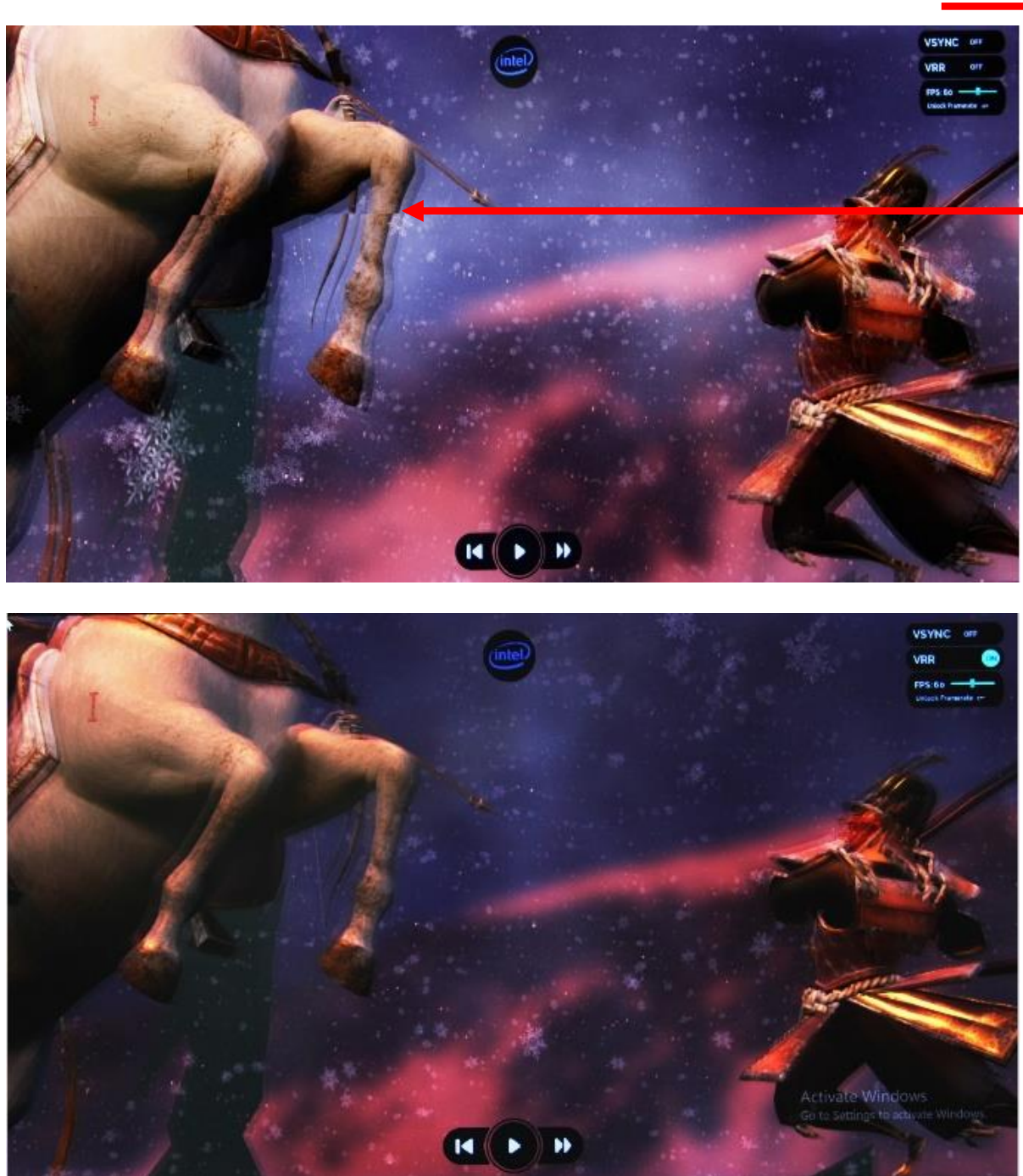
Contemporary display controllers support a mode where the updated frame can immediately be delivered to the display without waiting for the vertical blanking period and hence avoiding the delay but creating artifacts known as “tearing.”. This mode is known as Asynchronous (Async) mode. In Async mode, screen tearing is apparent when the rendered frame rate is not in sync with the display panel refresh rate. Asynchronous mode typically referred to as Vsync off. The display controller tries to always read the latest updated frame from the GPU but keep the display refresh rate constant. During such an operation, when the display controller receives a new frame buffer when a partial write out of the current frame is done, it immediately switches to fetch the data from the new frame buffer for the remaining portion of the current display frame. This creates the unwanted and annoying tearing effect on the display. The following pictures show the screen tearing effect when the Display controller is running in Async mode.

Adaptive sync is a VESA™\* DisplayPort™\* (DP) standard whose function is to dynamically synchronize the display panel refresh rate with the varying GPU render rate. In Gen11, it provides stutter and tear free gaming possible on eDP™\* panels that support the dynamically adjustable refresh rate range.

When the frame render rate by the GPU falls in the supported refresh rate range of the panel, the display controller adapts and syncs the display refresh rate to that of the GPU. Display controller makes this by increasing or decreasing the frame blanking period to either decrease or increase the display refresh rate respectively to match the GPU render rate. This totally alleviates the issue of the screen tear that would occur if the display controller runs on the Async mode of operation.

When the GPU render rate is lower than the minimum refresh rate supported by the panel, display controller's low frame rate compensation feature makes sure to fill in the additional frames to decrease visual artifacts. When the GPU render rate is higher than the maximum refresh rate, the frame refresh on the panel occurs at the maximum refresh rate of the panel.

Figure 13 shows an identical frame comparison of between Async and Adaptive Sync modes of operations respectively.



*Figure 13: Adaptive sync solution for frame tearing*



## 6 MORE INFORMATION

---

[Lake 2019] Adam Lake et al. Coarse Pixel Shading on Intel Graphics. To appear on Intel Developer Zone.

[Vaidyanathan 2014] Karthik Vaidyanathan, Marco Salvi, Robert Toth, Tim Foley, Tomas Akenine-Möller, Jim Nilsson, Jacob Munkberg, Jon Hasselgren, Masamichi Sugihara, Petrik Clarberg, Tomasz Janczak, and Aaron Lefohn. 2014. Coarse Pixel Shading. In High-Performance Graphics. 9–18.

Stephen Junkins 2015, “The Compute Architecture of Intel Processor Graphics Gen9”

## 7 NOTICES

---

Copyright © 20x19 Intel Corporation. All rights reserved

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

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.

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.

This document contains information on products, services and/or processes in development. All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest forecast, schedule, specifications and roadmaps.

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.

Copies of documents which have an order number and are referenced in this document may be obtained by calling 1-800-548-4725 or by visiting [www.intel.com/design/literature.htm](http://www.intel.com/design/literature.htm).

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.

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 information go to <http://www.intel.com/performance>.

Intel, the Intel logo, Iris™, Core™ 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.

Intel® Graphics 4600, Iris™ Graphics, and Iris™ Pro Graphics are available on select systems. Consult your system manufacturer. visit <http://www.intel.com/content/www/us/en/architecture-and-technology/microarchitecture/latest-microarchitecture.html>