SCALABILITY FOR ALL: UNREAL ENGINE 4 ON INTEL

Jeff Rous, Senior Developer Relations Engineer, Intel

Rolando Caloca, Rendering Systems Lead, Epic Games

Agenda

Rationale (Why are we doing this?)

Understanding the Threading Model of Unreal Engine 4

CPU Optimizations

GPU Optimizations

Wrap up

Why Work Together?

Benefits all games that use the engine

UE4 runs on more hardware

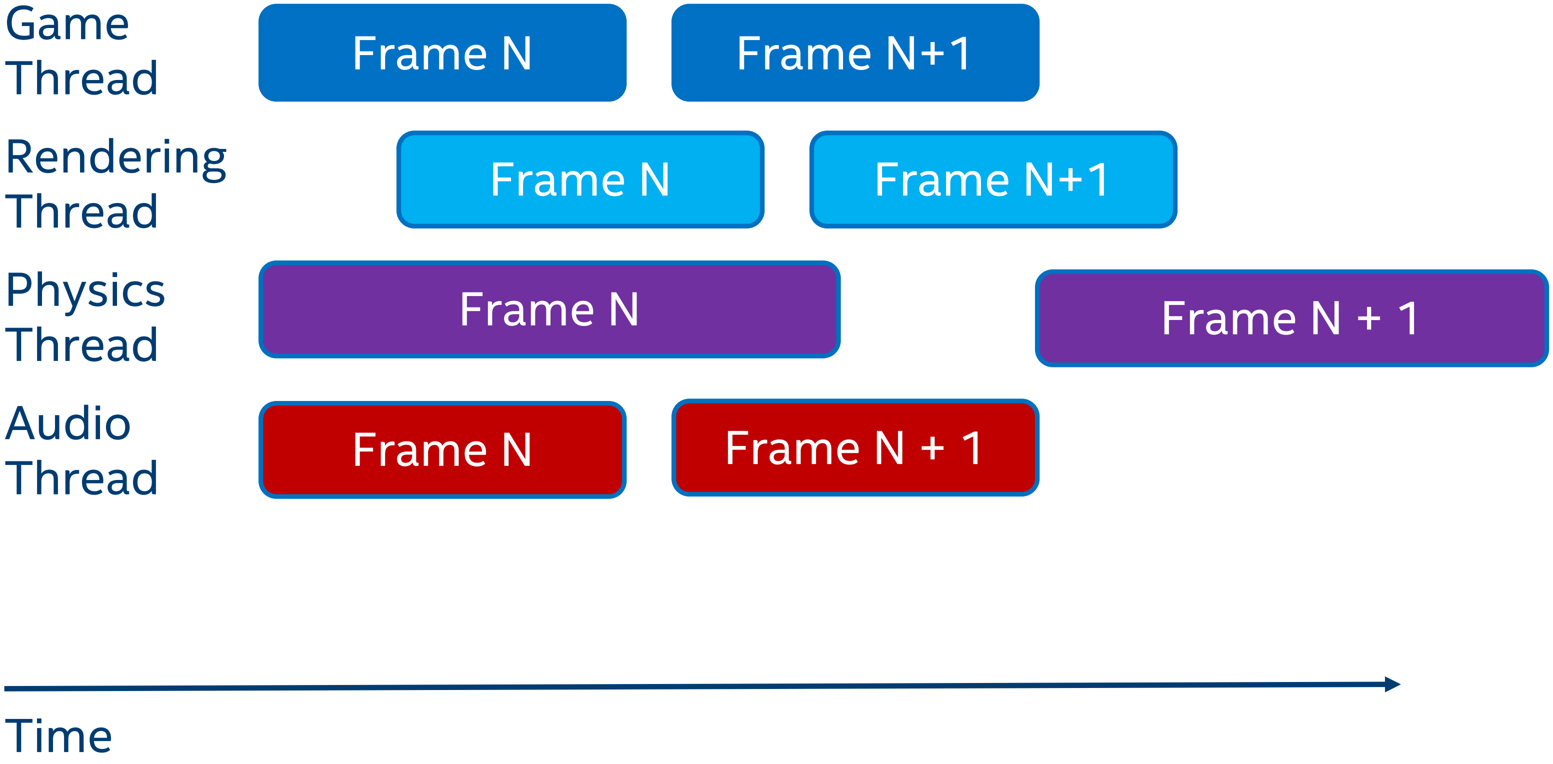
- Intel is 81% PC CPU share as of last Steam survey
- Optimizations help everyone – high end to phone

Common goals

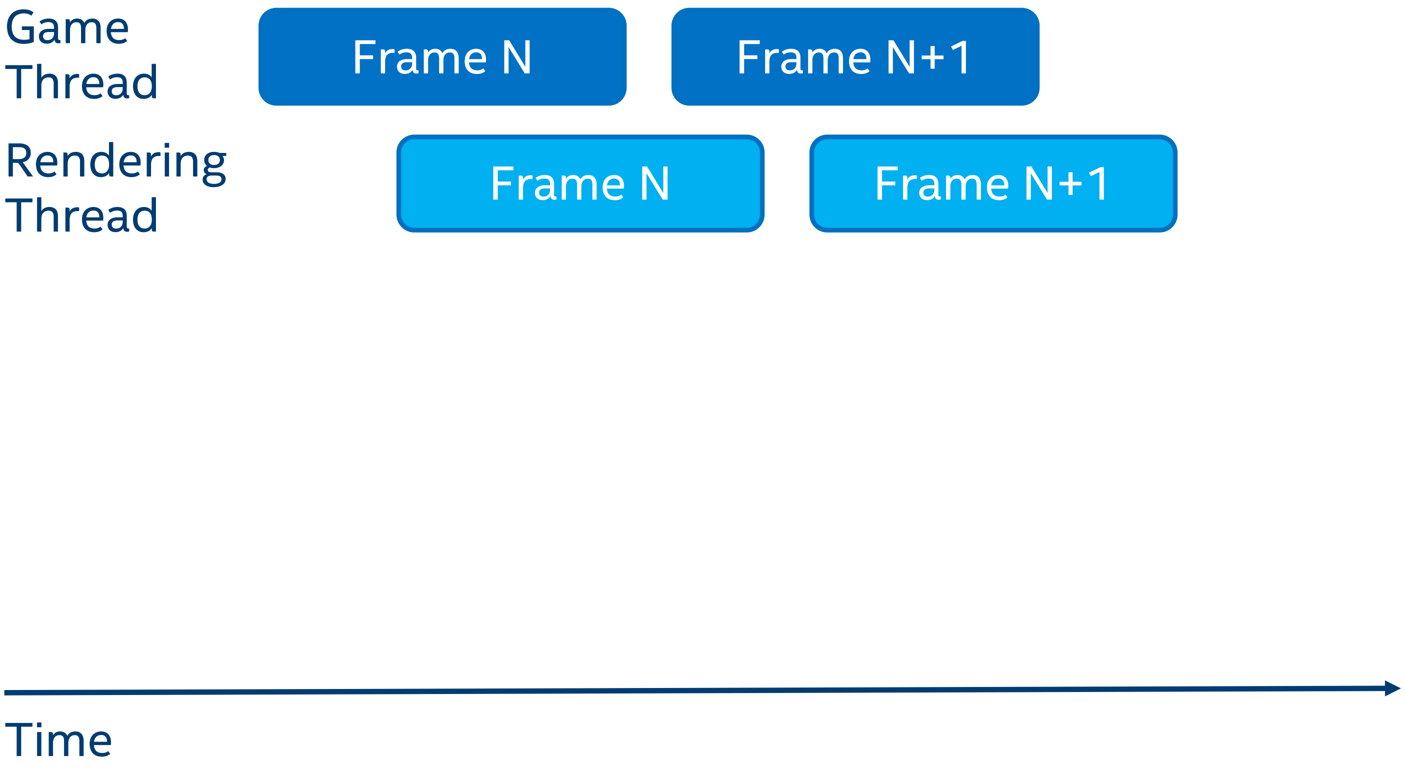
- Scalability means more reach and available market
- Leading edge APIs like DX12 are going to power tomorrow's games

UNDERSTANDING THE THREADING MODEL OF UNREAL ENGINE 4

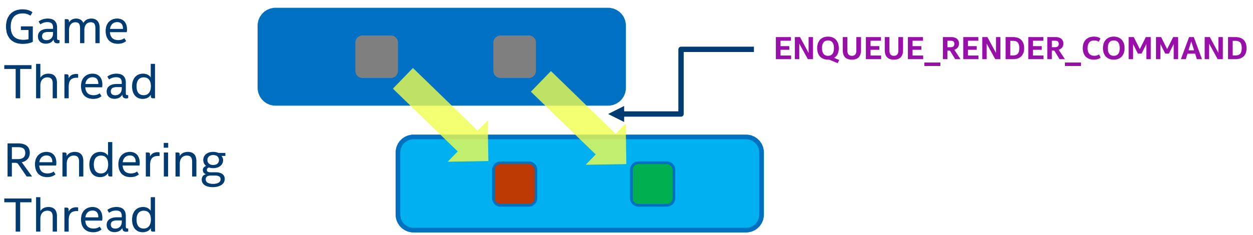
UE4’s Threading Model

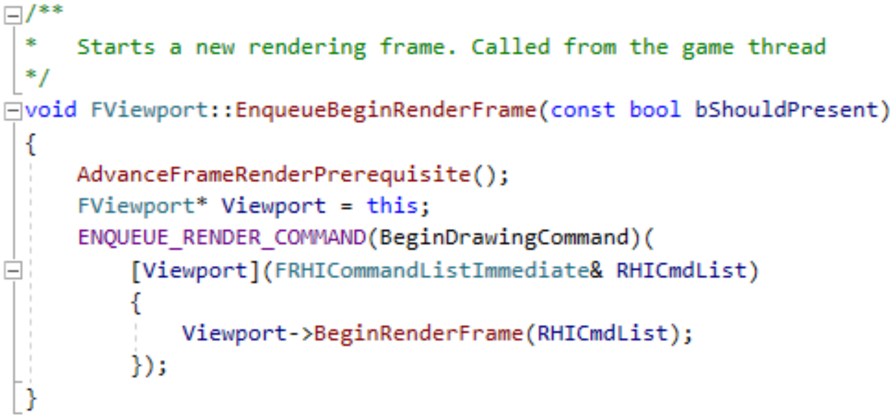


UE4’s Threading Model: Game -> Rendering Thread



UE4’s Threading Model: Game -> Rendering Thread

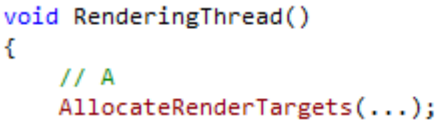



```
/**
*   Starts a new rendering frame. Called from the game thread
*/
void FViewport::EnqueueBeginRenderFrame(const bool bShouldPresent)
{
    AdvanceFrameRenderPrerequisite();
    FViewport* Viewport = this;
    ENQUEUE_RENDER_COMMAND(BeginDrawingCommand)(
        [Viewport](FRHICommandListImmediate& RHICmdList)
        {
            Viewport->BeginRenderFrame(RHICmdList);
        });
}
```

Time

UE4’s Threading Model: Rendering

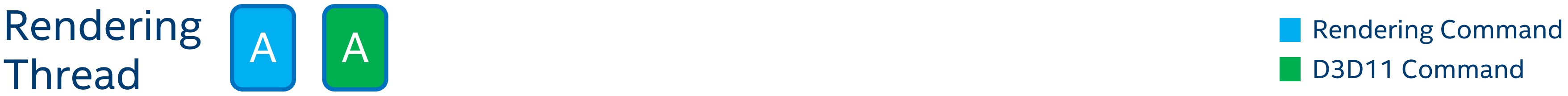
Rendering Thread
A
🟦 Rendering Command



```
void RenderingThread()
{
    // A
    AllocateRenderTargets(...);
```

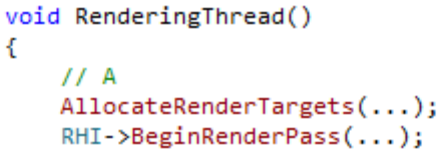
Time

UE4’s Threading Model: Rendering



🟦 Rendering Command

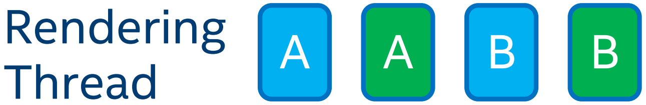
🟩 D3D11 Command



```
void RenderingThread()
{
    // A
    AllocateRenderTargets(...);
    RHI->BeginRenderPass(...);
```

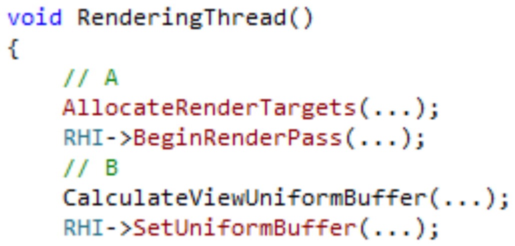
Time

UE4’s Threading Model: Rendering



🟦 Rendering Command

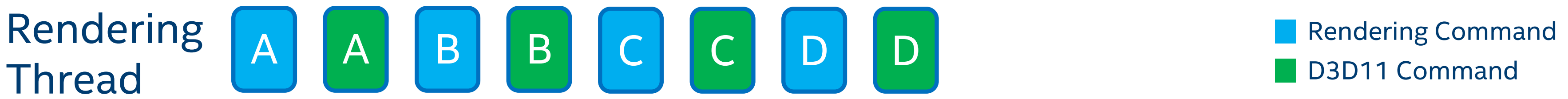
🟩 D3D11 Command

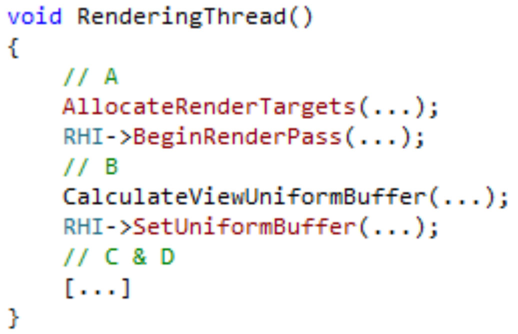


```
void RenderingThread()
{
    // A
    AllocateRenderTargets(...);
    RHI->BeginRenderPass(...);
    // B
    CalculateViewUniformBuffer(...);
    RHI->SetUniformBuffer(...);
```

Time

UE4’s Threading Model: Rendering

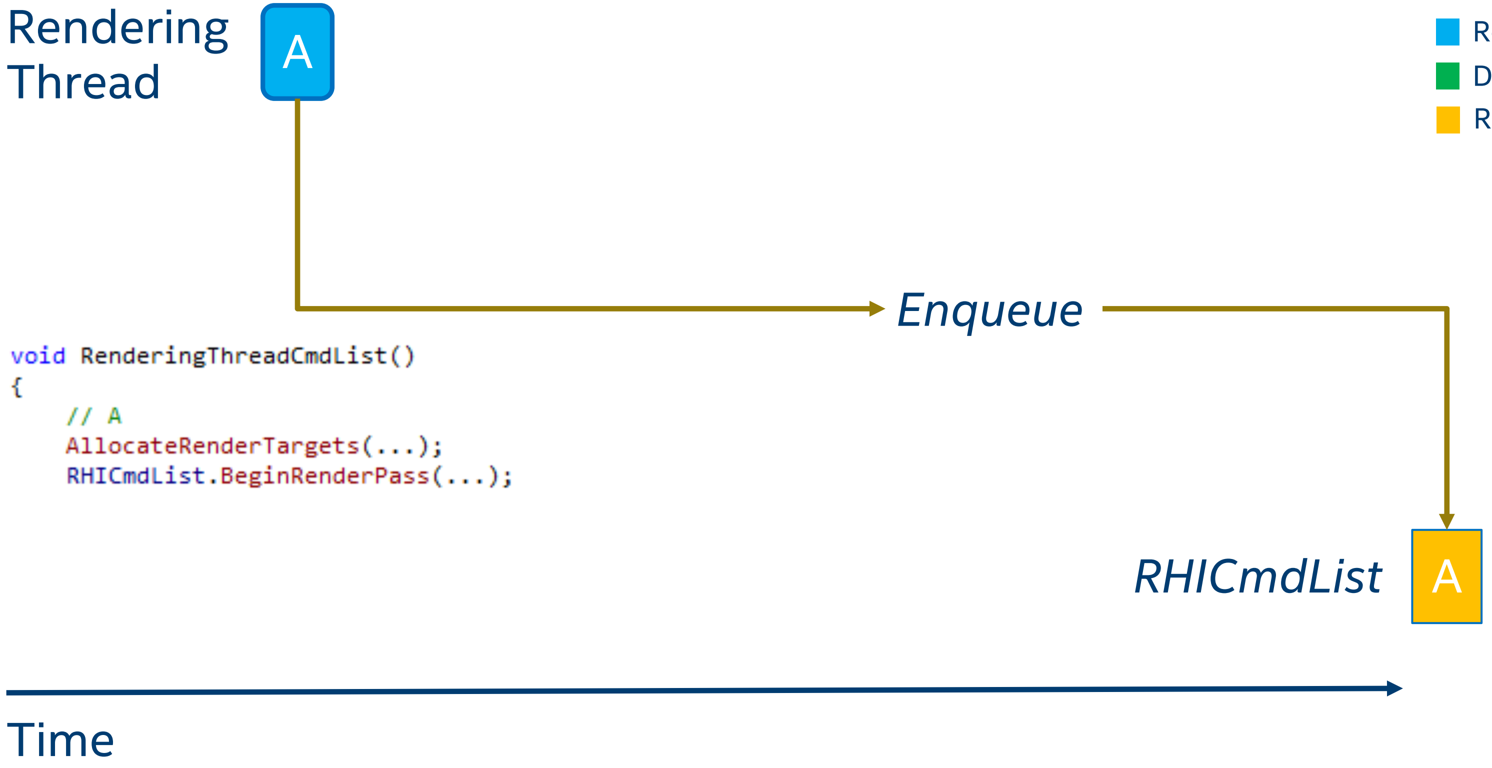




```
void RenderingThread()
{
    // A
    AllocateRenderTargets(...);
    RHI->BeginRenderPass(...);
    // B
    CalculateViewUniformBuffer(...);
    RHI->SetUniformBuffer(...);
    // C & D
    [...]
}
```

Time

UE4’s Threading Model: RHI Command List



Rendering Thread

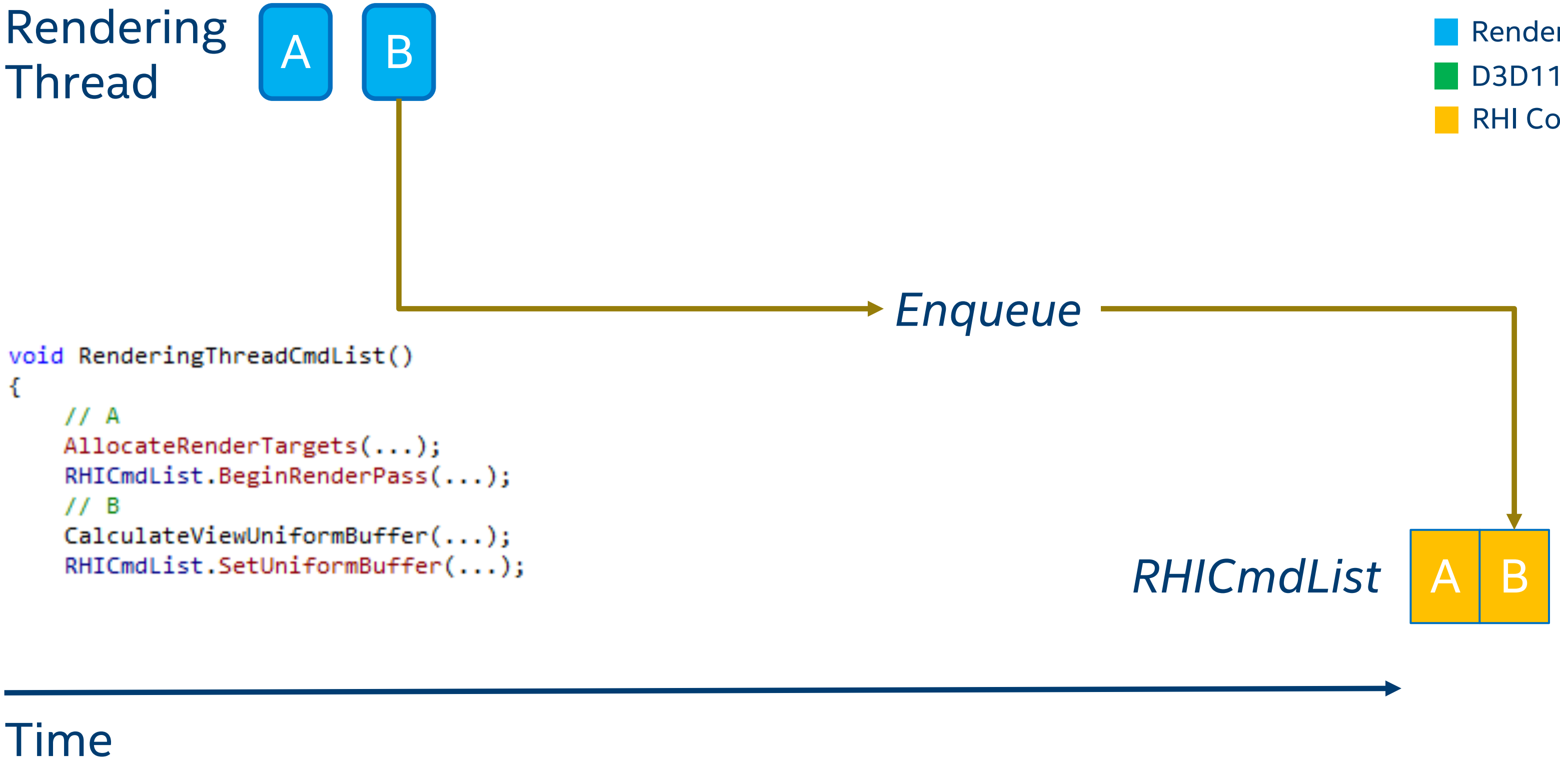
```
void RenderingThreadCmdList()
{
    // A
    AllocateRenderTargets(...);
    RHICmdList.BeginRenderPass(...);
```

🟦 Rendering Command

🟩 D3D11 Command

🟨 RHI Command

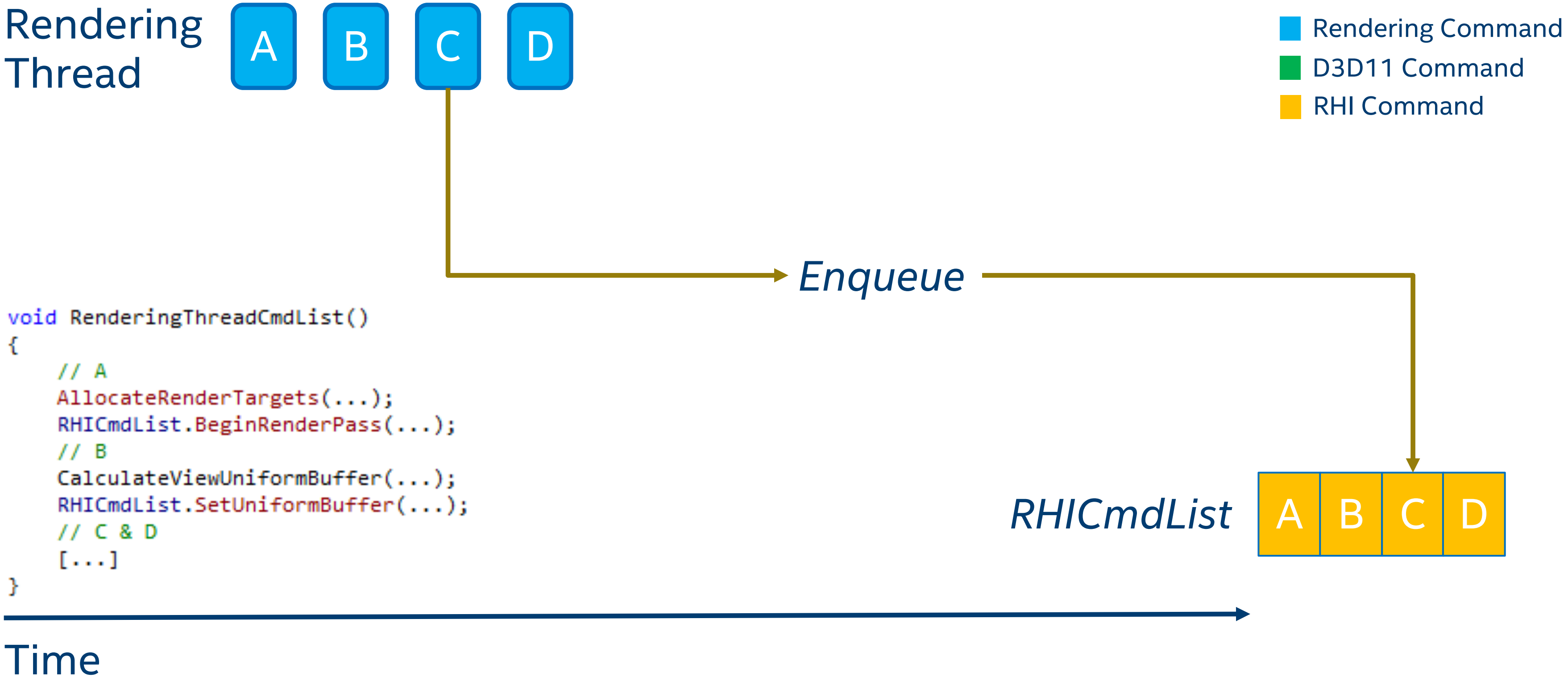
UE4's Threading Model: RHI Command List



```
void RenderingThreadCmdList()
{
    // A
    AllocateRenderTargets(...);
    RHICmdList.BeginRenderPass(...);
    // B
    CalculateViewUniformBuffer(...);
    RHICmdList.SetUniformBuffer(...);
```

🟦 Rendering Command
🟩 D3D11 Command
🟨 RHI Command

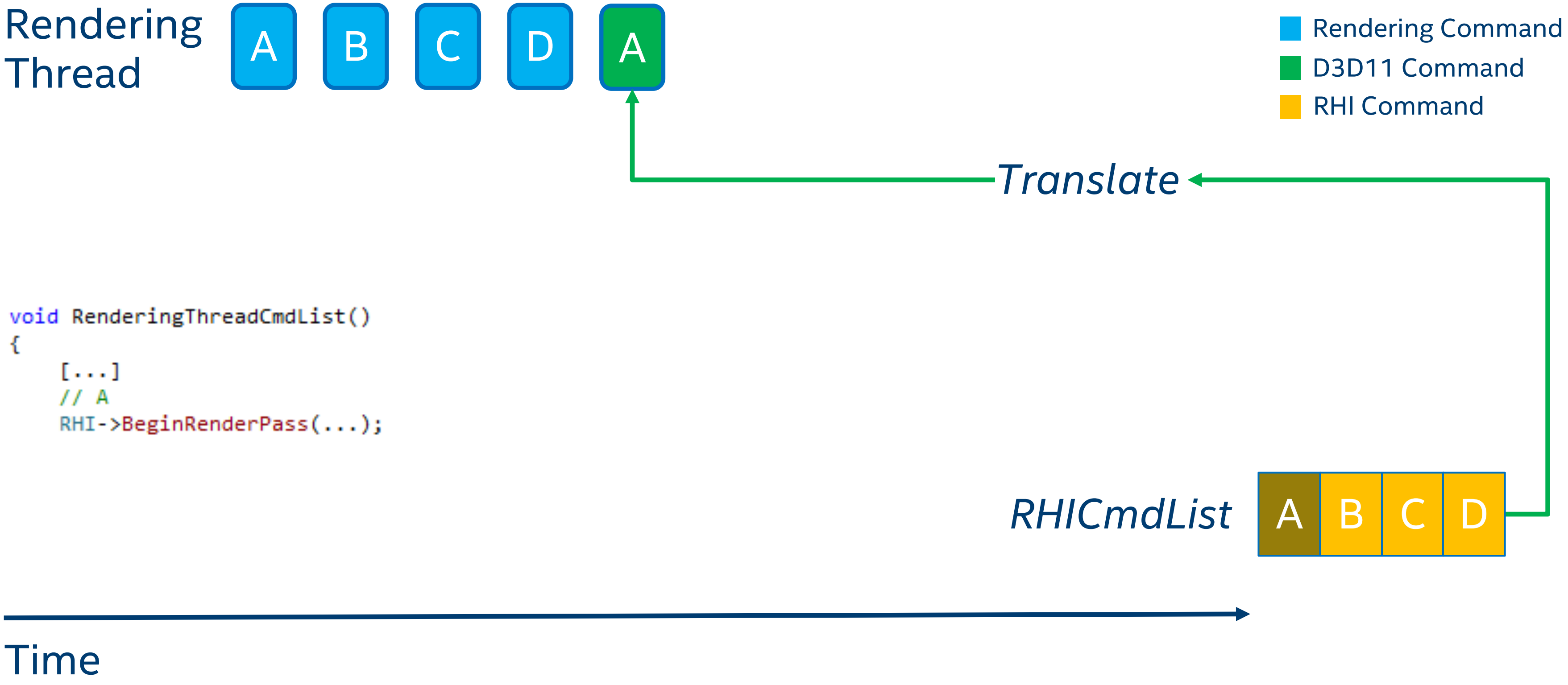
UE4’s Threading Model: RHI Command List



```
void RenderingThreadCmdList()
{
    // A
    AllocateRenderTargets(...);
    RHICmdList.BeginRenderPass(...);
    // B
    CalculateViewUniformBuffer(...);
    RHICmdList.SetUniformBuffer(...);
    // C & D
    [...]
}
```

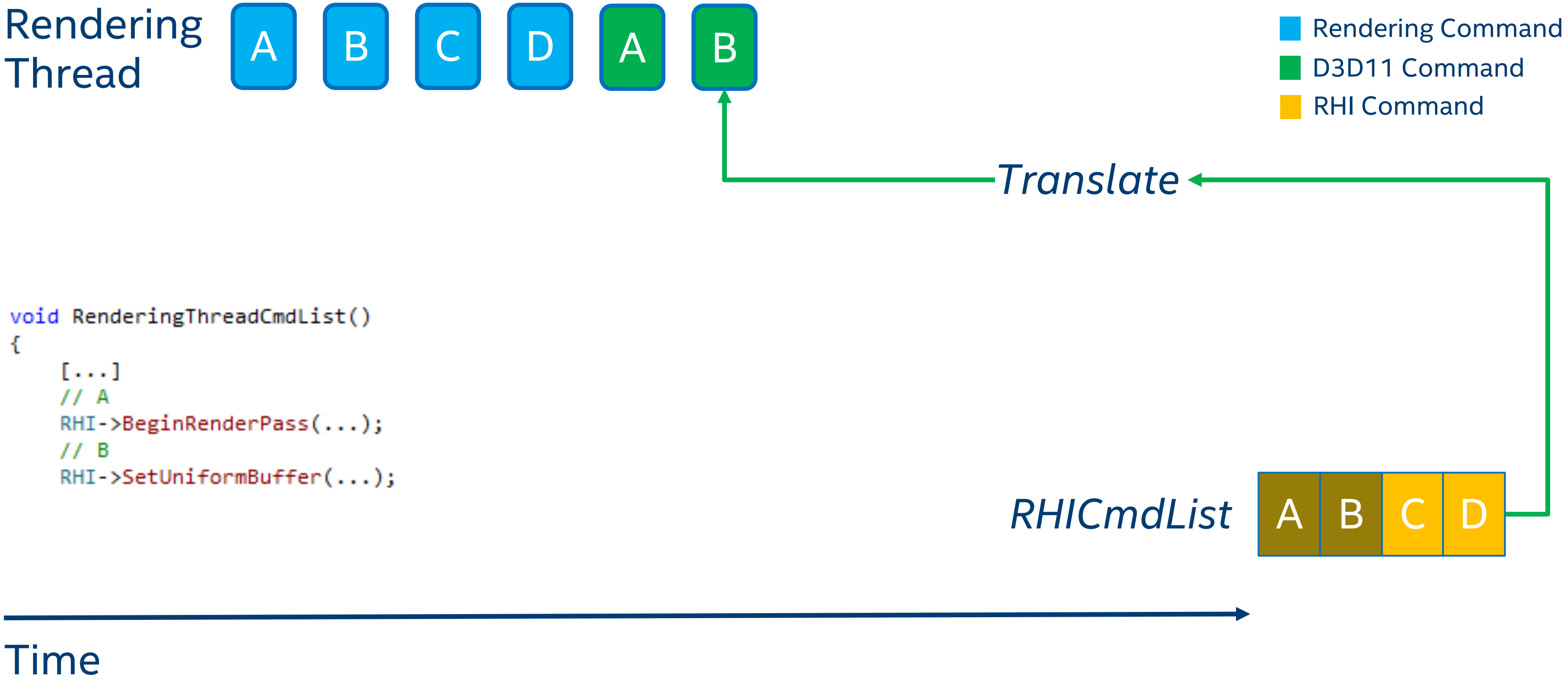
🟦 Rendering Command
🟩 D3D11 Command
🟨 RHI Command

UE4's Threading Model: RHI Command List



```
void RenderingThreadCmdList()
{
    [...]
    // A
    RHI->BeginRenderPass(...);
```

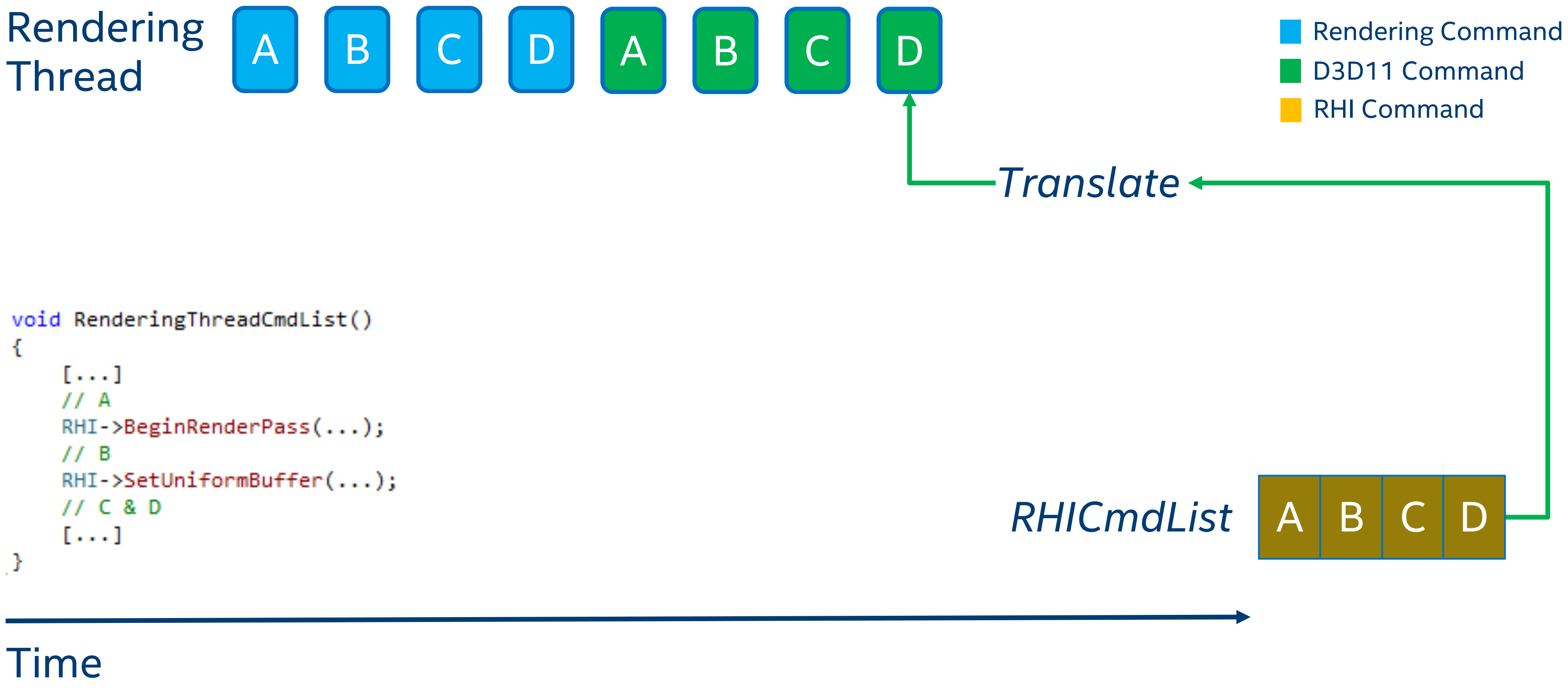
UE4's Threading Model: RHI Command List



🟦 Rendering Command
🟩 D3D11 Command
🟨 RHI Command

```
void RenderingThreadCmdList()
{
    [...]
    // A
    RHI->BeginRenderPass(...);
    // B
    RHI->SetUniformBuffer(...);
```

UE4's Threading Model: RHI Command List

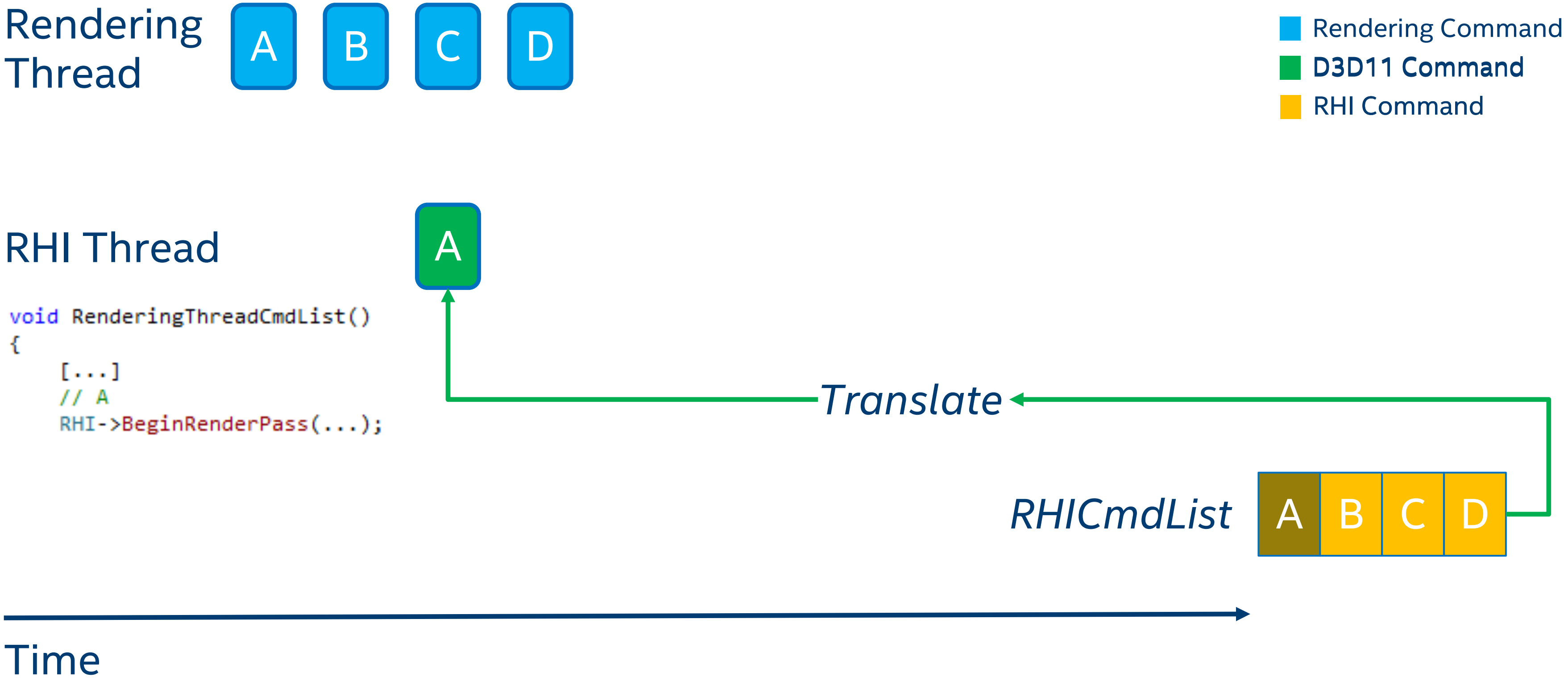


Rendering Thread

🟦 Rendering Command
🟩 D3D11 Command
🟨 RHI Command

```
void RenderingThreadCmdList()
{
    [...]
    // A
    RHI->BeginRenderPass(...);
    // B
    RHI->SetUniformBuffer(...);
    // C & D
    [...]
}
```

UE4's Threading Model: RHI Thread



```
void RenderingThreadCmdList()
{
    [...]
    // A
    RHI->BeginRenderPass(...);
```

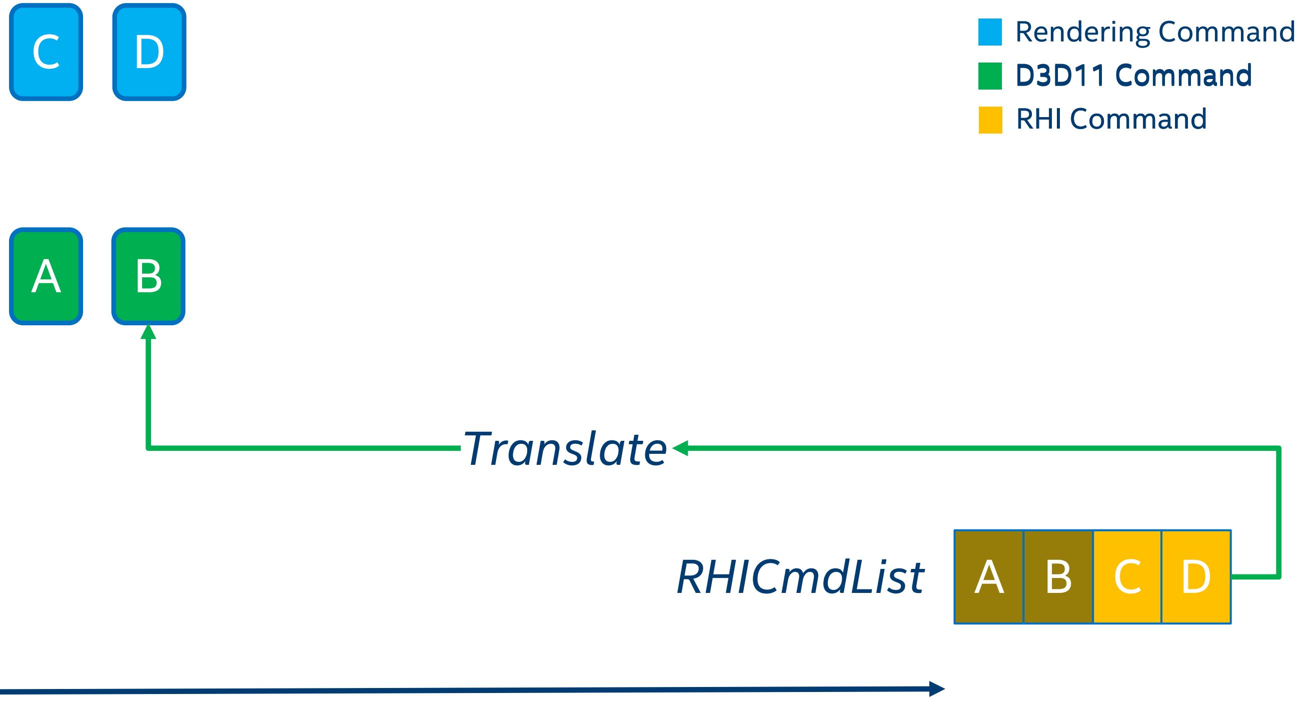
🟦 Rendering Command
🟩 D3D11 Command
🟨 RHI Command

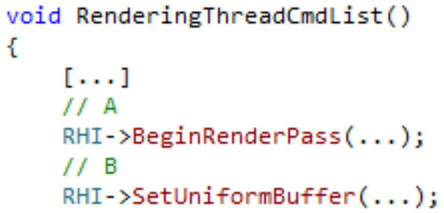
UE4’s Threading Model: RHI Thread

🟦 Rendering Command
🟩 **D3D11 Command**
🟨 RHI Command

Rendering Thread

RHI Thread

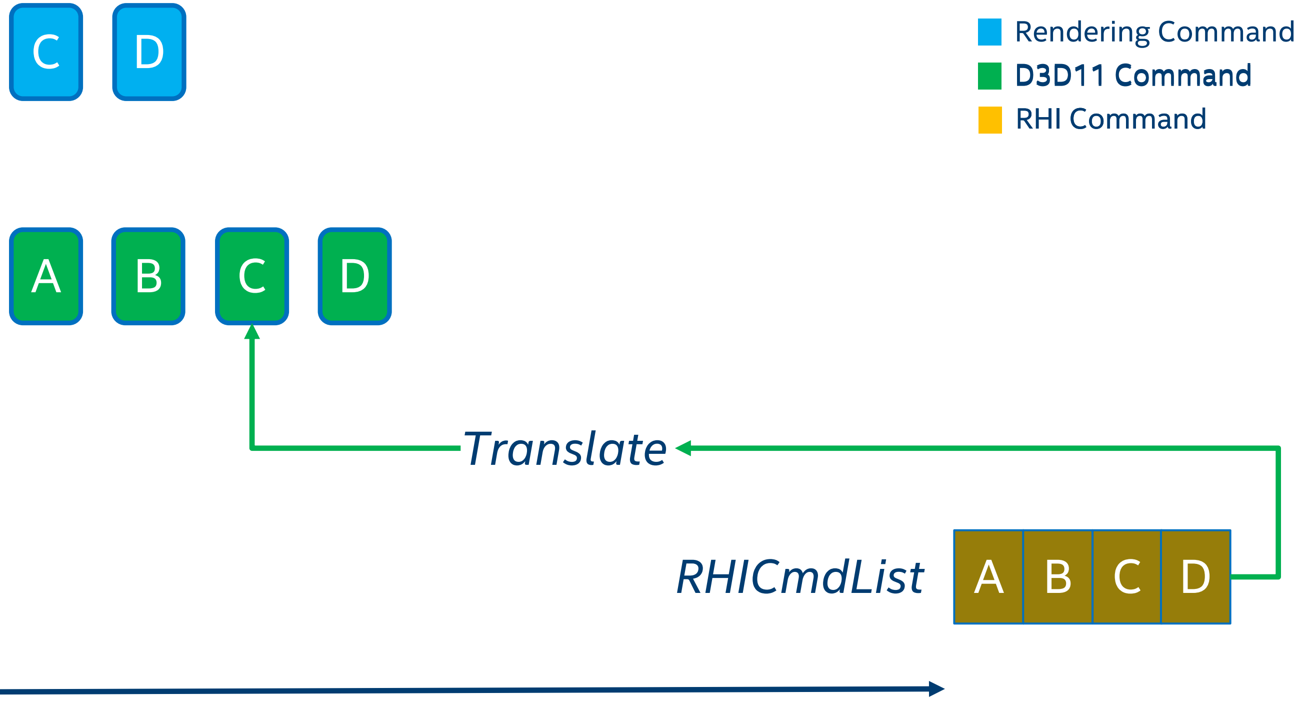


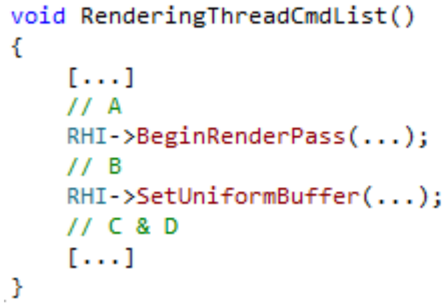


```
void RenderingThreadCmdList()
{
    [...]
    // A
    RHI->BeginRenderPass(...);
    // B
    RHI->SetUniformBuffer(...);
```

Time

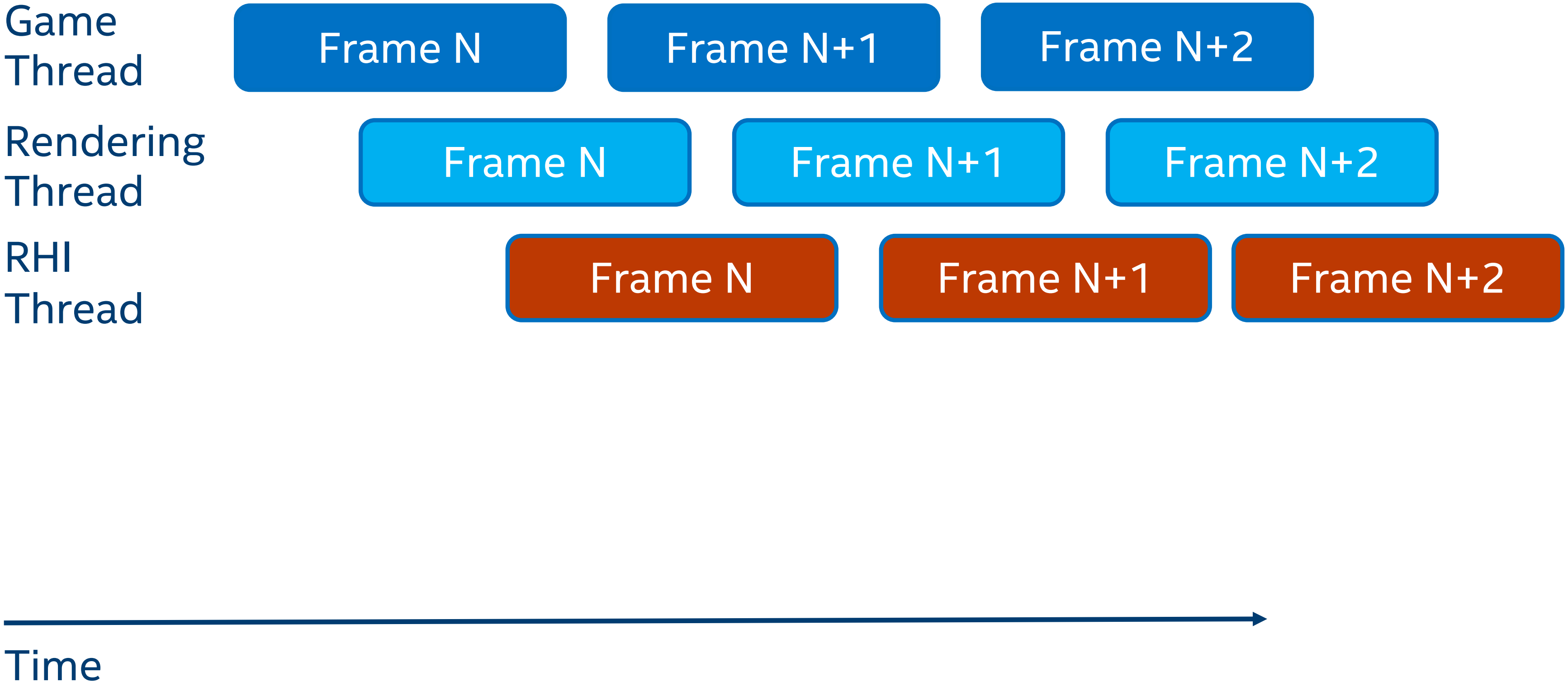
UE4's Threading Model: RHI Thread



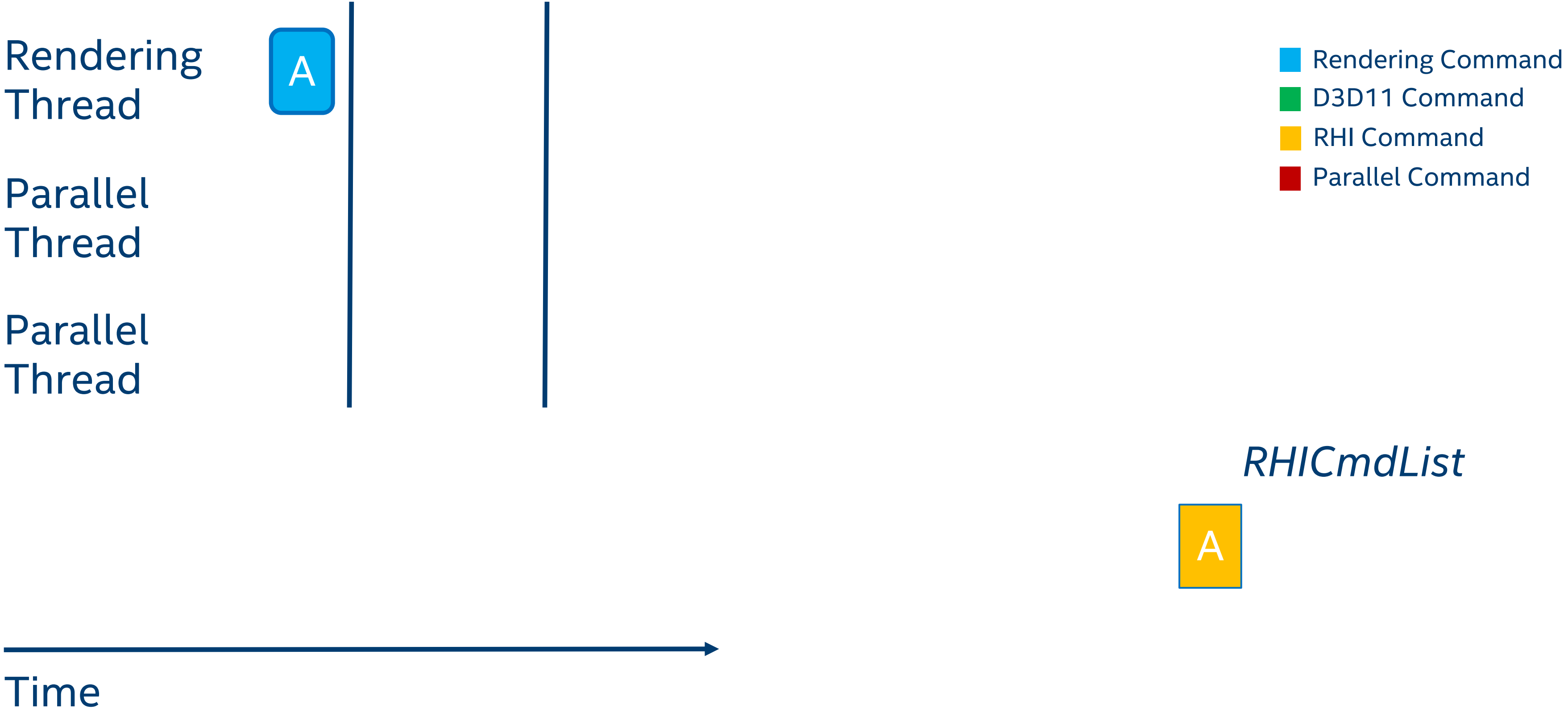


```
void RenderingThreadCmdList()
{
    [...]
    // A
    RHI->BeginRenderPass(...);
    // B
    RHI->SetUniformBuffer(...);
    // C & D
    [...]
}
```

UE4's Threading Model: Game -> Rendering -> RHI Thread



UE4's Threading Model: Parallel Frontend



Rendering Thread

Parallel Thread

Parallel Thread

A

RHICmdList

A

Time

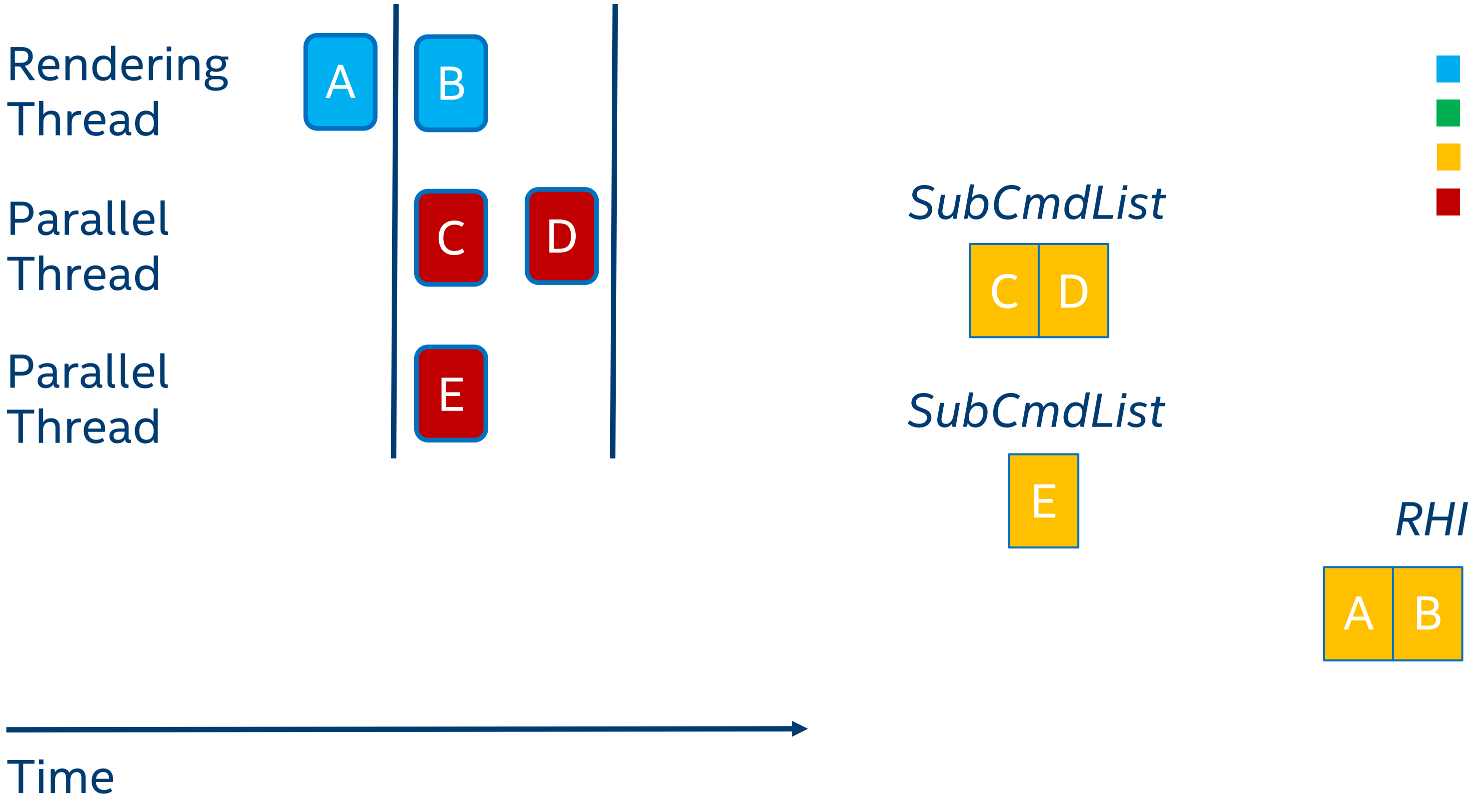
🟦 Rendering Command

🟩 D3D11 Command

🟨 RHI Command

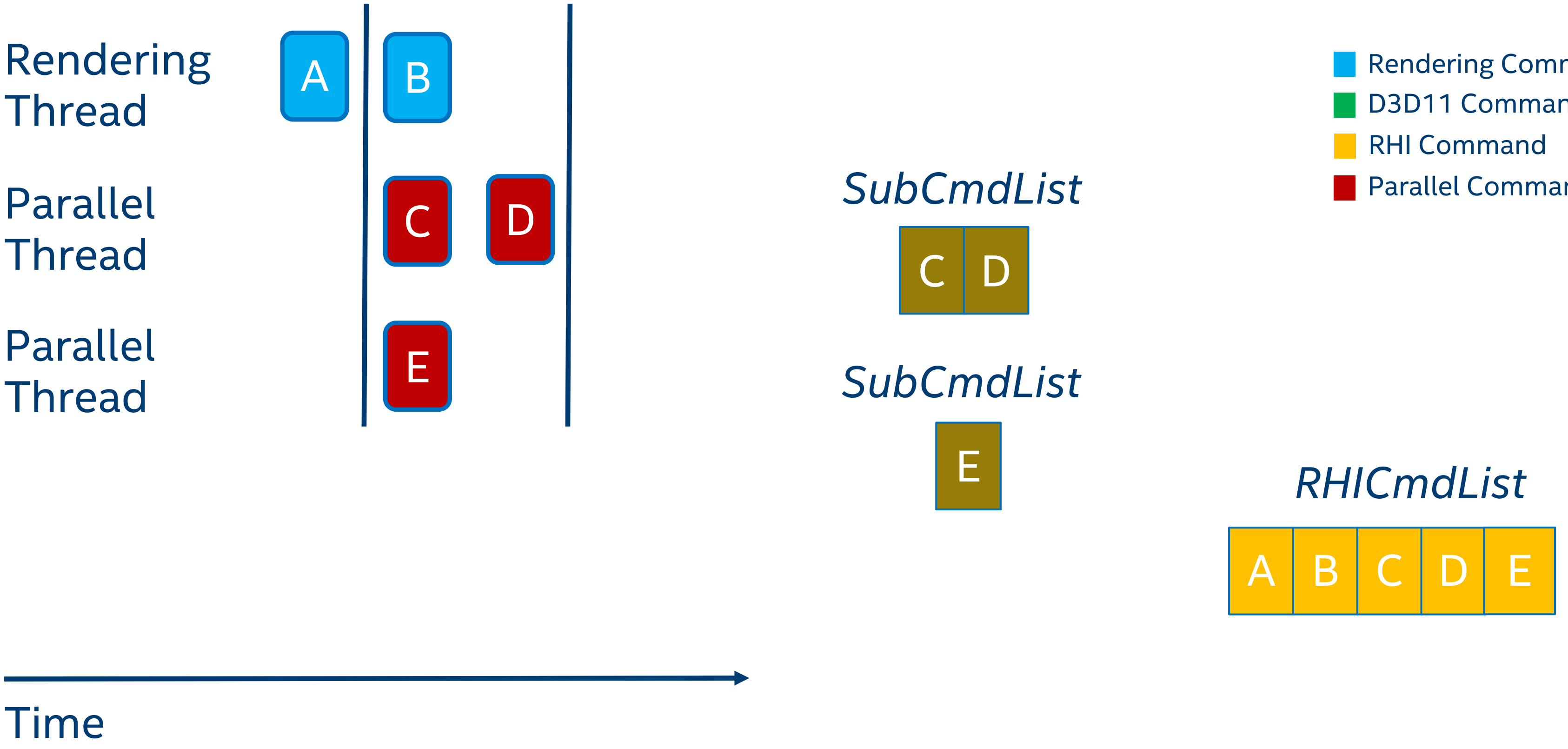
🟥 Parallel Command

UE4's Threading Model: Parallel Frontend



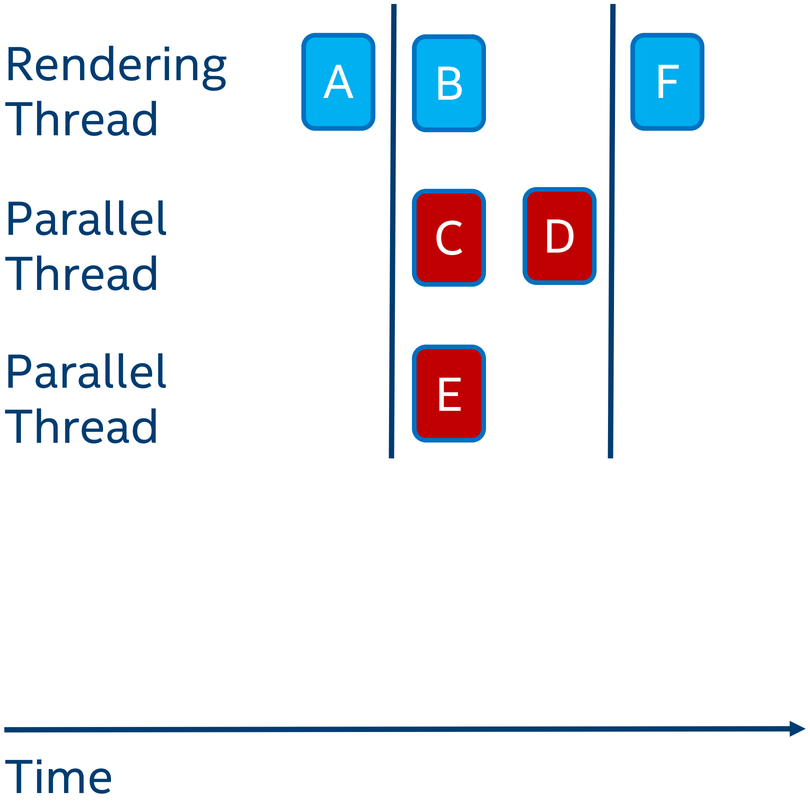
- 🟦 Rendering Command
- 🟩 D3D11 Command
- 🟨 RHI Command
- 🟥 Parallel Command

UE4's Threading Model: Parallel Frontend



- 🟦 Rendering Command
- 🟩 D3D11 Command
- 🟨 RHI Command
- 🟥 Parallel Command

UE4’s Threading Model: Parallel Frontend

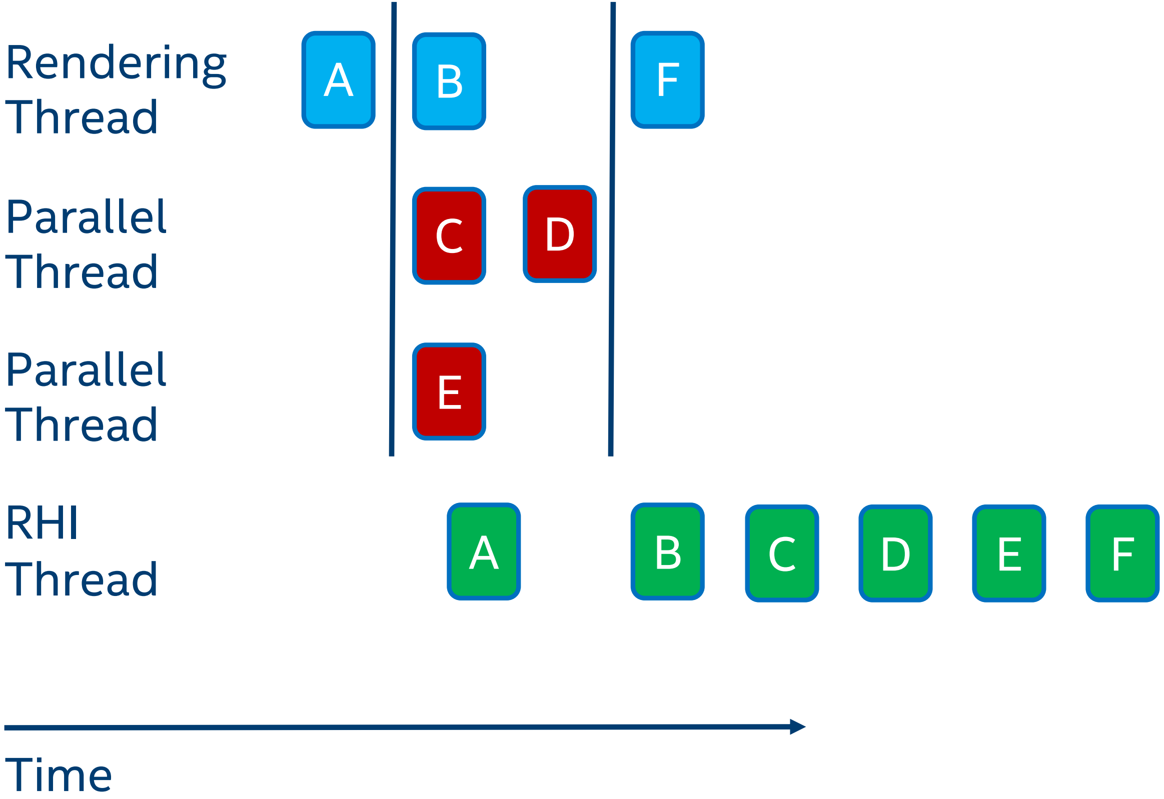


🟦 Rendering Command
🟩 D3D11 Command
🟨 RHI Command
🟥 Parallel Command

RHICmdList

A	B	C	D	E	F

UE4's Threading Model: Parallel Frontend



- 🟦 Rendering Command
- 🟩 D3D11 Command
- 🟨 RHI Command
- 🟥 Parallel Command

RHICmdList

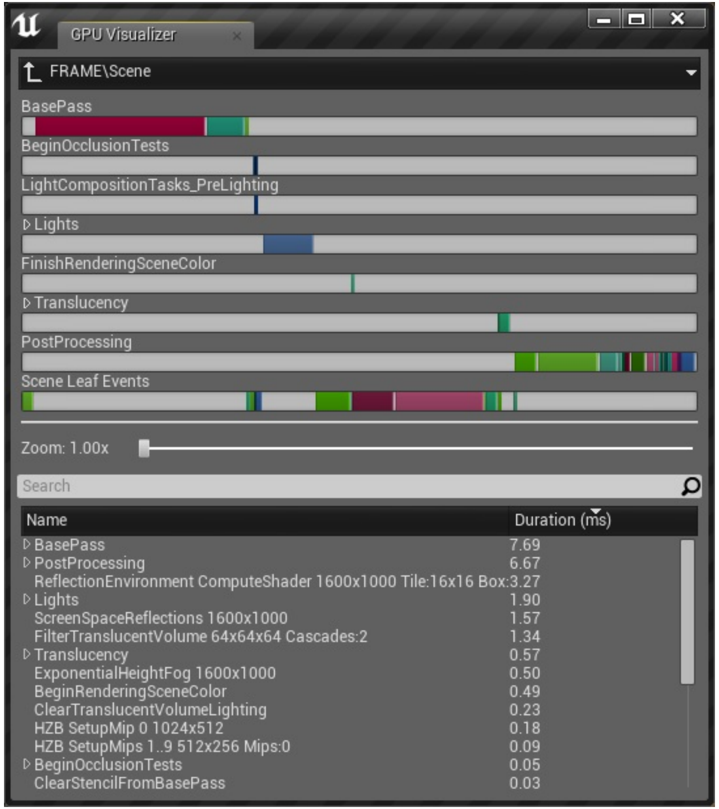
A	B	C	D	E	F

CPU OPTIMIZATIONS

DirectX12 Optimizations

Problem: DX11 Render Thread Bottlenecks are starving the GPU but DX12 path is behind in performance. New Microsoft features (RTX, VRS) require DX12.

Driver and Engine investment to improve DX12 performance, from shader compilation to runtime efficiency.



Name	Duration (ms)
▷ BasePass	7.69
▷ PostProcessing	6.67
ReflectionEnvironment ComputeShader 1600x1000 Tile:16x16 Box:	3.27
▷ Lights	1.90
ScreenSpaceReflections 1600x1000	1.57
FilterTranslucentVolume 64x64x64 Cascades:2	1.34
▷ Translucency	0.57
ExponentialHeightFog 1600x1000	0.50
BeginRenderingSceneColor	0.49
ClearTranslucentVolumeLighting	0.23
HZB SetupMip 0 1024x512	0.18
HZB SetupMips 1..9 512x256 Mips:0	0.09
▷ BeginOcclusionTests	0.05
ClearStencilFromBasePass	0.03

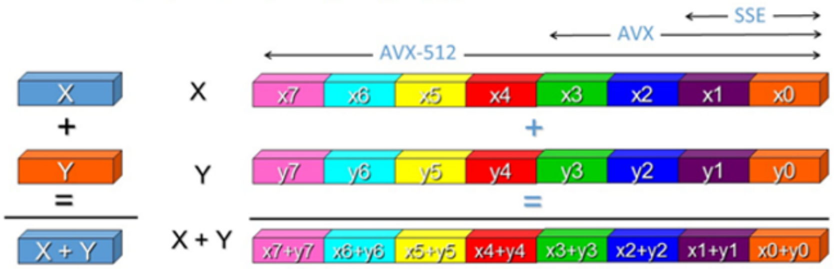
Chaos Physics

- Brand new physics system unveiled at Epic's GDC keynote.
- Worked closely with Epic to optimize low level solvers, data structures, and thread parallelism.
- Key learnings
  - C++ is bad at SIMD. Integrated Intel ISPC for ~3x gains in perf critical areas.
  - TSet is bad in high performance situations. Use TArray, Sort and RemoveSwap rather than guard dupes with TSet.Contains.
  - ParallelFor can be overused! Some oversubscription is good but don't go overboard with 1000s of jobs. Batch!

Intel SPMD Program Compiler (ISPC) Integration

- Problem: In perf critical code, C++ often doesn’t cut it. Usual solution is intrinsics. Not anymore!
- Implicit parallelism: SIMD lanes act similar to GPU shader invocations
- Write Once, Compile to many vectorized instruction sets (SSE4, AVX, AVX2)
- Used in Chaos, available in UE4 soon!
  - Include ISPC module in your build.cs
  - Add ispc files to your project
  - Include a generated C++ header
  - Unreal build tool handles the rest

```
double *x, *y, *z;
for (i=0;  i<n;  i++)      z[i] = x[i] + y[i];
```



Scalar and vectorized loop versions with Intel® SSE, AVX and AVX-512.

When to use ISPC?

- Good for dense compute-bound workloads. Heavy math like physics intersection testing, cloth or CPU vertex transformations
- Best with contiguous memory load, manipulate, store ie TArray
- Best when no data dependencies between operations. Especially useful when combined with ParallelFor and batching

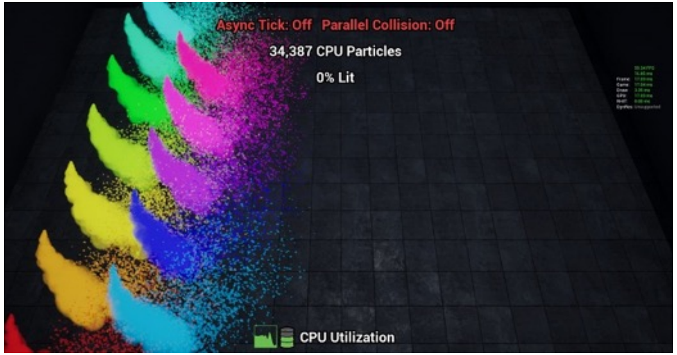
```
export void rgb2grey(uniform int N, uniform float r[],
uniform float g[], uniform float b[], uniform float grey[])
{
     foreach (idx = 0 ... N)    {
           grey[idx] = 0.3f * r[idx] +
           0.59f * g[idx] +
           0.11f * b[idx];
     }
}
```

```
…
vmulps (%rdx,%rax), %ymm1, %ymm3
vfmadd231ps (%rsi,%rax), %ymm0, %ymm3
vfmadd231ps (%rcx,%rax), %ymm2, %ymm3
vmovups %ymm3, (%r8,%rax)
…
```

Cascade CPU Particles

For enhanced realism (level lighting, bouncing, complex interactions).

- Legacy physics engine had issue with locking scene on reads, causing serialization through a critical section.
- No exiting support for ticking CPU particles in parallel.
- Fixing both gave up to 4x throughput improvement per frame
- Patch available for 4.19+



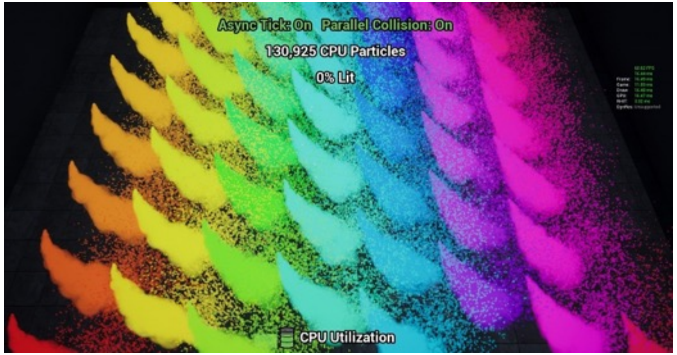
Async Tick: Off Parallel Collision: Off

34,387 CPU Particles

0% Lit

CPU Utilization

Before Optimization



Async Tick: On Parallel Collision: On

130,925 CPU Particles

0% Lit

CPU Utilization

After Optimization

Intel® VTune™ Amplifier

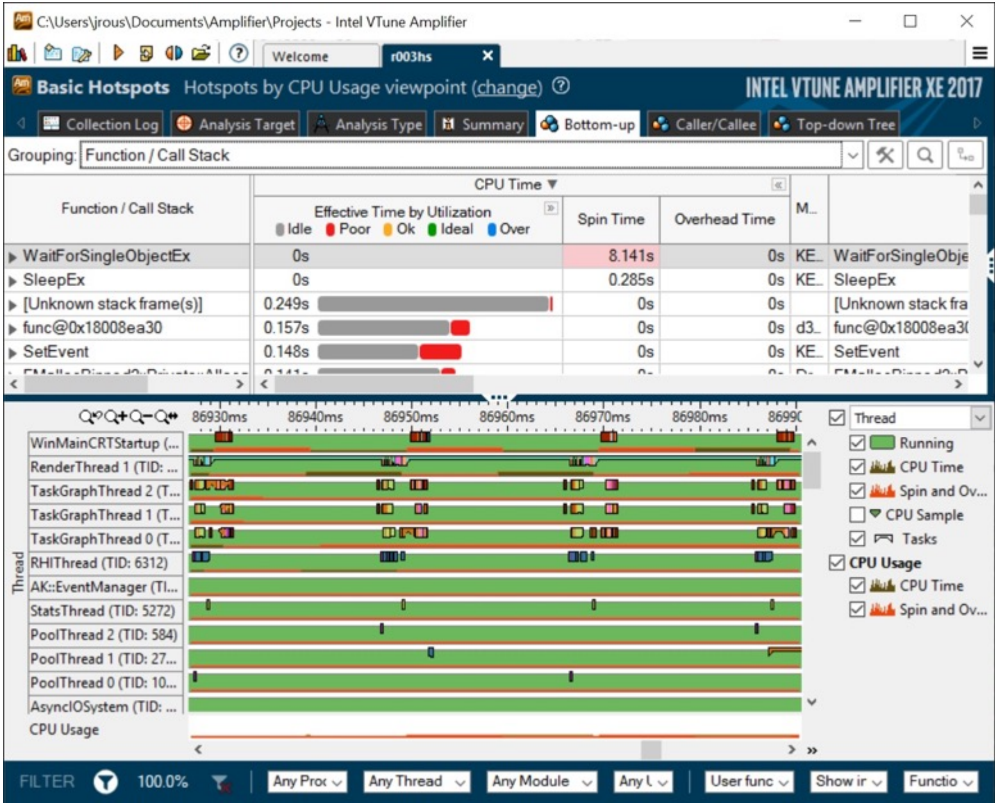
Intel® Vtune™ Amplifier enables deep profiling and problem identification.

Hotspots, locks, syncs, multithreading, even GPU data!

With 4.19, new support for event based CPU sampling using itt_notify framework.

Use **–VTune** on the command line and *stat NamedEvents* on the console.

Vtune™ is now free!



C:\Users\jrous\Documents\Amplifier\Projects - Intel VTune Amplifier

Welcome | r003hs

Basic Hotspots Hotspots by CPU Usage viewpoint (change)

INTEL VTUNE AMPLIFIER XE 2017

Collection Log | Analysis Target | Analysis Type | Summary | Bottom-up | Caller/Callee | Top-down Tree

Grouping: Function / Call Stack

Function / Call Stack	CPU Time ▼ Effective Time by Utilization (▮ Idle ▮ Poor ▮ Ok ▮ Ideal ▮ Over)	Spin Time	Overhead Time	M...	
▸ WaitForSingleObjectEx	0s	8.141s	0s	KE...	WaitForSingleObje
▸ SleepEx	0s	0.285s	0s	KE...	SleepEx
▸ [Unknown stack frame(s)]	0.249s	0s	0s		[Unknown stack fra
▸ func@0x18008ea30	0.157s	0s	0s	d3...	func@0x18008ea3(
▸ SetEvent	0.148s	0s	0s	KE...	SetEvent

☑ Thread
☑ Running
☑ CPU Time
☑ Spin and Ov...
☐ CPU Sample
☑ Tasks
☑ CPU Usage
☑ CPU Time
☑ Spin and Ov...

FILTER 100.0% | Any Proc | Any Thread | Any Module | Any U | User func | Show in | Functio

Intel Embree

Embree enables fast light baking for breathtaking visuals.

“Intel Embree is pretty good. 4x faster Lightmass ray tracing which results in 2.4x faster lighting builds” – Daniel Wright, Technical Director, Graphics, Epic Games

Embree is fully enabled for multicore

Used by default in static light builds

Embree accelerates lighting calculations using the full potential of multicore CPUs and new ISAs

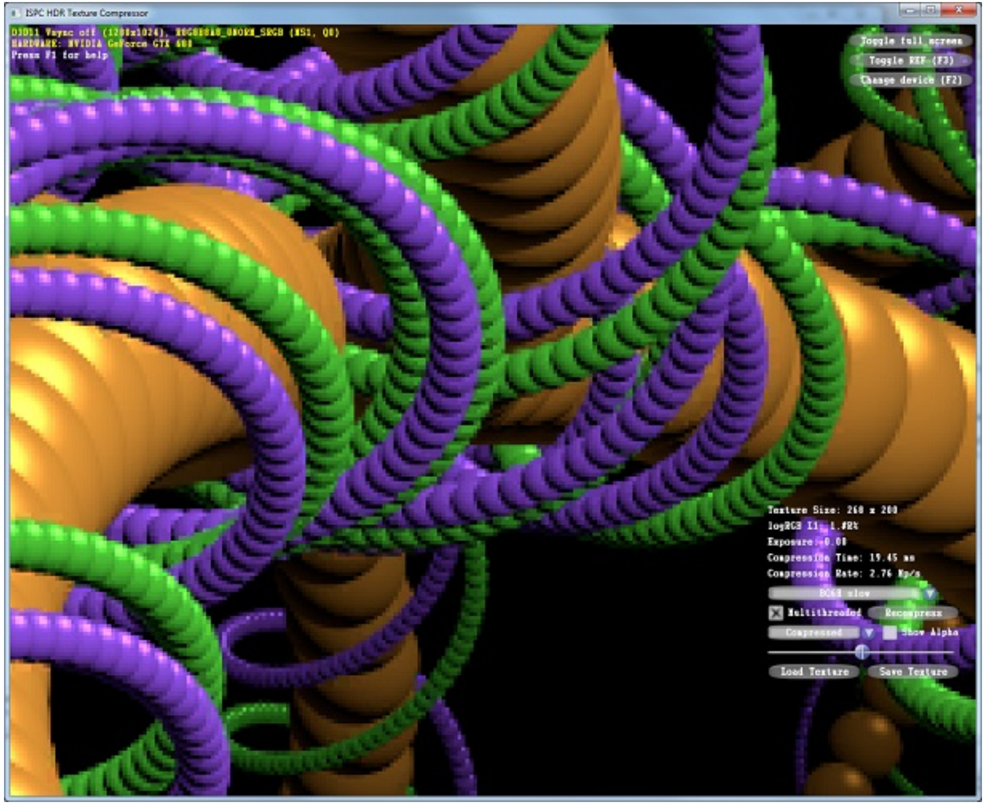
ISPC Texture Compression

Unreal Engine 4 now has support in the public engine release for Intel's fast texture compressor

Unreal needs multiple industry standard formats (BC6H/BC7/ASTC)

44x average speed improvement, making this the fastest in the industry on Intel

Epic found SunTemple texture compression time drop from **68 min** to **35 secs** on a Macbook Pro!

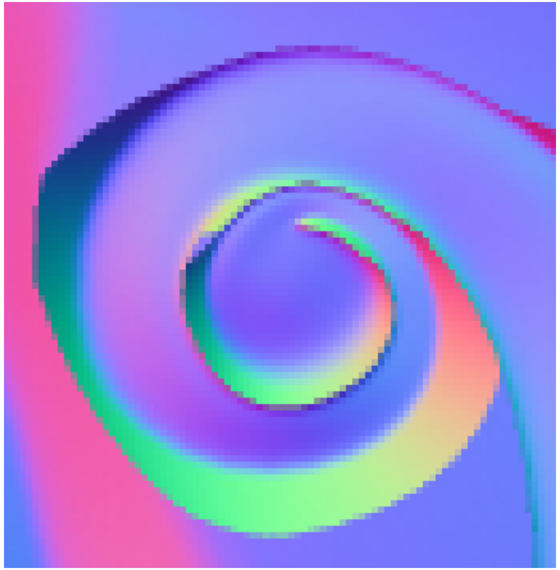


```
D3D11 Vsync off (1280x1024), R8G8B8A8_UNORM_SRGB (MS1, Q0)
HARDWARE: NVIDIA GeForce GTX 480
Press F1 for help
```

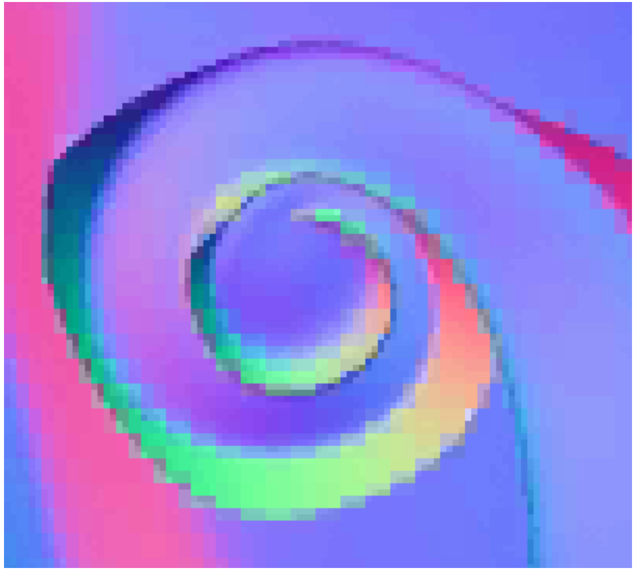
```
Texture Size: 268 x 200
logRGB L1: 1.#R%
Exposure: 0.00
Compression Time: 19.45 ms
Compression Rate: 2.76 Mp/s
```

ASTC Quality Comparison

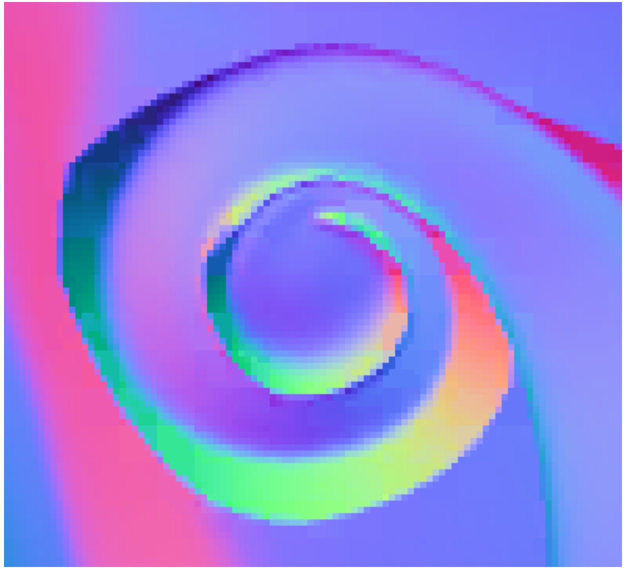
Zoomed in portion of a 2048x2048 normal map



Original: 12 MB



ETC1: 2 MB

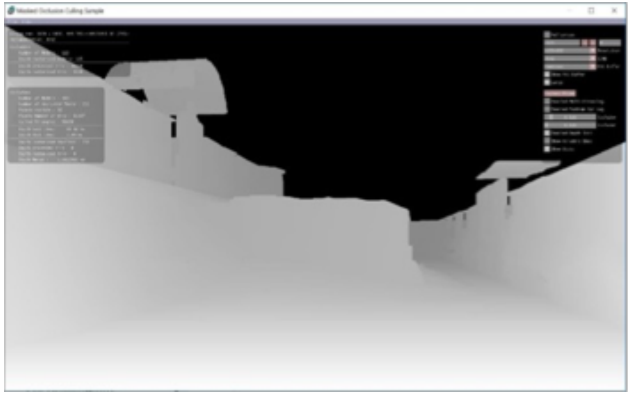


ASTC 6x6: 1.8 MB

GPU OPTIMIZATIONS

Masked Occlusion Culling

CPU-based alternative Hi-Z buffer representation for fast, low-latency occlusion queries.



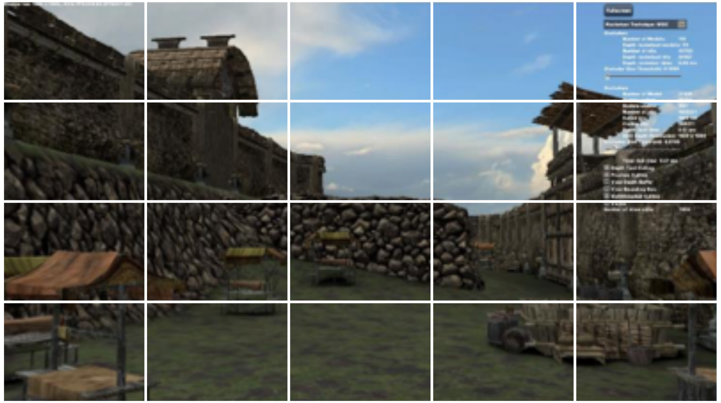
- Much less memory to read/write than full res z-buffer.
- Updates use bitmasks – can process many pixels in parallel (i.e. SSE4.1/AVX2).
- No need for conservative art assets (although faster if so).
- Integrated into UE4 threading systems.
- Compatible with LLVM/Clang for cross platform support.

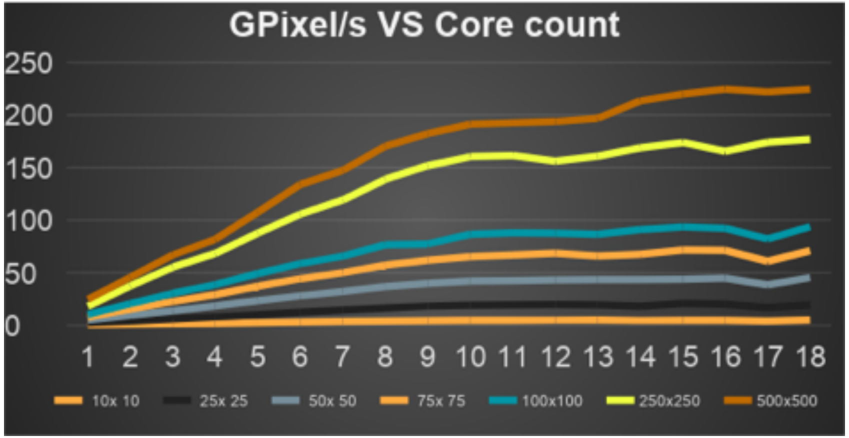
Intel Castle Sample, Performance Comparison

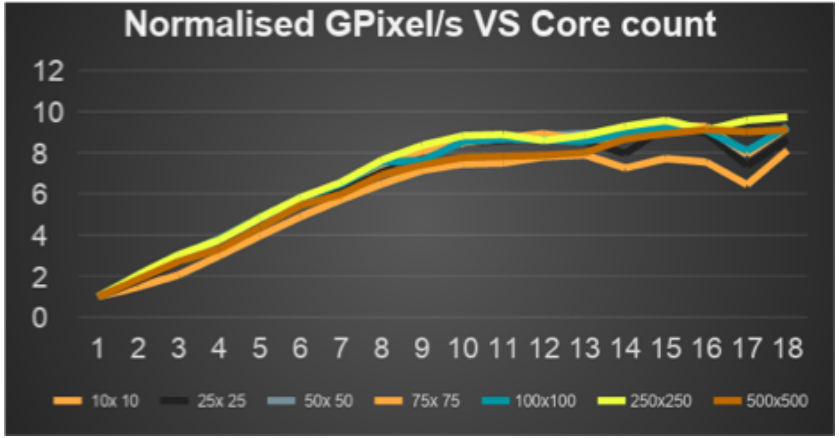
	Draw All	Frustum culling only	AVX2 Threaded (10C)
FPS	143	194	650
MS	6.9	5.14	1.53
Drawcalls	20801	6518	1512

MOC Binned rendering

- Transform triangles into screen space.
- Bin by screen-space tiles.
- Use ScissorRect to clip contents.
- 1 active tile per TaskGraph thread.





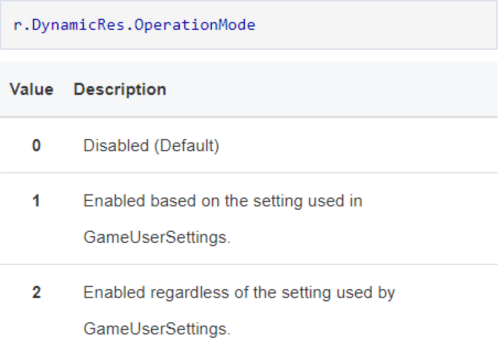


Problem: GPU cost varies over time

- In general, more content means longer frames
- If last frame was slow, potentially current one will be too:
  - Particles/fog/smoke
  - More draw calls
  - Higher density objects
  - … etc!

Dynamic Resolution

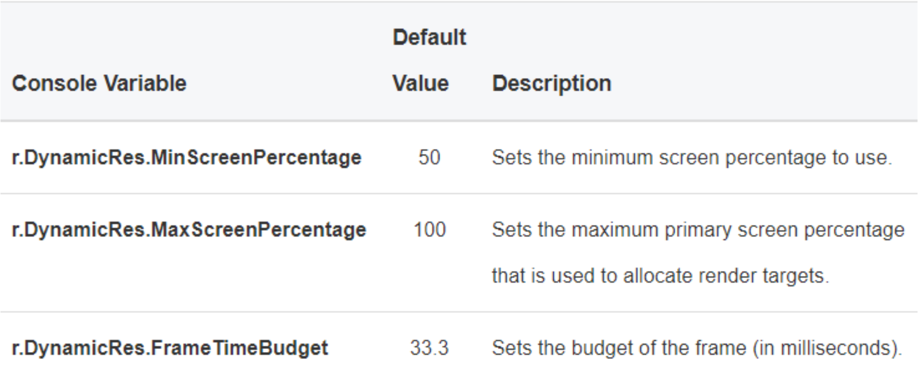
- Available since 4.21!
  - Adjusts the primary screen percentage according to the previous frames' GPU workload and/or game control



```
r.DynamicRes.OperationMode
```

Value	Description
0	Disabled (Default)
1	Enabled based on the setting used in GameUserSettings.
2	Enabled regardless of the setting used by GameUserSettings.

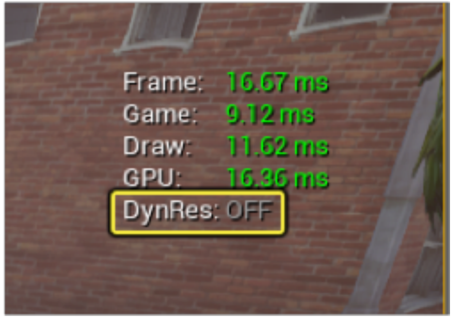
Dynamic Resolution



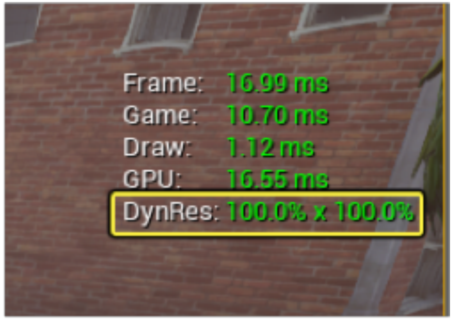
Console Variable	Default Value	Description
r.DynamicRes.MinScreenPercentage	50	Sets the minimum screen percentage to use.
r.DynamicRes.MaxScreenPercentage	100	Sets the maximum primary screen percentage that is used to allocate render targets.
r.DynamicRes.FrameTimeBudget	33.3	Sets the budget of the frame (in milliseconds).

Dynamic Resolution

- *stat unit* to show if it's enabled
- *When on, X% by Y%*
- *For more info:* https://docs.unrealengine.com/en-us/Engine/Rendering/DynamicResolution
- *Or Google "Unreal Engine Dynamic Resolution" ;)*



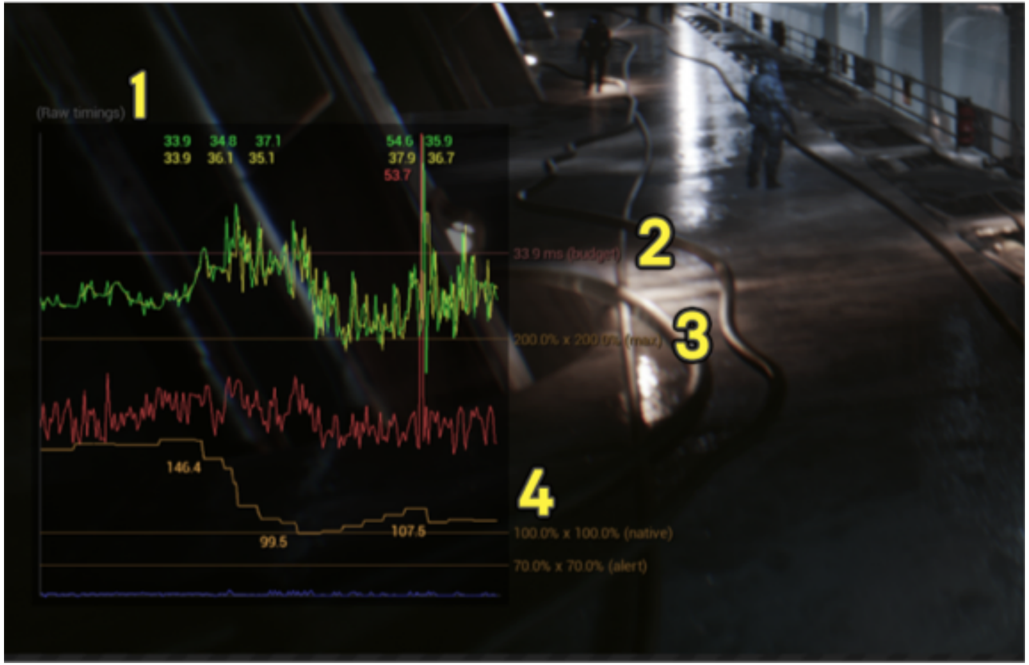
```
Frame:   16.67 ms
Game:    9.12 ms
Draw:    11.62 ms
GPU:     16.36 ms
DynRes: OFF
```



```
Frame:   16.99 ms
Game:    10.70 ms
Draw:    1.12 ms
GPU:     16.55 ms
DynRes: 100.0% x 100.0%
```

Dynamic Resolution

- *stat unitgraph*
  - *1- Timings (filtered or raw)*
  - *2- Target Frame Time*
  - *3- Dyn Res Max Screen %*
  - *4- Dyn Res Screen % curve*
- *Originally supported only on consoles…*



Dynamic Resolution

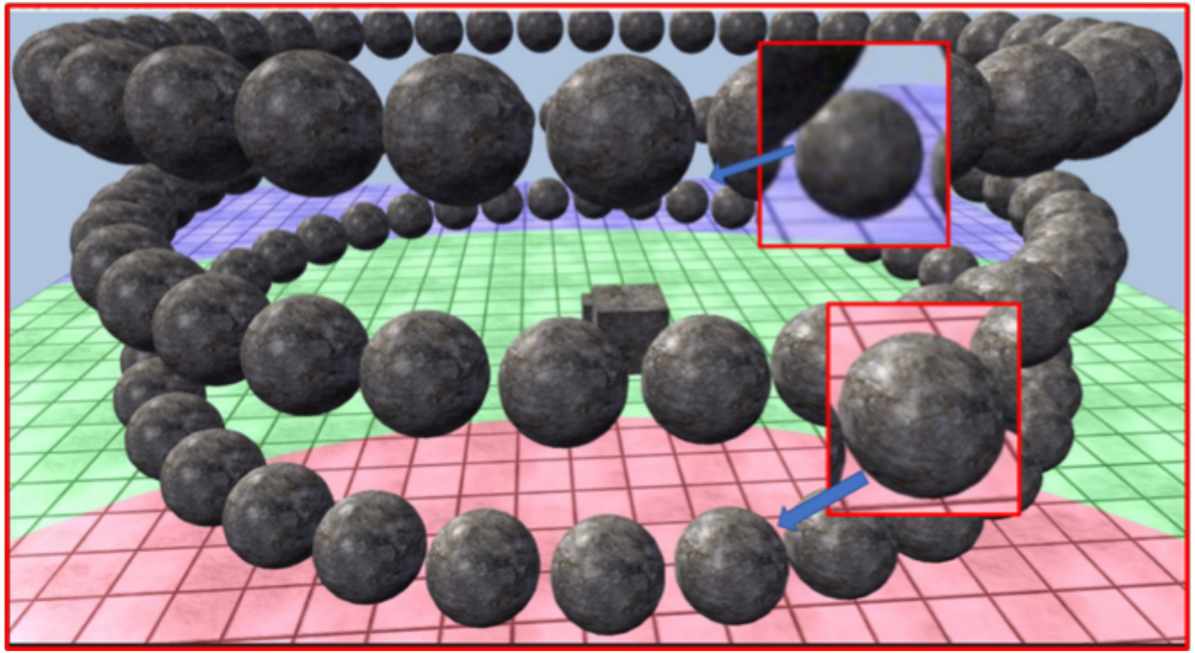
- Intel added driver API support to Unreal to access low level hardware counters
- Why isn't this a solved problem using API timestamps?
- Timestamp = GPU + CPU time. Want just GPU time to avoid CPU bubbles.
- Feature previously only available on console now on PC

100% scaling

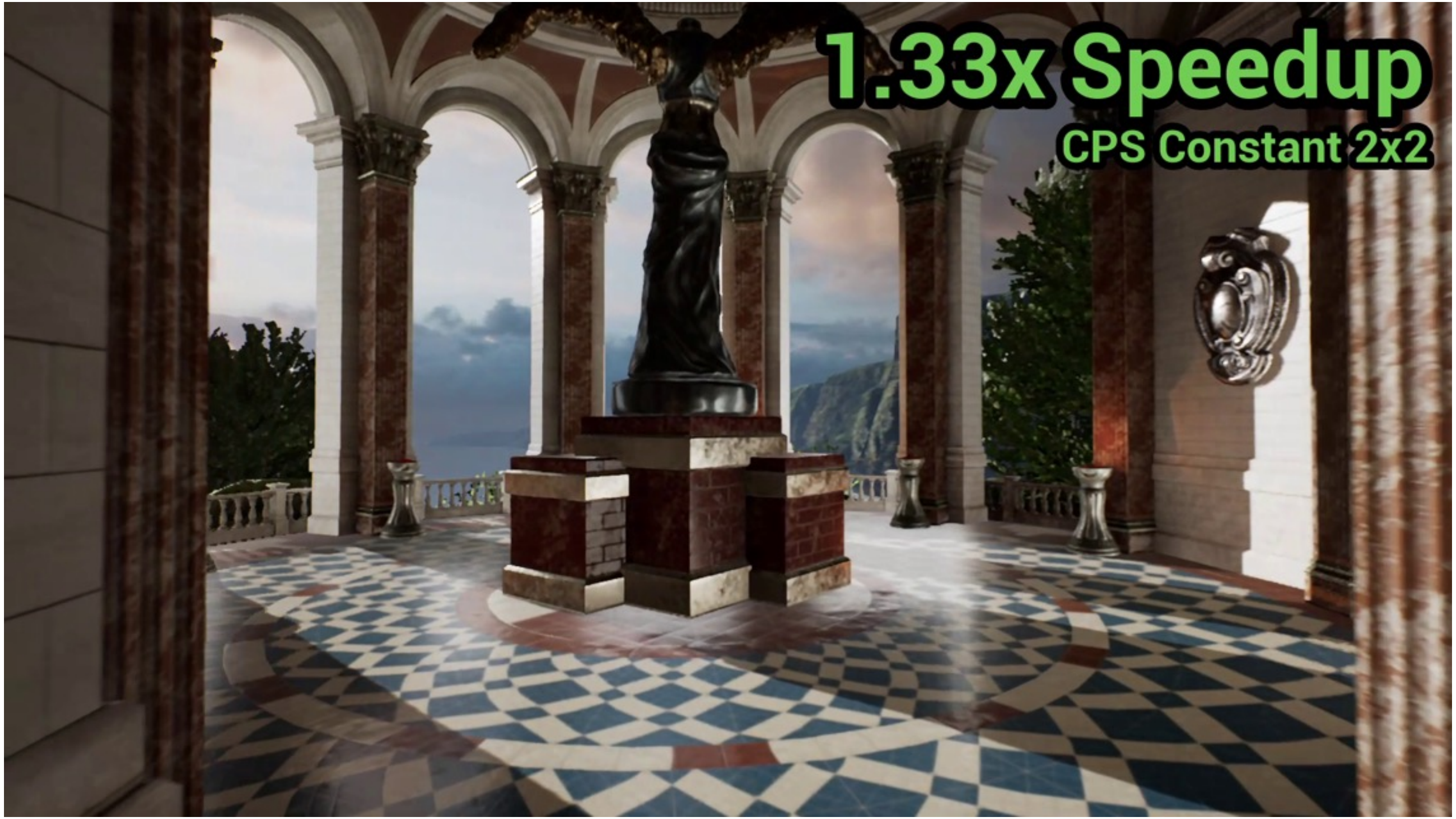
50% scaling

Variable Rate Shading

Problem: Spending pixel shading time on far away/motion-blurred objects.

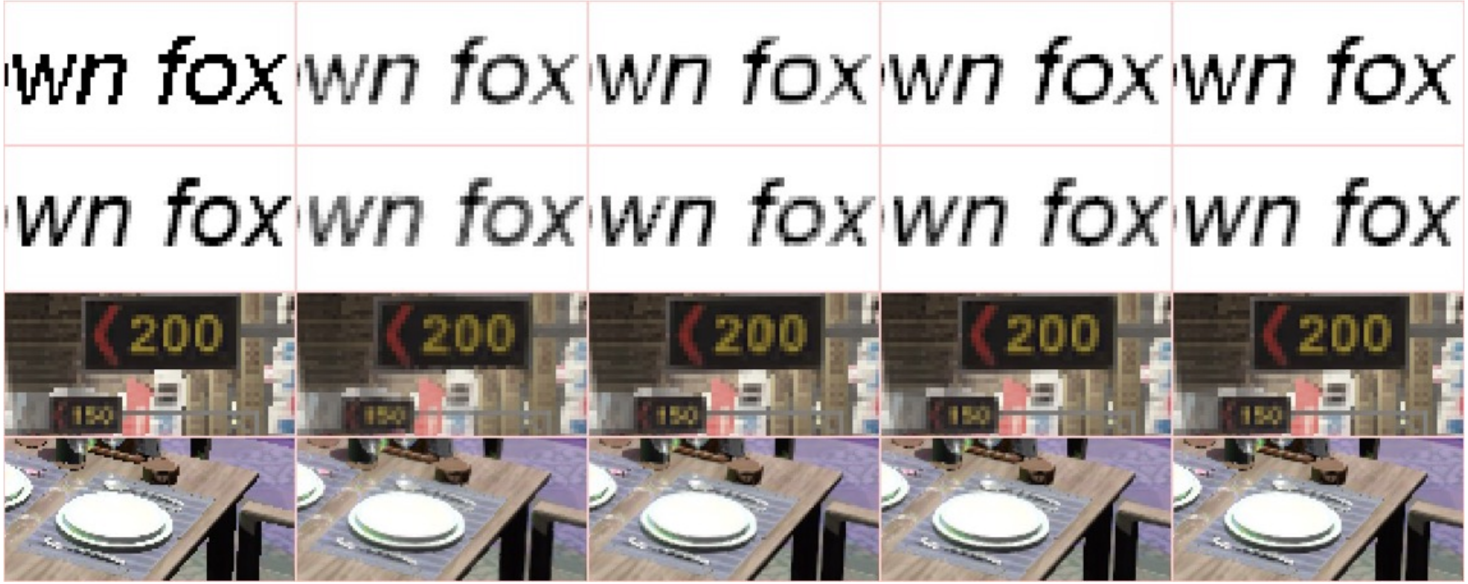


Variable Rate Shading



Problem: Anti-Aliasing Performance v. Quality

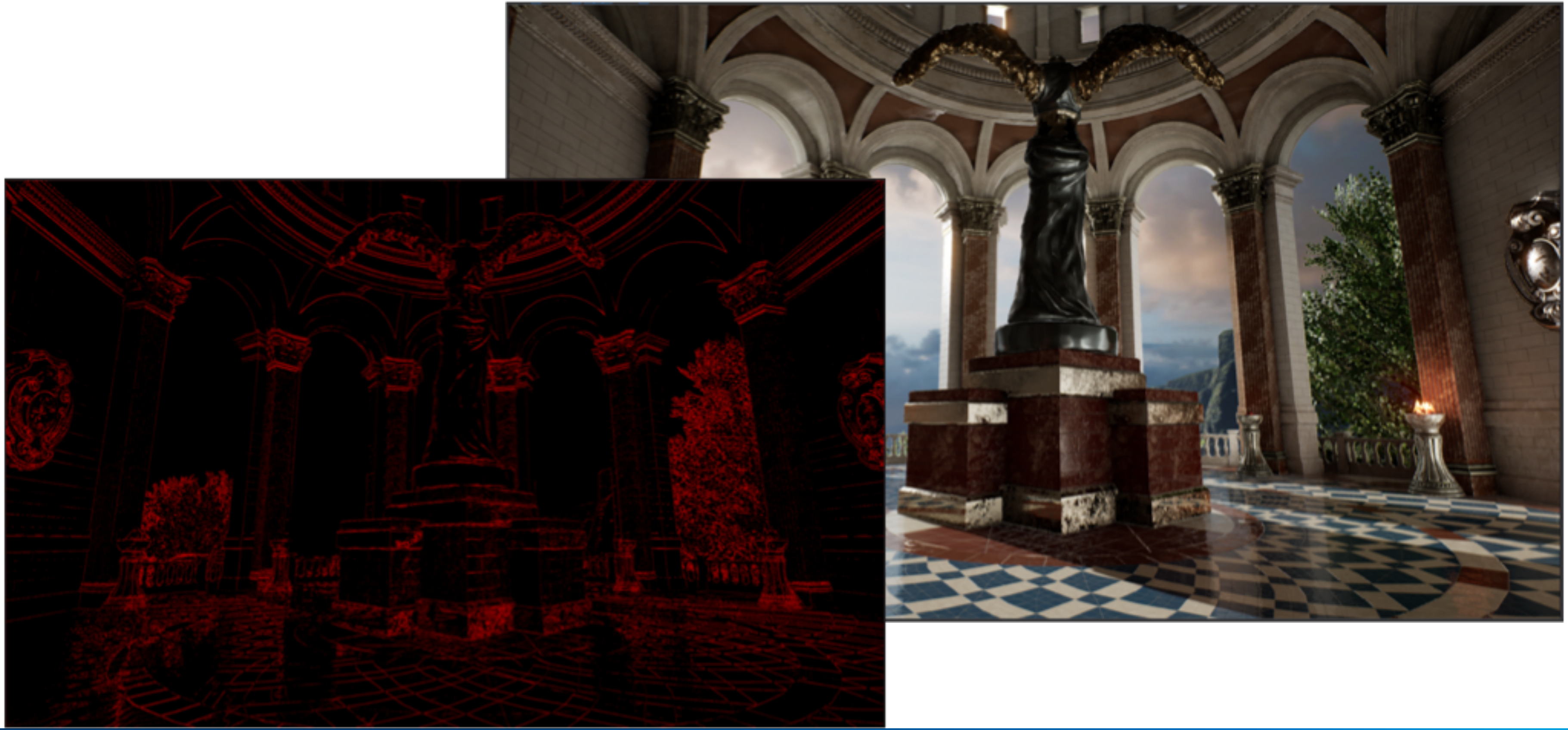
Tradeoff between extremes: temporal blurring or unreadable text?



Source FXAA 3.11 SMAA CMAA2 CMAA2-ExtraSharp

Sharpness vs anti-aliasing

Conservative Morphological Anti-Aliasing 2.0



Final Thoughts

- We’ve talked about a bunch of things that improve performance and help developer quality of life both on CPU and GPU
- Go try out the things we’ve worked on!
- Give us feedback on what we can work on next. Tell us about your pain points.

Talk back to us! Twitter handles **@jeff_rous** and **@rcalocao**

QUESTIONS?

Questions?

Unreal Engine 4 Samples and Whitepapers

CPU Particles (software.intel.com/en-us/articles/maximizing-visuals-with-cpu-particles-in-unreal-engine-4)

UE 4.19 Optimizations (software.intel.com/en-us/articles/intel-software-engineers-assist-with-unreal-engine-419-optimizations)

Optimization Guide (software.intel.com/en-us/articles/unreal-engine-4-optimization-tutorial-part-1)

CPU Optimizations for Cloth Simulations (software.intel.com/en-us/articles/unreal-engine-4-blueprint-cpu-optimizations-for-cloth-simulations)

Setting up Destructive Meshes (software.intel.com/en-us/articles/unreal-engine-4-setting-up-destructive-meshes)

CPU Scaling Sample (github.com/GameTechDev/RCRaceland)

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 intel®

experience
what's inside™

UNREAL
ENGINE