Console to PC VR: Lessons learned from The Unspoken

1. INTRODUCTION

*The Unspoken** is a first-person spell casting VR game from Insomniac Games* that features the Oculus Touch* controllers. Insomniac Games is a veteran console developer and their in-house engine was extended to support VR. Going from a 30-fps console experience to 90 fps on high-end VR rigs requires performance-aware design decisions and engine improvements to maximize the use of system resources. This article describes the detection of system-level engine bottlenecks, the reasoning behind them, and how they were addressed to improve performance.

2. ENGINE ARCHITECTURE

Before we dive into the details, let's talk about the Insomniac Games engine architecture. It is mainly composed of:

i) *Main thread (MT)*
The MT does the simulation of game objects and prepares the render work buffer for the next frame. The former includes issuing of skinning, physics, and vfx jobs, while the latter includes frustum and occlusion culling.

(i) *Render Submission thread (RS)*
The RS walks over the render work buffer and translates it into render commands (DirectX* 11). It includes updating various resources and currently uses an immediate context and no deferred contexts. The RS can help out as a worker thread when it is not submitting work to the GPU.

(ii) *Job threads*
There are four high-priority and four low-priority job threads that run skinning, vfx, physics, and so on.

The MT runs one frame ahead, which means that as it is working on frame N+1, the RS is submitting frame N's work to the GPU. While this adds a frame of input latency, the parallelism and loose decoupling helps with performance.

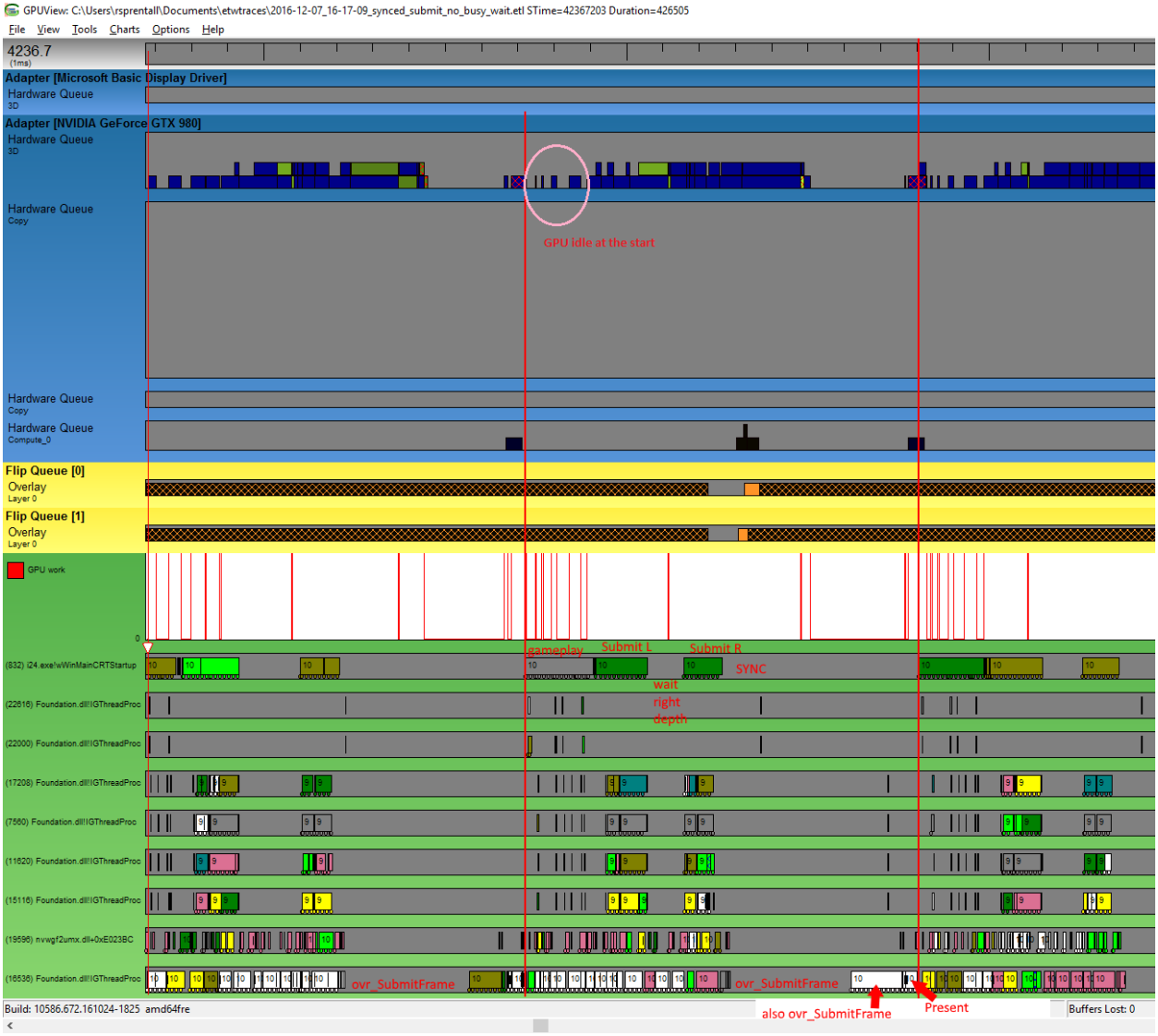
2.1 CONSOLE VS. PC

On the console, the GPU runs almost in lockstep with the RS. There is a tight synchronization between the MT and RS, wherein the RS wakes up the MT once the GPU reaches the post-processing work. This gives MT the head start needed to submit skinning work to update the vertex buffers for the frame RS to submit next, and reduces input latency while keeping the GPU fed (though there is a risk of starvation in doing so).

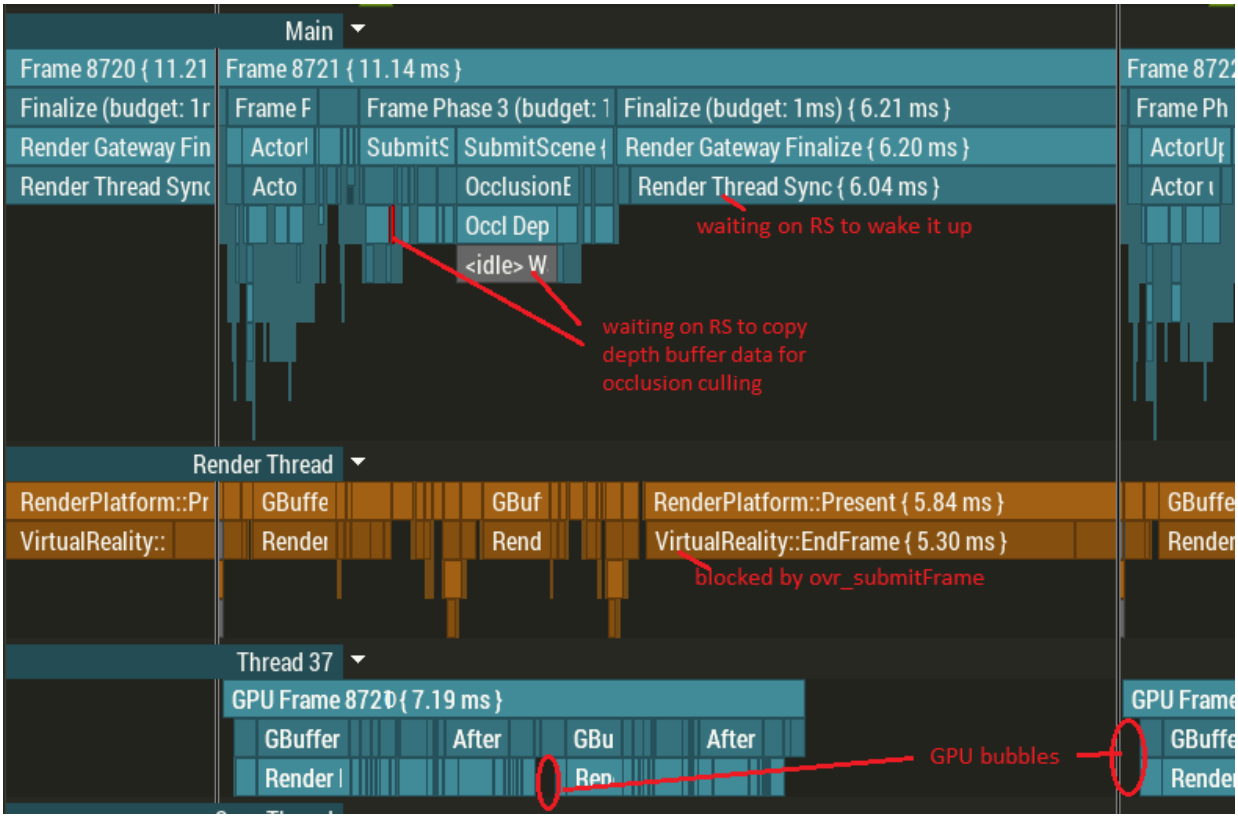
On the PC, depending on the arguments and maximum frame latency, the DirectX Present call can act as a throttle and block the game (CPU) from going too far ahead of the GPU. Without intrusive GPU queries that can stall the RS, there is no way to wake up the MT just in time; so, the PC path in the engine wakes up the MT after Present returns.

3. SYSTEM LEVEL PERFORMANCE ANALYSIS IN VR

When we looked at the game after the release of Oculus Connect* 3, we noticed bubbles (idle time) in the GPU queue around the beginning of the frame. In VR, the GPU budget for a game is under 10 ms to hit 90 fps, and ~1 ms of idle GPU time meant that 10 percent of that budget was being unutilized! Here are the GPUView* and RAD Telemetry* timelines showing the issue:



Before: GPU bubbles at the start of the frame (~1 ms). The frames shown take ~7 ms for the GTX 980 to render, and comprise a light workload from the game.*



Before: Task-based timeline view in RAD Telemetry showcasing the issue.

3.1 UNDERSTANDING THE ENGINE DEPENDENCIES

The bubbles were caused by a couple of dependencies between MT and RS.

i) *MT waits on RS to wake it post-Present*
In VR, before the mirror-Present (which shows the game on the non-HMD display) is a call to the Oculus* API ovr_submitFrame, which throttles (blocks) the app from submitting frames faster than the HMD refresh rate, which is 90Hz on the Rift*.

On an Intel® Core™ i7 6700K, the CPU frame time (RS submission time) is ~5–6 ms, which means that the ovr_submitFrame blocks for ~5 ms, during which the engine threads are almost completely idle.

ii) *RS waits on MT to submit and sync with skinning jobs*
With MT no longer having a head start, RS ends up waiting for the skinned vertex buffer data to be generated before it can submit the skinned objects in the G-Buffer pass.

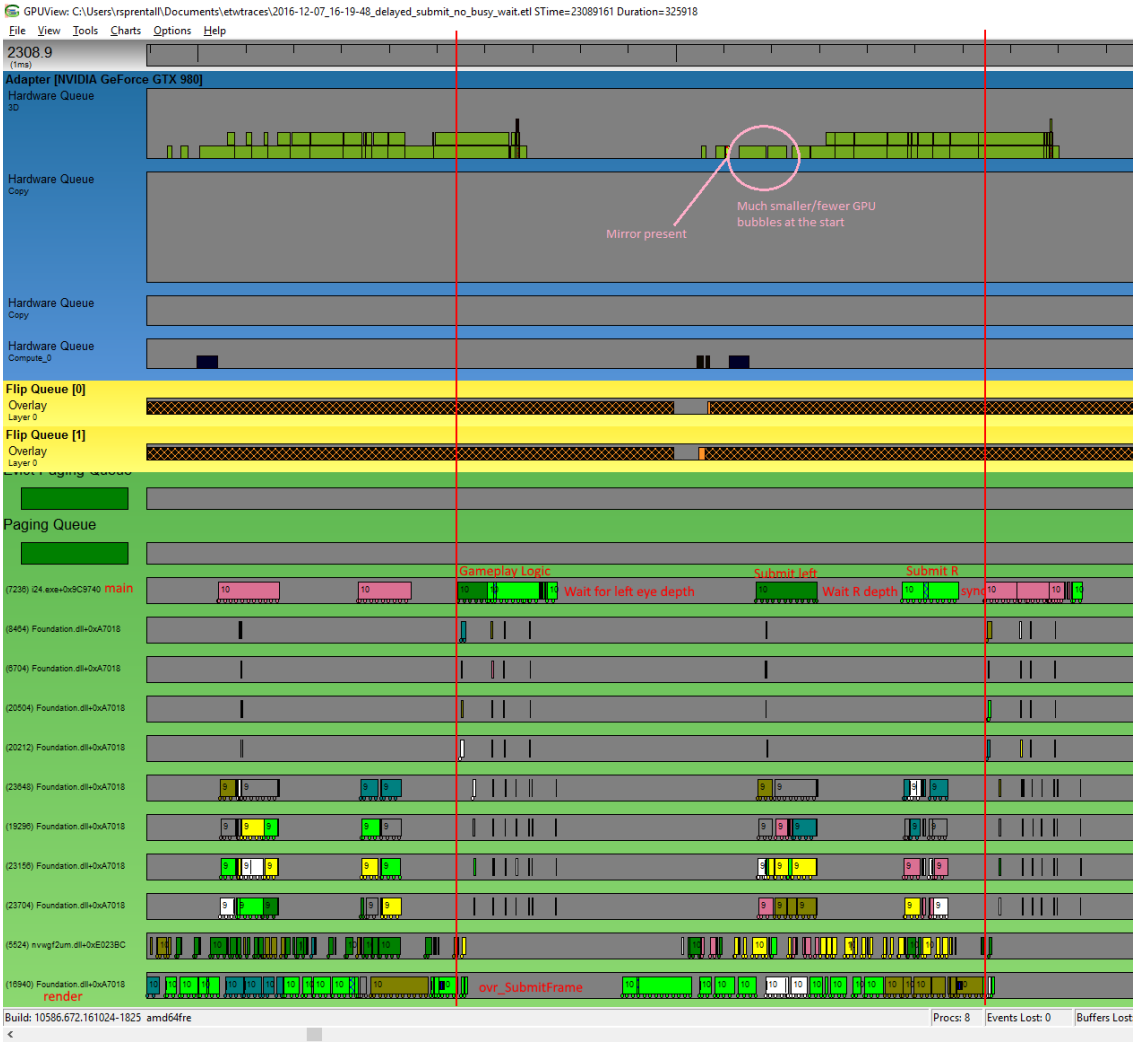
iii) *MT waits on RS to copy the downsampled depth data (per eye) for occlusion culling queries*
The engine uses an in-house occlusion culling system wherein the MT waits on RS to Map, copy, and Unmap the staging buffer holding the downsampled depth data per eye. The data used is two frames old; if MT is working on frame N, RS copies N-2's depth data for MT to reproject, hole fill, and cull occluded objects.

On lower-end VR rigs (Intel® Core™ i5 4500 + GTX 970*, for example), the GPU takes ~9–10 ms for an average frame workload, and ~1 ms of bubbles puts it beyond the 10 ms GPU budget and forces Asynchronous Space Warp* (ASW) to kick in, which clamps the game to update at 45Hz. The VR end-user experience is subjective in nature, and the keen observer can notice the delay with ASW on when moving hands or HMD quickly.

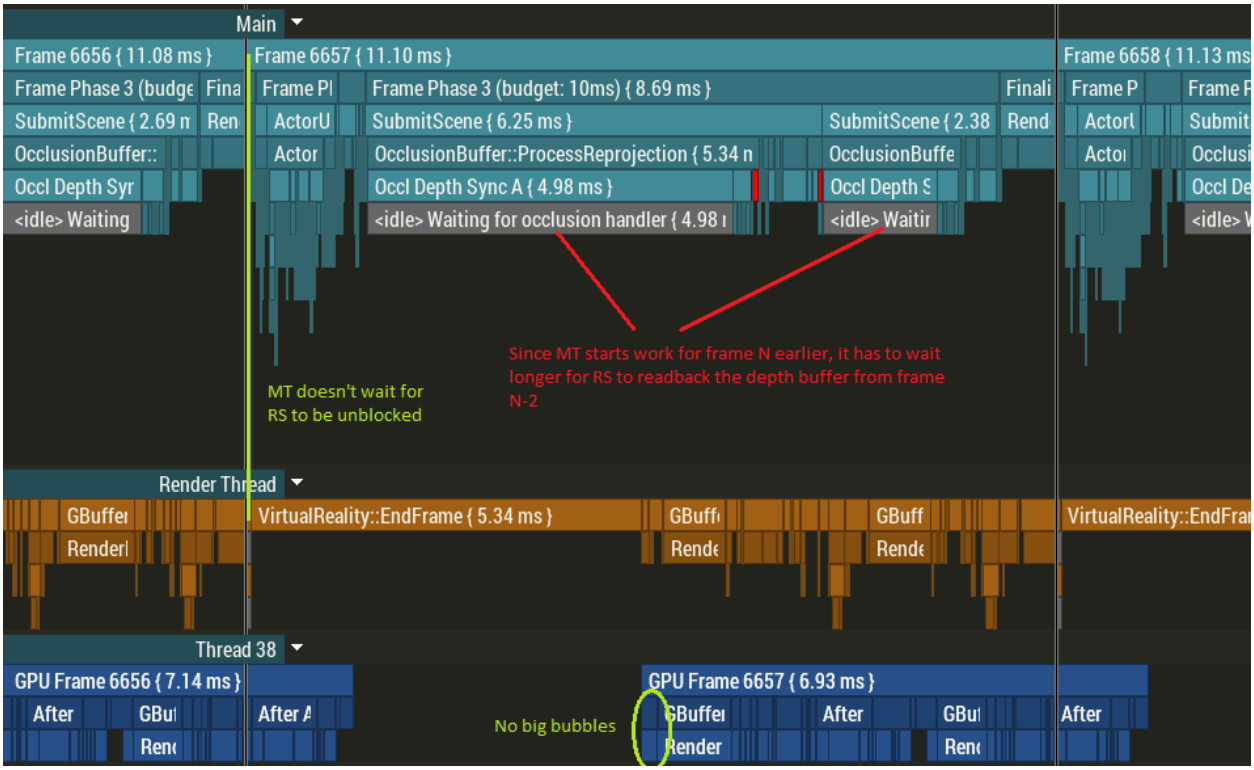
3.2 REMOVING UNNECESSARY ENGINE DEPENDENCIES

We refactored the engine to have RS wake up the MT and then call ovr_submitFrame and mirror-Present. This allowed the MT plenty of time to finish skinning and simulation work, after which it waited on RS to read back the depth buffer from the previous frame (see section 3.3). We added a toggle to let us dynamically switch between the two modes to verify the improvement, and for heavy frame workloads on the low-end rig, it prevented ASW from kicking in frequently.

Here're the GPUView and RAD Telemetry timelines after the fix that removes dependencies (i) and (ii):



After: MT starts before RS is blocked, which reduces the GPU bubbles at the start.



After: Bubbles at the start reduced; wait on depth buffer readback is more pronounced.

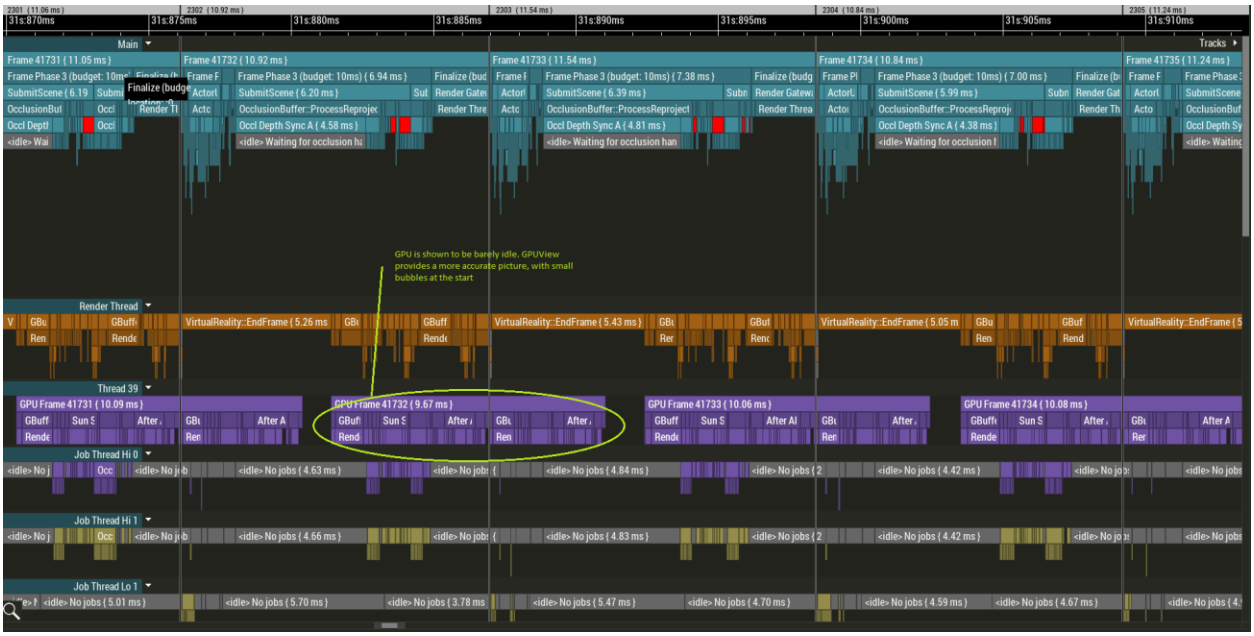
The bubbles in the GPU queue reduced at the beginning of the frame, but we see the occlusion readback system (main thread, working on frame N) waiting on the render thread (submitting work for frame N-1) to do the CPU copy of the staging resource (depth buffer from frame N-2). And that happens again for the right eye's depth buffer.

3.3 ADDRESSING THE OCCLUSION READBACK SYSTEM

While there's a lot of logic that can be shared for occlusion culling per eye, we didn't want the engine to diverge too much for VR. More importantly, MT isn't a bottleneck at all here. The game is still GPU-limited, and removing GPU bubbles lets us meet the 10 ms frame budget.

The reason RS doesn't read back the depth buffer earlier is to reduce the risk of GPU starvation. We want to ensure that the GPU is well fed before RS spends CPU time in copying the depth buffer, during which no GPU work is submitted. Instead of two staggered copies, they could be done back-to-back, after the G-Buffer submission for the left eye is done.

With this simple refactoring change, testing on a minimum spec machine in an area that was dropping a lot of frames **saved a solid 1 ms of GPU time** (bringing it to 9–9.3 ms, allowing for Oculus's post-process work to happen in time) and gave us a resulting 90 fps experience!



After occlusion readback changes: Telemetry timeline shows GPU barely idle.

3.4 TRUSTING YOUR TOOLS

While the Telemetry GPU timeline (which relies on GPU timer queries) shows it barely idle during the frame, GPUView with its Event Tracing for Windows* DirectX events provides a more accurate timeline, which shows us tiny bubbles at the start of the frame. It’s important to understand how various profiling and instrumentation tools work to not be misled by the data.

After occlusion readback changes: Still some idle GPU time to get back. Also, notice a lot of idle time on the various threads.

At this point, RS is not dependent on MT, and these bubbles are caused by resource updates that need to be done before DirectX commands can be dispatched. Resource updates are currently done in a serial manner by RS and don't make use of deferred contexts. While resources that are updated via the Map method need to be done on the immediate context (and hence RS), those using the UpdateSubresource method can be done on a deferred context, and save some valuable driver copy time if done on the job threads when RS is blocked by ovr_submitFrame. This would be a high priority change for any future patch.

4. CPU IDLENESS

In VR, given the harsh 10 ms GPU budget, it makes sense to having a CPU path for subsystems like physics, particle updates, and skinning. The Insomniac engine already used the CPU for simulating the subsystems, and given the small headroom on the GPU, we explored the option of having higher quality presets that improved immersion by making the levels feel alive.

While Oculus' decision to lower the minimum spec for VR is great for adoption, it does come at the cost of gamers with high-end rigs having an experience tailored to the minimum spec. In The Unspoken, the

presets are automatically set based on the underlying system hardware. However, the player is free to modify the quality setting to see what the high-end experience is like, though it may be at the cost of Oculus' runtime turning ASW on for a good chunk of the gameplay.

While an adaptive quality system that dynamically adjusts various rendering parameters is the ideal solution, having quality presets is the first step to give the player the best possible experience.

The ultra preset includes the following:

a) Physics destructibles: More objects in the arena respond physically to the players' spells.
b) Canned animations: While physics-based destruction is great, it can be very expensive to compute. Canned animations can provide a similar experience at a fraction of the runtime cost.
c) Particle effects (ambient and curl-based turbulence): A curl-based turbulence system was added to make particles emanating from the anarchist's hand and the winter wraith's body move in interesting ways.
d) Higher resolution rendering.
e) Temporal anti-aliasing.
f) Ground fog and distant model detail.

5. CONCLUSION

Over the past decade, a lot of game engines have moved work from the CPU to the GPU for subsystems like physics, skinning, particles, occlusion, etc. A 90 fps VR experience imposes a very challenging GPU frame budget of under 10ms. Most game engines have plenty of idle CPU time on various threads that can be used to offload work from the GPU. Since VR capable rigs span a much smaller subset of the PC ecosystem, a measurement driven approach to VR game development can help developers provide gamers the best possible experience.

6. FURTHER READING:

a) Stats exposed by Oculus (via API and HUD)
https://developer3.oculus.com/documentation/pcsdk/latest/concepts/dg-hud/

b) Asynchronous Space Warp (ASW)
https://developer.oculus.com/blog/asynchronous-spacewarp/

c) Controlling ASW via the SDK
https://developer3.oculus.com/documentation/pcsdk/latest/concepts/asynchronous-spacewarp/

If you're attending GDC 2017, you might be interested in the GDC talk on March 3rd (Friday) that covers the content in this article in more detail.

ACKNOWLEDGEMENTS

The author sends a shout out to Bob Sprentall, Yancy Young, Abdul Bezrati and Shaun Mccabe from Insomniac Games and Cristiano Ferreira, Chris Kirkpatrick, Brian Bruning and Dave Astle from Intel.

ABOUT THE AUTHOR

Raja Bala is a Game Developer Relations Engineer at Intel. He enjoys working with studios to improve engine and rendering performance, and educates developers on Intel® graphics architecture and game performance analysis.

Notices