Performance Lessons from Porting Source 2 to Vulkan

Dan Ginsburg

Overview

- Dota 2 Vulkan Performance Results
- Performance Lessons Learned

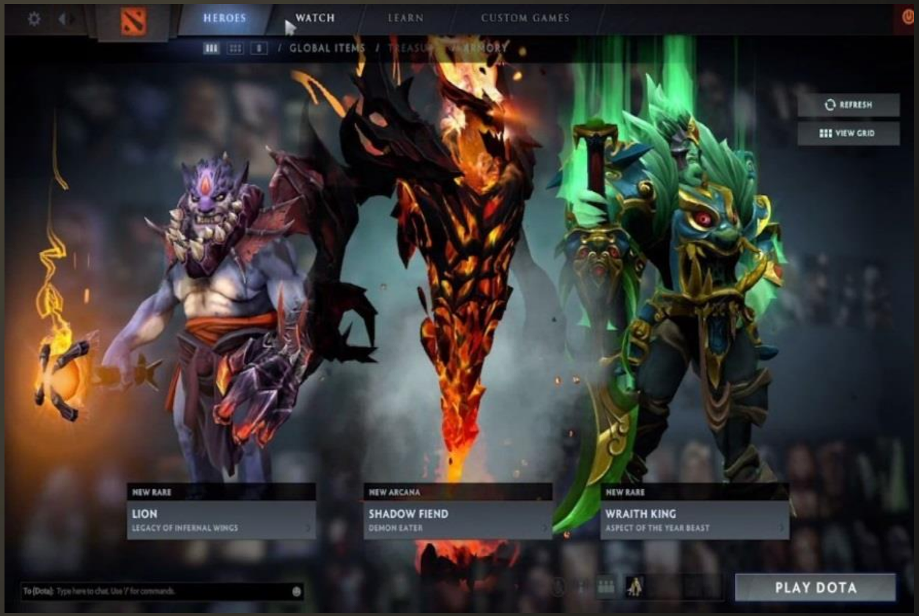
Overview

- Dota 2 Vulkan Performance Results
- Performance Lessons Learned

Source 2 Overview

- OpenGL, Direct3D 9, Direct3D 11, Vulkan
- Windows, Linux, Mac
- Dota 2 Reborn

Heavy Damage
Use a Salve to heal. Taking damage will cancel the healing effect
LUNA
SHOP



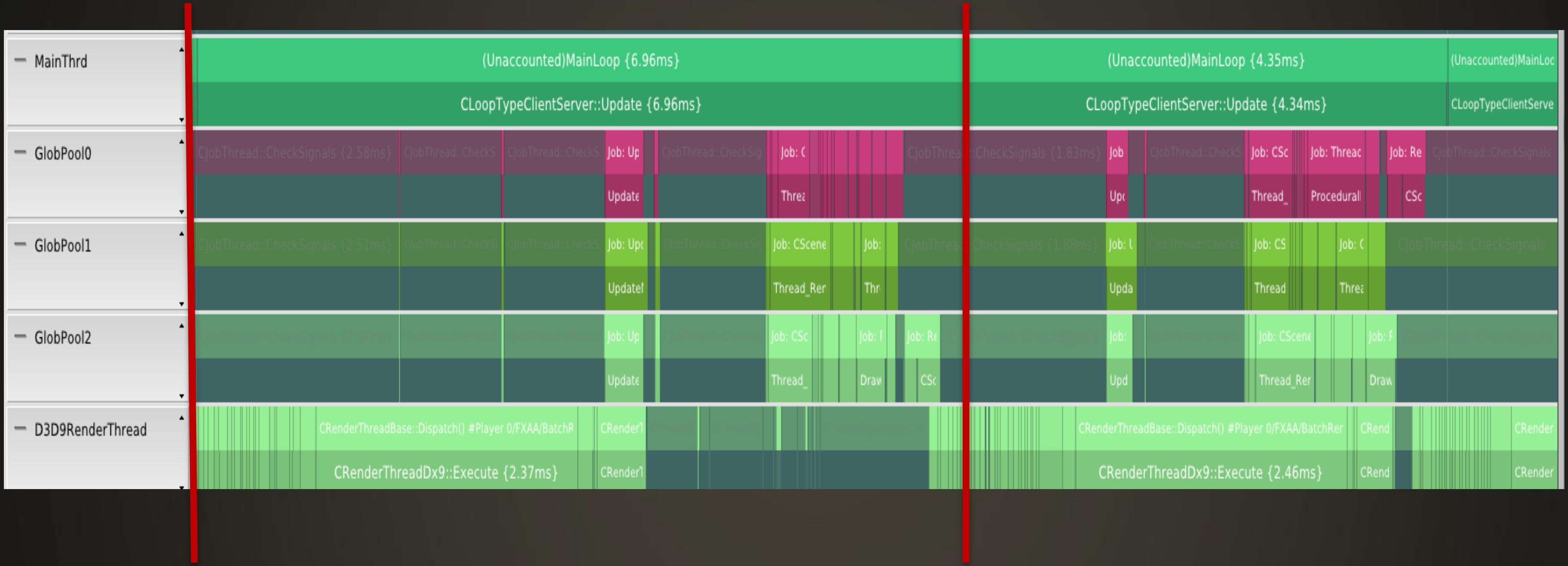
Dota 2 Performance Results - Disclaimer

- Not an ideal showcase for Vulkan
- Source 2 renderer is multithreaded, but…
  - Dota 2 is only ~1500 draw calls per frame
  - Allows DX/GL a frame of latency to avoid being renderthread bound
  - Does not (yet!) take advantage of:
    - Baking descriptors
    - Command buffer resubmission

Dota 2 Performance Results - Disclaimer

- Not an ideal showcase for Vulkan
- Source 2 renderer is multithreaded, but…
  - Dota 2 is only ~1500 draw calls per frame
  - Allows DX/GL a frame of latency to avoid being renderthread bound
  - Does not (yet!) take advantage of:
    - Baking descriptors
    - Command buffer resubmission
- Still very pleased with results!

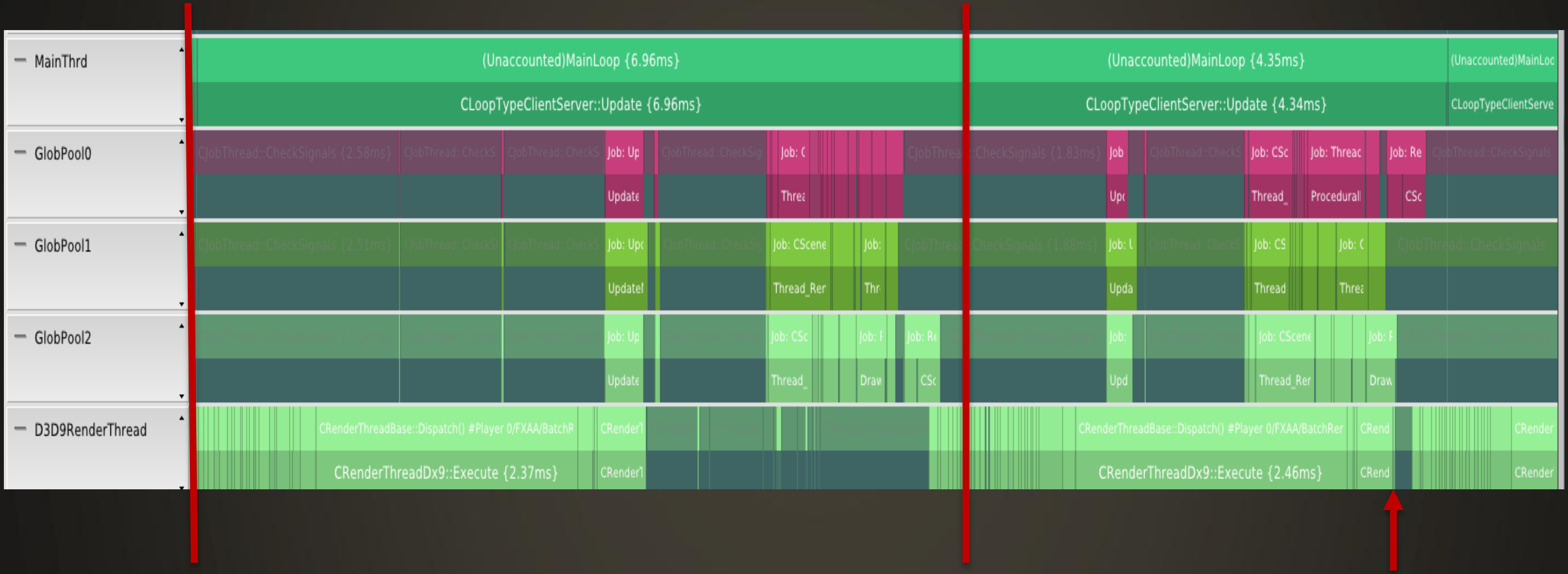
Dota 2 Vulkan Performance – DX9 Latency


MainThrd
GlobPool0
GlobPool1
GlobPool2
D3D9RenderThread
(Unaccounted)MainLoop {6.96ms}
CLoopTypeClientServer::Update {6.96ms}
(Unaccounted)MainLoop {4.35ms}
CLoopTypeClientServer::Update {4.34ms}
CJobThread::CheckSignals {2.58ms}
CJobThread::CheckSignals {1.83ms}
CJobThread::CheckSignals {2.51ms}
CJobThread::CheckSignals {1.88ms}
CRenderThreadBase::Dispatch() #Player 0/FXAA/BatchR
CRenderThreadDx9::Execute {2.37ms}
CRenderThreadDx9::Execute {2.46ms}

Frame Start

Frame End

Dota 2 Vulkan Performance – DX9 Latency

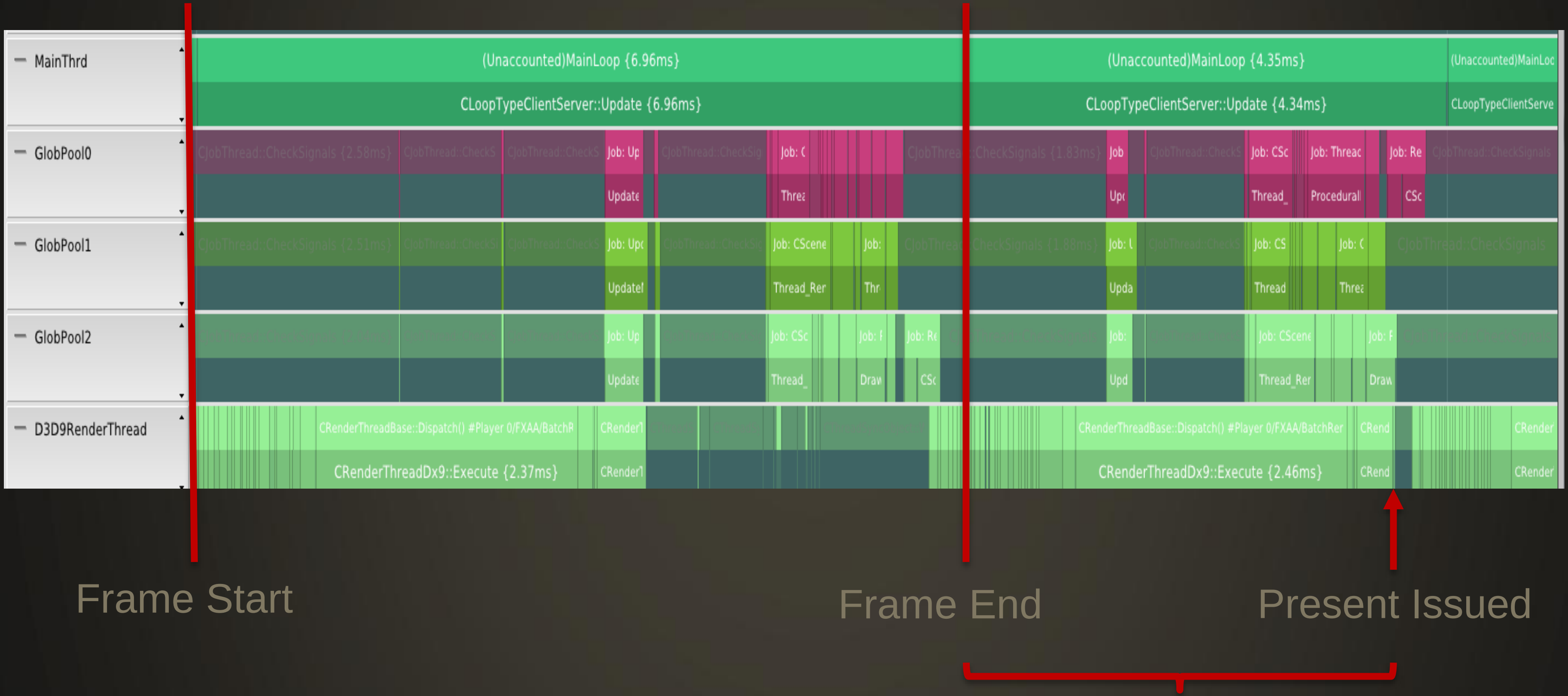

MainThrd
GlobPool0
GlobPool1
GlobPool2
D3D9RenderThread
(Unaccounted)MainLoop {6.96ms}
CLoopTypeClientServer::Update {6.96ms}
(Unaccounted)MainLoop {4.35ms}
CLoopTypeClientServer::Update {4.34ms}
CJobThread::CheckSignals {2.58ms}
CJobThread::CheckSignals {1.83ms}
CJobThread::CheckSignals {2.51ms}
CJobThread::CheckSignals {1.88ms}
CRenderThreadBase::Dispatch() #Player 0/FXAA/BatchR
CRenderThreadDx9::Execute {2.37ms}
CRenderThreadDx9::Execute {2.46ms}

Frame Start

Frame End

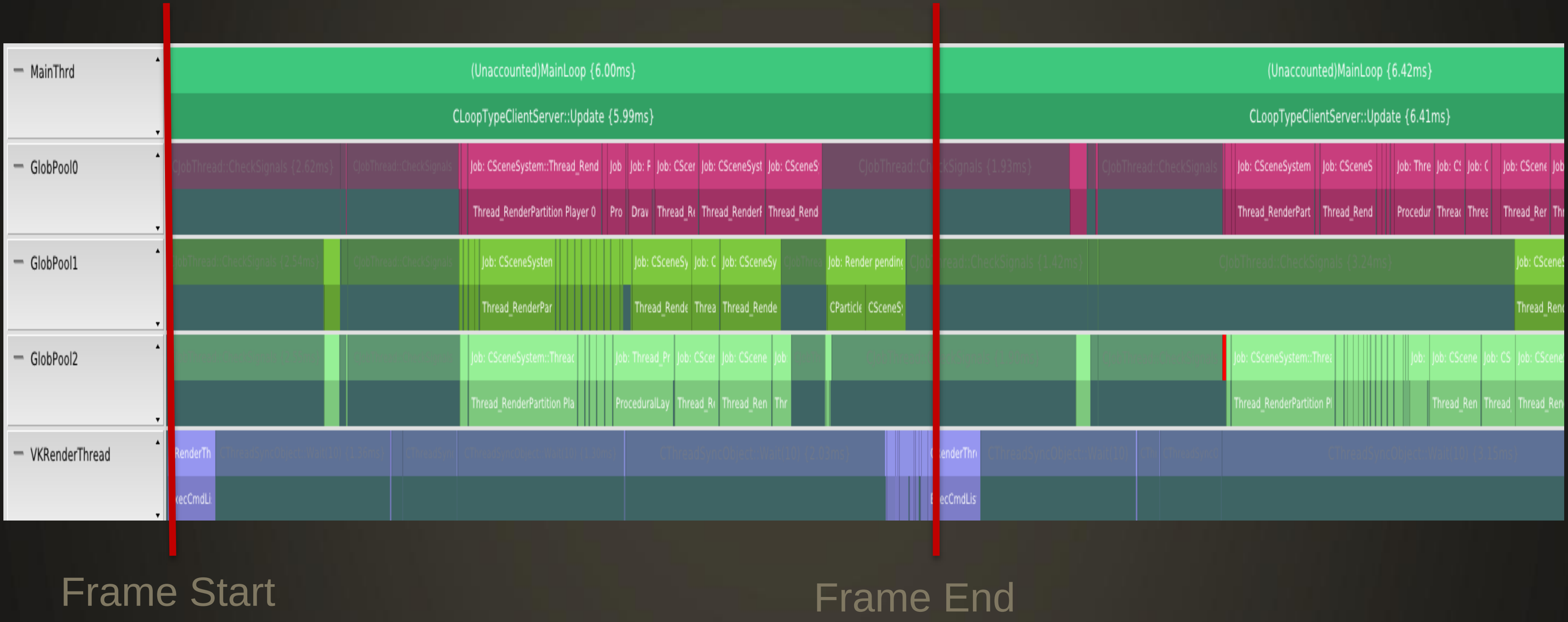
Present Issued

Dota 2 Vulkan Performance – DX9 Latency

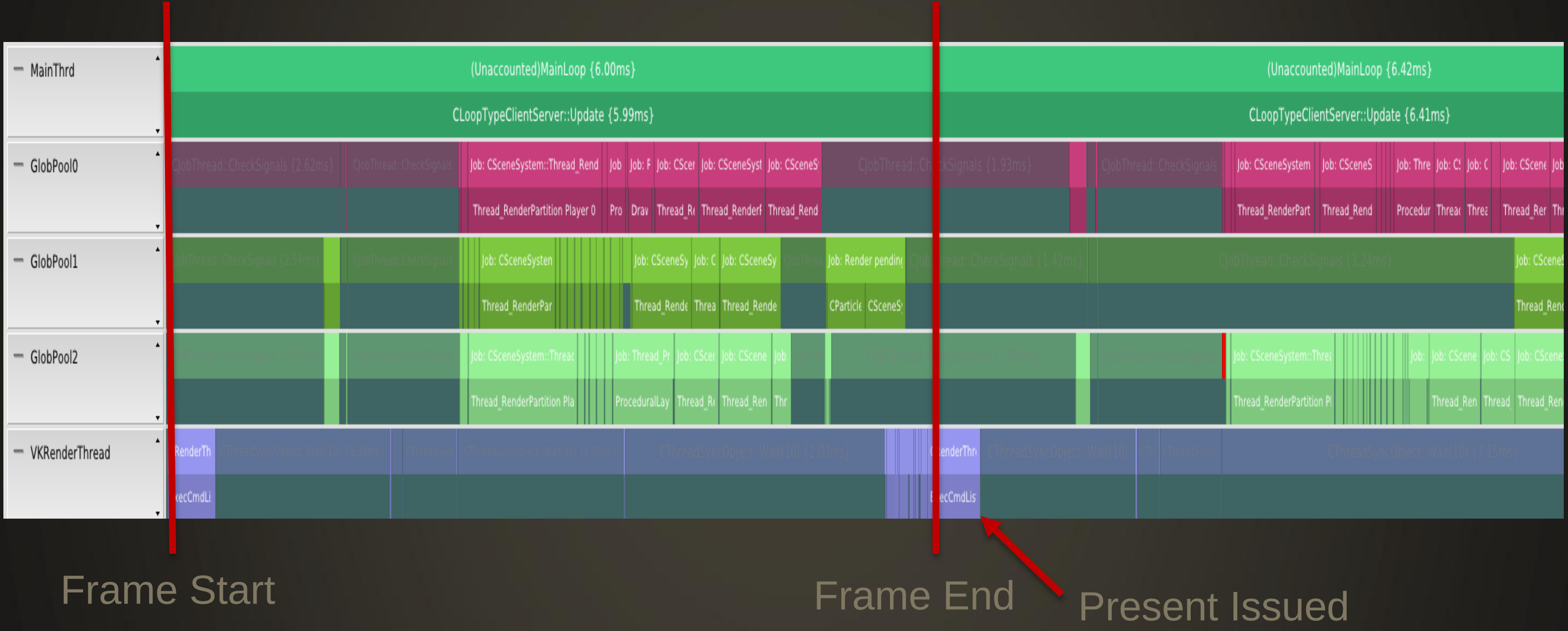

MainThrd
GlobPool0
GlobPool1
GlobPool2
D3D9RenderThread
(Unaccounted)MainLoop {6.96ms}
CLoopTypeClientServer::Update {6.96ms}
(Unaccounted)MainLoop {4.35ms}
CLoopTypeClientServer::Update {4.34ms}
CJobThread::CheckSignals {2.58ms}
CJobThread::CheckSignals {1.83ms}
CJobThread::CheckSignals {2.51ms}
CJobThread::CheckSignals {1.88ms}
CRenderThreadBase::Dispatch() #Player 0/FXAA/BatchR
CRenderThreadDx9::Execute {2.37ms}
CRenderThreadBase::Dispatch() #Player 0/FXAA/BatchRer
CRenderThreadDx9::Execute {2.46ms}
Frame Start
Frame End
Present Issued

DX9 Latency: 3.8ms

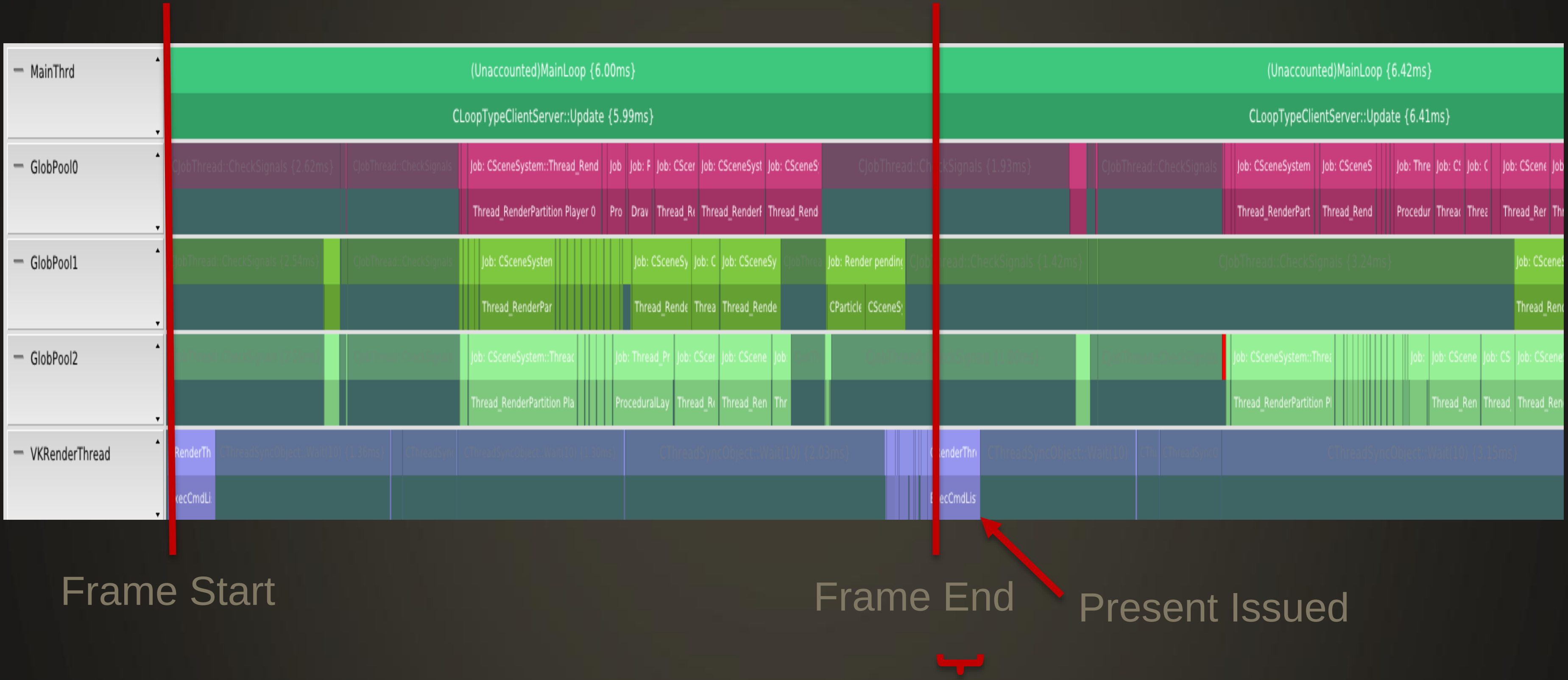
Dota 2 Vulkan Performance – Vulkan Latency


MainThrd
GlobPool0
GlobPool1
GlobPool2
VKRenderThread
(Unaccounted)MainLoop (6.00ms)
CLoopTypeClientServer::Update (5.99ms)
(Unaccounted)MainLoop (6.42ms)
CLoopTypeClientServer::Update (6.41ms)
Job: CSceneSystem::Thread_Rend
Thread_RenderPartition Player 0
CJobThread::CheckSignals (1.93ms)
CJobThread::CheckSignals (3.24ms)
Job: Render pending
CThreadSyncObject::Wait(10) (2.83ms)
CThreadSyncObject::Wait(10) (3.15ms)
Frame Start
Frame End

Dota 2 Vulkan Performance – Vulkan Latency


MainThrd
GlobPool0
GlobPool1
GlobPool2
VKRenderThread
(Unaccounted)MainLoop {6.00ms}
CLoopTypeClientServer::Update {5.99ms}
(Unaccounted)MainLoop {6.42ms}
CLoopTypeClientServer::Update {6.41ms}
CJobThread::CheckSignals {2.62ms}
CJobThread::CheckSignals {1.93ms}
Job: CSceneSystem::Thread_Rend
Thread_RenderPartition Player 0
CJobThread::CheckSignals {3.24ms}
Job: Render pending
CThreadSyncObject::Wait(10) {2.83ms}
Frame Start
Frame End
Present Issued

Dota 2 Vulkan Performance – Vulkan Latency


MainThrd
GlobPool0
GlobPool1
GlobPool2
VKRenderThread
(Unaccounted)MainLoop (6.00ms)
CLoopTypeClientServer::Update (5.99ms)
(Unaccounted)MainLoop (6.42ms)
CLoopTypeClientServer::Update (6.41ms)
Job: CSceneSystem::Thread_Rend
Thread_RenderPartition Player 0
Job: Render pending
CThreadSyncObject::Wait(10) (2.83ms)
Frame Start
Frame End
Present Issued

Vulkan Latency: 0.4ms (!)

Dota 2 Vulkan – Latency Reduction

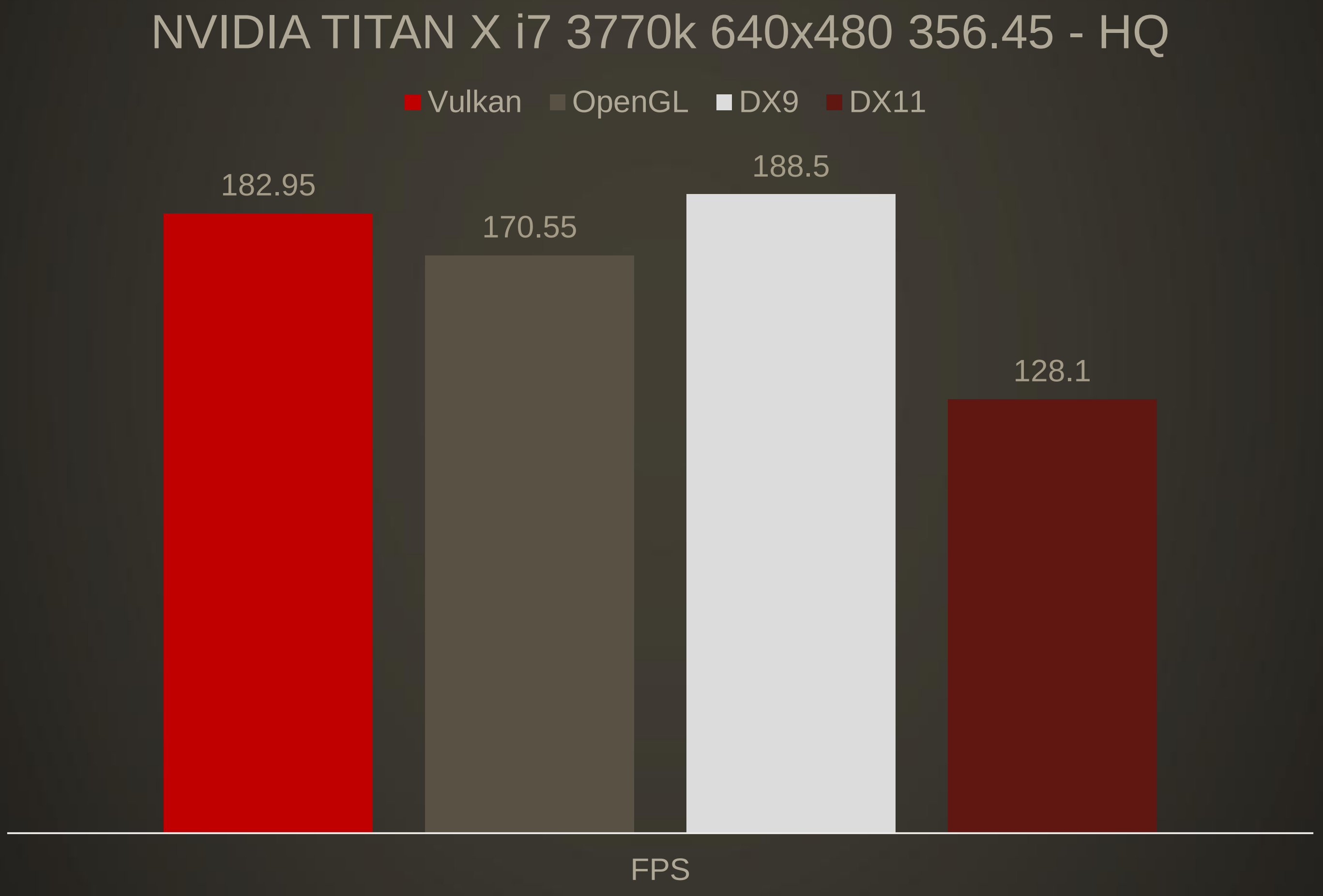
- Renderthread no longer a bottleneck
- Reduces “wallclock” time of frame
  - Time from end of frame to present reduced by 3.4ms
- Really important for:
  - Latency sensitive games (eSports)
  - VR

Dota 2 Vulkan - Framerate

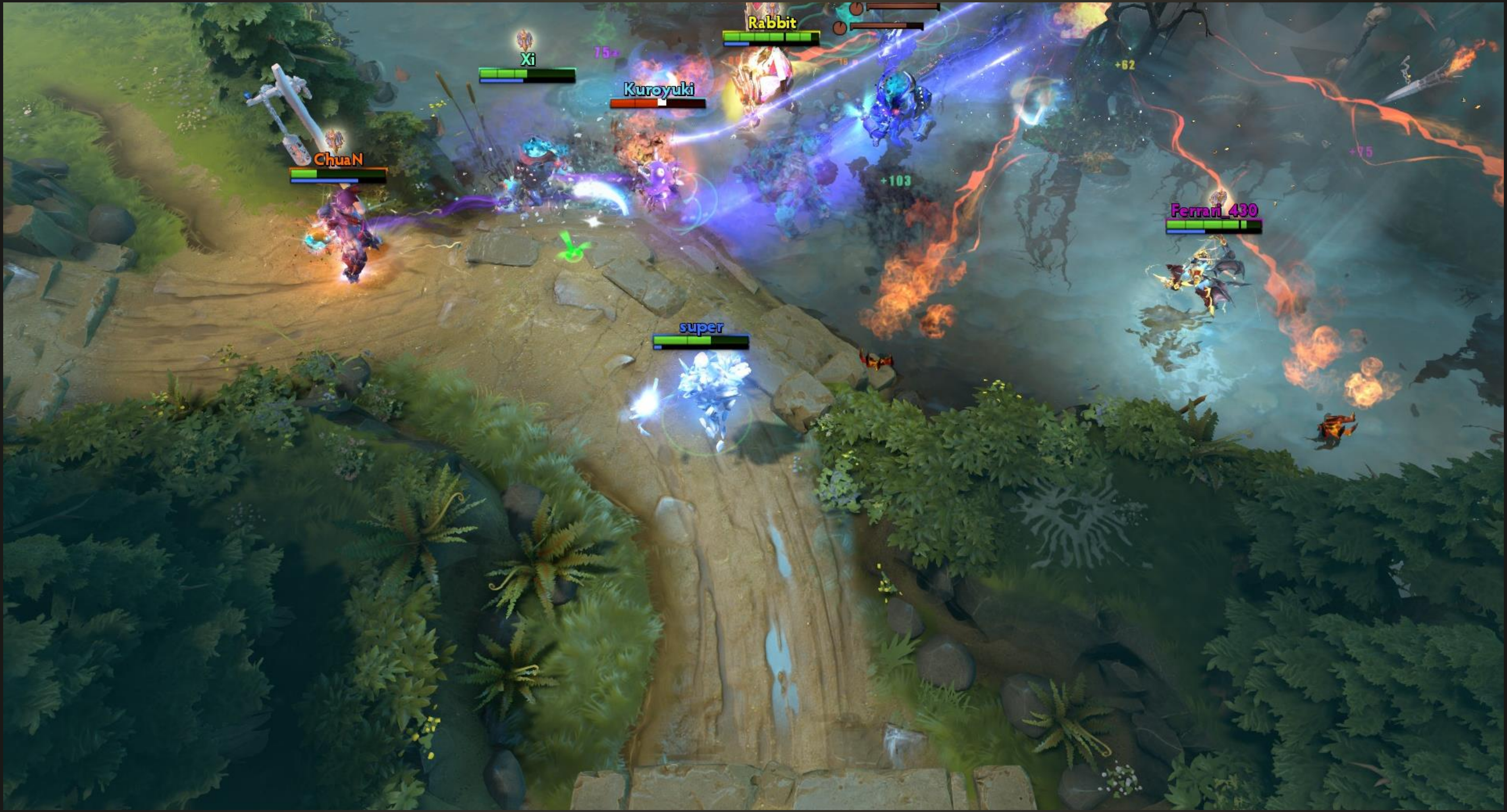
- Two timedemos:
  - Typical Dota 2 Match
  - High Drawcall Battle Scene
- Test system:
  - NVIDIA TITAN X 356.45
  - i7-3770k @ 3.50GHz
- Test settings:
  - Resolution: 640x480 (CPU Perf)
  - Highest Rendering Quality
  - Vulkan/GL/DX9/DX11

Dota 2 Timedemo – Typical Dota 2 Match

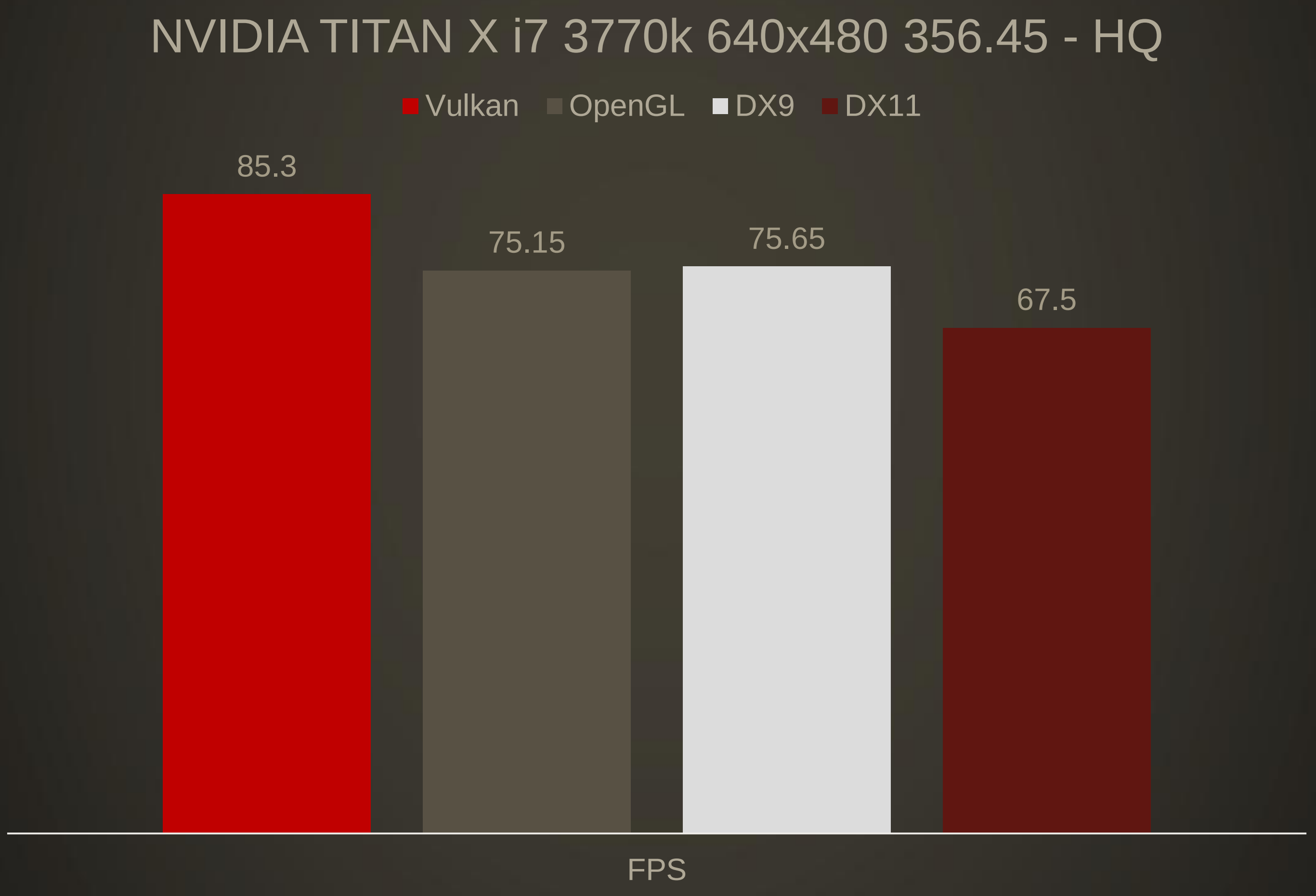
Dota 2 Timedemo – Typical Dota 2 Match



Dota 2 Timedemo – Battle Scene



Dota 2 – High Drawcall Timedemo



Dota 2 Vulkan Performance - Overall

- Significant latency reduction
- Improved framerate in heavy scenes
- Only going to get better…

Overview

- Dota 2 Vulkan Performance Results
- Performance Lessons Learned

Overview

- Dota 2 Vulkan Performance Results
- Performance Lessons Learned
  - Command Buffer Recycling
  - Command Buffer Batching
  - Redundant Call Filtering
  - Updating Descriptors
  - Pipeline Cache Usage

Command Buffer Recycling Overview

- At least one VkCommandPool per thread
- Recycling options:
  - vkResetCommandPool – resets all command buffers in pool
  - vkResetCommandBuffer – reset single command buffer
- Reset can either recycle or release resources

Command Buffer Recycling

- Souce 2 recycles individual command buffers after completion
- vkBeginCommandBuffer costly
    - Using VK_COMMAND_BUFFER_RESET_RELEASE_RESOURCES_BIT
    - Driver reallocates resources
    - Done to reduce memory footprint, but came at perf cost

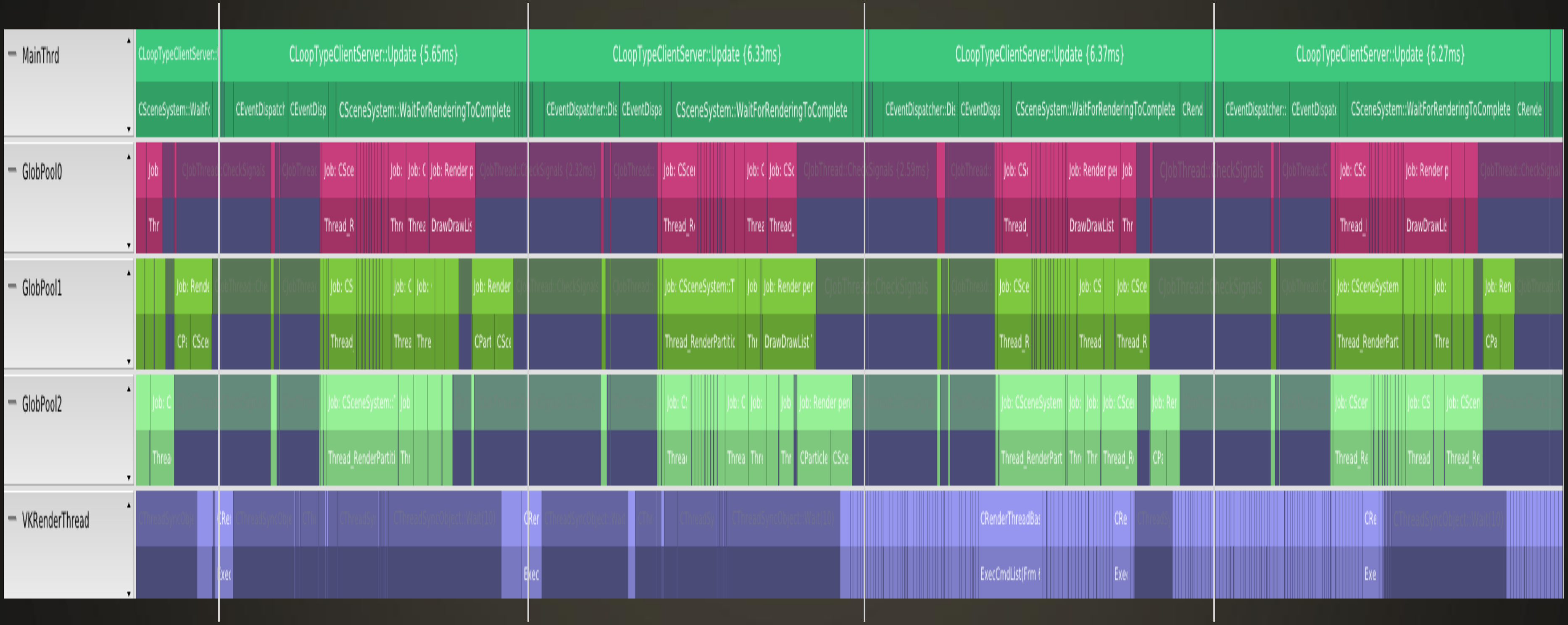
Fast Command Buffer Recycling

- vkCreateCommandPool
  - Use VK_COMMAND_POOL_CREATE_RESET_COMMAND_BUFFER_BIT
- vkResetCommandBuffer(pCmdBuffer, 0)
  - flags == 0, keeps resources for reuse
  - Downside: memory growth
- Source 2 strategy for handling memory growth:
  - Destroy command buffers no longer needed
  - Heuristic to destroy command buffers

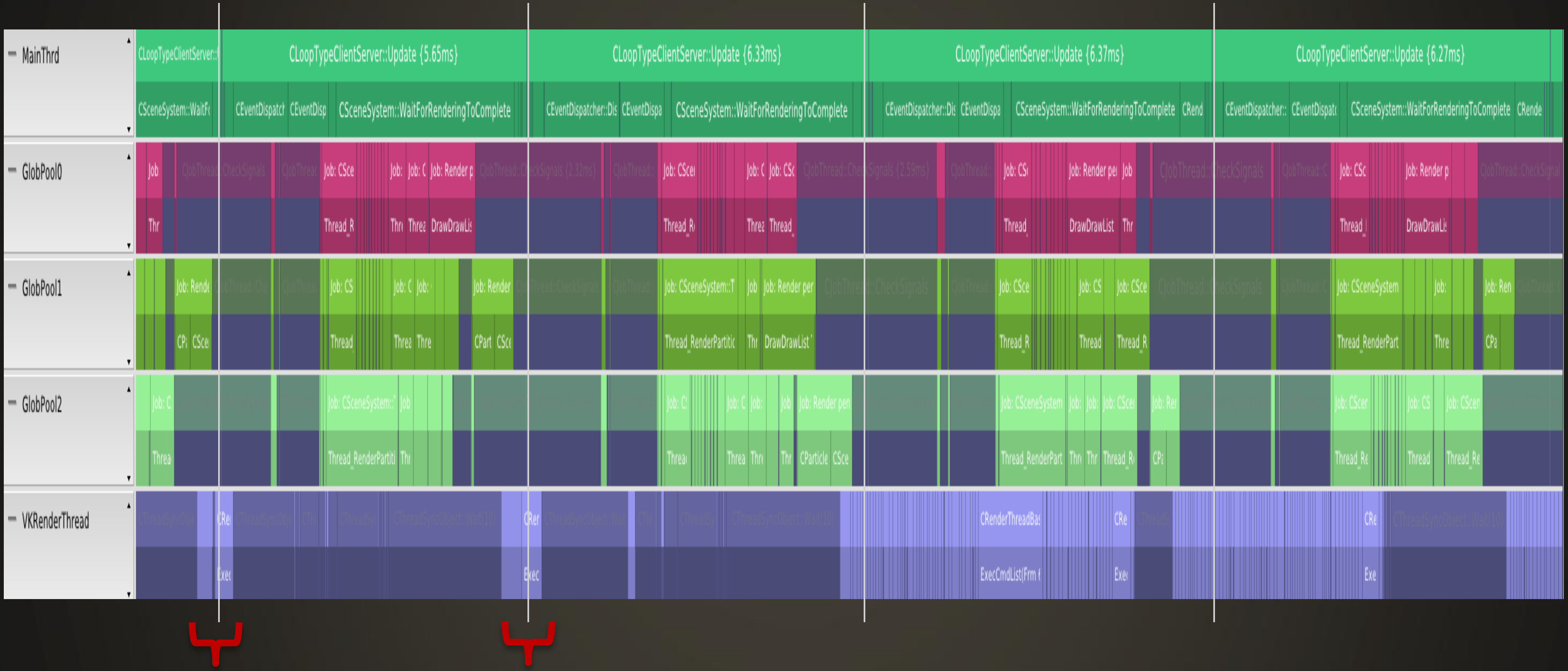
Command Buffer Batching

- vkQueueSubmit implies a flush
  - Also has CPU costs – memory residency
- Important to batch submits

Command Buffer Batching

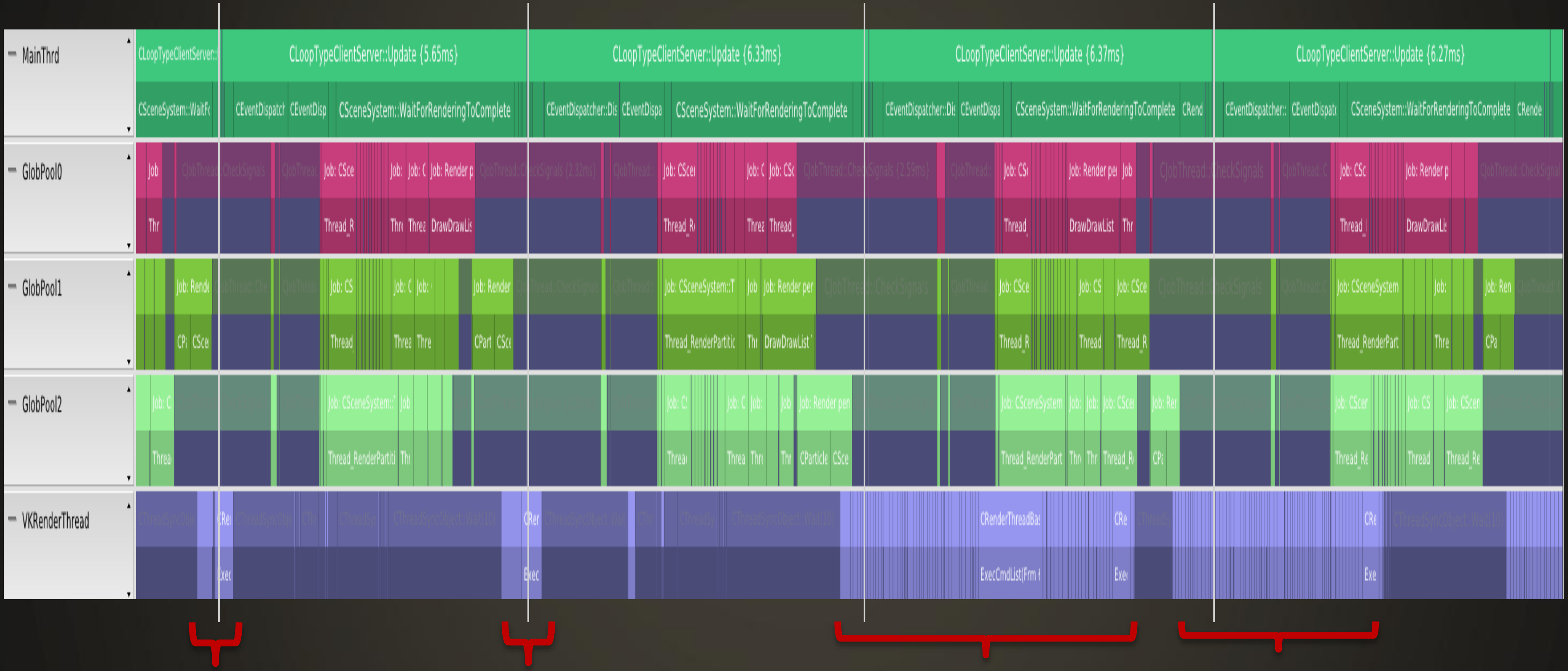

MainThrd
GlobPool0
GlobPool1
GlobPool2
VKRenderThread
CLoopTypeClientServer::Update (5.65ms)
CLoopTypeClientServer::Update (6.33ms)
CLoopTypeClientServer::Update (6.37ms)
CLoopTypeClientServer::Update (6.27ms)
CSceneSystem::WaitForRenderingToComplete
CEventDispatcher::Dis
CRenderThreadBa
ExecCmdList(Frm

Command Buffer Batching


MainThrd
GlobPool0
GlobPool1
GlobPool2
VKRenderThread
CLoopTypeClientServer::Update (5.65ms)
CLoopTypeClientServer::Update (6.33ms)
CLoopTypeClientServer::Update (6.37ms)
CLoopTypeClientServer::Update (6.27ms)
CSceneSystem::WaitForRenderingToComplete
CRenderThreadBa
ExecCmdList(Frm

Batched submit: ~0.7ms / frame

Command Buffer Batching


MainThrd
GlobPool0
GlobPool1
GlobPool2
VKRenderThread
CLoopTypeClientServer::Update (5.65ms)
CLoopTypeClientServer::Update (6.33ms)
CLoopTypeClientServer::Update (6.37ms)
CLoopTypeClientServer::Update (6.27ms)
CSceneSystem::WaitForRenderingToComplete
CRenderThreadBa
ExecCmdList(Frm

Batched submit: ~0.7ms / frame

Unbatched submits: ~4.5ms / frame

Source 2 Command Buffer Batching

- Gather command buffers on renderthread
  - Up to a threshold, needed during load time
- Wait for present request
- Issue single submit with all batched command buffers

Redundant Call Filtering

- Your job now!
  - Vulkan drivers may not (should not!) filter calls
  - If we don’t do it, we will force IHVs to
  - Hurts the good apps at the expense of the bad
- Examples from Source 2:
  - vkCmdBindIndexBuffer
  - vkCmdBindVertexBuffers
  - vkCmdBindPipeline
  - Dynamic render state
    - vkCmdSet*

Updating Descriptors

- vkUpdateDescriptorSets #1 hotspot
- vkCmdBindDescriptorSets #2 hotspot
- Source 2 approach:
  - Single pipeline layout shared across all pipelines
  - Descriptor sets will have unused entries
  - Update/bind descriptor set per draw
  - Not efficient!

Updating Descriptors – The Right Way

- In shaders, organize descriptor sets by update frequency
- Bake descriptor sets up front
- Use compatible pipeline layouts to simplify descriptor allocation

Updating Descriptors – The Right Way

- In shaders, organize descriptor sets by update frequency
- Bake descriptor sets up front
- Use compatible pipeline layouts to simplify descriptor allocation

- …we plan to do this in the future. Will help perf a lot.

Pipeline Creation

- vkCreateShaderModule is relatively fast
  - Loads in the SPIR-V, no heavy compilation
  - ~0.01ms in Dota 2
- vkCreateGraphicsPipelines is expensive
  - Driver performs shader compile here
  - 0.2 – 152ms in Dota 2 before cache is warmed

Vulkan Pipeline Cache

- Serialize compiled pipelines to disk
  - Preload to remove first-time stutters
  - Header contains VendorID/DeviceID/UUID
    - Otherwise opaque format
- Avoid unnecessary shader compiles
  - Driver de-duplicates
  - Only driver knows when recompile is needed based on state
  - Pipeline cache should contain only unique pipelines
- Allows compilation on multiple threads
  - Merge later using vkMergePipelineCaches

Summary

- Dota 2 Vulkan Performance Results
  - Reduced latency
  - Improved framerate in expensive scenes
- Performance Lessons Learned
  - Command Buffer Recycling
  - Command Buffer Batching
  - Redundant Call Filtering
  - Updating Descriptors
  - Pipeline Cache Usage

Questions?