Vulkan Timeline Semaphores

Jason Ekstrand
September 2018

Current Status of VkSemaphore

- **Current VkSemaphores require a strict signal, wait, signal, wait… pattern**
  - One wait per signal
  - After signal, it cannot be signaled again until after a wait
  - Signal must be queued before wait
  - Only for GPU waits; VkFence is for CPU waits

- **These restrictions were required to make all of the corner-cases well-defined**
  - When is the semaphore reset and can be used again?
  - What happens if you submit another signal while someone is waiting?

- **Behavior has to be well-defined and the same on all platforms**
  - Windows, Linux, etc.
  - Various hardware vendors with different implementation choices

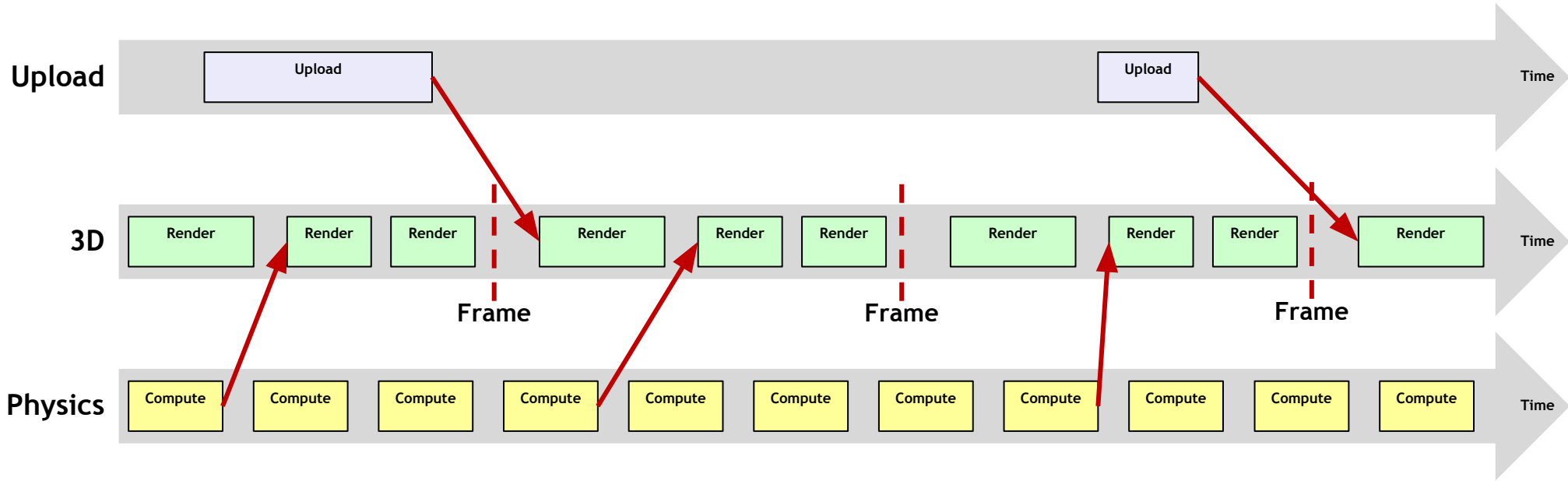
KHRONOS GROUP

An Example of Concurrent Work

Consider a theoretical app with three queues and threads:

- Main 3D render loop, runs once per frame
- Physics model thread, runs once per 5ms
- Resource upload thread, runs on demand

What happens when each thread goes to call vkQueueSubmit...



An Example of Concurrent Work

Upload Thread:

1. Receives a list of resources needed in the near future
2. Create command buffer with upload commands
3. Get a new VkSemaphore
4. vkQueueSubmit on DMA queue
5. Notify render thread and give it the semaphore

Render Thread:

1. Grab a physics snapshot and semaphore
2. Get fresh upload data and associated semaphores
3. Build command buffers to render
4. Get a new VkSemaphore
5. vkQueueSubmit with waits on the physics and upload semaphores and signals the new semaphore
6. Pass new semaphore to WSI

An Example of Concurrent Work

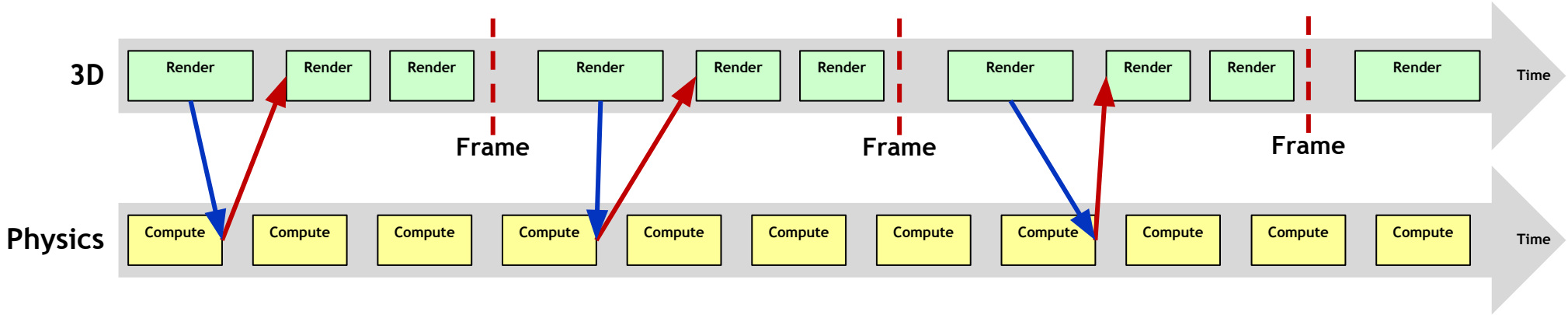
Physics Thread:

1. **Get new data on character motions from engine**
2. **Build command buffers to update physics model**
3. **Get a new VkSemaphore**
4. **vkQueueSubmit with a signal on the semaphore**
5. **Destroy the semaphore?**
    a. Will anyone use this thing?
    b. Is this a snapshot that 3D wants to use?
    c. What if 3D wants to use it twice?
    d. If, I want to destroy it, when is it safe to do so? I have commands pending that will signal it.
    e. Uh….?????

A Partial Solution: More Message Passing!

The easy answer (from a driver perspective) is to tell the app writer that they just need more mutex and condition variable message passing:

- Render thread must notify physics thread of which snapshots it's using
- Physics thread only allocates semaphores for those snapshots
- Due to latency, the render thread choose the physics snapshot as late as possible
- But not too late!
- If the render thread waits too long, it will block the physics thread
- All this message passing is expensive and adds latency!



Finding a Better Solution

- **The Khronos Vulkan working group has been working on a better solution**
  - Talking with game devs about their needs
  - Looking at alternative synchronization models
  - Discussing what's feasible on various platforms

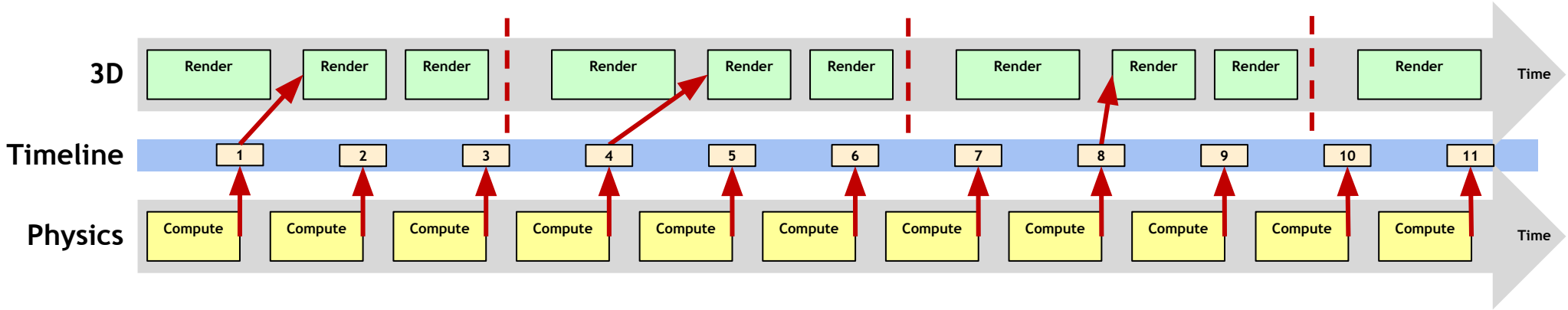
- **Determined we need a new synchronization programming model**

KHRONOS GROUP

Timelines: A New Programming Model

Replace the boolean single-use VkSeamaphore with a timeline:

- **Each timeline carries a monotonically increasing 64-bit value**
- **Signal operations increase the value to some arbitrary new value**
  - The signaler must be aware of the current value to ensure it increases
- **Wait operations wait for the timeline value to be greater than or equal to some client-specified wait value**
  - This implies allowing wait-before-signal behavior
- **Timelines will support both GPU and CPU wait/query**
- **They will be shareable across processes like current semaphores and fences**



Timelines: A New Programming Model

- **More use-cases become well-defined:**
  - Signal-after-signal and wait-after-wait “just work”
  - Can wait on previous signal operations, not just the latest
  - Signal-before-wait is well defined (more on this later)

- **Unifies GPU and CPU waits in a single object**
  - No more separation between VkFence and VkSemaphore

- **Easier for application developers**
  - Most VkSemaphore re-use pools can be replaced with a single timeline
  - In our previous example, each thread would have one timeline
  - Fewer object recycling headaches

Back to Our Example

With timelines, the structure of our example changes:

- **Each thread is associated with a single timeline**
- **The physics thread doesn't have to care about which semaphores get used**
- **Message passing (if any) can be done with simple atomics**
- **Threads can easily check the GPU progress of other threads by querying the timeline counter value**
- **Timeline values can be descriptive:**
  - The physics thread could make it's timeline value represent milliseconds since game launch if it wanted to.

Questions About Timelines?