CGDC

第五届中国游戏开发者大会
CHINA GAME DEVELOPERS CONFERENCE 2012

Stuttering in Game Graphics:
Detection and Solutions

Cem Cebenoyan
Director of Developer Technology, NVIDIA Corporation

汉威信恒
HOWELL EXPO

Stuttering – A Killer to Game Experience

- When people talk to you:
  - "For every few seconds, the game hitches…"
  - "The framerate is high, but it doesn't feel smooth…"
  - "The animation's choppy…"
  - "The response to input lags constantly…"
  - ……
- You know it's stuttering, but
  - What's going wrong?
  - It breaks your game in many ways
  - It's hard to find root causes and eliminate

In this talk,

- We are covering:
  - Top stuttering situations in graphics pipe
  - Methods to identify the root causes
  - Mitigation plans

- Not covering:
  - Stutters raised by disk/network IO, sound, and things other than graphics

Agenda

- A quick glimpse into the top stuttering causes
- Stutter diagnosis
- Causes & solutions
- Vsync, SLI & many other things

A Quick Glimpse into the Top Stuttering Causes

The Many Faces of Stutter

- Framerate hitching
  - Appearance: every so often, the framerate freezes and resumes
  - Possible causes: shader compilation, resource updating and/or vidmem paging
- Micro-stuttering
  - Appearance: the frames-per-second is high, but the overall feeling is laggy
  - Possible causes: highly uneven duration of each frame
- Timing discrepancy
  - Appearance: framerate is fine, but animation and simulation are choppy
  - Possible causes: incorrectly measured time interval and frame queuing

CGDC 第五届中国游戏开发者大会 CHINA GAME DEVELOPERS CONFERENCE 2012

Top 5 Stuttering Causes

1. *Shader compilation*
   - The driver translates D3D assembly into machine-level instructions, which will cause stalls
2. *Video memory oversubscription*
   - Heavily host-video memory paging occurs when running out of vidmem
3. *Resource management*
   - Creating, destroying & updating resources may thrash the performance
4. *Queued frames*
   - Uneven workload between CPU & GPU requires buffering, but which can also raise timing issues
5. *Improper queries*
   - Event & occlusion queries may change the default driver behavior, and sometimes block pipeline

Stutter Diagnosing

Identify Stuttering

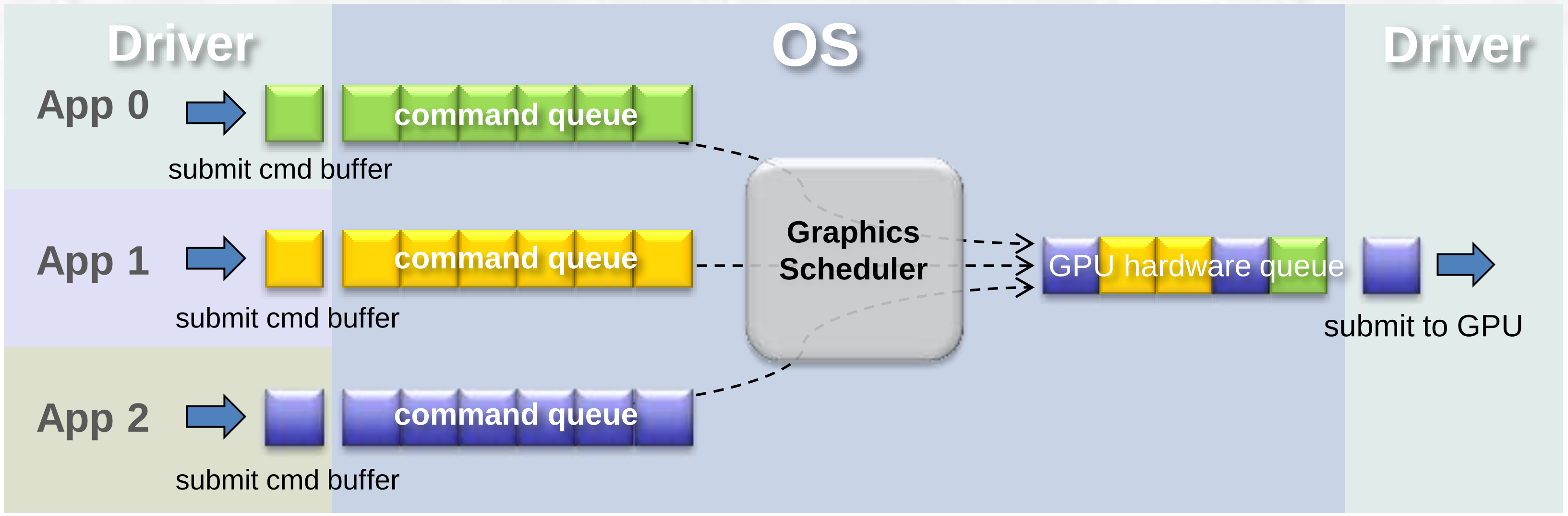
- Identifying stuttering is hard
  - It may only reproduce on some hardware under certain conditions
  - No convenient way to capture data for analysis
- Need to combine various tools and experiments for analysis
- Before covering the details, a few things to understand:
  - CPU/GPU communication
  - Windows Display Driver Model

CGDC

第五届中国游戏开发者大会

CHINA GAME DEVELOPERS CONFERENCE 2012

Preliminary: CPU/GPU Communication



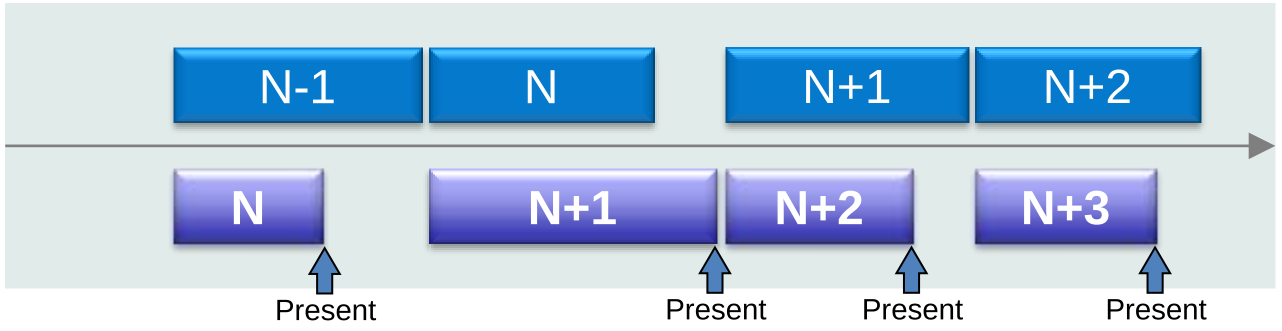
- Each D3D device has a graphics context
  - Maintaining a *command queue*
  - The driver builds API calls into *command buffers* and submits them to the command queue at a proper time
- Many applications share GPU resource
  - Global *graphics scheduler* (in OS) picks packets from many command queues into *GPU hardware queue*
  - GPU processes packets in order and remove them after finish

汉威信恒 HOWELL EXPO

gameworks.nvidia.com

Preliminary: CPU/GPU Communication (cont.)

- Command buffer flush
  - Usually, driver begins submitting (flushing) after Present
  - Sometimes flush happens at other places
- Frame latency
  - With no queued frames, GPU works 1 frame behind CPU



  - In practice, driver may queue up to 3 frames (4 frames behind) before flush

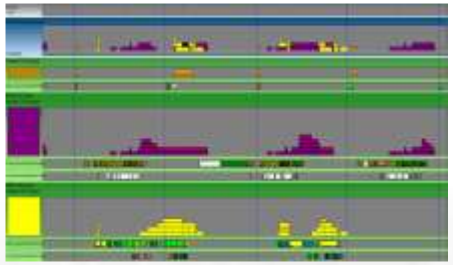
Preliminary: WDDM

- Windows Display Driver Model
  - Introduced since Vista
  - Virtualized video memory, better fault-tolerance, OS scheduled graphics task, …

- It’s two drivers
  - UMD: user mode driver
    Work with applications and D3D runtime
    Build & submit command buffers
  - KMD: kernel mode driver
    Work in OS kernel mode
    Manage hardware resources with OS
  - OS operates command queues between UMD & KMD

Tools for Stutter Diagnosing

- Fraps
  - Framerate recording
  - Quick stats
- Nsight
  - Static & dynamic analysis
  - GPU pipeline inspection
- GPUView
  - In-depth analysis



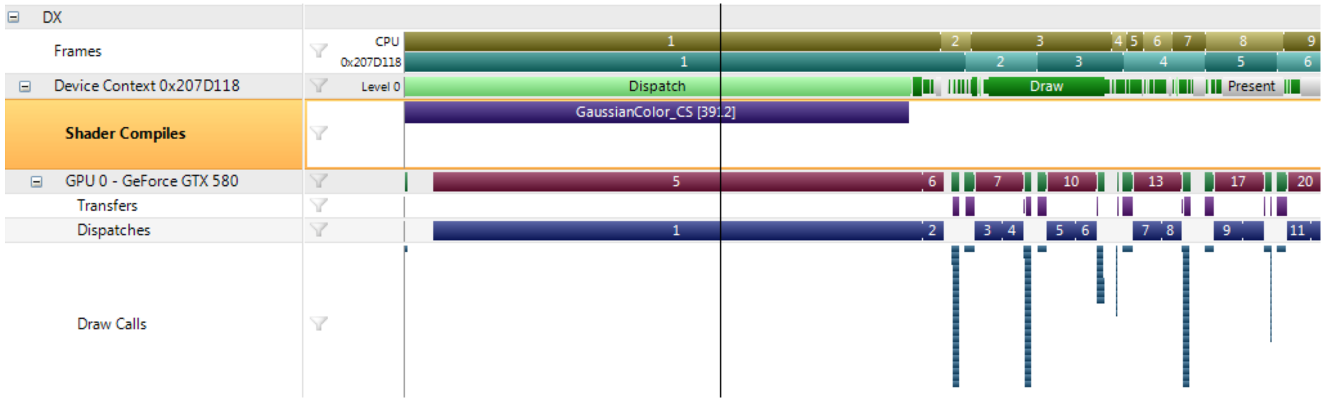
Framerate Hitching Diagnosing

- Appearance
  - Every so often, the framerate freezes and resumes
- Start from recording frametime
  - Add profiling code to game engine, recording the duration of each frame (present-to-present)
  - Or, use Fraps' *frametimes* function to benchmark
  - The lagged frames can be easily spotted
- Check the lagged frames:
  - Create new shaders (new material loaded?)
  - First time using a large chunk of resource (texture, render target, buffers, etc.)?
  - Render thread blocked by resource updating?
  - CPU or GPU has unusually large workload?

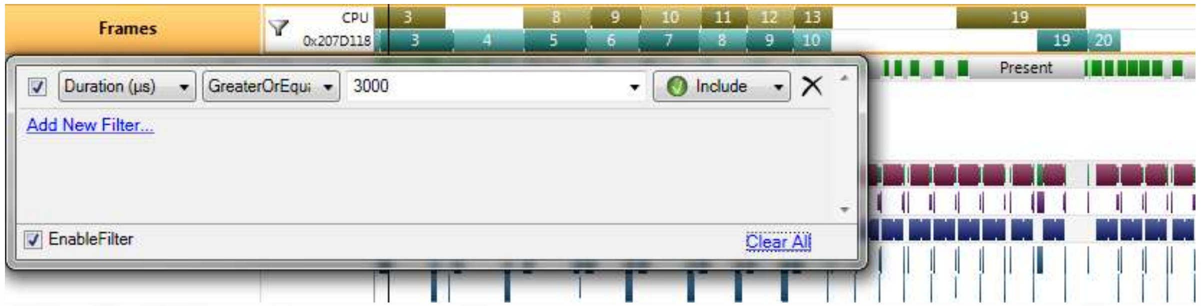
CHINA GAME DEVELOPERS CONFERENCE 2012 第五届中国游戏开发者大会

Framerate Hitching Diagnosing (cont. 1)

- Nsight can help to check lagged frames
  - Using "Trace Application" to record a period of game



  - Shader compilation time shown on the timeline



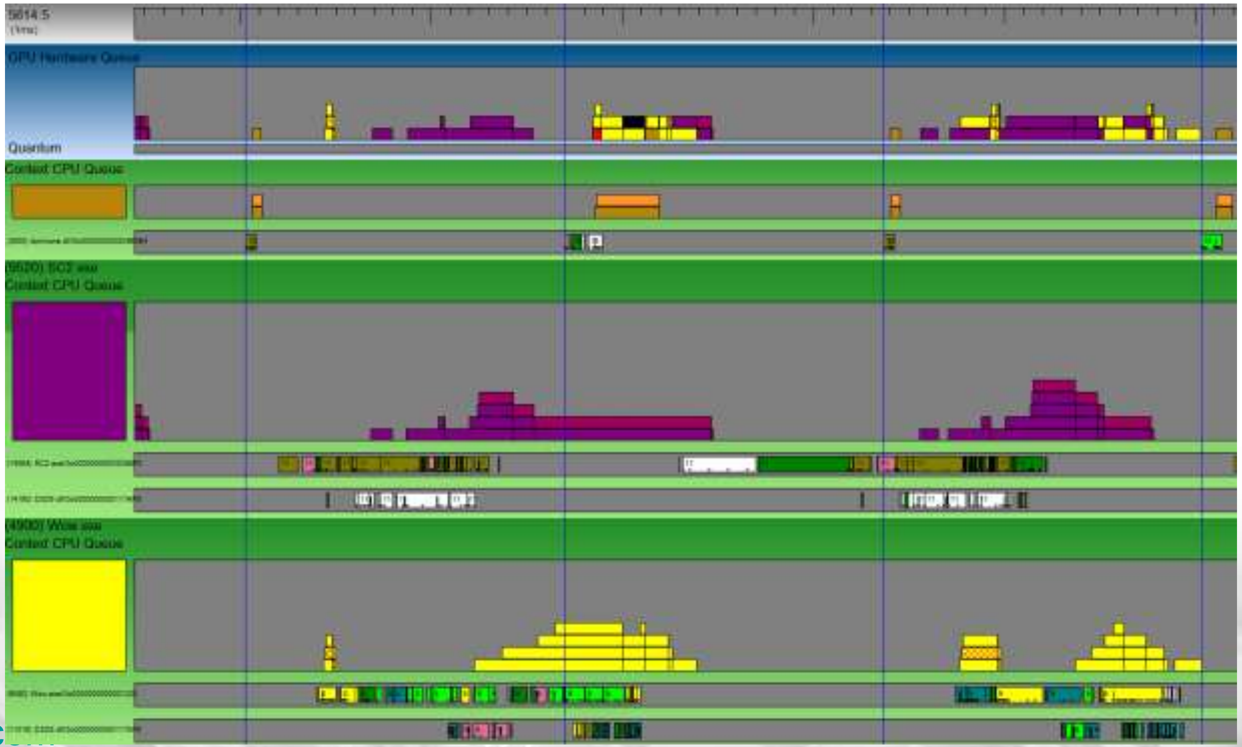
  - Filters allow to single out concerned frames and API calls

Framerate Hitching Diagnosing (cont. 2)

- Experiment with the usual suspects
  - Remove the shader being compiled if found in lagged frames
  - Remove the resources be referenced first time if found in lagged frames
  - Remove any Lock*, Map*, Update* functions if found in lagged frames
- The result of the experiments can show you the causes of stuttering
  - Shader compilation, resource management, etc.
  - We will discuss each cause in next section

Framerate Hitching Diagnosing (cont. 3)

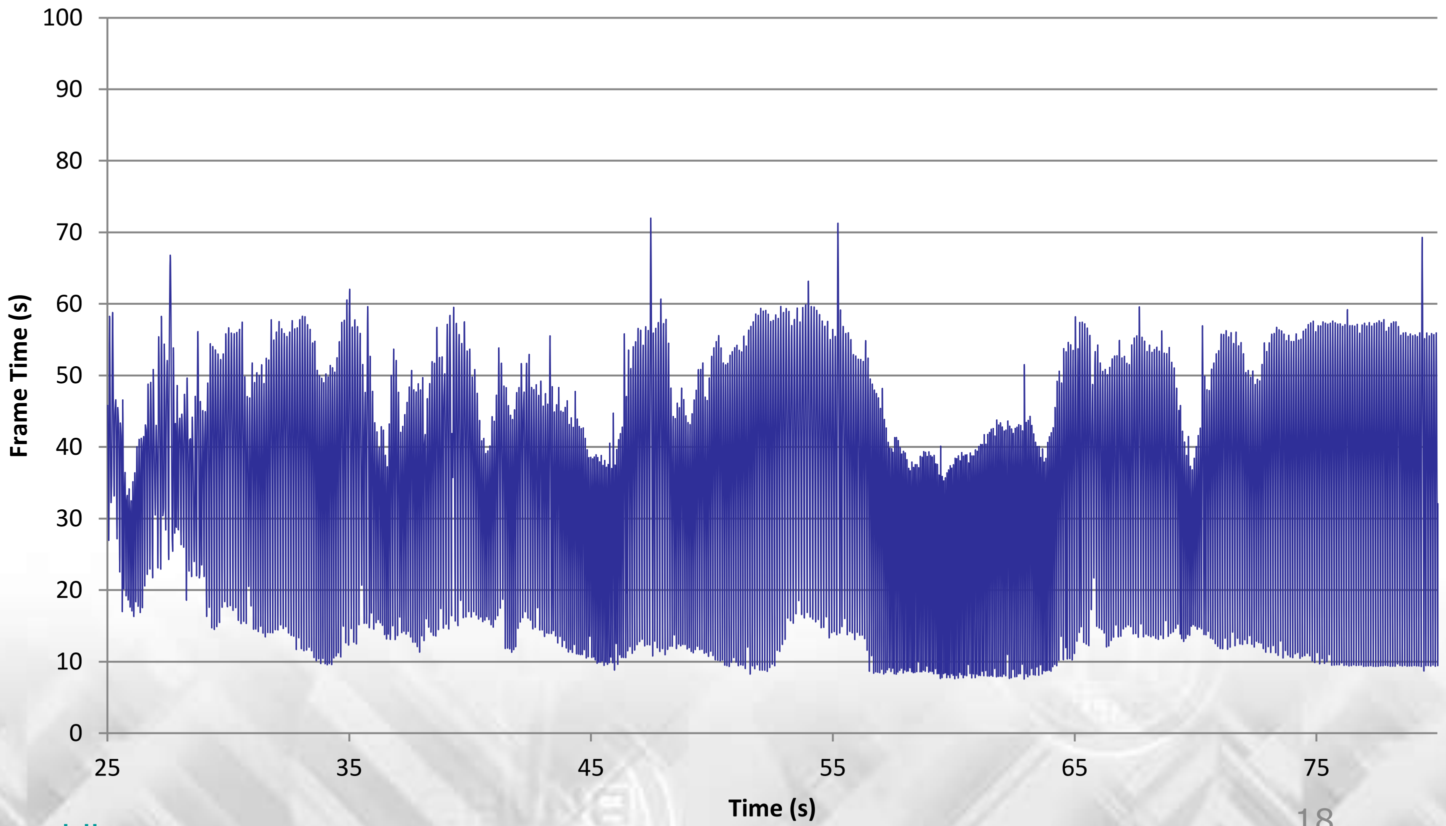
- GPUView is more advanced for inspection
  - *[PRO]* abundant information: process command queues, GPU hardware queue, content of each packet
    Less intrusive when recording
  - *[Con]* very challenging for new users
    Huge data set
  - Able to check entire system. For example: is my app's Present blocked by windows desktop?



Courtesy of
Matt Fisher

Micro-stuttering Diagnosing

- Appearance
  - The frames-per-second is high, but the overall feeling is laggy
- The frametime is extremely uneven



gameworks.nvidia.com

Micro-stuttering Diagnosing (cont. 1)

- Some possible causes
  - Uneven workload: AI, animation tasks not done on CPU for every frame
  - Game engine limits the number of buffered frames, the driver not able to cover non-uniform Present calls
  - Large amount resource updating
  - Video memory oversubscription, and resources are being continuously paged

Micro-stuttering Diagnosing (cont. 2)

- Use Nsight or GPUView to inspect
  - Nsight: check GPU frames
  - GPUView: check the GPU hardware queue

- Check the following facts
  - Is CPU workload very uneven from frame to frame?
  - Is the game engine use some methods to limit the number of queued frames?

  - Uneven CPU frames + no queued frames -> stutter

第五届中国游戏开发者大会 CHINA GAME DEVELOPERS CONFERENCE 2012

Micro-stuttering Diagnosing (cont. 3)

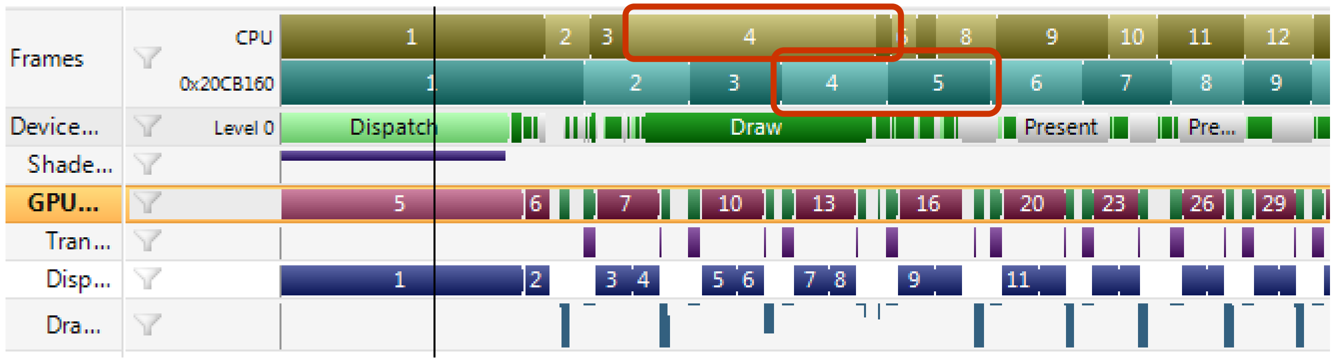
- Detect possible CPU stalls during resource updating
  - Write profiling code to enclose each Lock*, Map* and StretchRect calls. Sometimes CPU is stalled if the required resource is in use by GPU
  - Long CPU stall + queued frames -> stutter
- Estimate vidmem usage on fly
  - Use WMI interface and game engine’s own memory stats to estimate
  - Heavy paging -> stutter
  - But heavy paging only happens when resources largely exceed physical VRAM

Timing Discrepancy Diagnosing

- Appearance
  - Framerate all good, but animation and simulation are choppy
- Possible causes
  - The game engine uses incorrect time interval for scene updating (camera, animation, simulation, etc.)

Timing Discrepancy Diagnosing (cont.)

- Check the game engine's timing system
  - Is it measuring time interval using Present-to-Present time?
  - If so, use Nsight to inspect the timeline



  - CPUs notion of elapsed time is vastly different from GPUs actual elapsed time frame to frame -> animation stutter
  - In this case, CPU side Present-to-Present time is not the real time interval!

CGDC

第五届中国游戏开发者大会

CHINA GAME DEVELOPERS CONFERENCE 2012

Causes & Solutions

汉威信恒
HOWELL EXPO

Scenarios

- Recall the top 5 causes:
  1. *Shader compilation*
  2. *Video memory oversubscription*
  3. *Resource management*
  4. *Queued frames*
  5. *Improper queries*

Shader Compilation Basics

- Why compile shaders at runtime?
  - The driver needs to translate D3D assembly to machine instructions
  - Each GPU generation has drastically different instruction set
- When and how it gets compiled?
  - At the time Create***Shader invoked
  - The driver generates machine instructions once and saves them for later use.
  - For a complex shader, the driver may generate less optimized code first, and replace it with optimized code later

Shader Compilation Basics (cont.)

- How long does it take the driver to compile?
  - Depending on the shader complexity, tens of milliseconds to thousands of milliseconds
- Is there a way to pre-compile shaders and save them to disk?
  - No.
- Is the compilation “once-for-all”?
  - Not really. Some D3D9 state changes may trigger a compiled shader to be recompiled

State Dependent Recompile

- D3D9 states are not well mapped to GPU hardware states
  - Many state changes can trigger shader recompile
  - Earlier GPU generations (D3D9 class GPUs and older) have more such culprits
  - Doesn’t apply to D3D10.x and D3D11.x

State Dependent Recompile (cont.)

- "Dangerous" states (in order of severity)
  - Having shadow map bound/unbound
  - Changing bound texture between FP, non-FP format
  - Changing bound resource format from the compile time
  - sRGB state for render target and texture
  - Same pixel shader for different COLORWRITEENABLE settings
  - Shader contains static branches (using boolean varable): each static branch permutation require a compilation

(On D3D9 class GPUs and older)

- User clip plane
- Fixed function fog parameters
- MRT related states

Shader Compilation: Mitigation

- Old methods are still good:
  - At loading time, create all shaders that will be used in the level
  - Render everything in the scene with at least 1 primitive per mesh
  - For dynamic streaming, render a hidden object that contains mostly used materials at streaming time

- If you cannot do any of the above
  - The driver is okay to compile shaders on fly, just don’t use a shader immediately after its creation
  - Be sure to give the driver 500ms ~ 1000ms to compile the shader between Create***Shader and Set***Shader

Shader Compilation: Mitigation (cont.)

- For state dependent recompile:
  - Group objects by dangerous states
  - Avoid or reduce changing the states
  - Ensure the shader is created and rendered with under the states used most
- If using D3D11, async creates can help
  - Do so consistently since using async creates will turn off certain functionality in the driver. (i.e., don't do it once at startup and never again)
- Do not forget to use Nsight to verify the mitigation of runtime shader compilation

Resource Management Basics

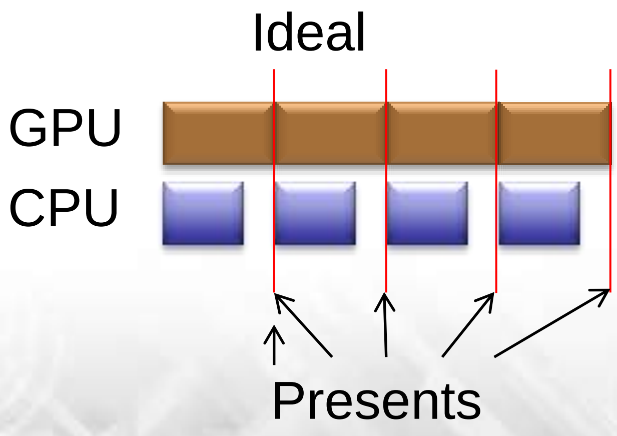
- Creating and destroying resources
  - The memory is not always allocated at creation time, but at the point the resource being referenced first time – may raise a stutter (for Vista and newer OSes)
  - Creating large resources at runtime is a huge cost
  - The release call only drops the reference counter by 1. The resource is destroyed when its counter reaches 0
  - Frequent creating/destroying resources will result in vidmem fragmentation
    - Whenever possible, reuse

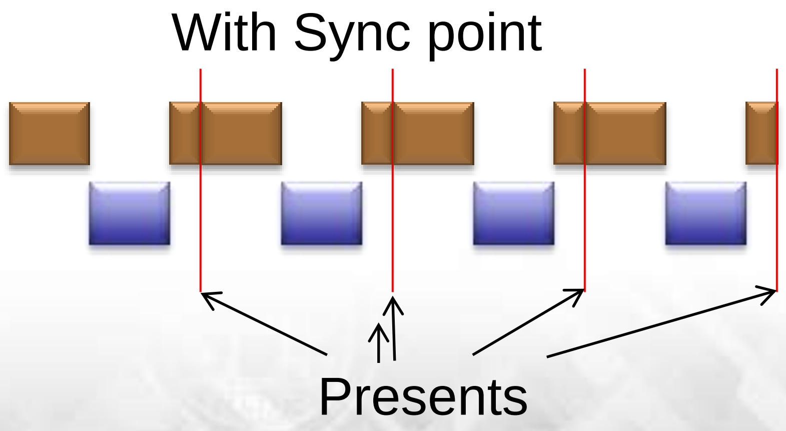
Resource Management Basics (cont. 1)

- CPU-GPU sync point
  - A CPU-GPU sync point is caused when the CPU needs the GPU to complete work before an API call can return
  - One bad sync point may **halve** your frame rate
- Various sync points
  - Immediate update of a buffer still in use by GPU
  - Read back the data in render target you just rendered to
  - Allocating a large resource after release a large resource

Resource Management Basics (cont. 2)

- Why are sync points so bad?
  - Ideal frame time should be max(CPU time, GPU time)
  - CPU-GPU Sync point turns this into CPU Time + GPU Time.
  - A long duration sync-point can introduce stutter





Resource Management Basics (cont. 3)

- Recall that GPU works 1~4 frames behind CPU. A random sync point occurring in a frame means:
  - Flush the command buffer
  - Wait for GPU up to 4 frames!
  - Stutter occurs

- Locking resources in D3D9
  - Locking any buffer with flags=0 guarantees CPU-GPU Sync point if that buffer is still in use.

CGDC

第五届中国游戏开发者大会

CHINA GAME DEVELOPERS CONFERENCE 2012

Resource Management: Mitigation

- General guidance 1
  - Use DISCARD flag when locking/mapping resources

    (When using DISCARD flag, you may get a new buffer instead the one you try to update. The new one will replace the original one later. This avoids sync point, but increases footprint.)

  - Apply DYNAMIC usage for frequently updated buffers and NOOVERWRITE flag during updating

    (The driver tends to place DYNAMIC resource in system memory. But many vertex/index buffers and small textures are fine to stay there.)

汉威信恒 HOWELL EXPO

Resource Management: Mitigation (cont. 1)

- General guidance 2
  - Avoid creating/destroying resource at runtime
  - Try allocating buffers at startup and reusing them at runtime
  - Before reusing a resource, issue a query to check if GPU has finished using it

Resource Management: Mitigation (cont. 2)

- Carefully managing small buffers
  - Animation, particle system, UI elements, etc.
  - Game engine can manage a vidmem pool for resource reusing & updating
- Self managed contention-free buffer
  - Game engine allocates a buffer as a memory pool. Treat it as a heap or circular buffer
  - Keep 3 lists for: free spaces, allocated spaces and spaces to be freed
  - To free a space, put it into to-be-freed list and issue a query to ensure GPU finished using it, and then move it to free-space list

Oversubscription: Mitigation

- If create / destroy is required at runtime:
  - Always destroy *then* create
  - Even momentary oversubscription can cause memory management problems
- Video memory allocation:
  First come, first serve

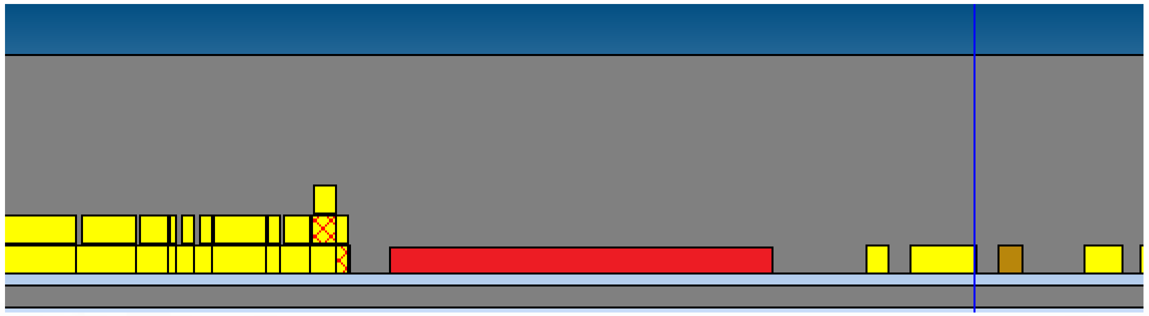
Oversubscription: Mitigation (cont. 1)

- Allocating resources in order of importance:
  1. Depth-stencil surface
  2. Render target
  3. Read-only random access resources: frequently used textures
  4. Read-only streams and less used resources: Vertex buffer, index buffer, small textures

- At same importance, allocating resources in order of size and format:
  – Larger, higher AA and FP format resources first.

Oversubscription: Mitigation (cont. 2)

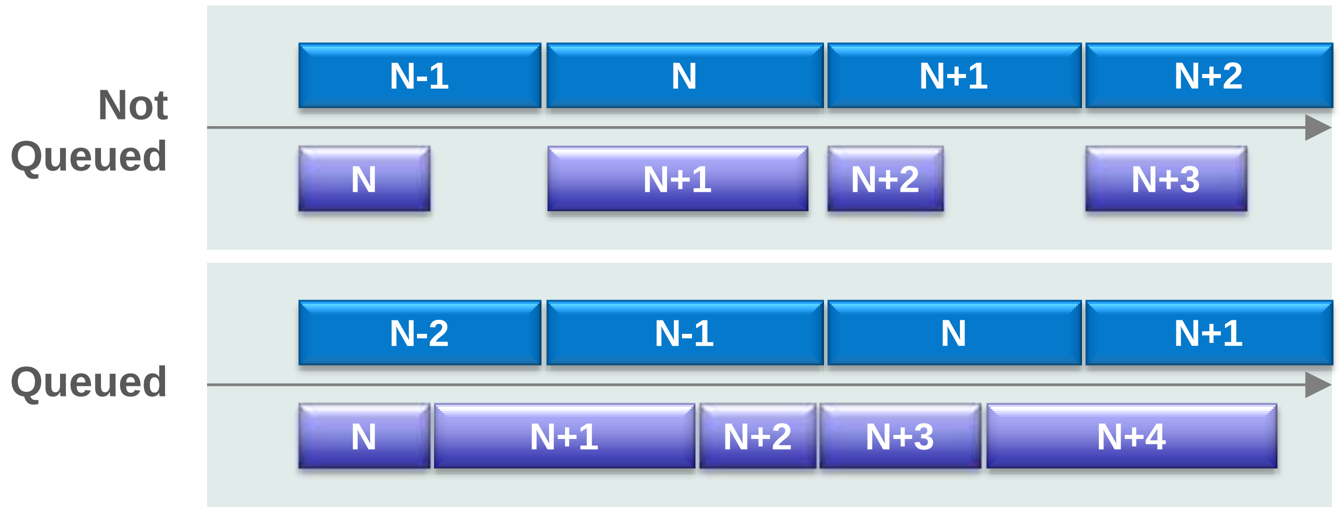
- Oversubscription not always a problem
  - As long as key resources that are frequently written to fit into vidmem, reading from other resources in hostmem should not noticeably slow performance, and paging between vidmem and hostmem should be minimal.
- GPUView is a great tool for tracking paging



  - Red block in GPU harware queue represents paging

Queued Frames

- The necessity of frame queuing
  - Why the driver always tries to buffer more frames?



  - The more frames being queued, the less chance CPU waiting on Present (less bubbles in timeline)
  - Less bubbles in GPU timeline, too
  - Driver can schedule command buffer flushing ahead to cover uneven CPU frames
  - In general, queuing -> better peformance

第五届中国游戏开发者大会
CHINA GAME DEVELOPERS CONFERENCE 2012

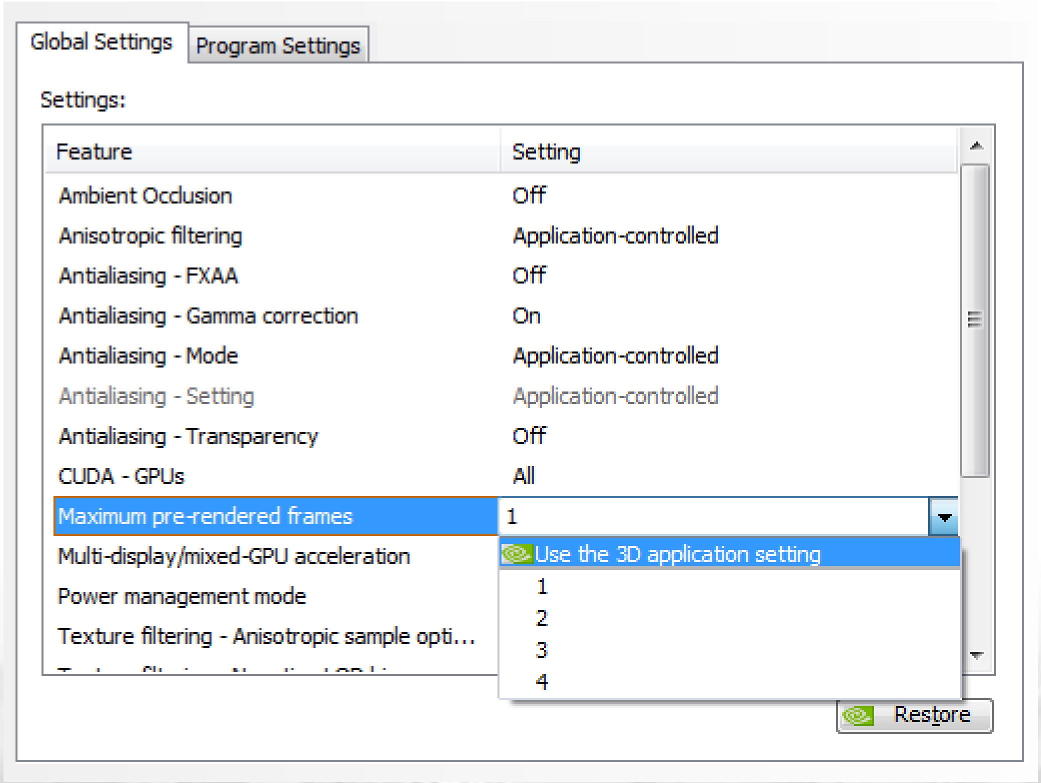
Dilemmas in Queued Frames

- Dilemma #1
  - Limiting buffered frames to 1 can shorten input latency
  - But it increases the chance of micro-stuttering, and idle bubbles in GPU processing, meaning lower performance
- Dilemma #2
  - Not limiting buffered frames can help smooth framerate
  - But a bad sync point will hurt more than no buffering
- If you decide to limit it,
  - Ensure your game engine distributes workload evenly
  - Enhance resource management to minimize sync-stall cost

CHINA GAME DEVELOPERS CONFERENCE 2012 第五届中国游戏开发者大会

Queued Frames: Mitigation

- Experiment on queued frames
  - Adjust the "maximum pre-rendered frames" setting in NV control panel
  - Is the stuttering getting better?



Feature	Setting
Ambient Occlusion	Off
Anisotropic filtering	Application-controlled
Antialiasing - FXAA	Off
Antialiasing - Gamma correction	On
Antialiasing - Mode	Application-controlled
Antialiasing - Setting	Application-controlled
Antialiasing - Transparency	Off
CUDA - GPUs	All
Maximum pre-rendered frames	1
Multi-display/mixed-GPU acceleration	
Power management mode	
Texture filtering - Anisotropic sample opti...	

Use the 3D application setting
1
2
3
4

gameworks.nvidia.com

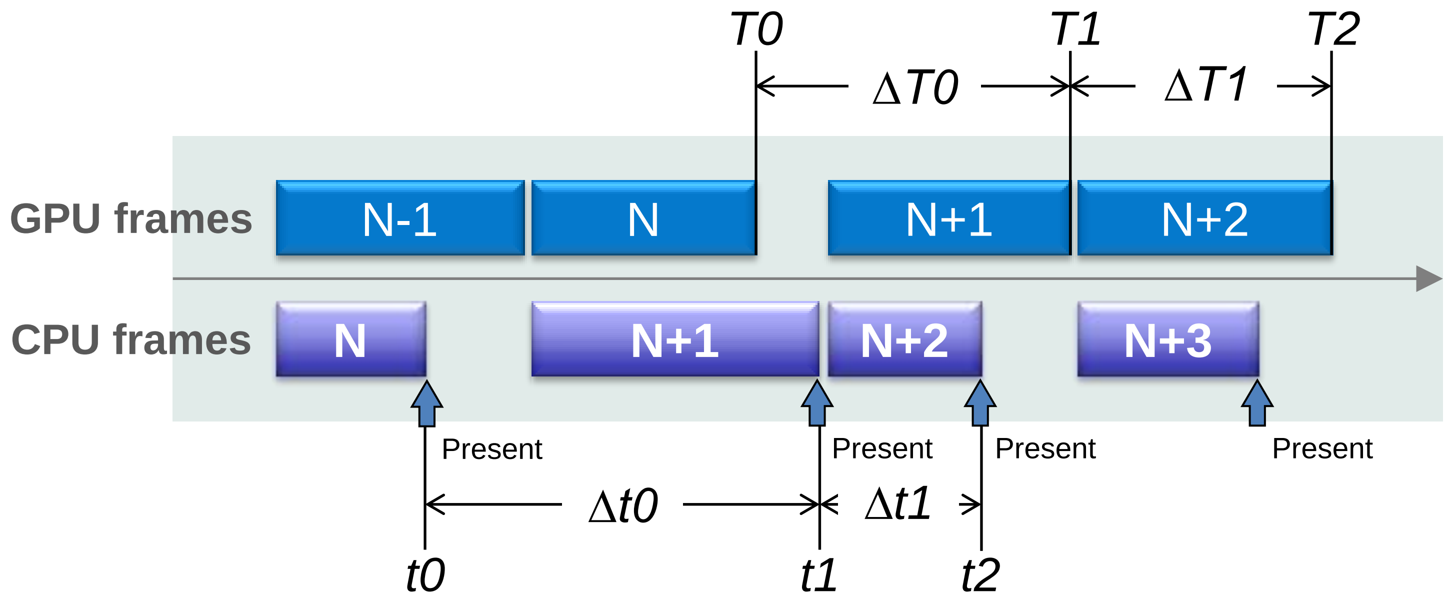
第五届中国游戏开发者大会
CHINA GAME DEVELOPERS CONFERENCE 2012

Queued Frames: Mitigation (cont.)

- Methods to limit buffered frames
  - Do not force it from NV control panel!
    It affects the entire system and other games
  - Use event query (see DXSDK document)
    But it’s not the best way – CPU gets blocked
  - Use API functions:
    IDirect3DDevice9Ex::SetMaximumFrameLatency
    IDXGIDevice1:: SetMaximumFrameLatency

Timing Issues

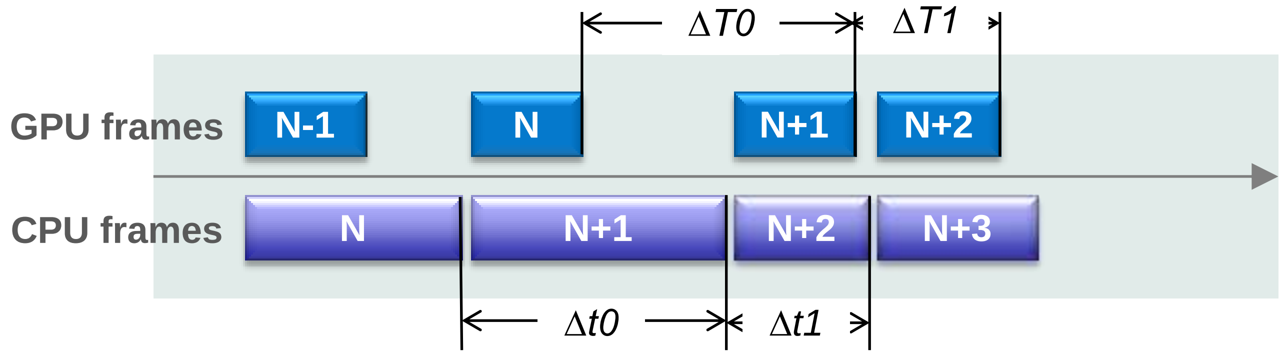
- CPU frametime vs. GPU frametime



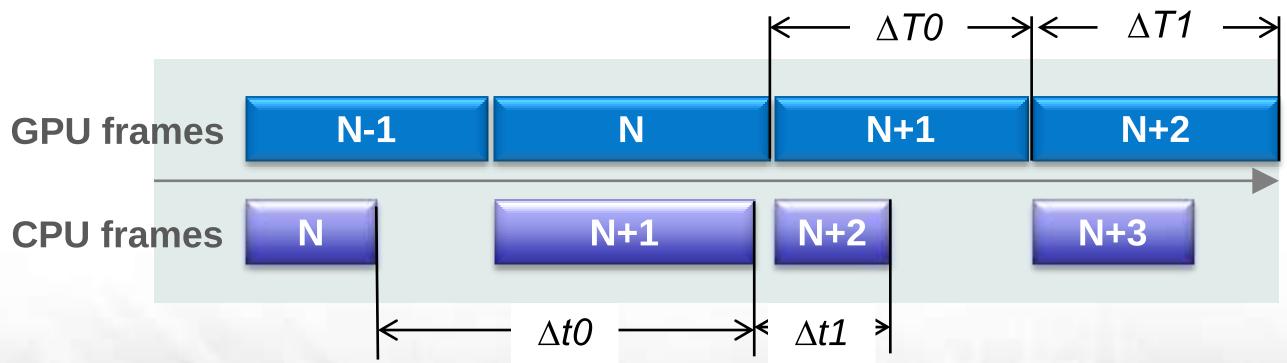
  - The game engine invokes Present at *t0*, *t1*, *t2*, …
  - The user sees the frames at *T0, T1, T2*, …
  - *Δt0, Δt1* cannot be used as elapsed time for updating since they are not the same values as *ΔT0, ΔT1*

Timing Issues (cont.)

- A couple of situations
  - Using CPU frametime is fine if the frame is CPU bound



  - CPU frametime has huge discrepancies from real frametime if GPU workload is much higher



Timing Issues: Mitigation

- Use GPU time stamps
  - After Present call, issue a time stamp query to get the time point GPU finish the present.
  - But GPU works behind CPU. The query result returns a few frames later and can only be used for estimation
  - To get the result quicker, invoke GetData with a FLUSH flag immediately after issuing, that hints the driver to flush command buffer
- Frametime estimation
  - Straightforward way: averaging frametimes in past several frames
  - More advanced way: comparing CPU frametimes to GPU timestamps to see the frame is CPU bound or GPU bound, and compute a weighted result

Query Basics

- Asynchronized queries in D3D
  - Async query introduced since D3D9 due to GPU working behind CPU
  - Spinning on retrieving query result can result in pipeline bubble

    while (S_FALSE == pQuery->GetData(…, D3DGETDATA_FLUSH));

  - This produces a CPU-GPU sync point and may become the source of stuttering

Event Queries

- Event queries can be used to eliminate queued frames in driver
  - It helps to reduce input latency, but…
  - It also exposes unbalanced frame-to-frame CPU workload -> micro-stuttering
  - CPU has to wait on the query return, thus the parallelism between CPU & GPU becomes lower -> lower performance
  - The driver is unable to perform certain optimizations without knowledge of multiple queued frames

Occlusion Queries

- Occlusion queries tend to have high latency
  - The result may return after 1~3 frames
  - Avoid spinning on GetData, which can cause much worse stalls:

    First, CPU waits for GPU on query results,
    Then, GPU waits for CPU for submitting new frames
    CPU-GPU almost work in serialized mode

    This may cancel the benefit of using occlusion query.

Queries: Mitigation

- Be cautious when using queries
  - Make sure your use of queries is optimal and not introducing bubbles in the pipe
  - Ideally, with optimized resource management and high framerate, you should not be limiting queued frames with event query.
  - Efficiently using occlusion query requires a complicated non-block system (not covered in this talk)

Check Your Middleware

- Middleware is generally written in a vacuum
  - What works best in a small environment might not scale well
- Especially check for CPU-GPU sync points

Vsync, SLI & Many Other Things

Vsync

- Vsync is a source of micro-stuttering
  - Framerate fluctuates between vsync points: 60fps, 30fps, 20fps, …
  - Applications can implement customized frame constraint system to avoid sudden framerate change

- Latest NVIDIA control panel offers the option of Adaptive Vsync
  - When framerate drops below the vsync point, vsync is disabled

SLI

- Micro-stuttering is much easier to trigger in multi-GPU environment
  - Two or more GPUs may present the rendering results at uneven cadences
  - Sync points raised by resource updating and query are harder to cover
  - Inter-GPU data transfer will place additional sync points
- The driver’s responsible for eliminating stutters in SLI
  - But the application needs to be well behaved
- In-depth SLI discussion is out of the scope of this talk. Please contact us if you have more questions

CGDC

第五届中国游戏开发者大会
CHINA GAME DEVELOPERS CONFERENCE 2012

Other causes of stuttering

- Some less common causes of stuttering:
  - GPU context switch
    For applications using compute shader, CUDA or other GPU computing tasks, the switching between graphics and computing contexts may flush command buffers at improper time
  - Contention among multiple D3D devices
    Games running with multiple windows and D3D devices may suffer resource contention
  - Driver running out of paged/non-paged pool
    Under XP 32bit, the low availability of paged/non-paged pool can be troublesome for games keeping lots of resources in flight
  - There’re more possible causes, but we can’t cover all of them here.

汉威信恒 HOWELL EXPO

CGDC

第五届中国游戏开发者大会

CHINA GAME DEVELOPERS CONFERENCE 2012

汉威信恒 HOWELL EXPO

Thanks!

Q & A

gameworks.nvidia.com