NVIDIA GEFORCE GTX

NVIDIA SLI AND STUTTER AVOIDANCE:

A Recipe for Smooth Gaming and Perfect Scaling with Multiple GPUs

NVIDIA SLI AND STUTTER AVOIDANCE:

Iain Cantlay (Developer Technology Engineer)
Lars Nordskog (Developer Technology Engineer)

WHY SLI?

- “SLI” – Set of multi-GPU technologies
- Pixel counts increasing at a staggering rate (4K+)
- Emulating the “hardware of tomorrow”
- VR – 2 eyes, 2 GPUs

SLI BASICS

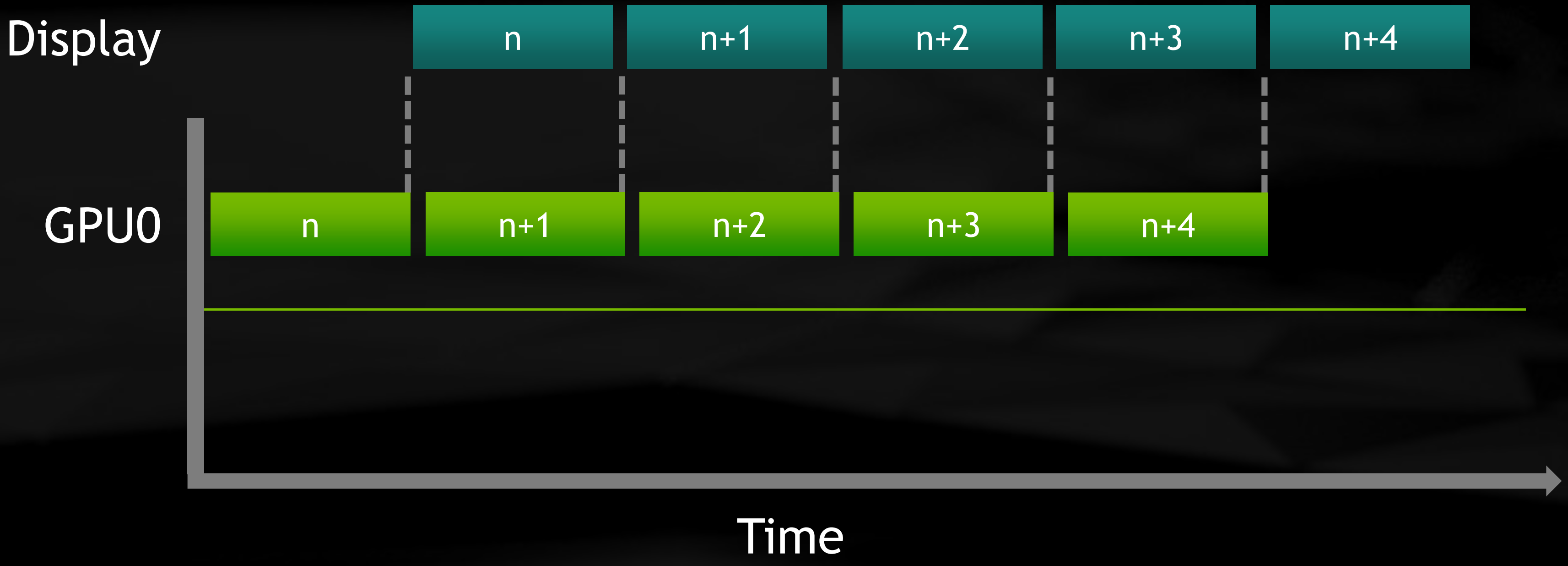
- AFR – Alternate Frame Rendering
  - One frame per GPU in parallel

- Want linear performance improvements for each GPU added
  - “SLI scaling”

- AFR SLI abstracts all non-primary GPUs away from the runtime
  - Game sees one GPU
  - Driver does the “magic”

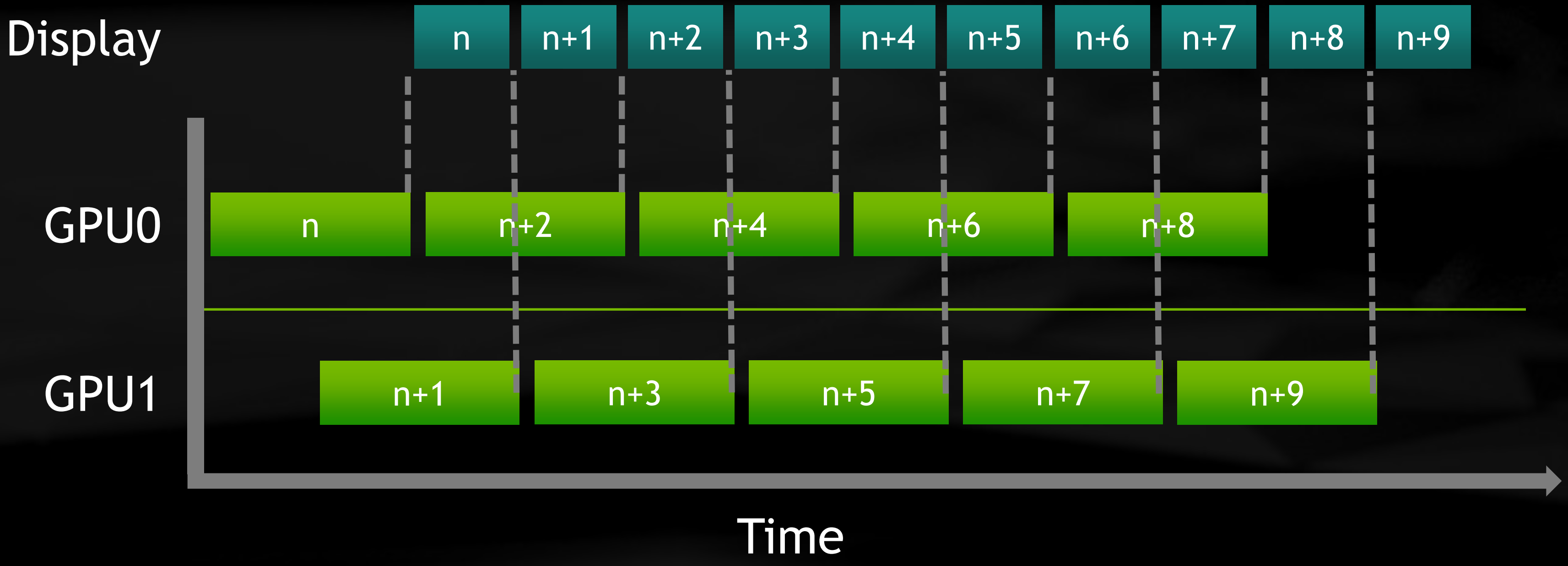
SLI BASICS

- Single GPU frame rendering



SLI BASICS

- 2-way Alternate Frame Rendering
  - Parallelism



SLI BASICS

- Allocated resources replicated per AFR GPU

- Static GPU resource mirrored between GPUs
  - Reading from local memory is optimal
  - Static textures, IBs, VBs, etc.

- Dynamic GPU resources can diverge
  - RTs, UAVs

SLI BASICS

AFR Pros

- Up to linear performance scaling
- "Frame" provides natural data dependency boundary*
- Uniform workloads (frames similar)

AFR Cons

- Non-uniform flip intervals (microstutter)
- *Interframe dependencies
- Input latency does not reduce with increased performance

SLI BASICS

AFR Pros

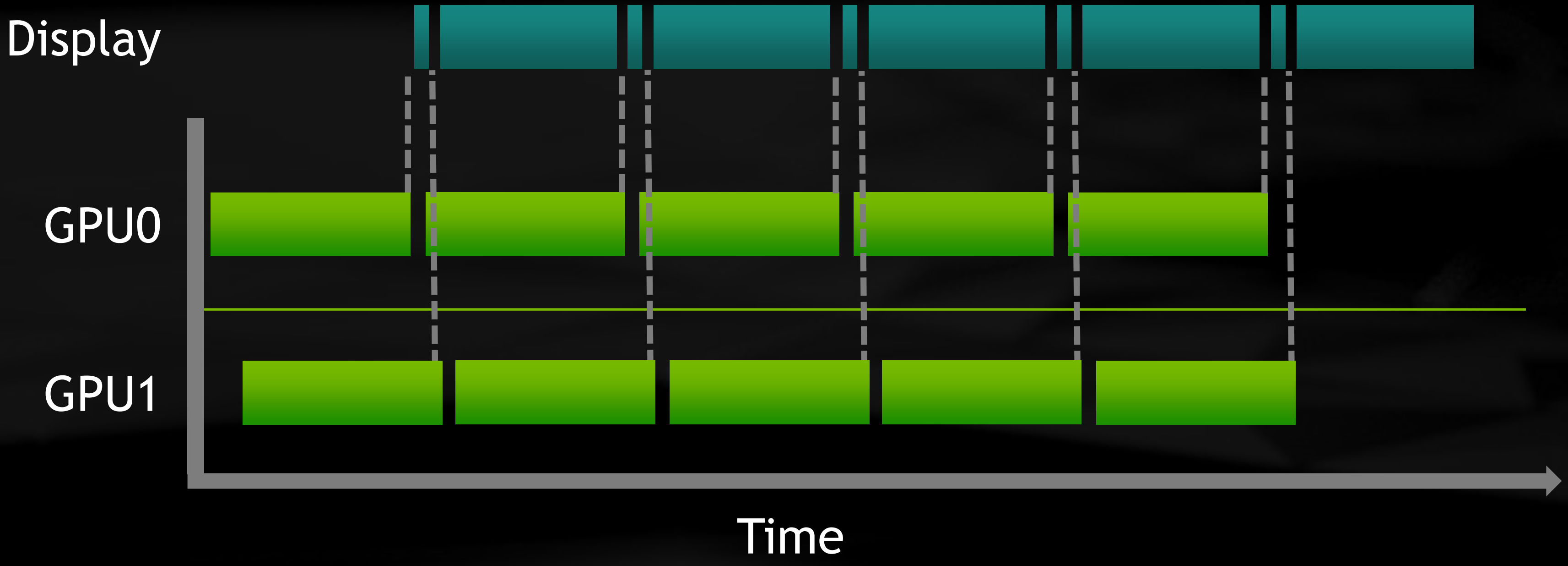
- Up to linear performance scaling
- "Frame" provides natural data dependency boundary*
- Uniform workloads (frames similar)

AFR Cons

- Non-uniform flip intervals (microstutter)
- *Interframe dependencies
- Input latency does not reduce with increased performance

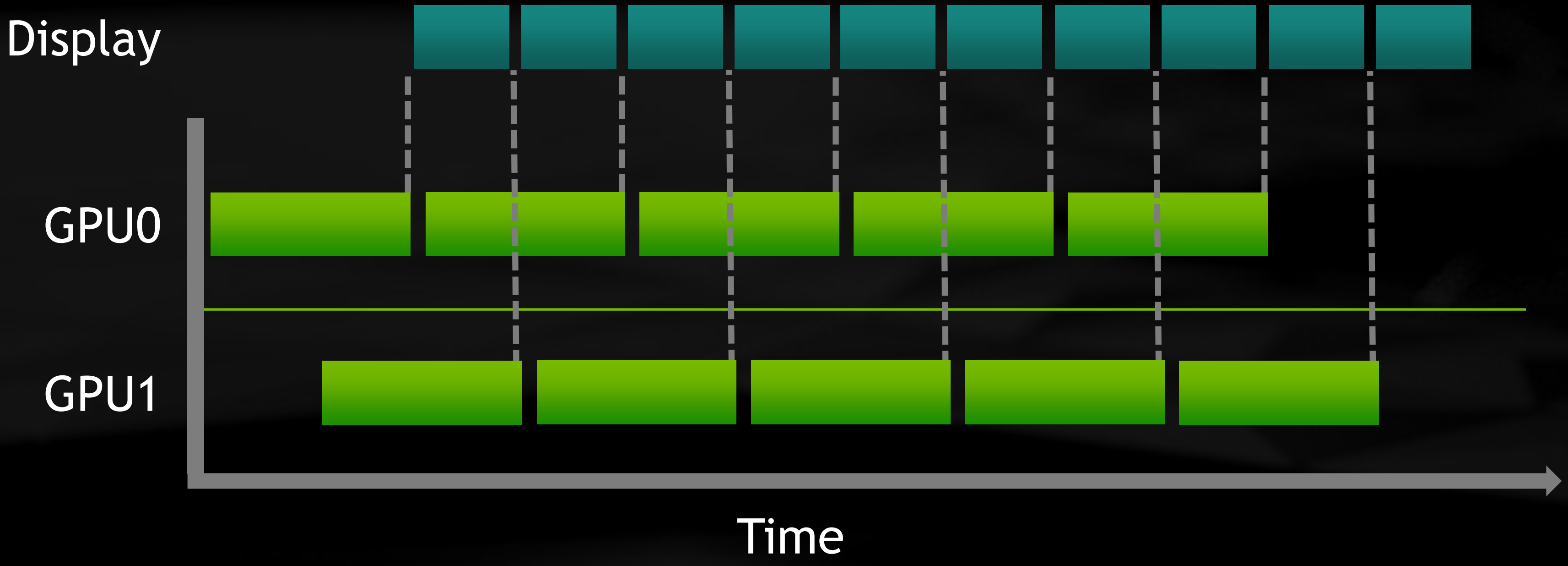
MICROSTUTTER

- Naïve parallelism -> non-uniform flip intervals…
  - Reported framerate 2x, but perceived framerate closer to single GPU



FLIP METERING

- … but SLI driver handles frame flip metering, so you don't have to!
    - Back pressure to application avoids animation stutter



INTERFRAME DEPENDENCIES

- We know static resources replicated to all GPUs
  - Never change, so no problem

- …but some RTs/UAVs are modified by GPU
  - Correctness! Driver must keep RTs/UAVs in sync between GPUs
  - Sustain “illusion” of single GPU
  - Data transferred GPU->GPU when reference “dirty”

INTERFRAME DEPENDENCIES

- Transferring data hurts SLI performance

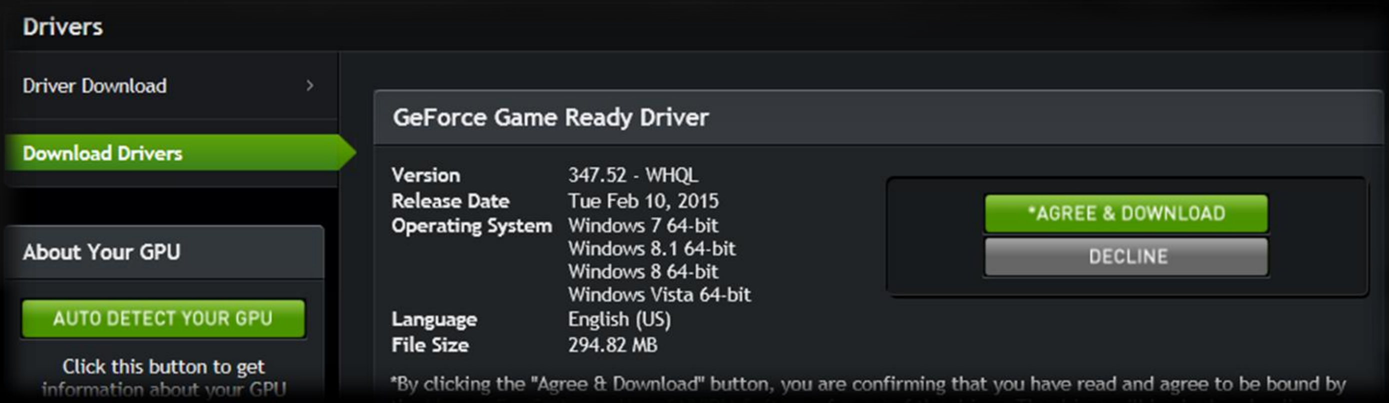
- Some transfers not necessary
  - Game updates resource entirely each frame

- ...But other transfers are necessary
  - Techniques that need previous frame results as input
  - Temporal feedback (luminance adaptation, TXAA)
  - Compute (simulations)
  - Partial updates (tiled shadowmap, cubemap, atlas textures)

- Driver transfers entire mip slice/buffer

INTERFRAME DEPENDENCIES

- SLI Profile skips transfers deemed unnecessary
  - Blunt instrument
  - Prioritize correctness
- NVIDIA tests, ships official SLI profile with driver
  - Profiles usually more complicated than AFR1/AFR2

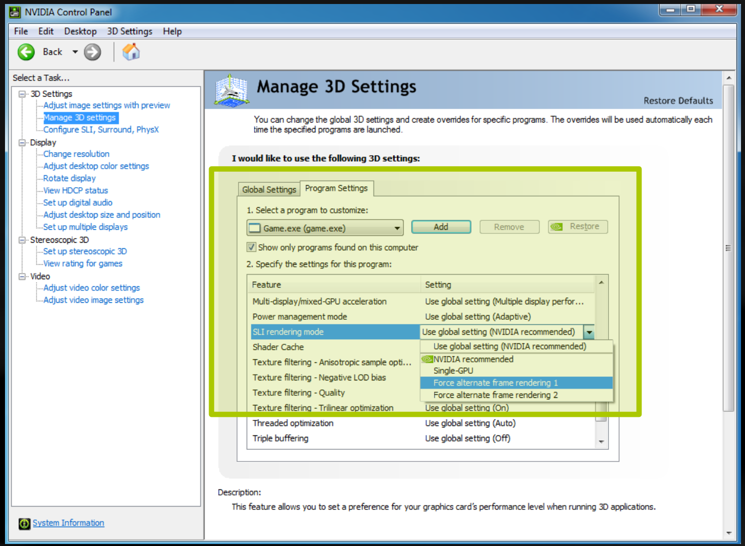


Version	347.52 - WHQL
Release Date	Tue Feb 10, 2015
Operating System	Windows 7 64-bit Windows 8.1 64-bit Windows 8 64-bit Windows Vista 64-bit
Language	English (US)
File Size	294.82 MB

*By clicking the "Agree & Download" button, you are confirming that you have read and agree to be bound by

INTERFRAME DEPENDENCIES

- SLI-enabled in Control Panel without SLI profile = single GPU
- NVIDIA Control Panel AFR Modes
  - AFR1 transfers all dirty resources -> low scaling, but no corruption
  - AFR2 skips some transfers -> better scaling, but possible corruption



Manage 3D Settings

Restore Defaults

You can change the global 3D settings and create overrides for specific programs. The overrides will be used automatically each time the specified programs are launched.

I would like to use the following 3D settings:

Global Settings | Program Settings

1. Select a program to customize: Game.exe (game.exe) — Add — Remove — Restore

[✓] Show only programs found on this computer

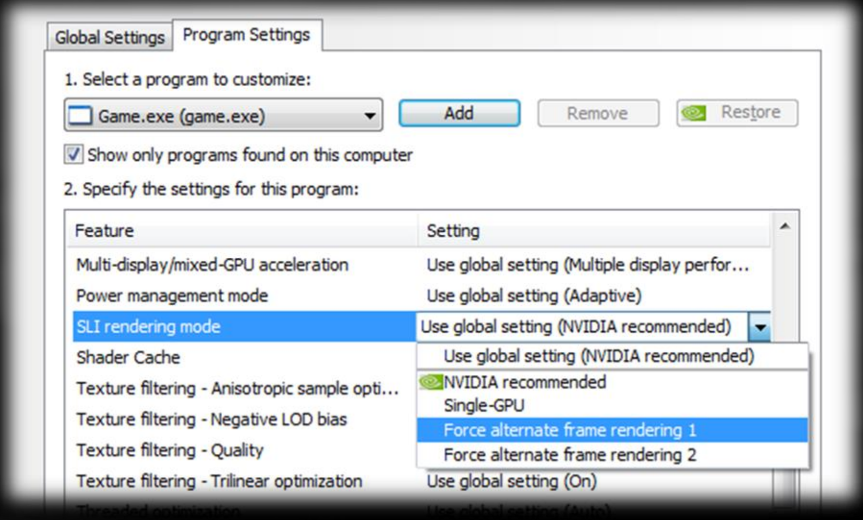
2. Specify the settings for this program:

Feature	Setting
Multi-display/mixed-GPU acceleration	Use global setting (Multiple display perfor...
Power management mode	Use global setting (Adaptive)
SLI rendering mode	Use global setting (NVIDIA recommended)
Shader Cache	
Texture filtering - Anisotropic sample opti...	
Texture filtering - Negative LOD bias	
Texture filtering - Quality	
Texture filtering - Trilinear optimization	Use global setting (On)
Threaded optimization	Use global setting (Auto)
Triple buffering	Use global setting (Off)

Use global setting (NVIDIA recommended)
NVIDIA recommended
Single-GPU
Force alternate frame rendering 1
Force alternate frame rendering 2

Description:

This feature allows you to set a preference for your graphics card's performance level when running 3D applications.


Global Settings | Program Settings

1. Select a program to customize: Game.exe (game.exe) — Add — Remove — Restore

[✓] Show only programs found on this computer

2. Specify the settings for this program:

Feature	Setting
Multi-display/mixed-GPU acceleration	Use global setting (Multiple display perfor...
Power management mode	Use global setting (Adaptive)
SLI rendering mode	Use global setting (NVIDIA recommended)
Shader Cache	
Texture filtering - Anisotropic sample opti...	
Texture filtering - Negative LOD bias	
Texture filtering - Quality	
Texture filtering - Trilinear optimization	Use global setting (On)

Use global setting (NVIDIA recommended)
NVIDIA recommended
Single-GPU
Force alternate frame rendering 1
Force alternate frame rendering 2

SO WHERE'S THE PROBLEM?

Driver doesn't have all the information about game's intent ☹

Need game to behave well, provide hints to driver, and become "AFR-aware" for optimal performance!

COMMON SCALING PITFALLS (CPU)

- Queries or APIs preventing queuing of frames (Bad!)
  - Solution: SLI input latency same as single, so allow n+1 frames in flight for n GPUs
- Readbacks to CPU (Dangerous!)
  - Solution: Avoid, or delay readback by n frames via buffering to avoid CPU stalling
- Stalling Map() writes (Bad!)
  - Solution: Use WRITE_DISCARD/WRITE_NO_OVERWRITE

COMMON SCALING PITFALLS (GPU)

- Necessary transfer causing GPU->GPU serialization
  - Solution: Decouple GPUs, look back n frames on n-way config (input local)
- Mod of per-frame ping-pong buffer -> n-way dependent transfers
  - Solution: Always Discard/Clear dynamic resource before bind, QA SLI with 3-way config
- GPU-generated data not regenerated every frame
  - Solution: Regenerate data on each GPU, or hint to keep

INITIAL STEPS

- Renaming EXE to AFR-FriendlyD3D.exe
  - Enables n-way AFR, skips all transfers
  - Corruption, but ideal for checking “speed of light”
  - No scaling with rename -> CPU-GPU serialization or CPU-boundedness

- Query NVAPI to detect SLI via number of GPUs
  - NvAPI_D3D_GetCurrentSLIState()
  - SLI profile “aware”… no profile returns “1 GPU”

SLI COMPATIBILITY PROCESS

1. Think through your interframe dependent effects/systems

2. Run with exe renamed to “AFR-FriendlyD3D.exe” to skip all GPU->GPU transfers of dirty resources

3. Test thoroughly, looking for corruption

4. Address resources that are not “AFR Friendly”
   - Regenerate data for all GPUs, or hint to driver that data must persist
   - Hint what data can be discarded

TAKING ACTION

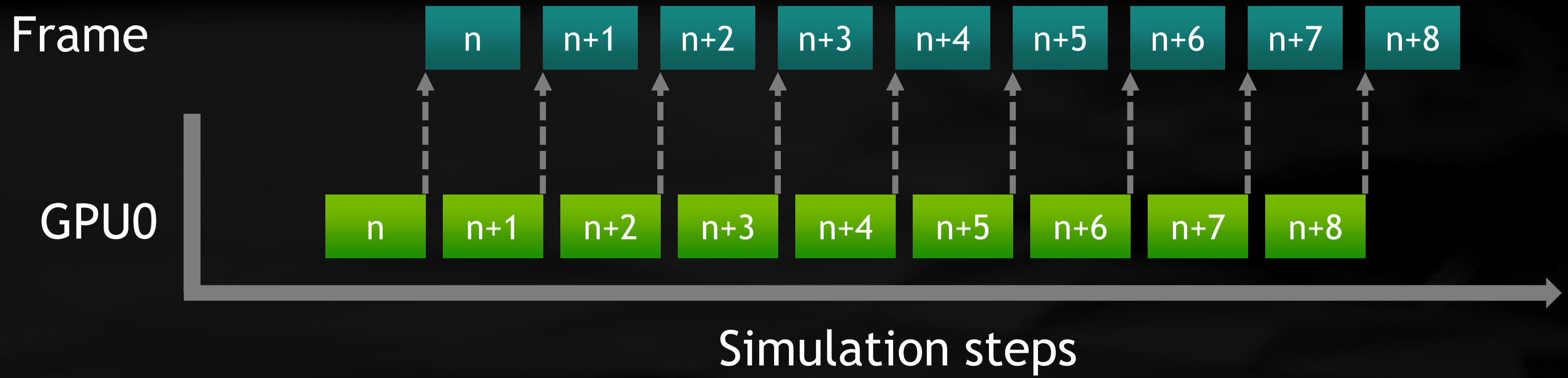
"How do I regenerate data for each GPU?"

- "Explicit synchronization"
  - Keep track of which GPUs receive updates
  - Re-issue for each "dirty" GPU
  - Allows discarding of transfers + GPU coherency

- Regenerate work or hint to driver to keep?
  - Generally regenerating better, but case by case
  - Only so much data practically transferred per frame (performance)

TAKING ACTION

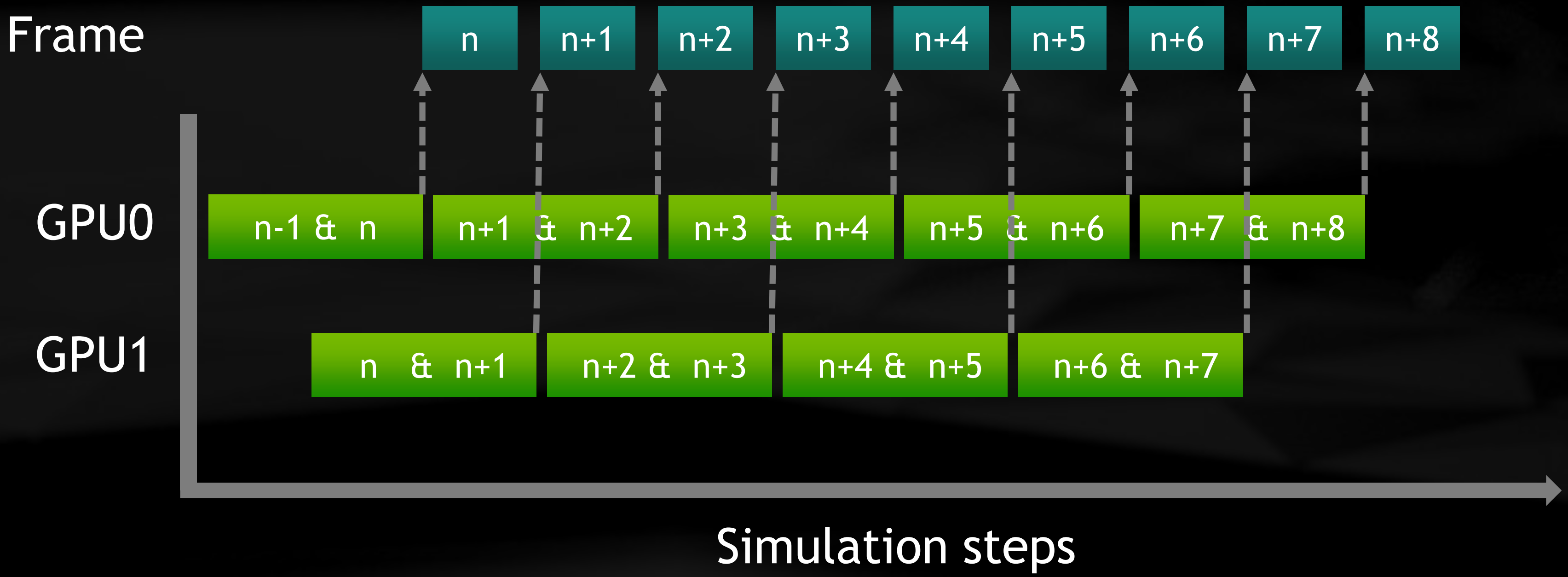
"How do I regenerate data for each GPU?" (cont)



TAKING ACTION

"How do I regenerate data for each GPU?" (cont)

- Simulation is duplicated for each GPU
- Still faster than doing a transfer!



TAKING ACTION

*“How do I hint what I *DO* need to persist between frames?”*

- NvAPI_D3D_BeginResourceRendering()/NvAPI_D3D_EndResourceRendering()
  - Wrap update -> driver transfers early/efficiently
  - NVAPI_D3D_RR_FLAG_FORCE_DISCARD_CONTENT works as Discard/Clear for ping-pong case

- Begin/End assume only next GPU needs data
  - NVAPI_D3D_RR_FLAG_MULTI_FRAME if used for multiple frames

- USE WITH CAUTION!!!
  - Final update of resource in frame
  - Don’t Begin/End > 1 time per frame, per resource
  - Begin/End takes precedence over profile
  - Entire resource transferred

TAKING ACTION

*“How do I hint what I *DON’T* need to persist between frames?”*

- ID3D11DeviceContext1::DiscardView()/DiscardResource()
  - Ideal solution
  - Before bind in current frame
  - Only supported in DX11.1

- ID3D11DeviceContext::Clear*()
  - Before bind in current frame

- NvAPI_D3D_SetResourceHint()
  - Driver excludes resource from SLI “dirty” state tracking (never transfers)
  - Sticky through allocation lifetime

TAKEAWAYS

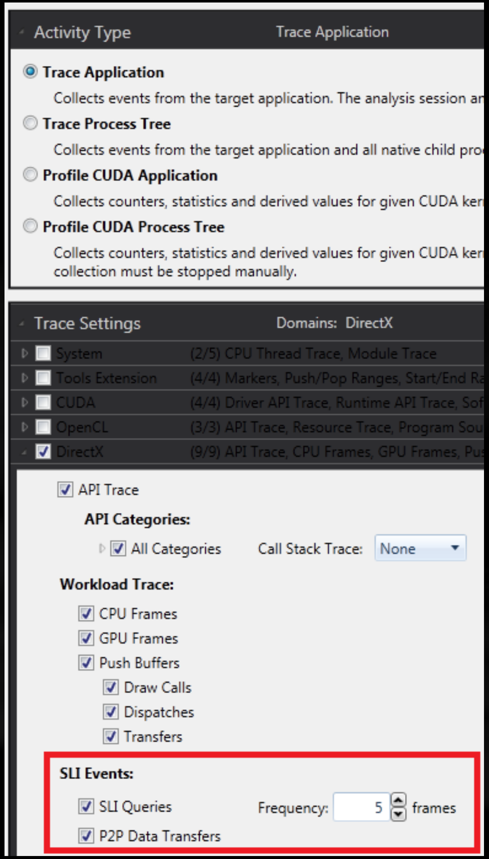
- SLI excellent for substantially increasing GPU performance
- Ensure AFR friendly CPU behavior
- Use AFR-FriendlyD3D.exe
- Anticipate interframe dependent effects/systems
  - Design them to be AFR friendly
- At minimum, focus on regenerating data or hinting what to keep between frames
  - BeginResourceRendering/EndResourceRendering hints
  - NVIDIA can remove the rest with profiles, but Discard APIs better

OH YEAH…

- Getting testing builds to NVIDIA early (Please ☺)
  - For SLI profiling, identification issues, advice
- QA SLI on 3-way configuration
  - Needs profile or AFR-FriendlyD3D.exe to scale

OTHER RESOURCES

- Nsight SLILog
  - Plans to expand functionality and clarity
- GPUView



Activity Type — Trace Application

[✓] Trace Application
Collects events from the target application. The analysis session ar

Trace Process Tree
Collects events from the target application and all native child pro

Profile CUDA Application
Collects counters, statistics and derived values for given CUDA ker

Profile CUDA Process Tree
Collects counters, statistics and derived values for given CUDA ker
collection must be stopped manually.

Trace Settings — Domains: DirectX

System	(2/5) CPU Thread Trace, Module Trace
Tools Extension	(4/4) Markers, Push/Pop Ranges, Start/End Ra
CUDA	(4/4) Driver API Trace, Runtime API Trace, Sof
OpenCL	(3/3) API Trace, Resource Trace, Program Sou
✓ DirectX	(9/9) API Trace, CPU Frames, GPU Frames, Pu

✓ API Trace

API Categories:

✓ All Categories — Call Stack Trace: None

Workload Trace:

✓ CPU Frames
✓ GPU Frames
✓ Push Buffers
✓ Draw Calls
✓ Dispatches
✓ Transfers

SLI Events:

✓ SLI Queries — Frequency: 5 frames
✓ P2P Data Transfers

QUESTIONS?

- Thank you!

- Iain Cantlay (icantlay@nvidia.com)
- Lars Nordskog (lnordskog@nvidia.com)

GAMEWORKS

- Get the latest information for developers from NVIDIA and continue the discussion
- gameworks.nvidia.com