Care and Feeding of Exotic Animals

Matthias Wloka

Developer Technology, NVIDIA

The PC Graphics Zoo

- Multiple hardware vendors
- Desktop, mobile, integrated
  - Different performance characteristics

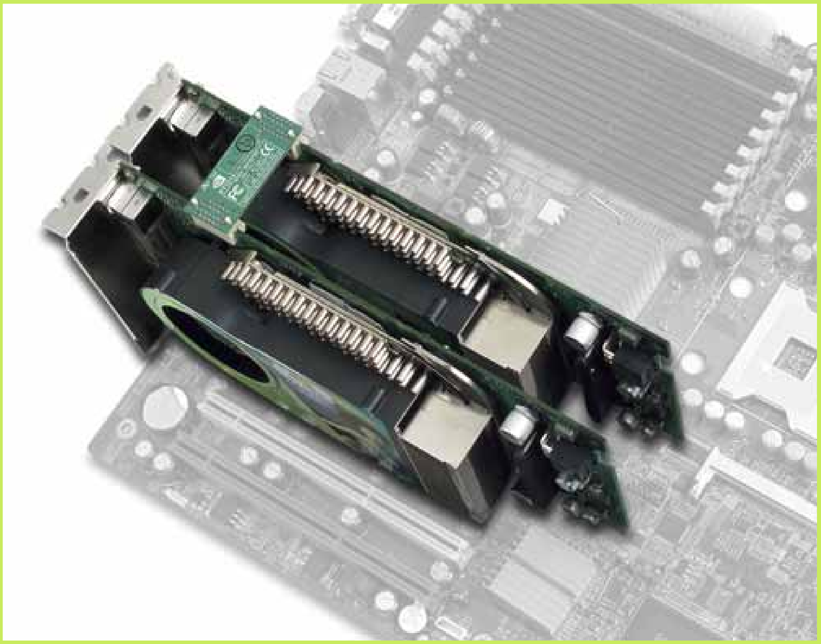
- Multiple price points
  - Different performance characteristics per price point
- Many, many drivers

And Now Even More Exotic Beasts

- SLI Multi GPU
  - Scalable Link Interface

- Stereo
  - 3D Stereo Driver
- The good news?
  - Automatic: drivers take care of (most) everything
  - Little things you can help with: hence this talk

What Is SLI?

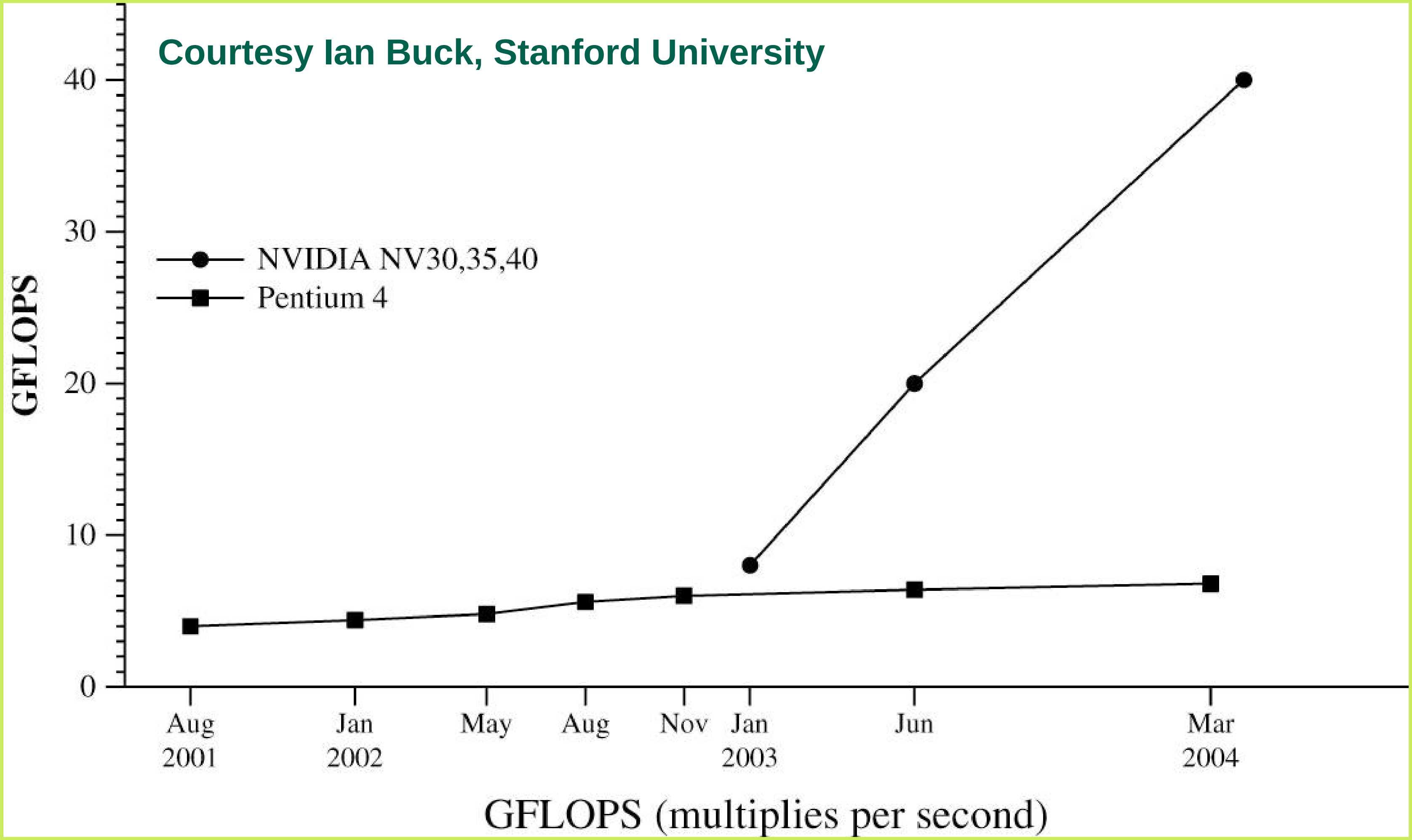


- Plug 2 identical GPUs into PCI-E motherboard
- Driver still reports only a single (logical) device
- That runs (much) faster
- Video memory does NOT double

Why SLI Is Hot

- Up to 1.9x of single GPU performance
- SLI becomes readily available at game development offices
  - Just wait for your colleague to go home early
- Get a taste for next year's GPU performance

Are You CPU Limited?



Game Development Cycle

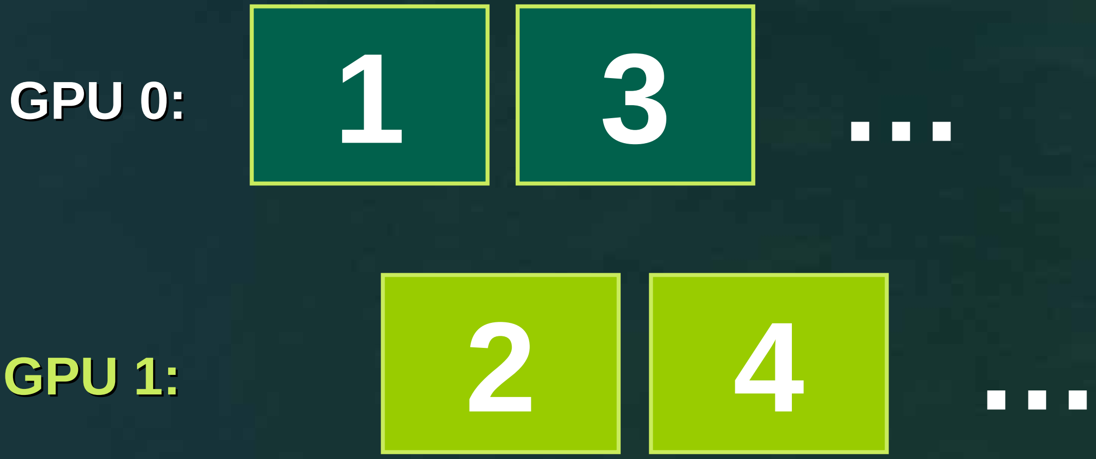
- 2 years (or more):
  - CPU performance doubles
  - GPU performance quadruples (or more)
- CPU/GPU balance shifts over 2 years!
  - More CPU-hungry modules come online later: e.g., AI, full game play, physics, etc.
- How to aim your product at the target spec?
  - SLI gives hint about future ‘mainstream’ machine, today

Ok, How Does SLI Work?

- Driver decides what mode to run an app in
- Compatibility mode:
  - Only uses one GPU
  - No SLI benefits, but guaranteed to work
- Alternate frame rendering (AFR)
- Split frame rendering (SFR)

AFR

- Each GPU works on its own frame



GPU 0: 1 3 …

GPU 1: 2 4 …

- Scan-Out toggles from where to read framebuffer

General Rendering Case for AFR

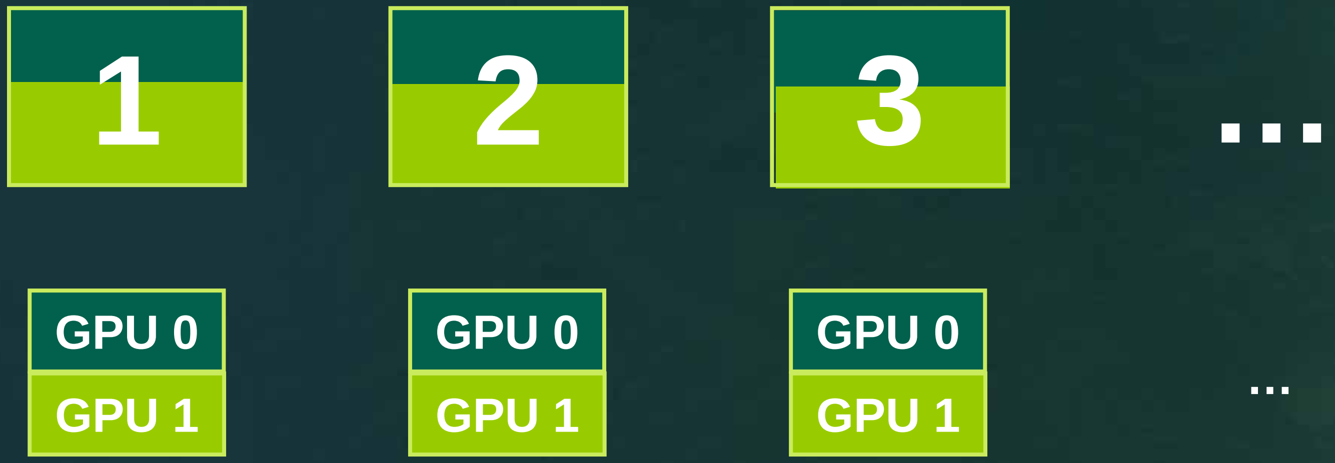
- If not self-contained, push necessary data to other GPU
  - E.g., updating render-to-texture targets only every other frame
- Pushing data to other GPU is overhead
  - Hence not 2x speed-up

AFR Advantages

- All work is parallelized
  - Pixel fill
  - Raster
  - Vertex
- Preferred SLI mode
- Works best when each frame is self-contained
  - No prior work is re-used
  - No communications overhead between GPUs

SFR

- Both GPUs work on the same frame
  - GPU 0 renders top portion
  - GPU 1 renders bottom portion



- Scan-Out combines framebuffer data

General Rendering Case for SFR

- ‘Top’ vs. ‘Bottom’ is load-balanced
  - If one GPU took longer to render
  - Adjust load accordingly (make it work less)
- Clipping avoids each GPU processing all vertices per frame
  - But not perfect so avg vertex load/GPU > half
- Still requires data sharing:
  - E.g., render to texture

SFR Compared to AFR

- Works even when number of frames buffered is limited
  - Or when AFR otherwise fails
- In general, more communications overhead
  - Less speed-up
- Applications with heavy vertex load benefit less

Overview: Things Interfering with SLI

- CPU-Bound applications
- VSync enabled
- Limiting number of frames buffered
- Updating render-targets every other frame

CPU-Bound Applications

- SLI cannot help
- Reduce CPU work or better:
- Move CPU work onto the GPU
  - http://GPGPU.org
- Don’t throttle frame-rate in application

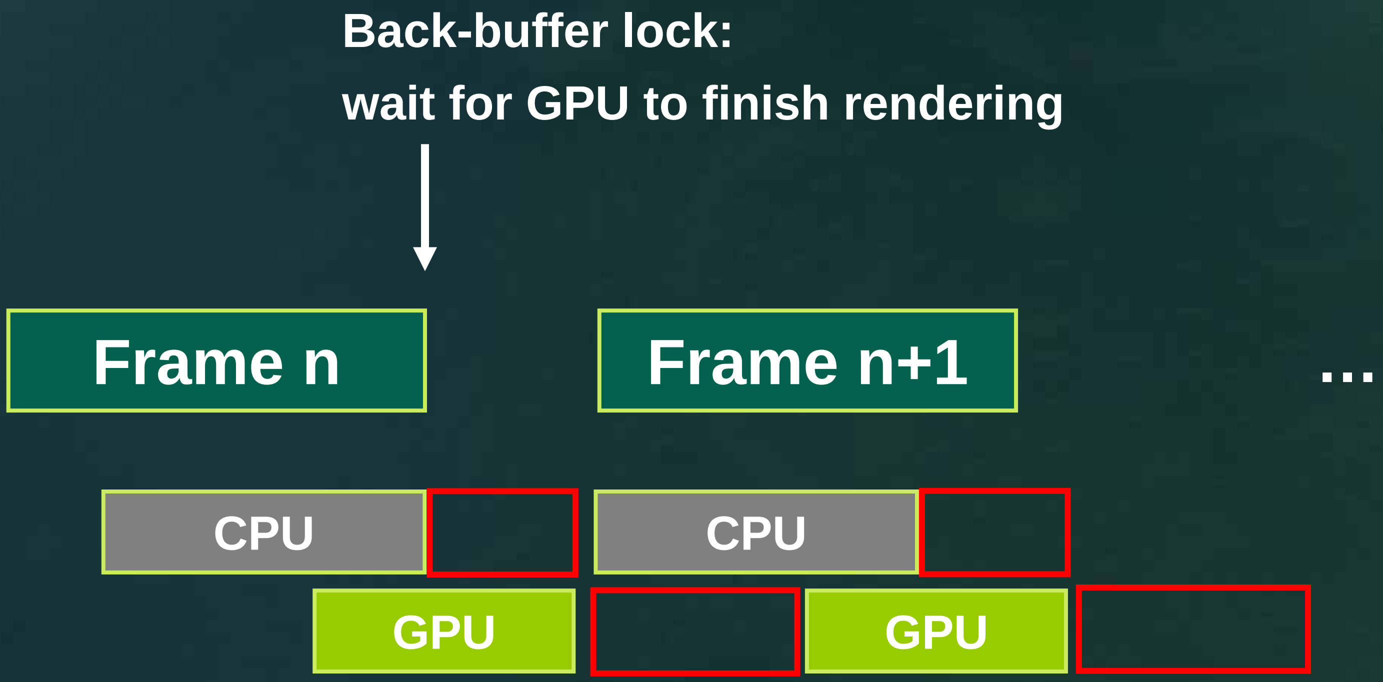
VSync Enabled

- Throttles frame-rate to monitor refresh
- Enabling triple-buffering does NOT offset enabling vsync:
  - If render-rate is faster than monitor refresh,
  - Then vsync still gates GPU
- Worse, triple-buffering
  - Increases lag
  - Consumes (much) more video-memory

Limiting Number of Frames Buffered

- Some people allow at most one frame buffered
  - Via event queries
  - To reduce lag
  - Don’t lock/read back-buffer: causes CPU stall
- But SLI is up to ~1.9x faster
  - I.e., SLI systems ~1.9x less lag
- Instead limit frames buffered to number of SLI GPUs:
  - Single GPU system buffers at most 1 frame
  - Dual GPU system buffers at most 2 frames, etc.

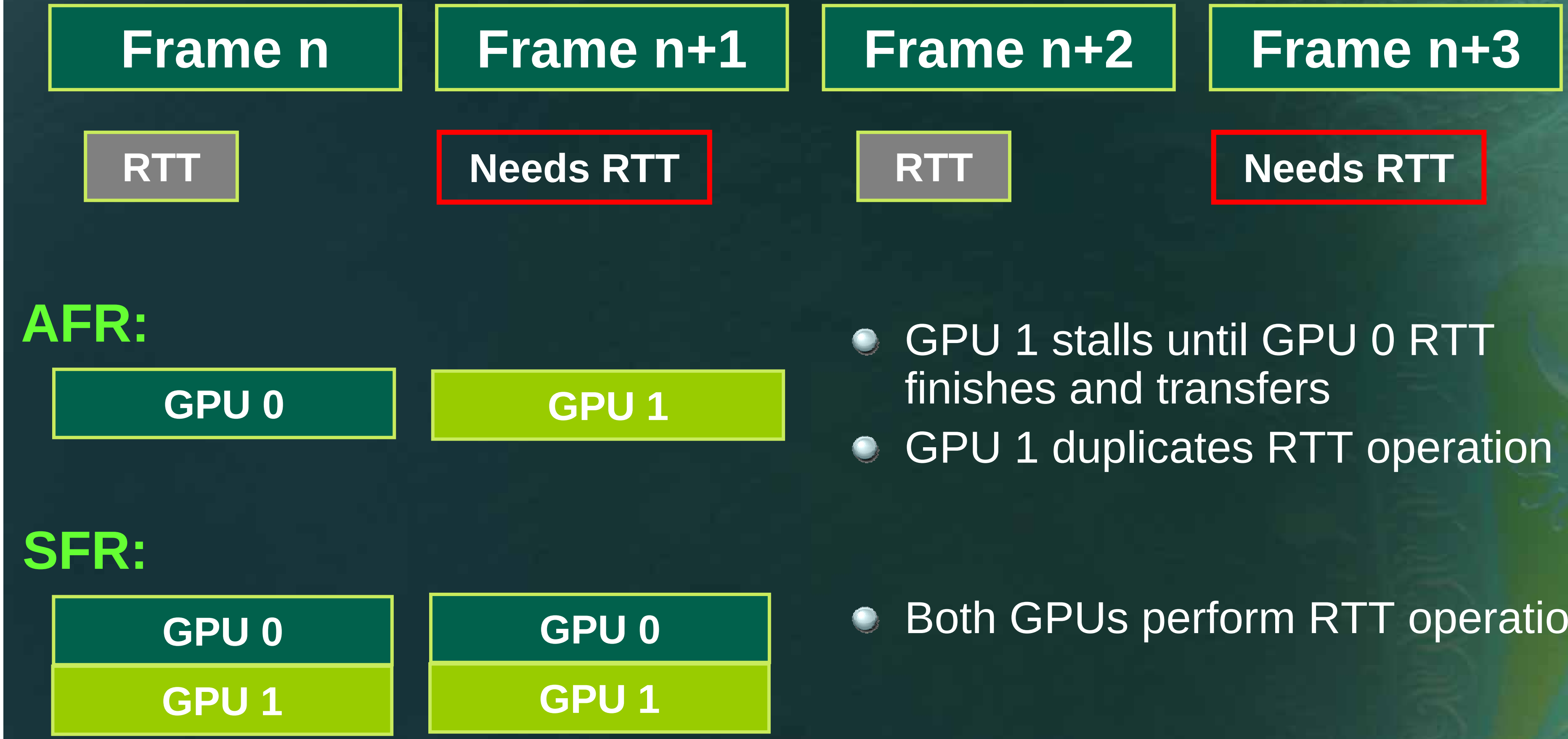
Locking the Back-Buffer Is Bad



Updating Render-Targets

- For maximum SLI efficiency:
  - Share as little data as possible
  - I.e., make frames completely independent of previous frames
  - Generate all render-targets in same frame that they are used
  - Clear your render-targets! Tells GPU not to push
- Skipping render-target update for performance?
  - Actually hurts SLI

Update-Skipping



Frame n

Frame n+1

Frame n+2

Frame n+3

RTT

Needs RTT

RTT

Needs RTT

AFR:

GPU 0

GPU 1

SFR:

GPU 0

GPU 1

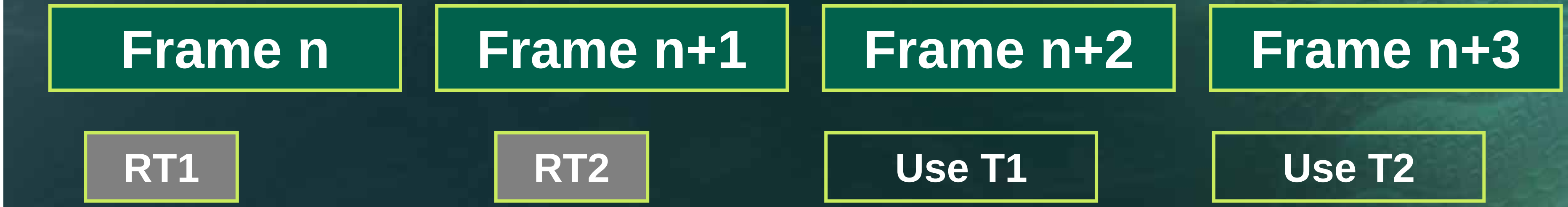
GPU 0

GPU 1

- GPU 1 stalls until GPU 0 RTT finishes and transfers
- GPU 1 duplicates RTT operation

- Both GPUs perform RTT operation

Modified Update-Skipping



Frame n

RT1

Frame n+1

RT2

Frame n+2

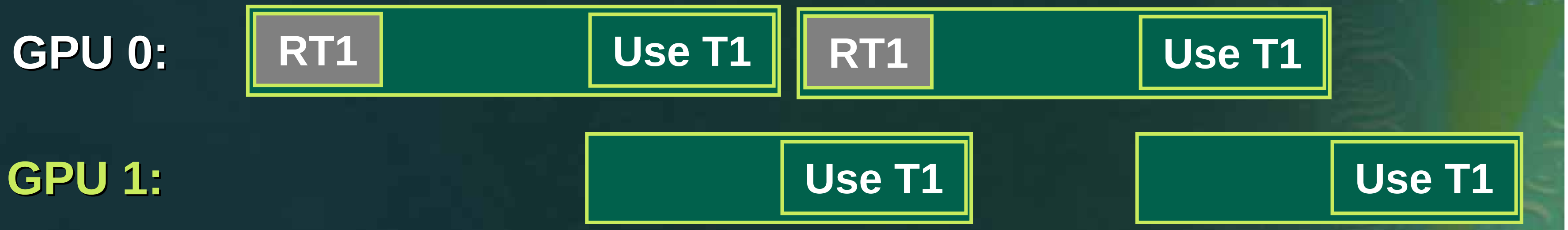
Use T1

Frame n+3

Use T2

- But doubles number of render-targets

- Or render early, use late:
  - Render every 3rd frame for better load balancing



GPU 0:

RT1

Use T1

RT1

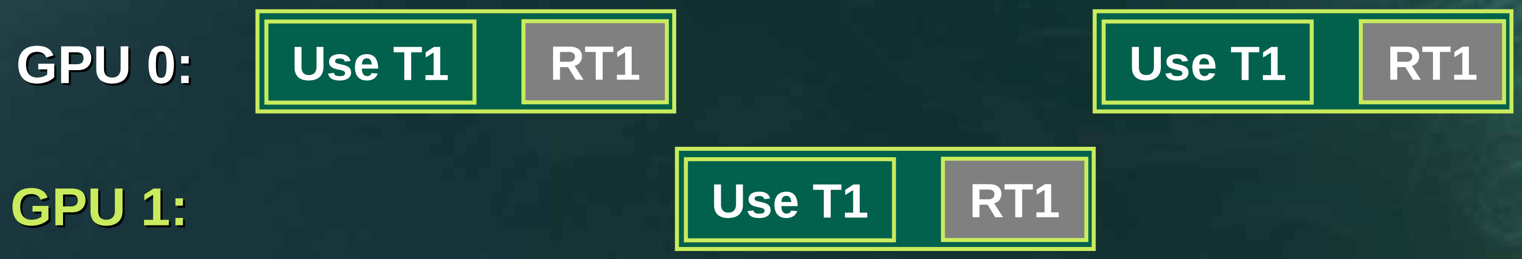
Use T1

GPU 1:

Use T1

Use T1

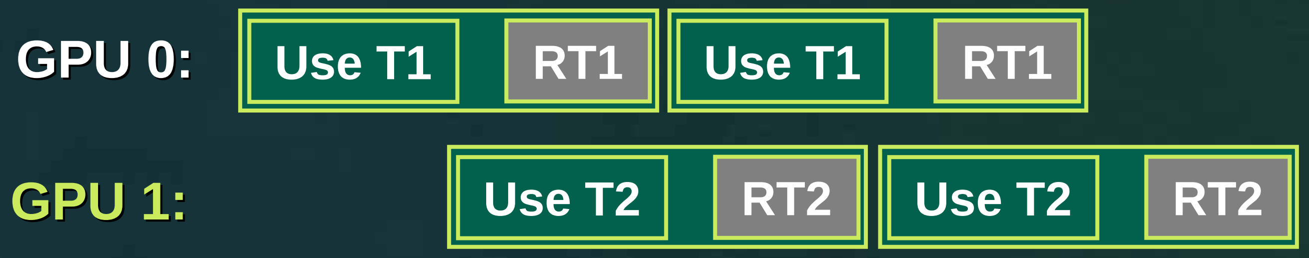
Also Bad: Use Early, Render Late



GPU 0: Use T1 RT1 Use T1 RT1

GPU 1: Use T1 RT1

- Double-buffer your textures instead:
  - E.g., HDR exposure control in MS DirectX9c SDK



GPU 0: Use T1 RT1 Use T1 RT1

GPU 1: Use T2 RT2 Use T2 RT2

Other SLI Performance Advice

- Allocate vertex buffers in POOL_MANAGED
  - Especially if it is dynamically and partially updated
- Allows data to come from system-mem copy
  - As needed

How to Detect SLI Systems?

- NVCPL API:
  - NVIDIA-specific API supported by all NV drivers
- Functions supported for:
  - Detecting that NVCPL API is available
  - Bus mode (PCI/AGP/PCI-E) and rate (1x-8x)
  - Video RAM size
  - SLI
- SDK sample and full documentation available

SLI Questions?

Stereo?

- Special NVIDIA drivers that generate stereo: http://www.nvidia.com/object/3d_stereo.html
- Works with variety of stereo outputs:
  - 3D stereo (shutter) glasses
  - Red/Blue glasses
  - 3D stereo monitors
  - 3D notebook PCs
- For the stereo enthusiast crowd

How Does the Stereo Driver Work?

- Driver parses camera position
  - Easy for fixed function: intercept D3DTS_VIEW
  - Harder for vertex shader: parse oPos generation
- Offset camera position every other frame
  - And toggle red/blue or shutter setting
- Just works, unless…

Rendering at Incorrect Depth

- Background images or sky domes
  - Place at farthest possible depth
- Don’t render UI at near-plane
  - Name labels hovering over characters
  - Render HUD as far into scene as possible
  - Laser sights, cross hairs, cursors, etc. at depth of object they are pointing at
  - Highlight objects at object’s depth

Billboards and Screen-Space Effects

- Billboards look flat (since they are)
  - Prefer low-res geometry over billboards
- Post-Processing effects
  - Bloom, glow, image-based motion blur etc.
  - Cool, but do not work at all in stereo
  - Option to disable for stereo crowd
- Screen-Space effects
  - Halos, coronas, explosions, weapons-fire, etc.
  - Make sure these have meaningful depth

Things Bypassing the Stereo Driver

- Sub-View rendering
  - PIP displays, rear-view mirrors, etc.
  - Set viewport accordingly
- Dirty rectangles, manually writing RTs, no depth data on vertices
  - Driver gets no depth info
- Windowed mode
  - Allow full-screen mode

Other Things to Watch Out For

- Resolving collisions with too much separation
  - Very obvious in stereo
- Small gaps in meshes
- Dark scenes become darker in stereo
  - Provide brightness/gamma adjustment
- High-Contrast causes ghosting in stereo
  - Provide brightness/gamma adjustment

Test Your Game in Stereo

- Easy to do via red/blue glasses
- Above problems immediately jump out
- Fixing them
  - Is usually easy
  - Also benefits the non-stereo game
- Look up current issues with your game
  - Stereo driver's "Stereo Game Configuration" lists known issues with released games

Stereo SDK/API Coming Soon

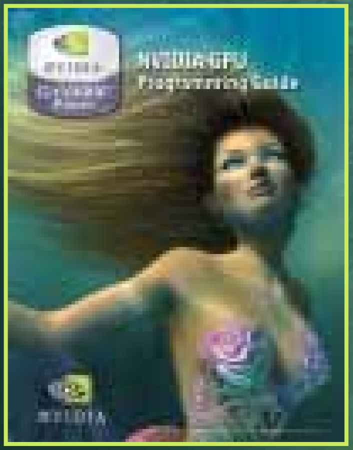
- StereoBLT API
  - Display pre-rendered stereo images in 3D
  - Code sample for DirectX
- IStereoAPI
  - Real-Time control over stereoscopic rendering
  - Header and library
  - Query and control: convergence, etc.

More Stereo Information

- 3DStereo@nvidia.com
- http://developer.nvidia.com/object/3D_Stereoscopic_Dev.html
- Low cost (< $100) stereo developer kits: http://www.i-glasses.com

Questions?

- NVIDIA GPU Programming Guide:
- http://developer.nvidia.com/object/gpu_programming_guide.html



- Matthias Wloka (mwloka@nvidia.com)

- http://developer.nvidia.com