Next Generation Shading and Rendering

Bryan Dudash

NVIDIA

Session Overview

- 3.0 Shader Model Overview
  - ps.3.0 vs. ps.2.0
  - vs.3.0 vs. vs.2.0
- Next-Gen Rendering Examples
  - Dynamic Water
    - Vertex Texture Fetch
    - Floating-point filtering / blending
    - GPU-based physics simulation
  - Volumetric Fog
    - MRT and branching for speed
  - Deferred Rendering
    - MRT and branching for speed
- Geometry Instancing
  - Added visual complexity
  - Performance optimization

Pixel Shader 3.0 Feature Comparison

Pixel Shader Feature	Shader 2.0	Shader 3.0	Description
Shader length	96	65535+	Allows more complex shading, lighting, and procedural materials
Dynamic branching	No	Yes	Saves performance by skipping complex shading on irrelevant pixels
Shader anti-aliasing	Not supported	Built-in derivative instructions	Developers can calculate the screen space derivatives of any function, allowing them to adjust shading frequencies or over-sampling to eliminate artifacts
Minimum Precision	fp24	fp32	Fewer artifacts, more dynamic range
Back-face register	No	Yes	Allows two-sided lighting in a single pass

Pixel Shader 3.0 Feature Comparison

Pixel Shader Feature	Shader 2.0	Shader 3.0	Description
Back-face register	No	Yes	Allows two-sided lighting in a single pass
Interpolated color format	8-bit integer minimum	32-bit floating point minimum	Higher range and precision color allows high-dynamic range lighting at the vertex level
Multiple render targets	Optional	4 required	Allows advanced lighting algorithms to save filtering and vertex work – thus more lights for minimal cost
Fog and specular	8-bit fixed function minimum	Custom fp16-fp32 shader program	Shader Model 3.0 gives developers full and precise control over specular and fog computations, previously fixed-function
Texture coordinate count	8	10	More per-pixel inputs allows more realistic rendering, especially for skin

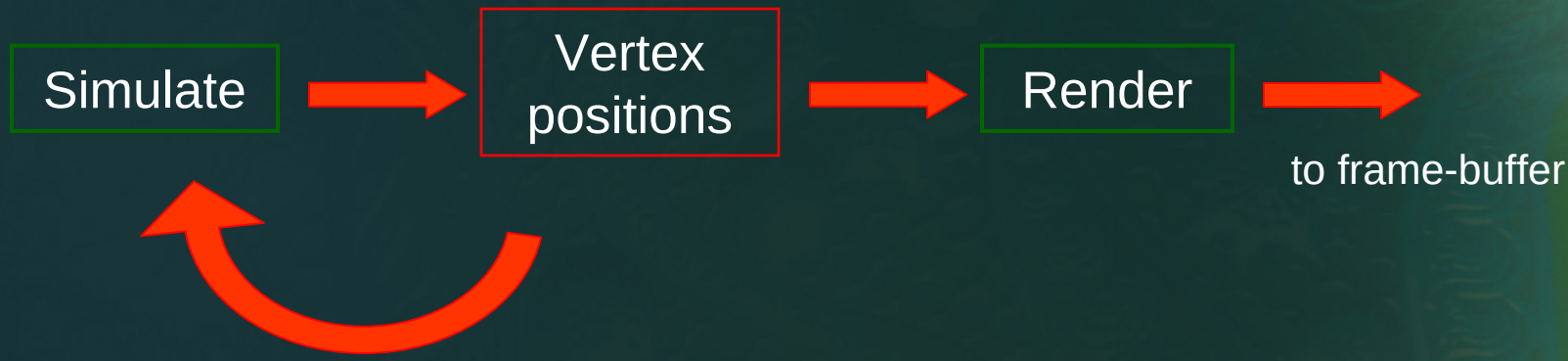
Vertex Shader 3.0 Feature Comparison

Vertex shader feature	Shader 2.0	Shader 3.0	Description
Shader length	256 Instructions	65535 instructions	More instructions allow more detailed character lighting and animation
Dynamic branching	No	Yes	Saves performance by skipping animation and calculations on irrelevant vertices
Vertex texture	No	Any number of lookups from up to 4 textures	Allows displacement mapping, particle effects
Instancing support	No	Required	Allows many varied objects to be drawn with only a single command

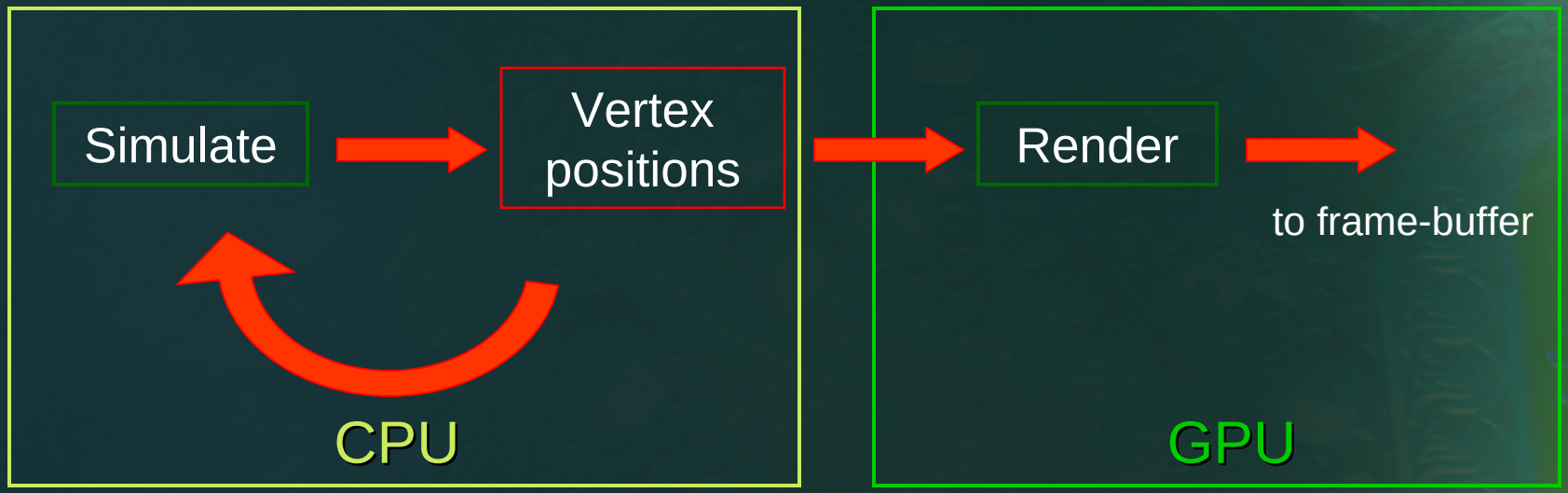
So what can we do with all this?

- Dynamic Water Rendering
  - VS3.0 Vertex Texture Fetch
  - Floating-point filtering / blending
  - GPU-based physics simulation
- Animated Volumetric Fog
  - Use polygon primitives to bound fog
  - MRT and branching for speed
- Geometry Instancing
  - Draw many “instances” of a mesh with one draw call

Typical Workflow

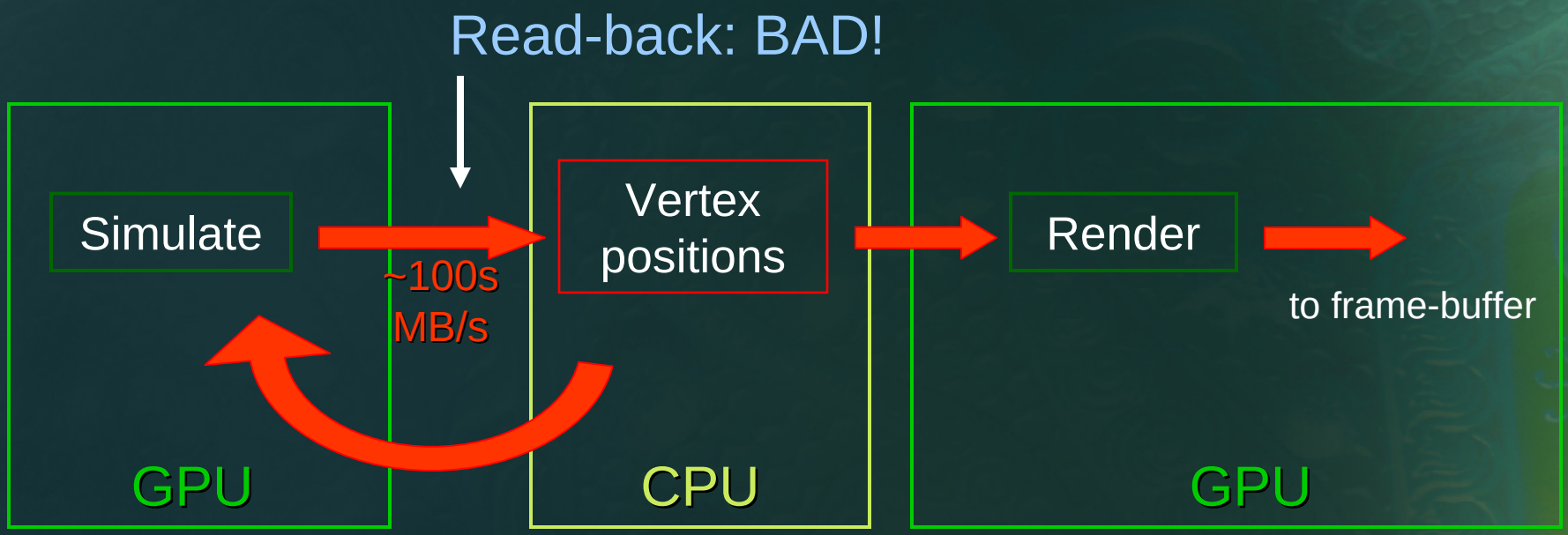


Typical Processing Allocation

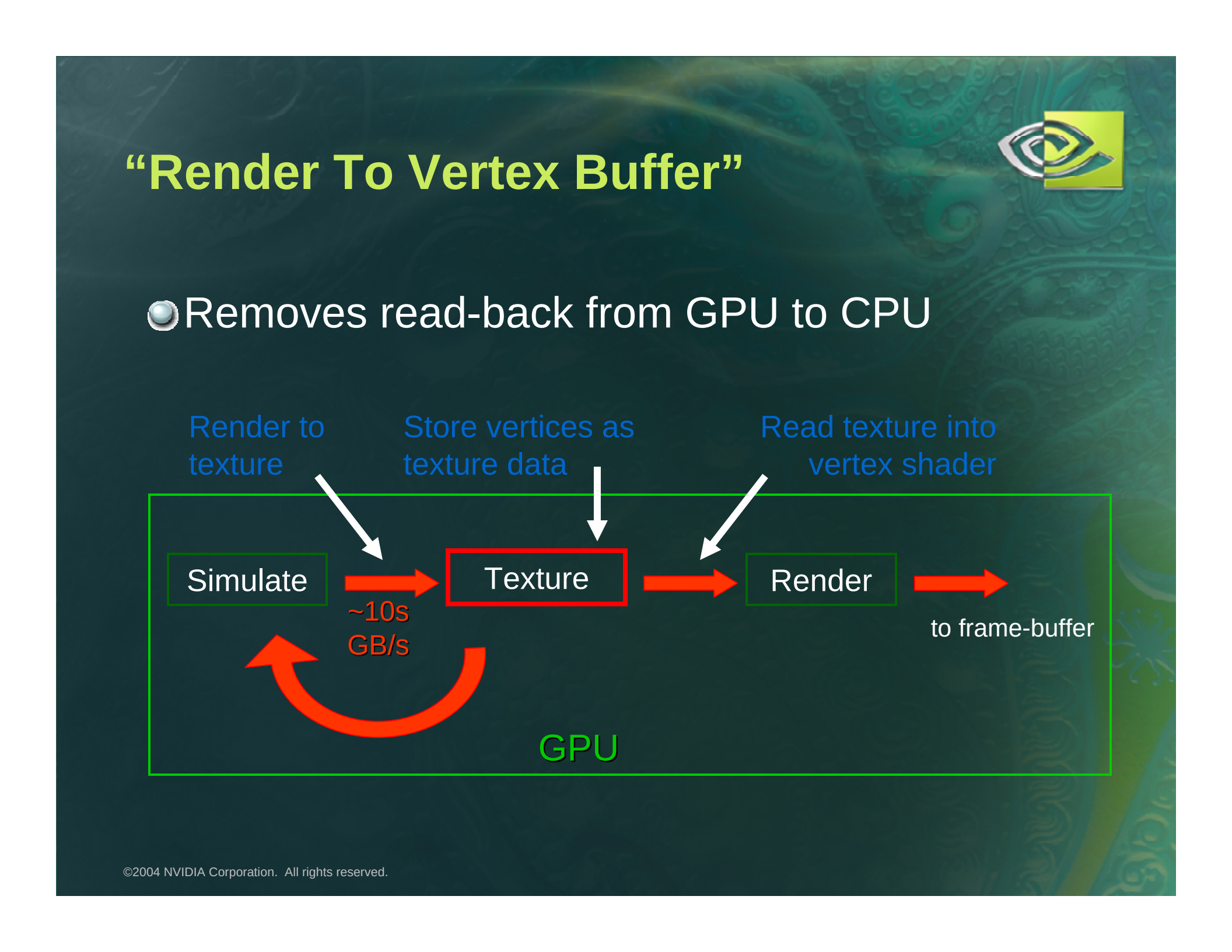


Simulating On the GPU?

- Use them programmable shaders!
  - The read-back can kill you
  - This is for PCI. PCI Express is better.

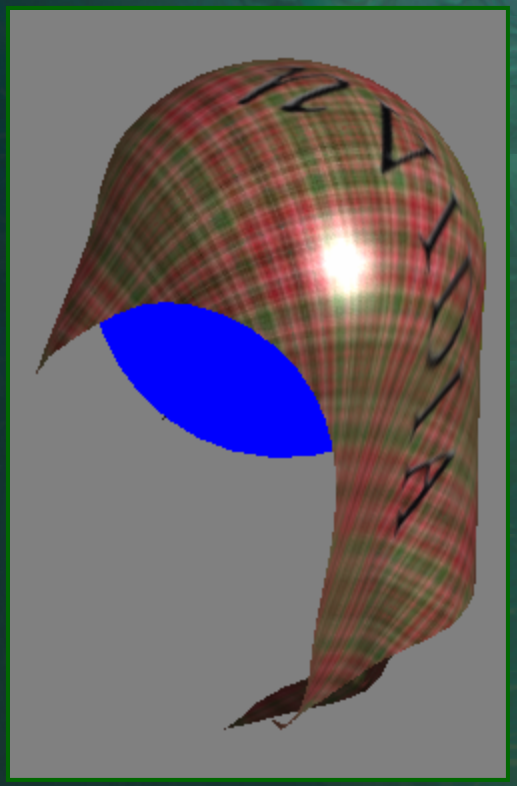


“Render To Vertex Buffer”



- Removes read-back from GPU to CPU

Examples



- Cloth
  - Collide cloth against scene
  - Run cloth physics: damped springs
- Displacement Mapping
  - Displace vertices

More Examples

- Snow/Sand accumulation
  - Simulate friction/sliding
- Wind (simulation) bending grass
- Particle Systems
- Water waves/wakes

Rendering Water – Algorithm Overview

- Perform water simulation in pixel shader
  - Render to texture (D3DFMT_A16B16G16R16F)
- Render refraction and reflection maps
- Render water surface
  - Use simulation results via VS3.0 vertex texture fetch
  - Compute perturbed texture coordinates
  - Combine refraction and reflection using Fresnel term