Interactive Ray Tracing with CUDA

David Luebke and Steven Parker
NVIDIA Research

Ray Tracing & Rasterization

Rasterization

- For each triangle:
  - Find the pixels it covers
  - For each pixel: compare to closest triangle so far

Ray tracing

- For each pixel:
  - Find the triangles that might be closest
  - For each triangle: compute distance to pixel

When all triangles/pixels have been processed, we know the closest triangle at all pixels

Ray Tracing & Rasterization

Rasterization

- For each triangle:
  - Find the pixels it covers
  - For each pixel: compare to closest triangle so far

Requires Z-buffer: track distance per pixel

Ray tracing

- For each pixel:
  - Find the triangles that might be closest
  - For each triangle: compute distance to pixel

Requires spatial index: a spatially sorted arrangement of triangles

When all triangles/pixels have been processed, we know the closest triangle at all pixels

Myths of Ray Tracing & Rasterization

- Ray tracing is clean, rasterization is ugly
  - Both are ugly
- Ray tracing is sublinear, rasterization linear in primitives
  - Rasterization uses culling techniques
- Ray tracing is linear, rasterization sublinear in pixels
  - Ray tracing uses packets & frustum tracing

Ray Tracing vs. Rasterization

- Rasterization is fast
  - but needs cleverness to support complex visual effects
- Ray tracing supports complex visual effects
  - but needs cleverness to be fast

Why Rasterization?

- Fast & Efficient
- Ubiquitous – part of workflow, pipeline
- Great for displacement-mapped geometry
- Developers know how to make beautiful pictures...

Why Rasterization?

From Battlefield: Bad Company, EA Digital Illusions CE AB

Why Rasterization?

From Battlefield: Bad Company, EA Digital Illusions CE AB

Why Rasterization?

From Crysis, Crytek GmbH

Why Rasterization?

From Crysis, Crytek GmbH

Why ray tracing?

- Ray tracing unifies rendering of visual phenomena
  - fewer algorithms with fewer interactions between algorithms
- Easier to combine advanced visual effects robustly
  - soft shadows
  - subsurface scattering
  - indirect illumination
  - transparency
  - reflective & glossy surfaces
  - depth of field
  - …

Ray Tracing vs. Rasterization

- ~~Rasterization is fast~~
  - ~~but needs cleverness to support complex visual effects~~
- ~~Ray tracing supports complex visual effects~~
  - ~~but needs cleverness to be fast~~

Use both!

Ray tracing (Appel 1968, Whitted 1980)

Distributed Ray Tracing (Cook, 1984)

Path Tracing (Kajiya, 1986)

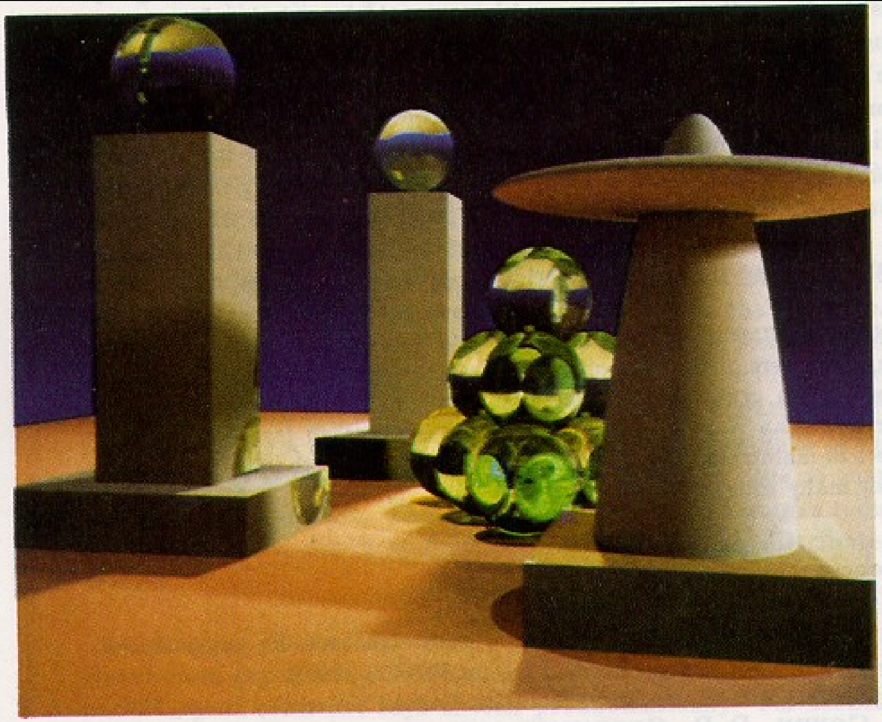
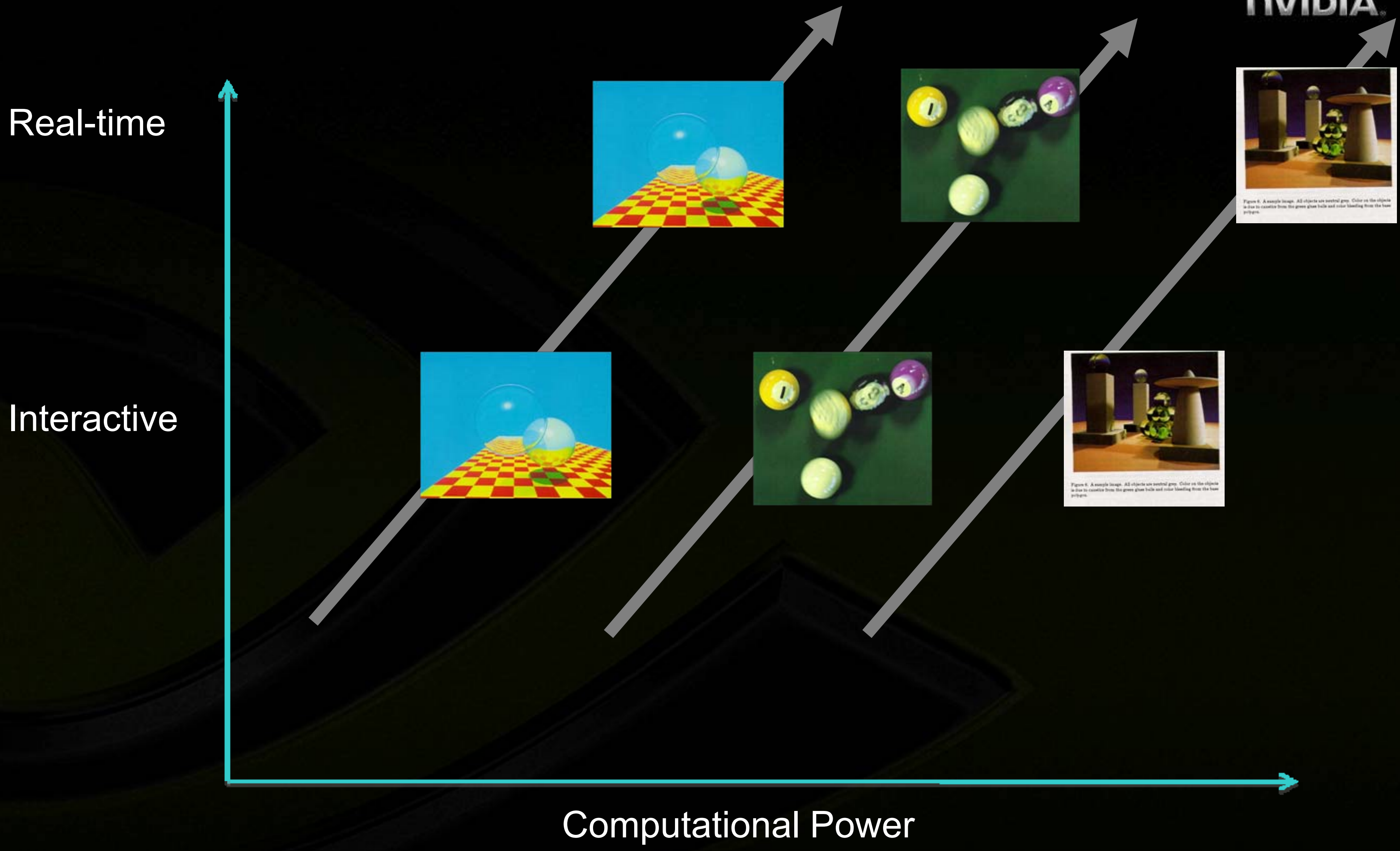


Figure 6. A sample image. All objects are neutral grey. Color on the objects is due to caustics from the green glass balls and color bleeding from the base polygon.

Ray Tracing Regimes



Industrial strength ray tracing

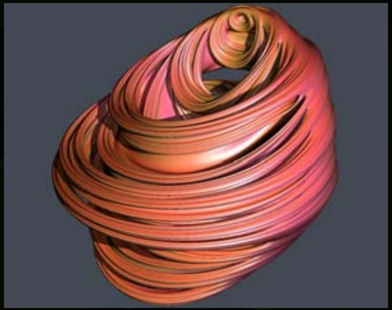
- mental images is market leader for ray tracing software
- Applicable in numerous markets: automotive, design, architecture, film

Importance










Importance





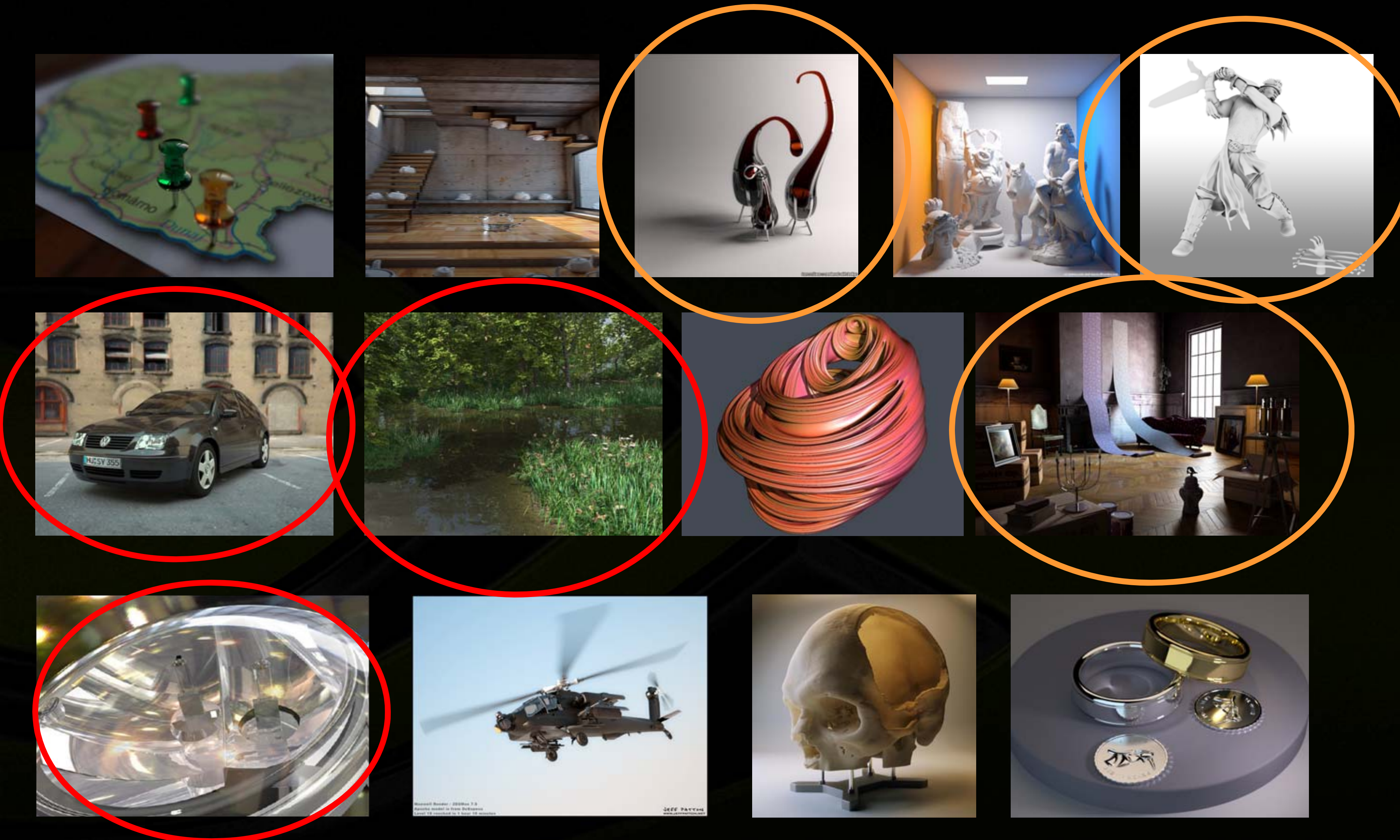






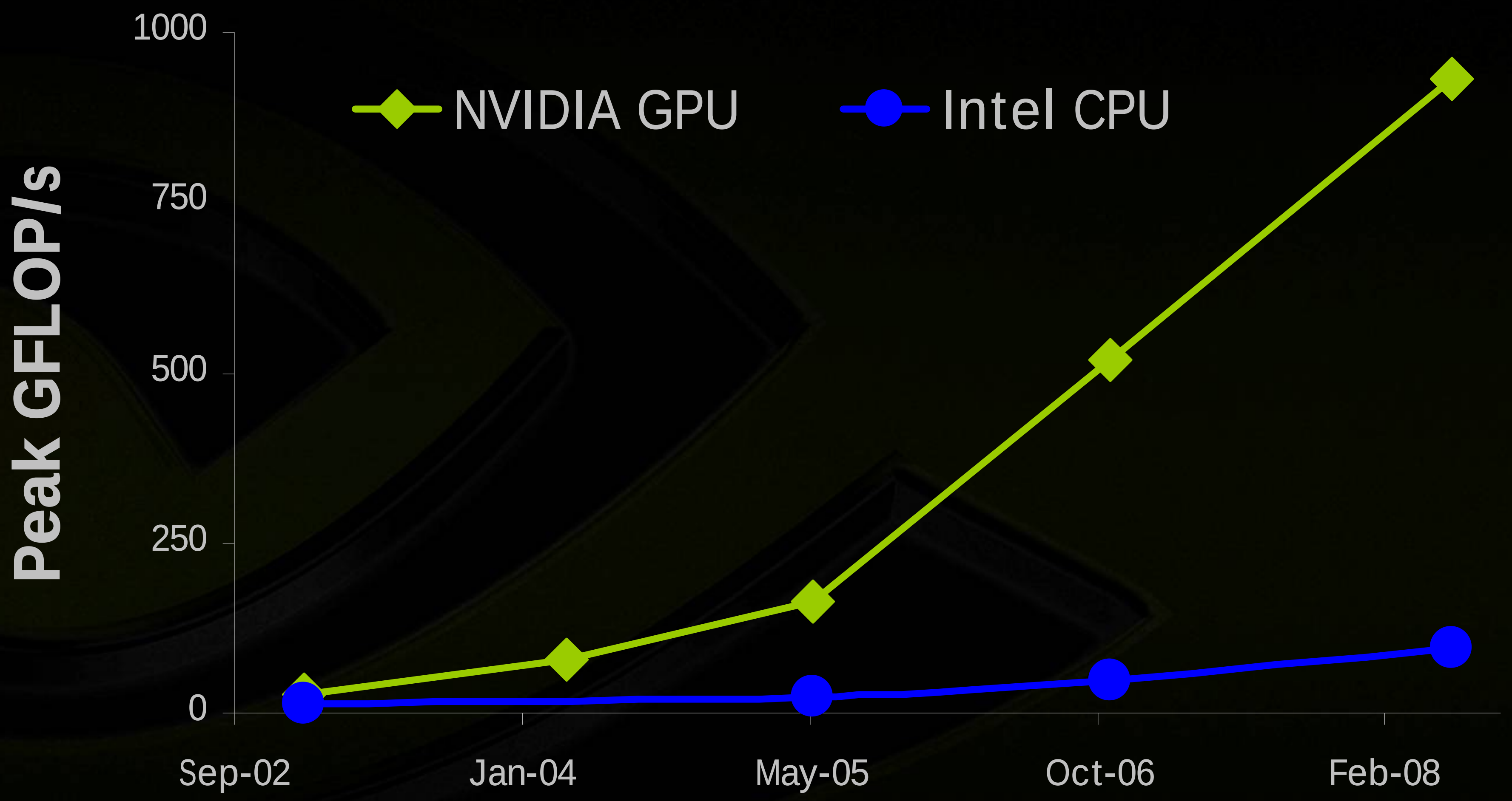


Importance



Interactive Ray Tracing

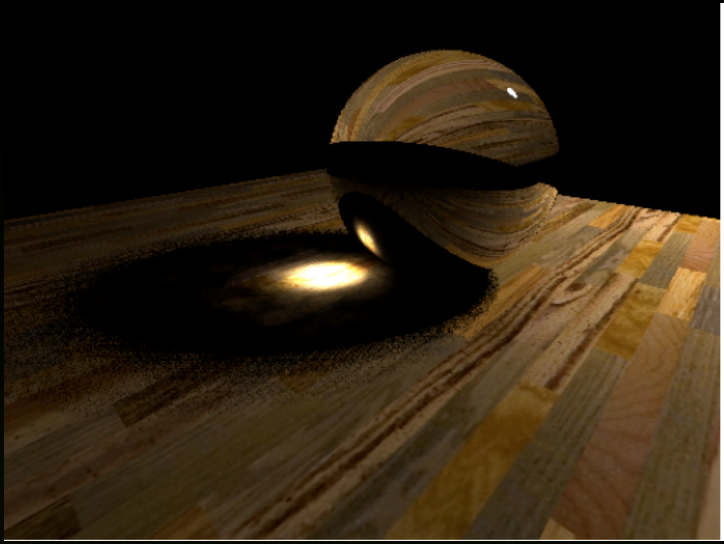
GPUs Are Fast & Getting Faster



Why GPU Ray Tracing?

- Abundant parallelism, massive computational power
- GPUs excel at shading
- Opportunity for hybrid algorithms

GPU Ray Tracing



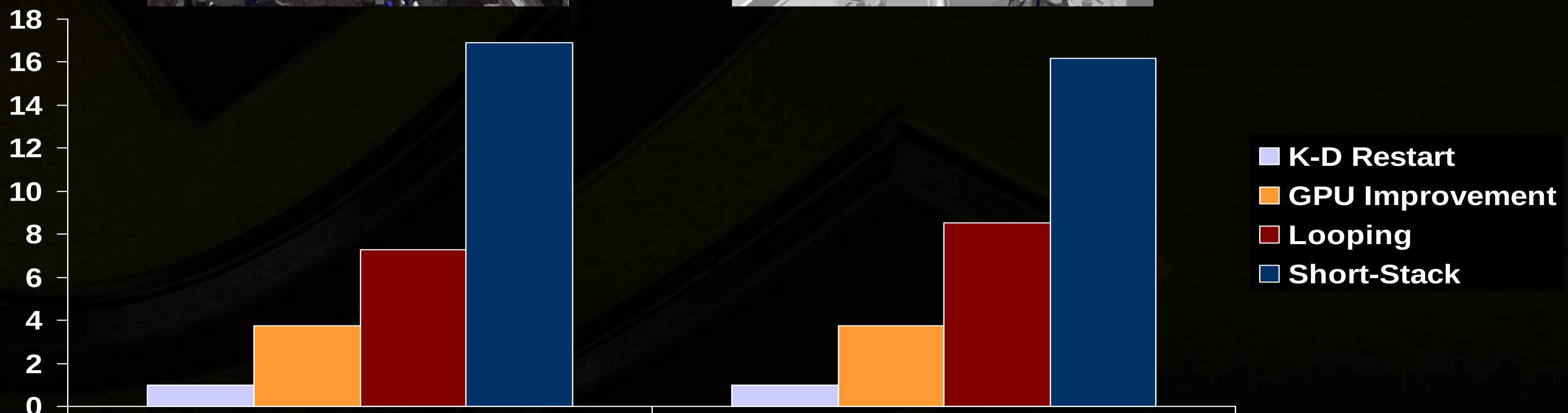
Purcell et al., *Ray Tracing on Programmable Graphics Hardware*, SIGGRAPH 2002

Purcell et al., *Photon Mapping on Programmable Graphics Hardware*, Graphics Hardware 2004

Popov et al., *Stackless KD-Tree Traversal for High Performance GPU Ray Tracing*, Computer Graphics Forum, Oct 2007

Popov et al., *Realtime Ray Tracing on GPU with BVH-based Packet Traversal*, Symposium on Interactive Ray Tracing 2007

GPU Ray Tracing



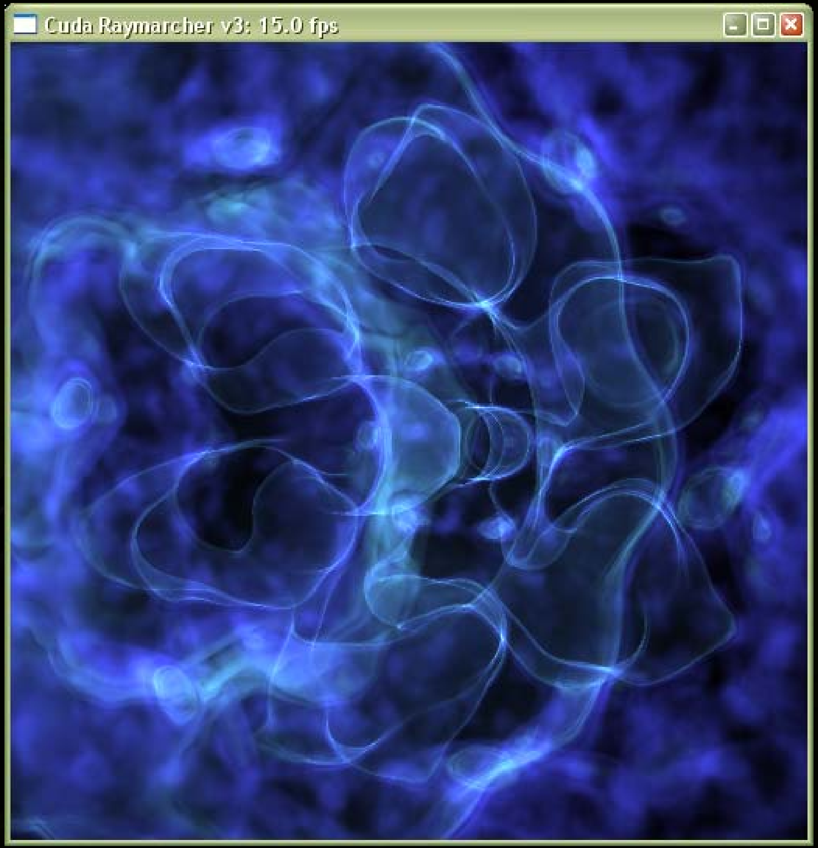
Horn et al., *Interactive k-D Tree GPU Raytracing*
ACM SIGGRAPH Symposium on Interactive 3D Graphics 2007

GPU Ray Tracing

Zhou et al., *Real-Time KD-Tree Construction on Graphics Hardware*
Microsoft Research Asia Tech Report 2008-52

Volume Ray Casting

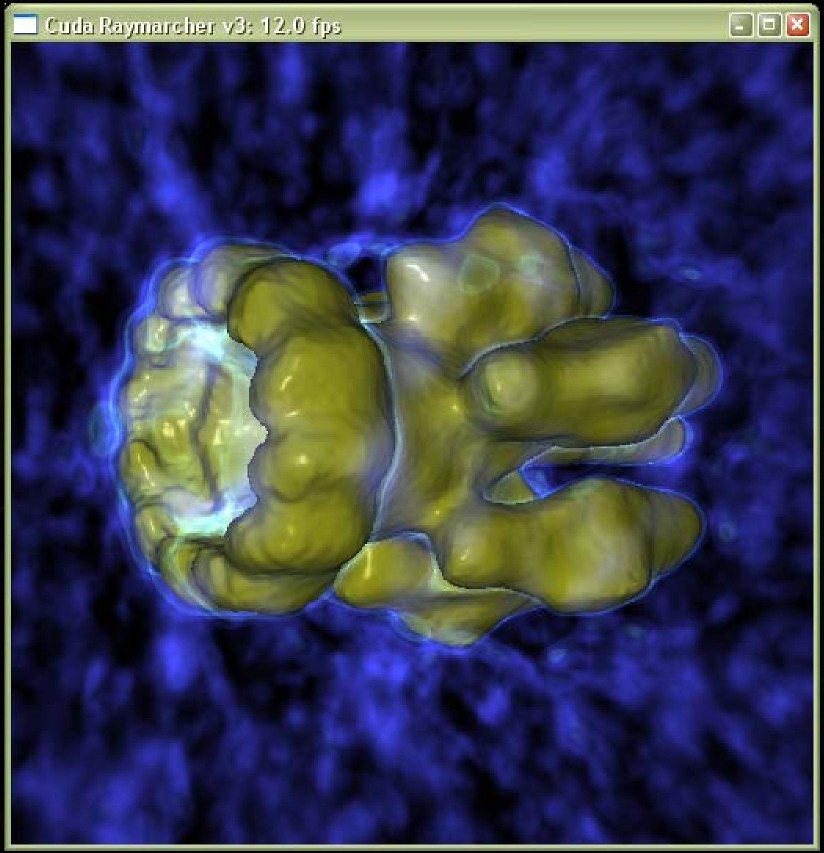
- Ray marching for isosurfaces + direct volume rendering
- Electron density of virus from cryoelectroscopy
- Vital to change isosurface interactively
- Great match for CUDA



Volume Ray Casting With CUDA
Marsalek & Slusallek 2008

Volume Ray Casting

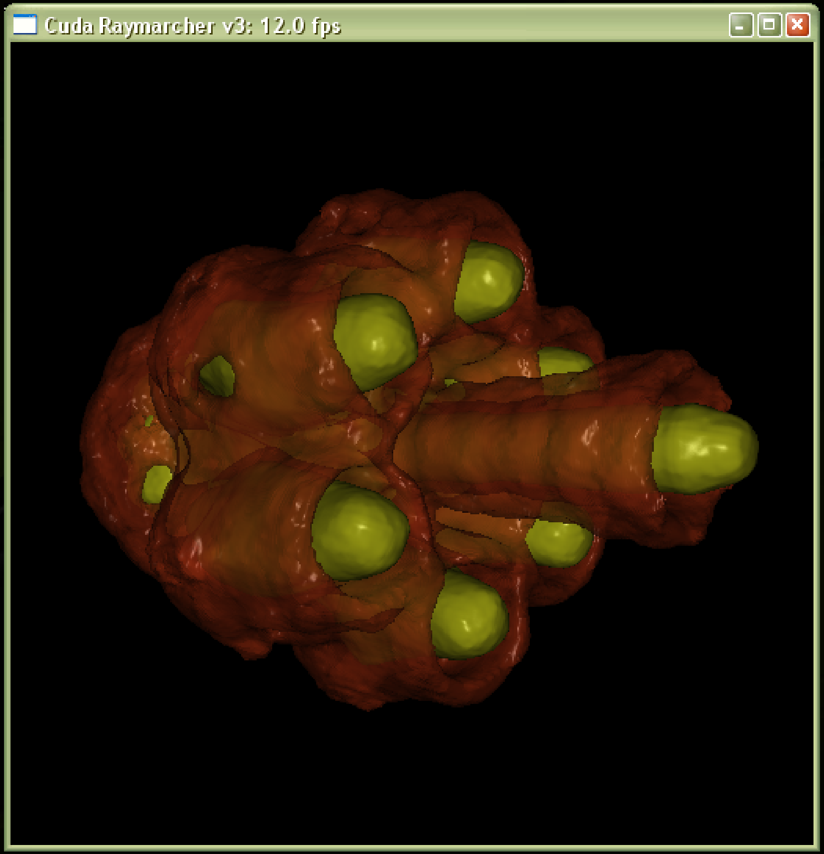
- Ray marching for isosurfaces + direct volume rendering
- Electron density of virus from cryoelectroscopy
- Vital to change isosurface interactively
- Great match for CUDA



Volume Ray Casting With CUDA
Marsalek & Slusallek 2008

Volume Ray Casting

- Ray marching for isosurfaces + direct volume rendering
- Electron density of virus from cryoelectroscopy
- Vital to change isosurface interactively
- Great match for CUDA



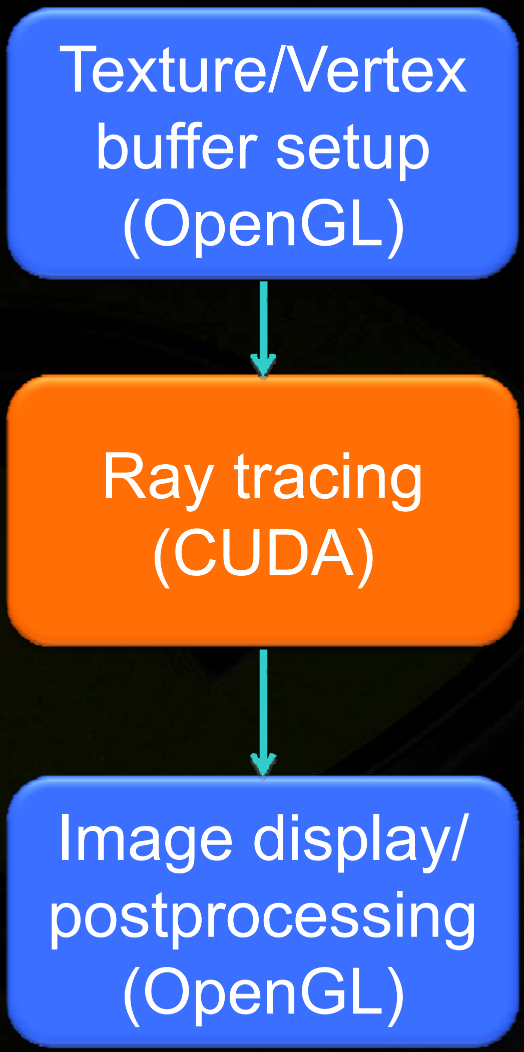
Volume Ray Casting With CUDA
Marsalek & Slusallek 2008

City demo

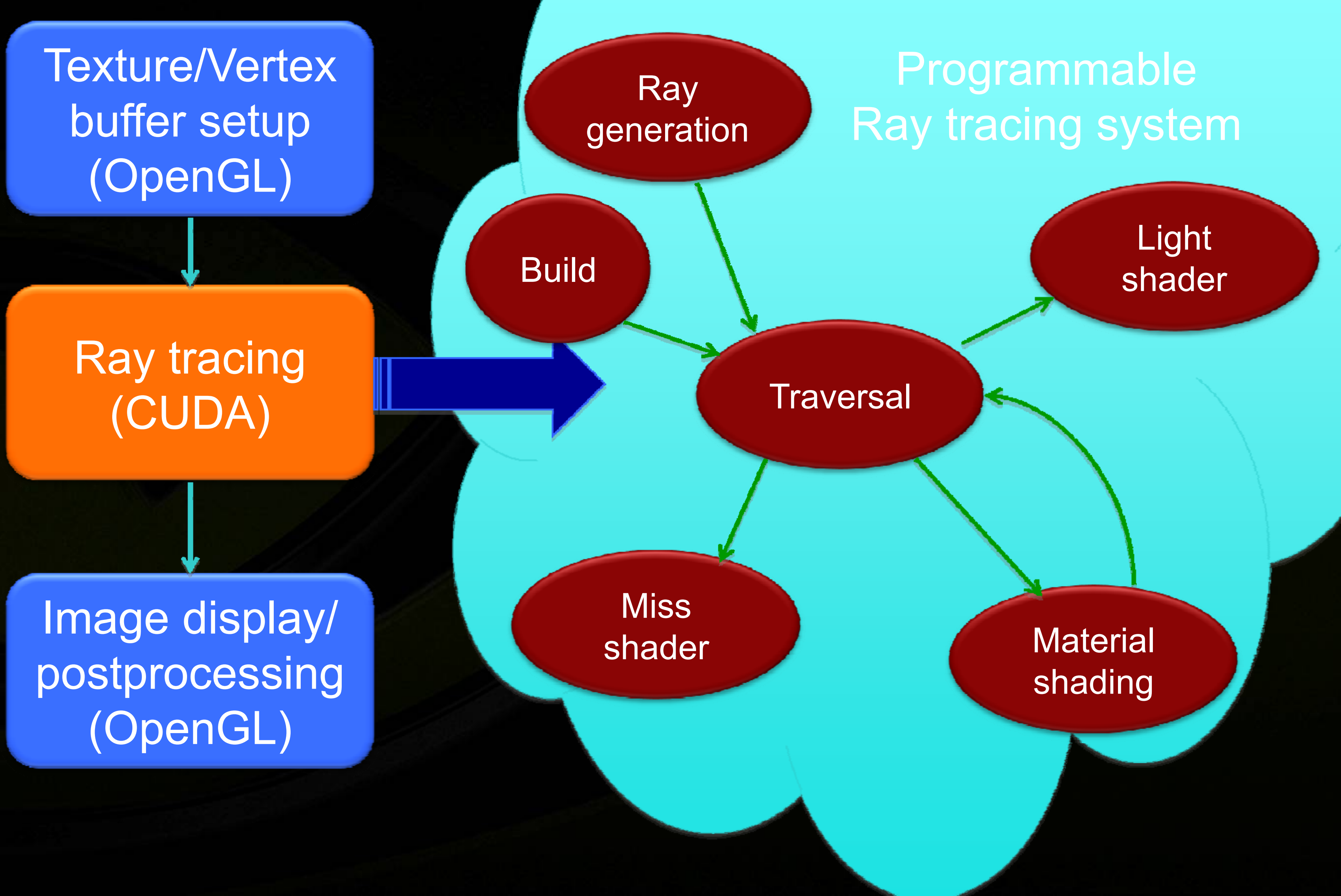
- Real system
- NVSG-driven animation and interaction
- Programmable shading
- Modeled in Maya, imported through COLLADA
- Fully ray traced

2 million polygons
Bump-mapping
Movable light source
5 bounce reflection/refraction
Adaptive antialiasing

System Diagram – ray tracing



System Diagram – ray tracing



Key Parallel Abstractions in CUDA

0. Zillions of lightweight threads

1. Hierarchy of concurrent threads

2. Lightweight synchronization primitives

3. Shared memory model for cooperating threads

Key Parallel Abstractions in CUDA

0. Zillions of lightweight threads
   → Simple decomposition model

1. Hierarchy of concurrent threads
   → Simple execution model

2. Lightweight synchronization primitives
   → Simple synchronization model

3. Shared memory model for cooperating threads
   → Simple communication model

Hierarchy of concurrent threads

- Parallel kernels composed of many threads
  - all threads execute the same sequential program

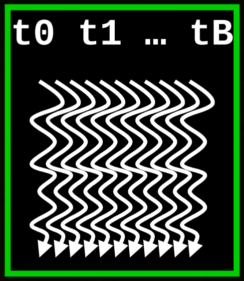
Hierarchy of concurrent threads

- Parallel kernels composed of many threads
  - all threads execute the same sequential program

Thread *t*

- Threads are grouped into thread blocks
  - threads in the same block can cooperate

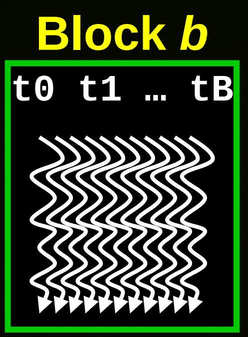
Block *b*



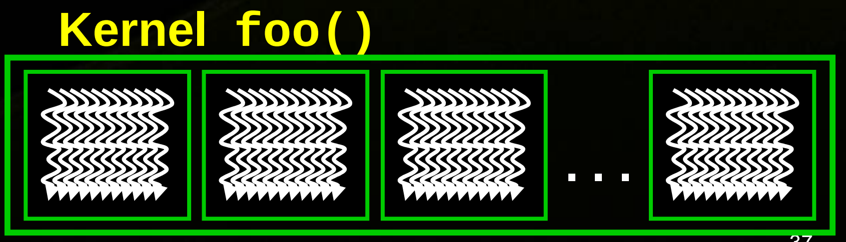
Hierarchy of concurrent threads

- Parallel kernels composed of many threads
  - all threads execute the same sequential program

- Threads are grouped into thread blocks
  - threads in the same block can cooperate



- Threads/blocks have unique IDs



Big Picture

GTX 280 supports up to 30,720 concurrent threads!

1. Big strategic optimization: minimize per-thread state

2. Otherwise, take simplest option
   - Clever optimizations usually violate rule 1

3. *Lots* of opportunity for further research
   - Coalescing work for increased coherence (work queues)
     - Data coherence
     - Execution coherence
   - Ray space hierarchies
   - Radical departures from traditional methods (see RT08)

Details – Algorithmic

- Top-level BVH + subtrees (BVH or k-d tree)
  - Supports rigid motion, instancing
  - Rebuild/refit easy to add
- Traversal + intersection + shading “megakernel”
  - while – while vs. if – if
- Highly variable thread lifetimes!
  - Software load-balancing

Details - Implementation

- Triangle & hierarchy data through texture cache
- Ray tree recursion
  - Stack in local memory to store shader live variables

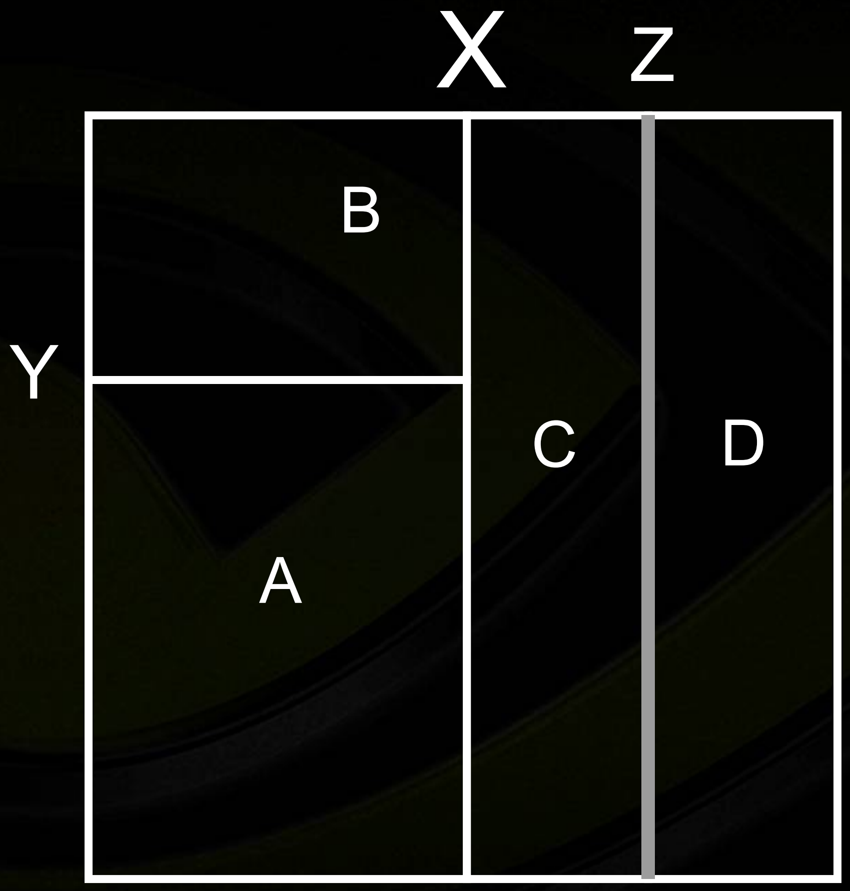
Short Stack

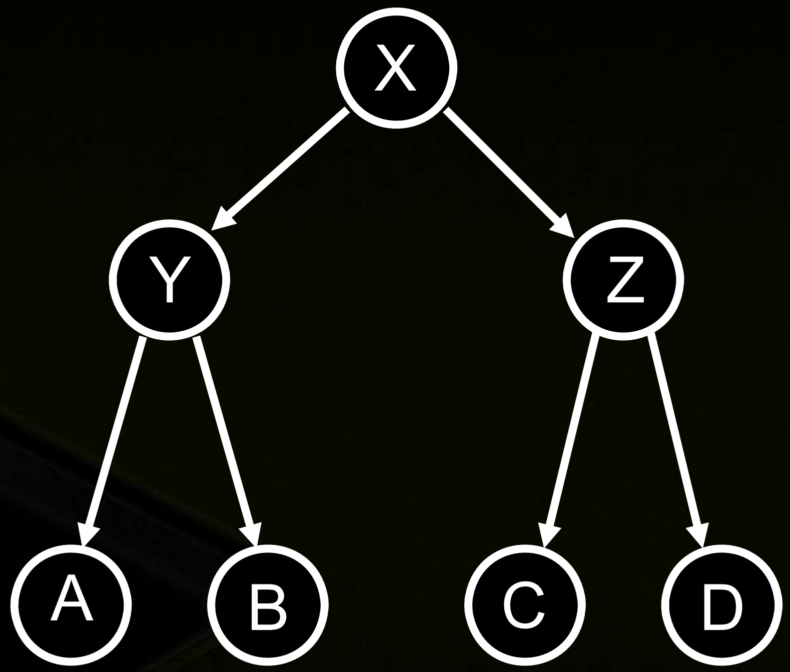
- Goal: minimize state per thread
- Strategy: replace traversal stack with *short stack*

Horn et al., *Interactive k-D Tree GPU Raytracing*, I3D 2008

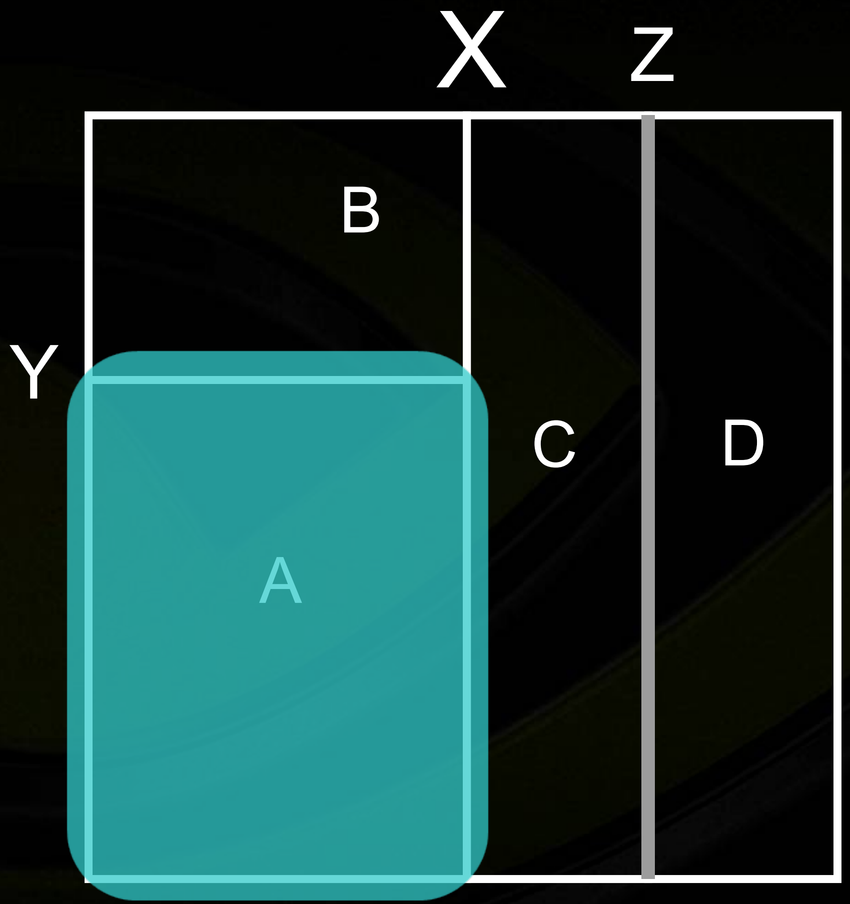
Slides courtesy Daniel Horn

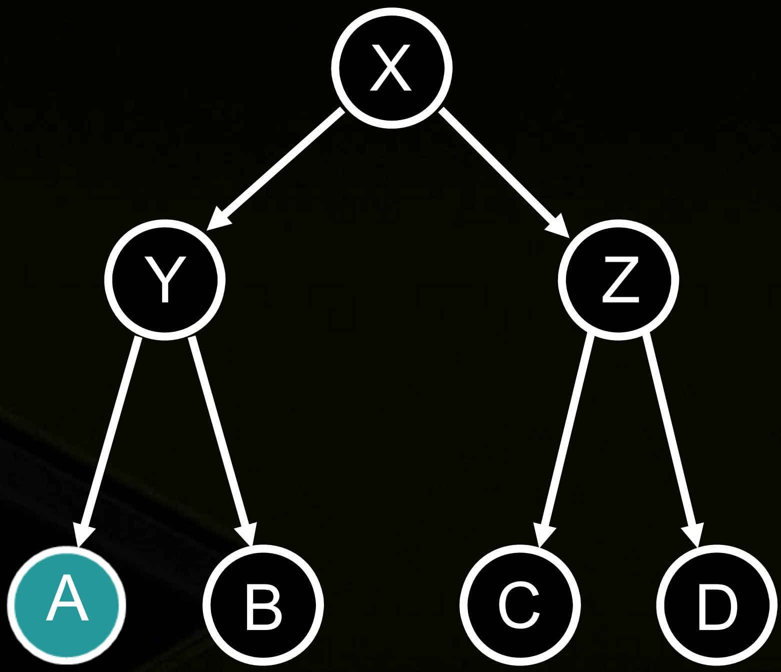
KD-Tree



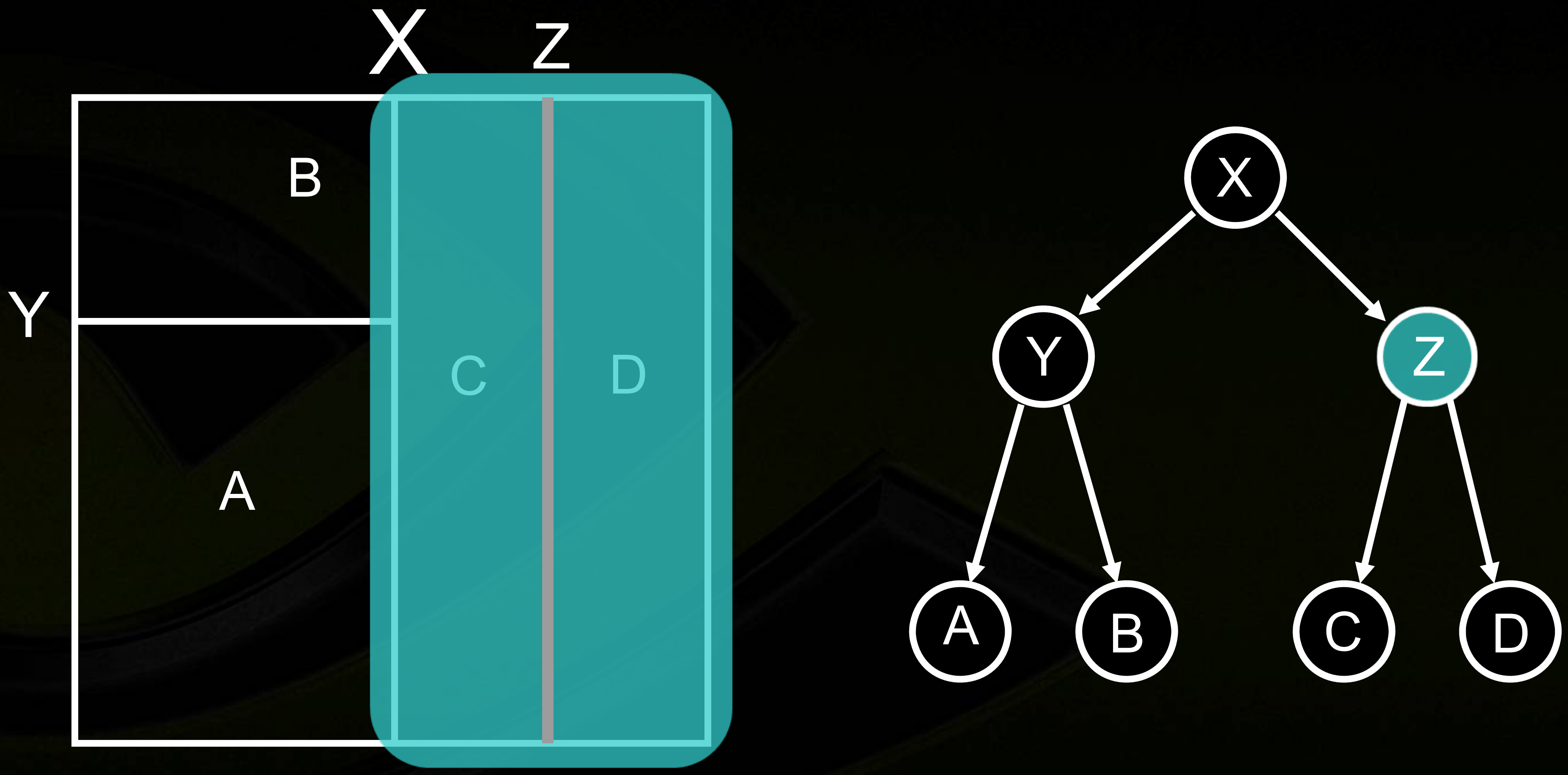


KD-Tree

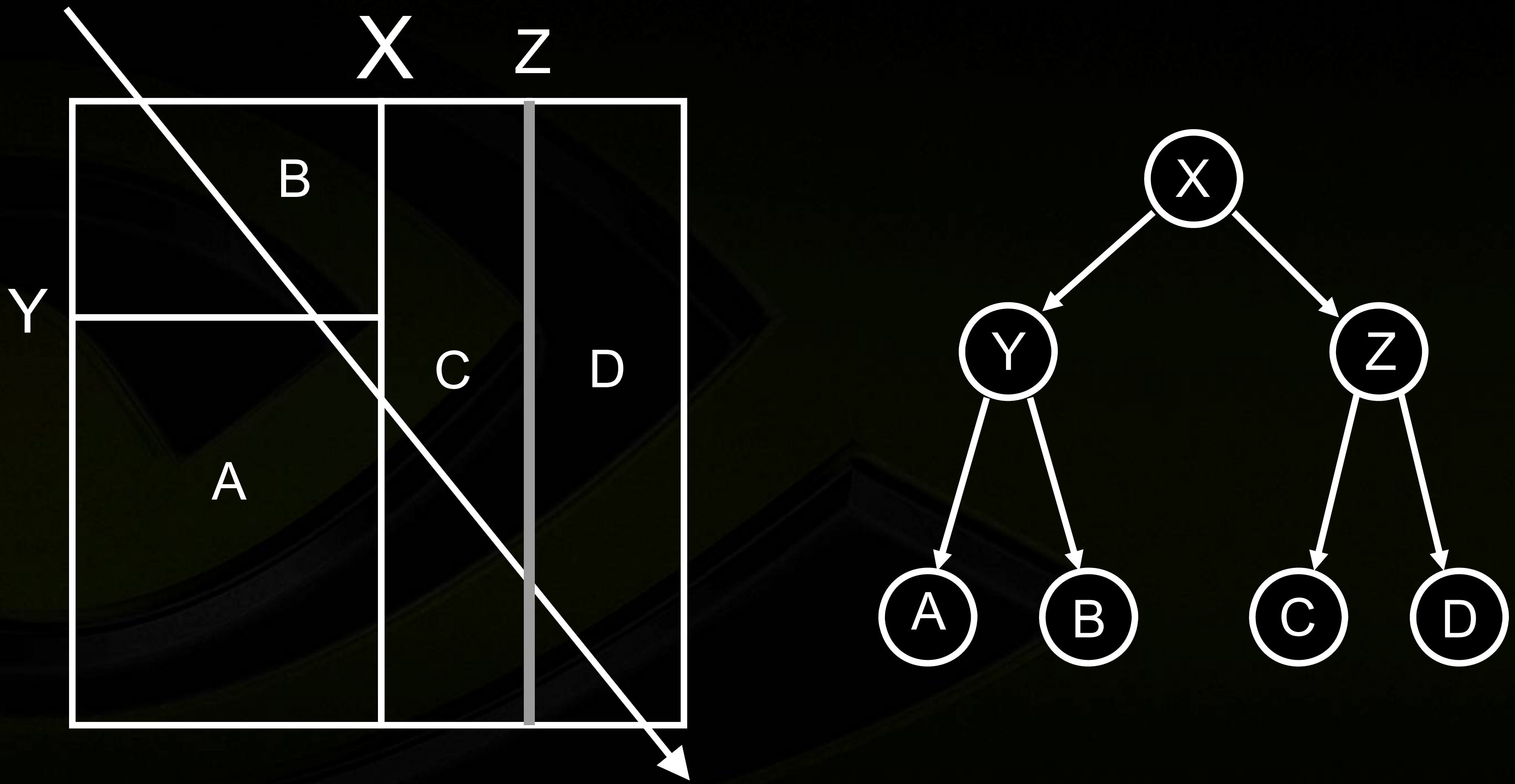




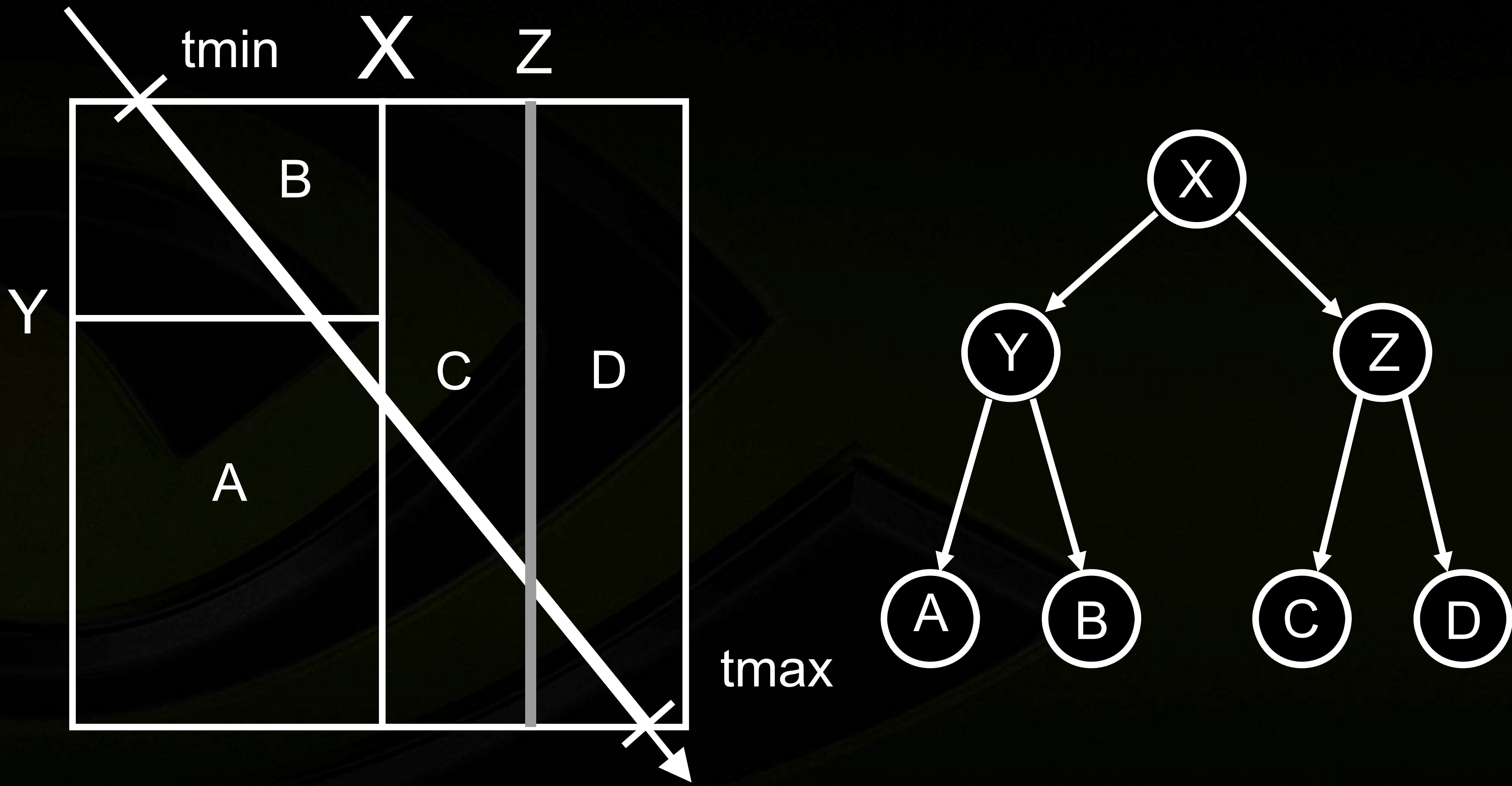
KD-Tree



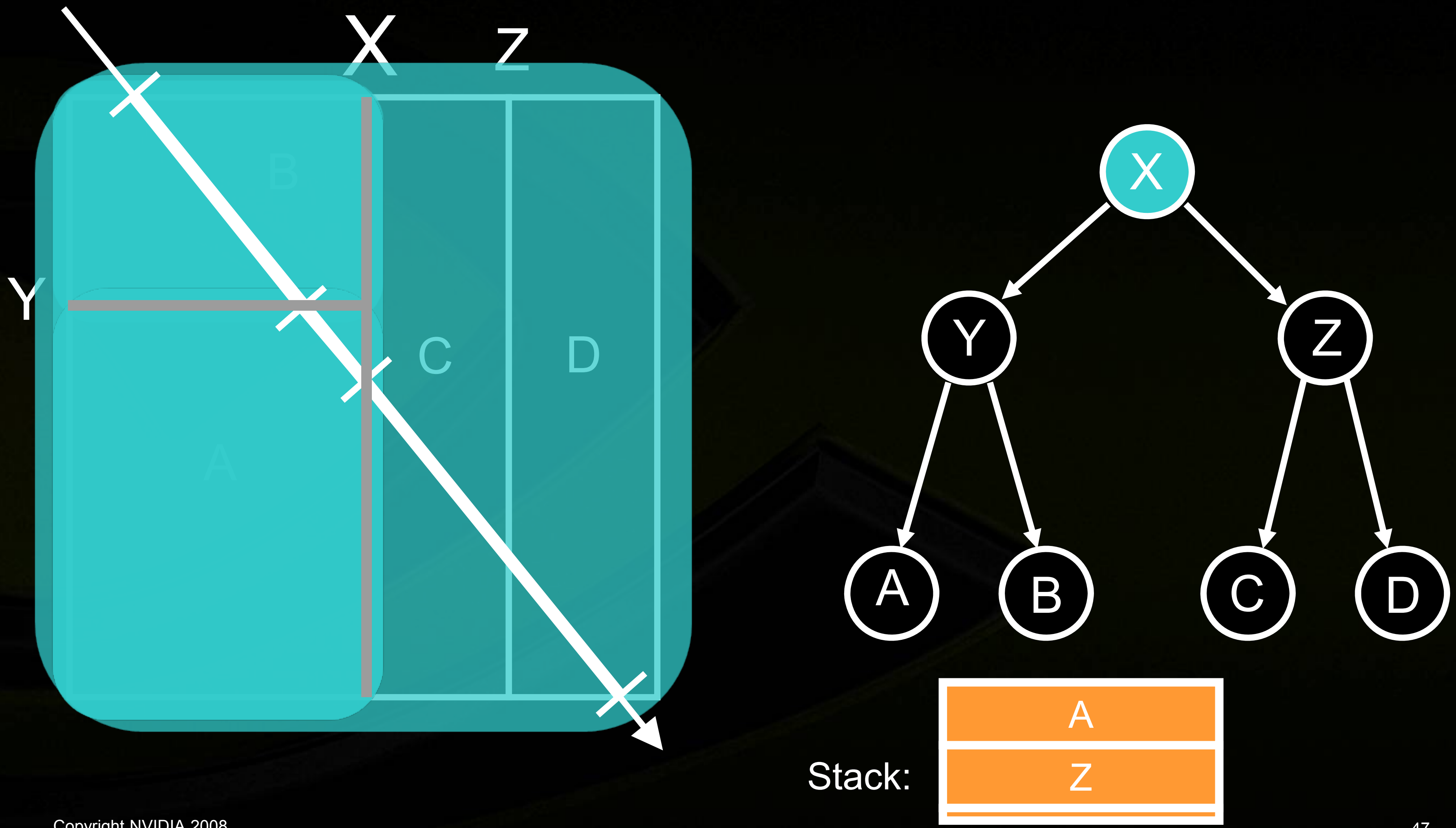
KD-Tree



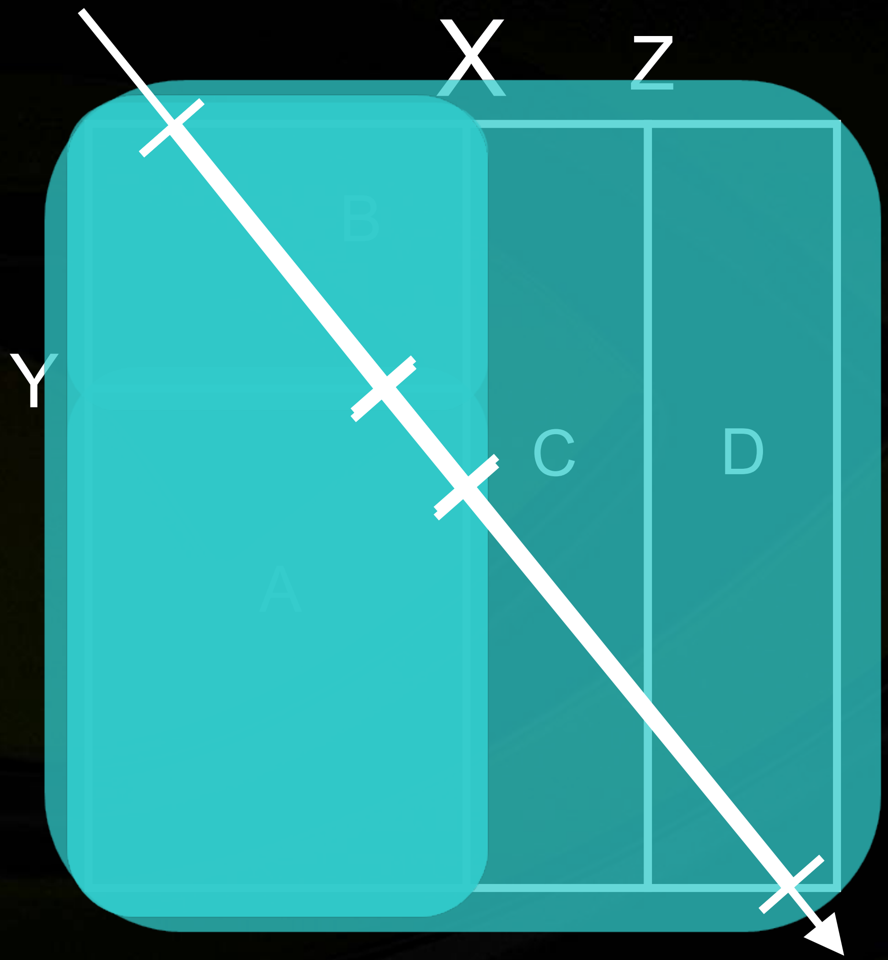
KD-Tree



KD-Tree Traversal

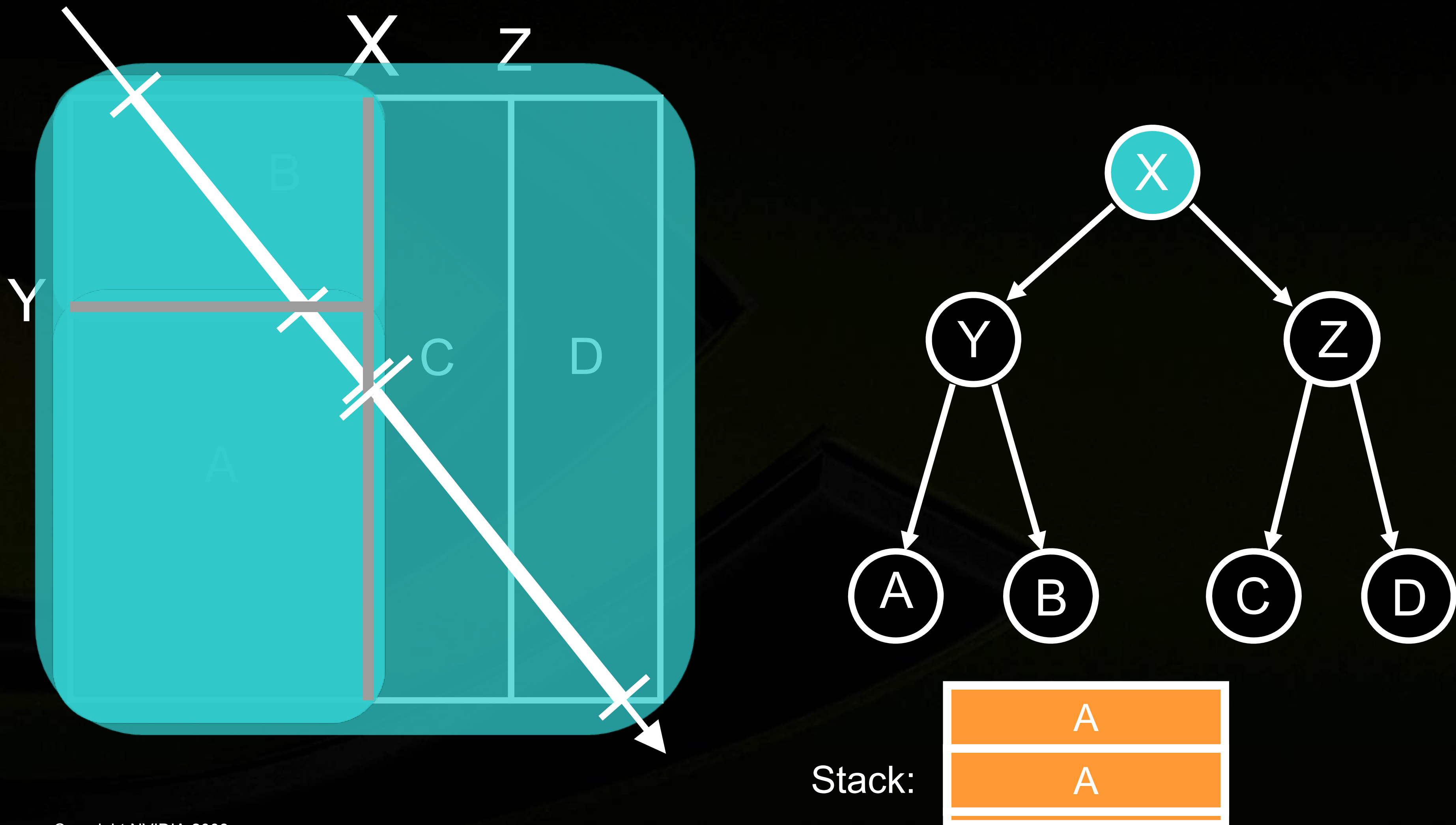


KD-Restart



- Standard traversal
  - Omit stack operations
  - Proceed to 1st leaf
- If no intersection
  - Advance (tmin,tmax)
  - Restart from root
- Proceed to next leaf

KD-Restart with short stack (size 1)



Short Stack Cache

- **Even better:**
  - Each thread stores full stack in memory non-blocking writes
  - Cache top of stack locally (registers or shared memory)
- Enables BVHs as well as k-d trees
  - 5-10% faster in our current implementation

Details – Algorithmic

- Top-level BVH + subtrees (BVH or k-d tree)
  - Supports rigid motion, instancing
  - Rebuild/refit easy to add
- Traversal + intersection + shading “megakernel”
  - while – while vs. if – if
- Highly variable thread lifetimes!
  - Software load-balancing

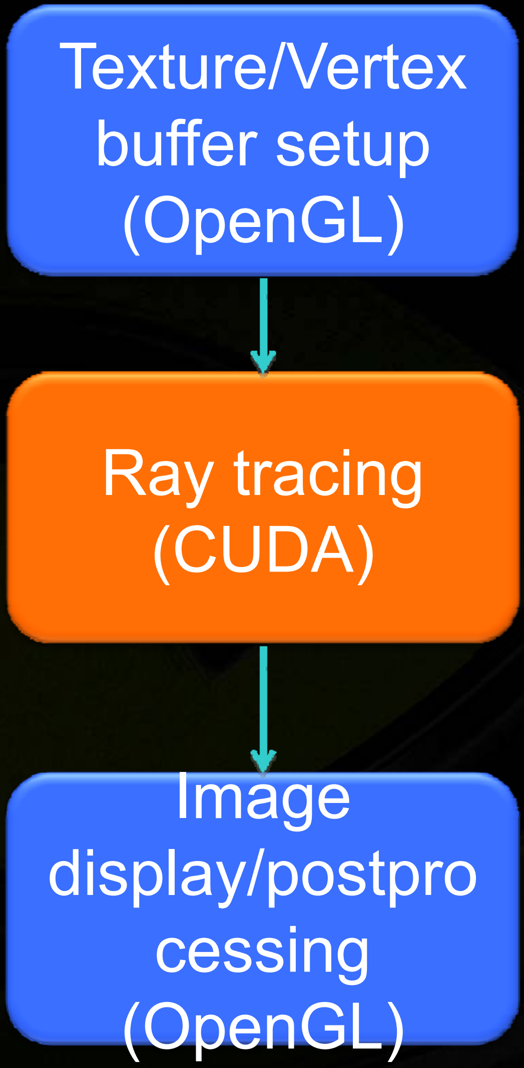
Details - Implementation

- Triangle & hierarchy data through texture cache
- Ray tree recursion
  - Stack in local memory to store shader live variables

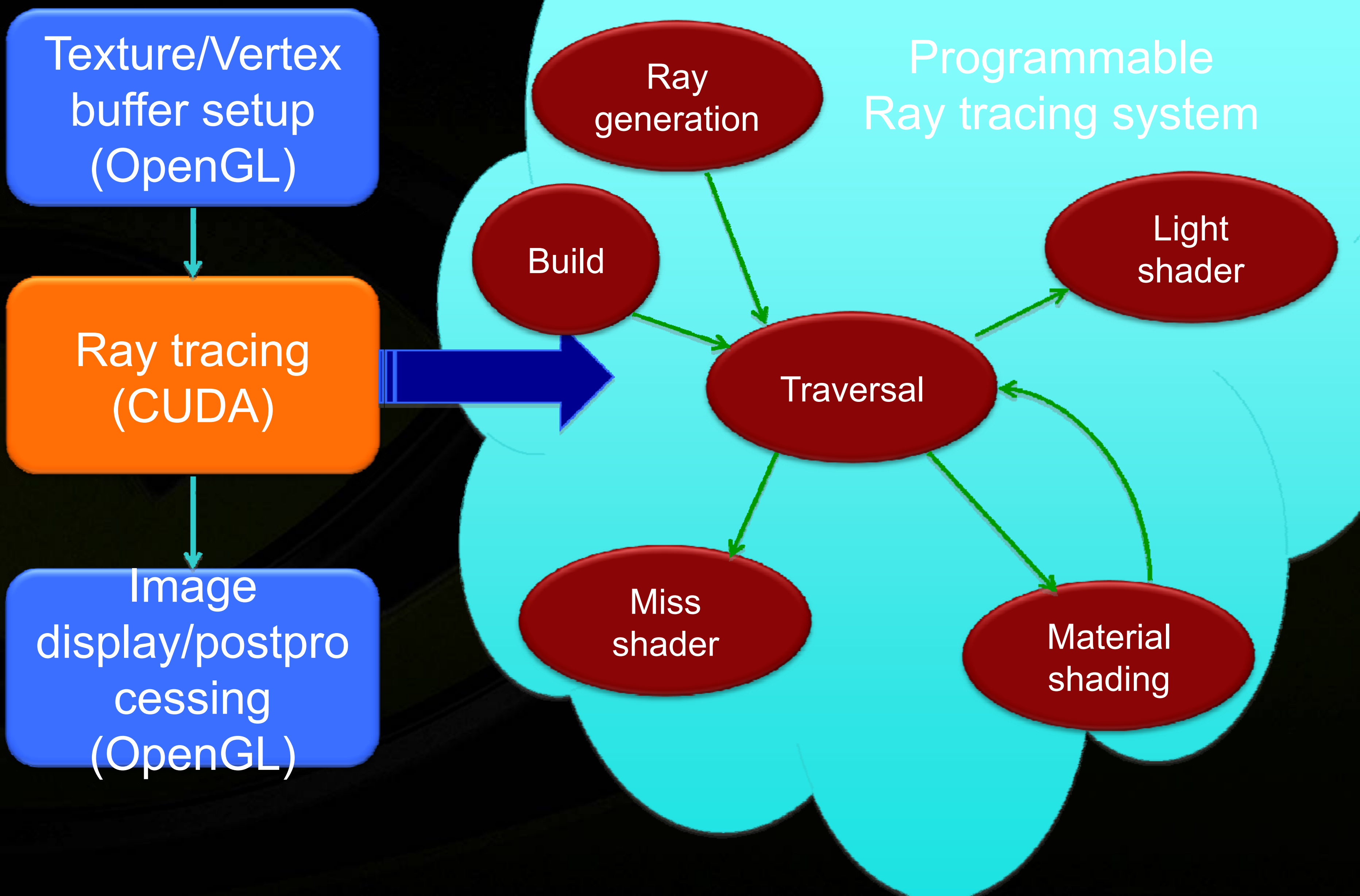
Big Picture

1. Big strategic optimization: minimize per-thread state
2. Otherwise, take simplest option
   - Clever optimizations usually violate rule 1
3. *Lots* of opportunity for further research
   - Coalescing work for increased coherence (work queues)
     - Data coherence
     - Execution coherence
   - Ray space hierarchies
   - Radical departures from traditional methods (see RT08)

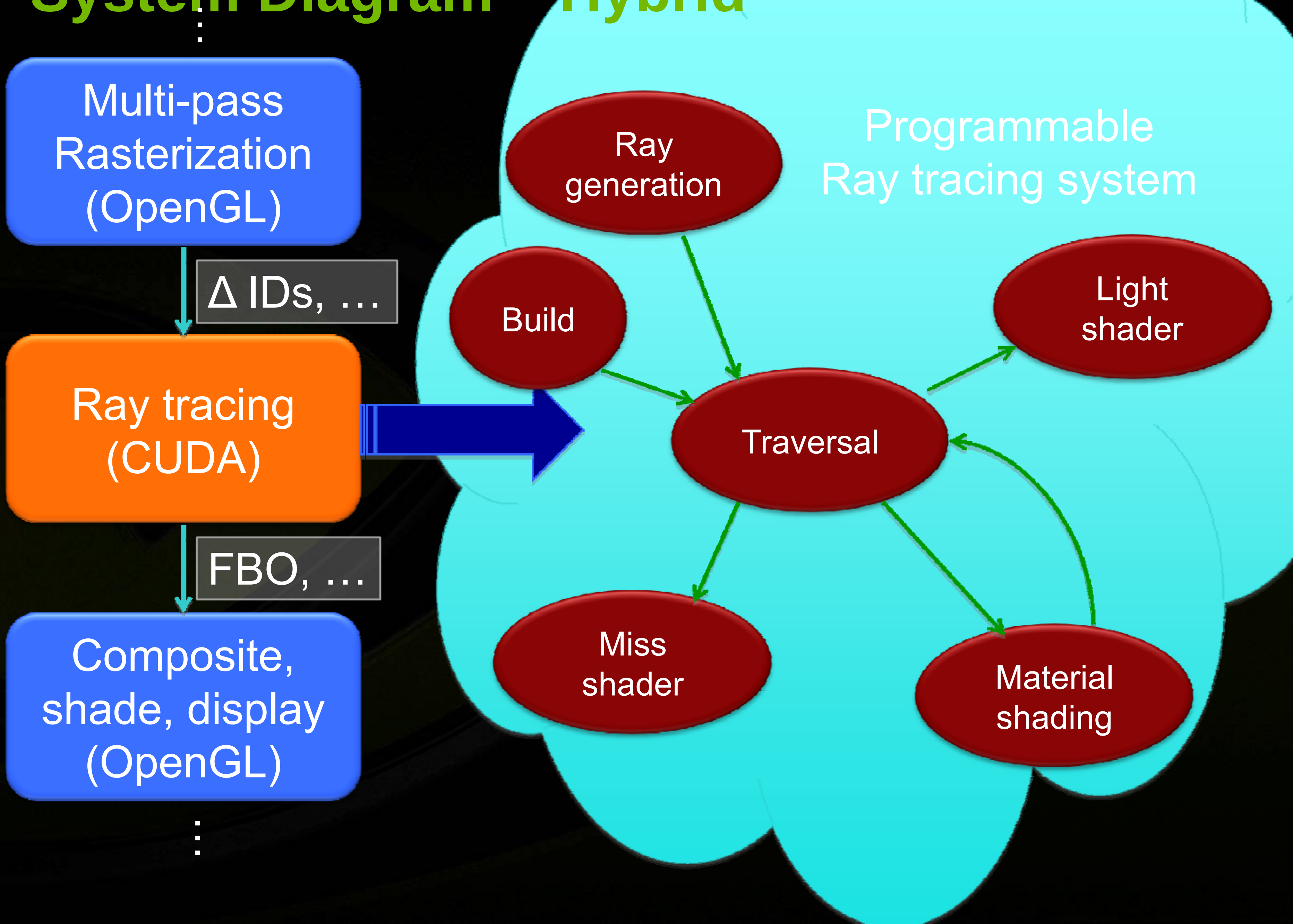
System Diagram – ray tracing



System Diagram – ray tracing

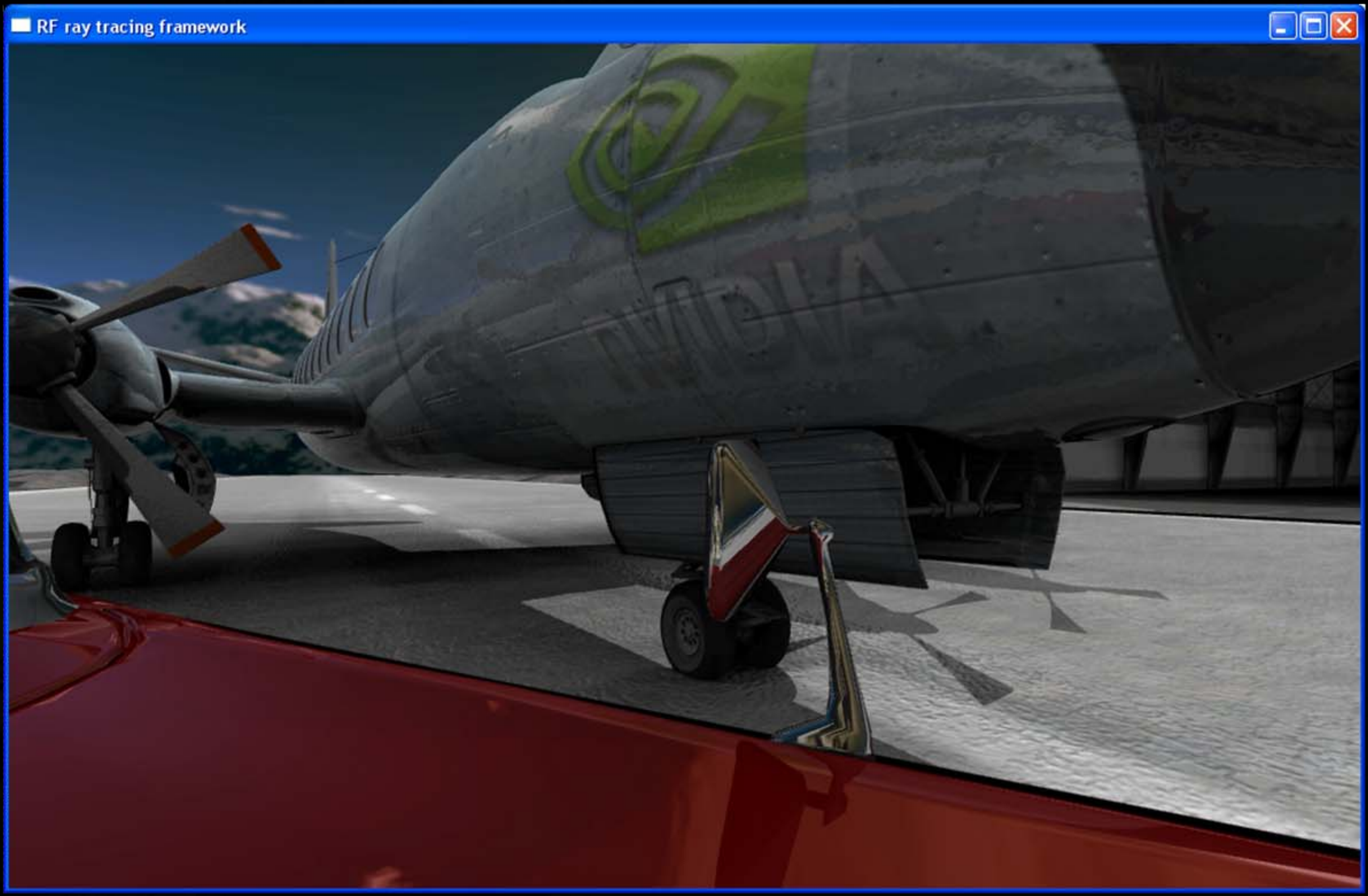


System Diagram – Hybrid



Hybrid Rendering – Primary Rays

Hybrid Rendering – Primary Rays

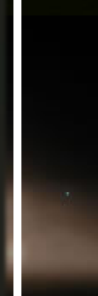


Hybrid Rendering – “God Rays”
Wyman & Ramsey, RT08

Creative Commons Image: Mila Zinkova

Hybrid Rendering – “God Rays”

Wyman & Ramsey, RT08



Creative Commons Image: Mila Zinkova

Indirect Illumination != Ray Tracing

No indirect lighting

With indirect lighting

Laine et al., *Incremental Instant Radiosity for Real-Time Indirect Illumination*
Eurographics Symposium on Rendering 2007

Solve the Right Problems!

- Tracing eye rays is uninteresting
  - rasterization wins, use it
- Scenes change dynamically at run time
  - can’t lovingly craft all spatial indices in off-line process
- Complex shaders & texturing are mandatory
  - a big weakness of CPU software tracers to date
- Need to provide a complete solution
  - construction, shading, application integration, hardware

Summary

- CUDA makes GPU ray tracing fast and practical
- A powerful tool in the interactive graphics toolbox
- Hybrid algorithms are the future
  - Leverage the power of rasterization with the flexibility of CUDA
  - Together they provide tremendous scope for innovation

Thank You!