

Ray Tracing on Programmable Graphics Hardware

Tim Purcell, Ian Buck,
Bill Mark*, Pat Hanrahan

Stanford University
* Currently at NVIDIA

Why Ray Tracing?

- Global illumination
- Good shadows!
 - Doom 3 will be using shadow volumes
 - Expensive!
 - Shadow maps are hard to use and prone to artifacts
- Efficient ray tracing based shadows could be the next killer feature for GPU



Doom 3 [id Software]

Why Ray Tracing?

- Output-sensitive algorithm
 - Sublinear in depth complexity
- Selective sampling
 - Frameless rendering [Bishop et al. 1994]
 - Render Cache [Walter et al. 1995]
 - Shading Cache [Tole et al. 2002]
- Interactive on clusters of PCs [Wald et al. 2001] and supercomputers [Parker et al. 1999]



Power Plant
[Wald et al. 2001]

Beyond Moore's Law

NVIDIA Historicals

Season	Product	MT/s	Yr rate	MF/s	Yr rate
2H97	Riva 128	5	-	100	-
1H98	Riva ZX	5	1.0	100	1.0
2H98	Riva TNT	5	1.0	180	3.2
1H99	Riva TNT2	8	1.0	333	3.4
2H99	GeForce	15	3.5	480	2.1
1H00	GeForce2 GTS	25	2.8	666	1.9
2H00	GeForce2 Ultra	31	1.5	1000	2.3
1H01	GeForce3	40	1.7	3200	10.2
1H02	GeForce4	65	1.6	4800	1.5

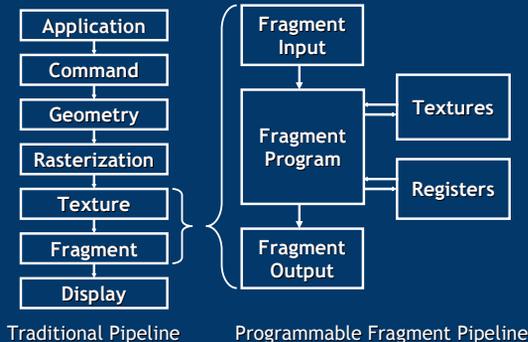
Courtesy of Kurt Akeley

1.8

2.4

Yearly growth well above Moore's Law (1.5)

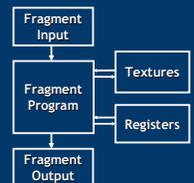
Graphics Pipeline



Fragment Programs Tomorrow

Hypothetical DirectX 9 / OpenGL 2.0 class GPU

- General, orthogonal instruction set
- Floating-point data types
- Resources
 - Large number of registers
 - Long program length
 - Multiple outputs
 - Unlimited texture lookups
 - Multiple levels of dependent texture lookup
- Data dependent loops and branches (OGL2)



Contributions

- Map complete ray tracer onto GPU
 - Ray tracing generally thought to be incompatible with the traditional graphics pipeline
- Abstract programmable fragment processor as a stream processor
- Map ray tracing to streaming computation
- Show that streaming GPU-based ray tracer is competitive with CPU-based ray tracer

Assumptions

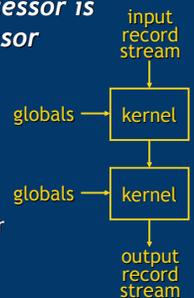
- Static scenes
- Triangle primitives only
- Uniform grid acceleration structure

Demo

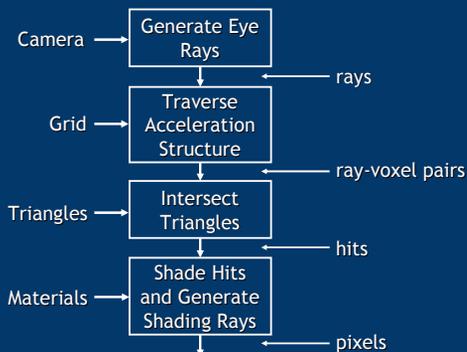
Stream Programming Model

Programmable fragment processor is essentially a stream processor

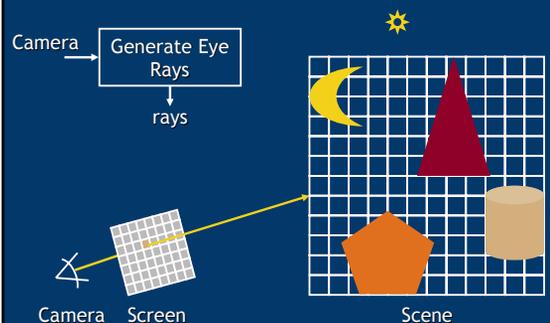
- **Kernels and streams**
 - Stream is a set of data records
 - Kernels operate on records
 - Streams connect kernels together
 - Kernels can read global memory



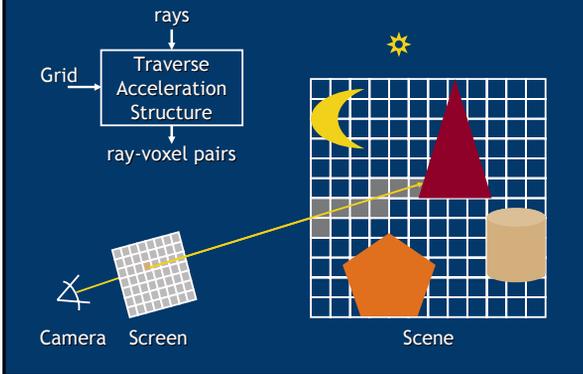
Streaming Ray Tracer (Simplified)



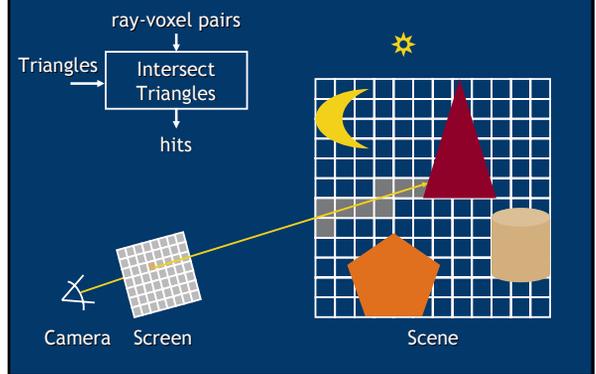
Eye Ray Generator



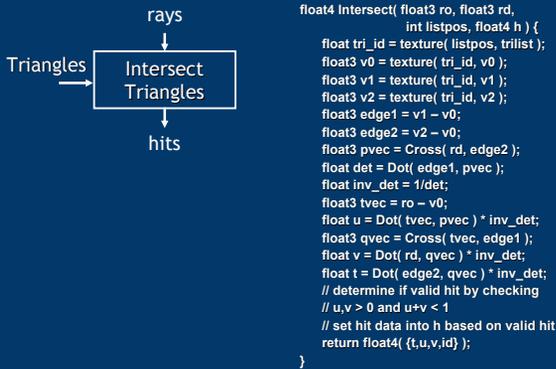
Traverser



Intersector



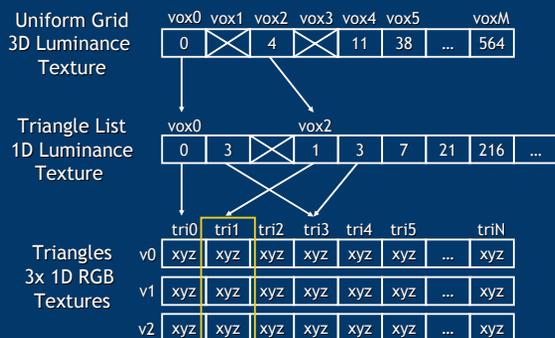
Intersection Code



Ray Tracing on a GPU

- Store scene data in texture memory
 - Dependent texturing is key
- Multipass rendering for flow control
 - Branching would eliminate this need

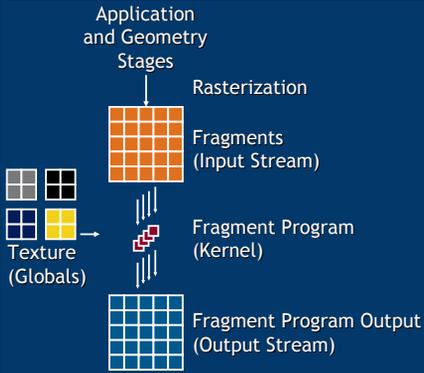
Scene in Texture Memory



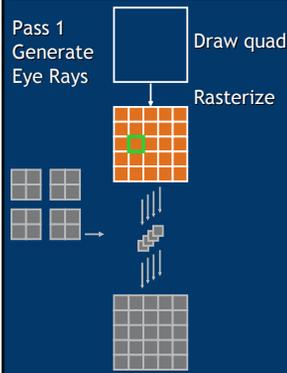
Texture As Memory

- Currently limited in size - 128MB
 - About 3M triangles @ 36 bytes per triangle
- Uniform grid
 - Maps naturally to 3D textures
 - Requires 4 levels of dependent texture lookups
- 1D textures limited in length
 - Emulate larger address space with 2D textures
- Want integer addressing - not floating point
 - Efficient access without interpolation
- Integer arithmetic

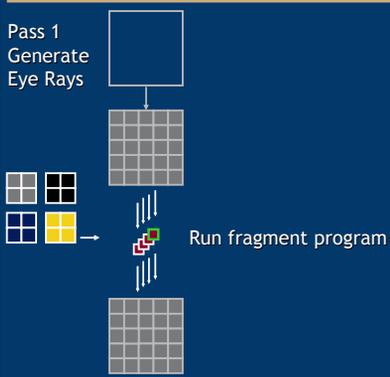
Streaming Flow Control



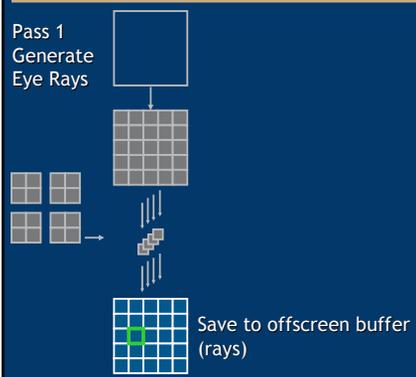
Multiple Rendering Passes



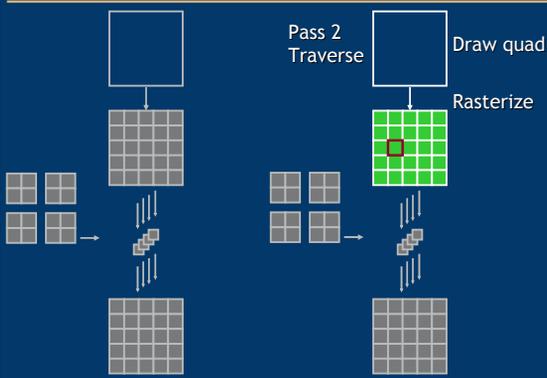
Multiple Rendering Passes



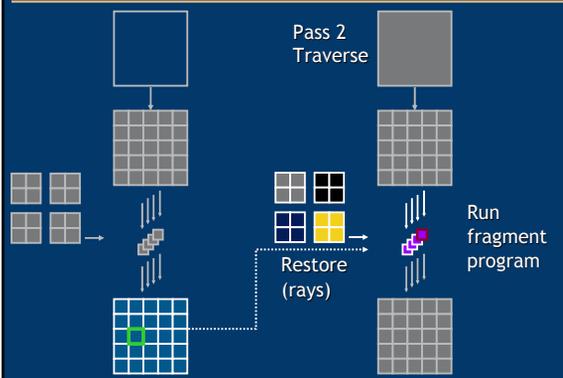
Multiple Rendering Passes



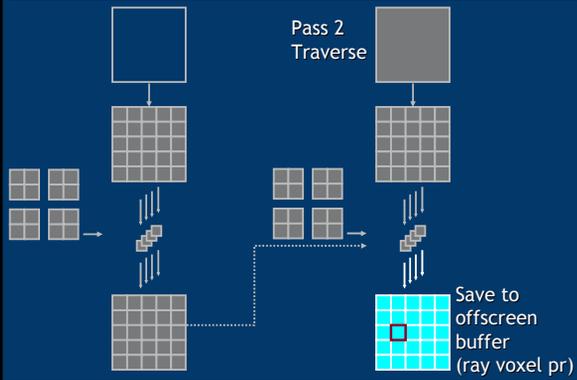
Multiple Rendering Passes



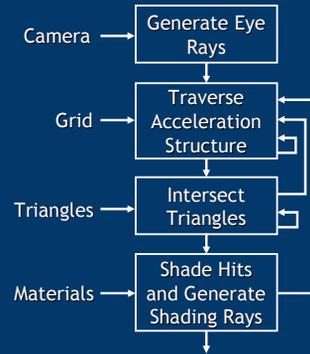
Multiple Rendering Passes



Multiple Rendering Passes



Streaming Ray Tracer



Multipass Optimization

- **Reduce the number of passes**
 - Choose to traverse or intersect based on work to be done for each type of pass
 - Connection Machine ray tracer [Delany 1988]
 - Intersect once 20% of active rays need intersecting
- **Make each pass less expensive**
 - Most passes involve only a few rays
 - Early fragment kill based on fragment mask
 - Saves compute and bandwidth

Scene Statistics



v	14.41	26.11	81.29	130.7	93.93
t	2.52	40.46	34.07	47.90	13.88
s	0.44	1.00	0.96	0.97	0.82
P	2443	1198	1999	2835	1085

v - average number of voxels a ray pierces
 t - average triangles a ray intersects
 s - average number of shading evaluations per ray
 P - number of rendering passes

$$C = R * (C_r + v * C_v + t * C_t + s * C_s) + R * P * C_{mask}$$

Performance Estimates

- **Pentium III 800 MHz CPU implementation**
 - 20M intersections/s [Wald et al. 2001]
- **Simulated performance**
 - 2G instructions/s and 8GB/s bandwidth
 - Instruction limited
 - 56M intersections/s
 - Nearly bandwidth limited
 - 222M intersections/s
- **Streaming ray tracing is compute limited!**

Demo Analysis

- **Prototype Performance (ATI R300)**
 - 500K - 1.4M raycast/s
 - 94M intersections/s
 - Only three weeks of coding effort
- **ATI Radeon 8500 GPU (R200)**
 - 114M intersections/s [Carr et al. 2002]
 - Fixed point operations
 - Only ray-triangle intersection kernel

Summary

- Programmable GPU is a stream processor
- Ray tracing can be a streaming computation
- Complete ray tracer can map onto the GPU
 - Ray tracing generally thought to be incompatible with the traditional graphics pipeline
- Streaming GPU-based ray tracer is competitive with CPU-based ray tracer

Architectural Results

- Fragment mask proposed for efficient multipass
 - Stream buffer eliminates this need
- Stream data should not go through standard texture cache
- Triangles cache well for primary rays, secondary less so
- Branching architecture
 - More cache coherence than the multipass architecture for scene data
 - Reduces memory bandwidth for stream data
 - But has its own costs...

Future Work

- Acceleration Structure
 - Multi-level grids or k-d trees
 - Dynamic acceleration structure
 - Building acceleration structure on GPU
 - Requires scatter (i.e. dependent texture write)
- Photon mapping
 - Ask me again in January...

Final Thoughts

- Ray tracing maps into current GPU architecture
 - Does not require fundamentally different hardware
 - Hybrid algorithms possible
- What else can the GPU do?
 - Given you can do ray tracing, you can do anything
 - Fluid flow, molecular dynamics, etc.
- GPU performance increase will continue to outpace CPU performance increase

Acknowledgements

- Demo
 - James 'RTD' Percy, Pradeep Sen, Eric Chan
- People
 - Matt Papakipos, Kurt Akeley - NVIDIA
 - Bob Drebin, Mark Peercy - ATI
 - Katie Tillman
- Sponsors
 - ATI, MERL, NVIDIA, Sony, Sun
 - DARPA

Questions?