Advancements in V-Ray RT GPU

Vlado Koylazov, CTO & Co-founder
Blagovest Taskov, RT GPU Team Lead
Alexander Soklev, RT GPU R&D
Agenda

- Recent improvements in RT GPU
  - Rounded edges
  - MDL material support
- Next-gen GPU raytracing kernels architecture R&D
  - Multi-kernel vs mega kernel
  - On demand texture loading
- And other stuff
Rounded corners

- Works at render time
- Works for disconnected meshes, displacement etc.
- Works between different objects
- No additional mesh-related data structures needed
Raytraced rounded corners

- Base technology licensed from nVidia...
- ...with two improvements:
  - Randomly jitter the rotation of the sampling pattern for "feeler" rays
  - Trace feeler rays in a cone around the shaded point
    - Removes the need for offsetting the feeler rays along the surface normal
Raytraced rounded corners

- Original method
- Our method
Raytraced rounded corners

Original method

Our method
MDL

- Support coming soon
  - CPU and GPU
- Thanks to nVidia for making the API available for us
- Hopefully available in our products in Fall 2016
Texture Baking

- QMC Sampler
- VRayFur
- VRayPlane
- VRayMultiSubTexture
- Subdivision
- VRayUserColor
- GLSL Textures
- Better Light Cache
- OS X support
- Better Caustics
- Faster updates
- Texture mapped IOR
- Anisotropy

New adaptive image sampling algorithm

Service Pack 2
Service Pack 3

Part of the features in RT GPU for 2015

Lights Decay
Displacement
ProjectionTex
Composite Map
Disc Light
Better OpenCL
Cleaner VRayBlendMtl
PhysicalCamera bitmap aperture

Lights cast shadows option

Cleaner glossy reflections

Particles from VRayProxy

Procedural environment textures

VR Ready
Next-gen GPU raytrace kernels

- This talk – very technical - kernel architectures overview, targeted at developers

- Building up on “Optimizing large scale CUDA applications using input data specific optimizations”
  
  (ACM doi 10.1145/2668904.2668941).

- Papers are energy consuming
What has changed since GTC’15

- PTX recompiling
  - V-Ray 3.3 does not do this anymore. No recompiling during rendering, faster updates
  - No performance loss – control spilling with no-inlined functions (this works as if it is multi-kernel, but calling functions is faster)
  - Still useful – helped us add support for GLSL and MDL
Gathering statistical data

- Important for making our code faster
  - How do we reduce divergence?
- In-house x86-64 CUDA implementation (GTC’15)
  - Flexible, native x86-64 tools support
- Record the state of each ray for each bounce
  - Perfectly accurate divergence data
- Pareto principle
Multi-kernel against divergence

• Why multi-kernel?
  – A lot of papers on the topic
  – Less register pressure, probably smaller ray context
  – Having ray contexts in global memory gives room for additional processing e.g. sorting rays by material ID before shading.
  – It allows on-demand loading of resources (more on this a bit later)
  – Allows us to use the stats gathered to minimize divergence.
  – Allows usage of Shared Memory!
    • We know which data is hot. Put that in shared memory, and use a pointer to global memory for the rest of the raystate (+15%)
    • Sort rays in shared memory!
The results:

- Multi kernel pros:
  - Is much better when rendering interiors and VFX
  - On-demand resource loading allows rendering of scenes that didn’t fit in memory before.

- Mega kernel pros:
  - Is much better for cases such as: Automotive, exteriors, product design
  - Allows ray contexts to be kept in local memory. Yields performance boost of ~40%!
  - Very compiler friendly (Compilers love predictability).
  - No time consuming kernel calls, no need for cudaDeviceSynchronize()
On-demand texture loading

• Build on top of the memory manager we presented at GTC’15
• Can work with Pixel/Texel Streaming
• Before
  – 4.07 GB of memory (needs at least 4GB GPU)
• After
  – <2.8GB of memory
  – Filtered textures
  – Same render time
• Auto detects num channels

Scene kindly provided by Dabarti CGI
Mega-kernel vs. Multi-kernel*

- Mega kernel excels where multi-kernel fails
  - Automotive, exteriors, product design

- Multi kernel excels where mega-kernel fails
  - Interiors, VFX
  - On-demand resource loading

- Making the user choose kernel type is awful
  - The artist should not care what a kernel is at all

So which one should we use?

*it is “Torvalds vs Tanenbaum” all over again (Torvalds won)
Why don’t we have both?
What we propose
Heterogeneous kernel architecture

• We start renders with multi-kernel (6+ kernels)
• Load all the resources on-the-fly. Auto-generating mip-maps for the textures
• Measure how fast the render goes
• Switch to mega-kernel (if necessary) – happens instantly without re-transfers, measure how fast the render goes
  – Choose dynamically if ray sorting is needed
• This process is not noticeable from user point of view as the rendering is not being stopped.
What we propose

Divergence solution for mega-kernel

• Store rays in shared memory
• Keep block size as big as possible
• Sort inside the block only – much faster and easier
• Warp size is 32
• Block is up to 1024
• 32 groups of sorted rays – more than enough
GPU acceleration not only for V-Ray RT

- VDenoise for V-Ray and V-Ray RT
- GPU Accelerated. More than x25 speedup compared to CPU.
- No need of OpenCL devices
- Interactive, non-destructive denoising during render time

More later this year ...
Different flavor of RT (OpenCL)

- V-Ray RT GPU has supported CUDA and OpenCL for a long time
- RT CUDA is faster and has more features compared to RT OpenCL
- We did a major breakthrough with the RT OpenCL that made our OpenCL implementation far more robust and reliable (available in V-Ray 3.30.04 and later)
Guide to GPU

• Tips and answers to a lot of questions regarding rendering on the GPU
• Free download from labs.chaosgroup.com
• Coming soon @CG_LABS
Q&A

Please complete the Presenter Evaluation sent to you by email or through the GTC Mobile App. Your feedback is important!