HIPRT: A Ray Tracing Framework in HIP
High Performance Graphics 2024

Daniel Meister
Paritosh Kulkarni
Aaryaman Vasishta
Takahiro Harada
HIPRT: Ray Tracing in HIP

- Professional rendering features
  - Multi-level instancing
  - Motion blur
  - Intersection filters
  - Custom primitives

- Bounding volume hierarchy (BVH)
  - Scalable BVH construction
  - Novel SBVH builder on GPU

- Cross-platform
  - HIP ≈ “CUDA on AMD/Nvidia HW”
  - Windows and Linux OSs
  - AMD (including MI series) and Nvidia (SW emulation)
    - Scientific computing

- Small codebase
  - ~17k lines of code
  - HIP supports modern C++ standards
  - Open source
Renderers using HIPRT
API Design

- Not limited by standards by third parties
  - We can design our own API focusing on *ease of use*

- Ray tracing programmable pipeline
  - Opaque and difficult to setup and debug
  - *Shader binding table* (SBT) is the most difficult part
    - Whole book chapters and blogs about how to set it up
  - Not suitable for professional rendering
    - Coupling ray tracing and shading
    - Shading is typically very complex

- HIPRT follows a similar philosophy as Embree
  - Minimal host code setup
  - Providing only the ray tracing functionality (a.k.a. ray queries)
    - Shading and data assignment on the application side
  - SBT reduced to a 2D table
    - Custom intersections and intersection filters
Example

Host code

```cpp
// Triangle mesh
hiprtTriangleMeshPrimitive mesh;
mesh.triangleIndices = ...;
mesh.vertices = ...;
...

// Create and build geometry
hiprtGeometry geom;
hiprtCreateGeometry(..., geom);
hiprtBuildGeometry(..., geom);

// Build trace kernel
hiprtBuildTraceKernels(...);
```

Device code

```cpp
__global__ void RayTraceKernel(hiprtGeometry geom, ...) {
    // Generate ray
    hiprtRay ray = generateRay(...);

    // Traversal object
    hiprtGeomTraversalClosest tr(geom, ray, ...);

    // Find hit
    hiprtHit hit = tr.getNextHit();
    ...
}
```
BVH Builders

LBVH
- Fast build
- Spatial median splits via Morton codes
- One bottom-up pass [Apetrei 2014]
  - Building topology
  - Refitting bounding boxes

PLOC
- Balanced build
- Iterative agglomerative clustering
  - One kernel launch per iteration (a.k.a PLOC++)
  - Morton codes to find nearest neighbors
**BVH Builders**

**SBVH**
- High-quality build
  - Slow and high-memory usage
- Object and spatial splits
  - Robust to diagonal and oblong primitives
- GPU implementation using binning
  - Quality very close to SBVH on CPU [Stich et al. 2009]
  - Only un-splitting is not implemented
  - Spatial binning is the bottleneck (global atomics)
  - Iterative top-down build with multiple kernel launches

**Custom BVH**
- Import own BVH using HIPRT API
- Useful for benchmarking or research
Multi-Level Instancing

- Arbitrary number of levels
- Additional stack needed for more than two levels
  - Storing ray and a pointer to acceleration struct. above

- Moana Island on AMD Radeon PRO W7900
  - 3-level hierarchy
  - 156M unique primitives and 31B instantiated primitives
Motion Blur

- Multi-segment motion blur with non-uniform intervals
  - You can explicitly specify time for each key frame
  - For example, key frames with times 0.0, 0.1, and 1.0
    - HIPRT uses 3 key frames
    - OptiX needs to explicitly resample to 11 key frames
- Correct component-wise interpolation even for matrices
  - Internal matrix decomposition

\[
(1 - t) \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} + t \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \neq \begin{bmatrix} \cos(t\pi) & \sin(t\pi) & 0 \\ -\sin(t\pi) & \cos(t\pi) & 0 \\ 0 & 0 & 1 \end{bmatrix}
\]

OptiX (linear matrix interpolation)

HIPRT (component-wise interpolation)

A singularity for \( t = 0.5 \)
Ray Traversal

- Stack-based algorithm
  - Traversal stack as a template argument
  - Best performance with the *global stack*
    - Rolling stack in shared memory (top-most entries)
    - Global memory as backup (bottom-most entries)

- Intersection filters
  - A custom functions filtering found intersections
    - Inspired by Embree
  - Useful for alpha masking or filtering self-intersections
  - In the programmable pipeline you have no other choice than to use any-hit shader

A cutout filter alpha masking based on texture coordinates
Evaluation Setup

• Wavefront path tracer
  • Isolating ray tracing and shading
    • Various tracer implementations
  • Scene graph
    • Pre-transformed geometry (one large instance)
    • Original two-level partitioning

• Ray tracing backends
  • HIPRT
    • Fast, balanced, and high-quality builds
    • Embree BVH as imported BVH
      • High-quality build with spatial splits built on CPU
  • Vulkan
    • Fast build and fast trace (HQ) options

• HW & SW
  • AMD Radeon PRO W7900 (48GB)
  • ROCm 5.7 & Vulkan 1.3
Test Scenes

- Trains: 836k tris
- Bistro Interior: 1207k tris
- Hangar Ship: 1235k tris
- Opera House: 2512k tris
- Bistro Exterior: 2829k tris
- Museum: 3650k tris
- Sci-fi: 4809k tris
- Zero Day: 5165k tris
- Toy Shop: 5212k tris
- Yokohama: 8217k tris
Trace Times – Two Levels

• Averaged normalized trace times per wave
  • Normalized by PLOC
  • Averaged over all scenes

• Vulkan faster for primary rays

• HIPRT faster for shadow and secondary rays

• SBVH is faster than Embree
Build Times and SAH Cost

- Averaged normalized build times (pre-transformed)
  - Normalized by PLOC
  - Averaged over all scenes

- LBVH provides the fastest build overall

- PLOC is faster than both Vulkan options

- SBVH is slow but provides lowest SAH cost
Time-to-Image = Build Time + Trace time

Yokohama (Pre-transformed)
- Build time corresponds to the offset at zero
- SBVH outweighs the higher build overhead at around 64 samples
Conclusion

HIPRT is an open-source ray tracing framework tailored for AMD GPUs

- Performance comparable with Vulkan yet API is a way more user-friendly
  - SBVH provides excellent performance but the construction is slow
- Professional rendering
  - Motion blur, multi-level instancing, intersection filters
- Pointing out some of the drawbacks of existing APIs
  - Shader binding table or motion blur

Future Work

- H-PLOC
- Curve primitive
- Optimization of advanced features
Thank you for your attention!

• The project webpage
  • https://gpuopen.com/hiprt/

• The source codes
  • https://github.com/GPUOpen-LibrariesAndSDKs/HIPRT

• The PBRT-v4 port
  • https://github.com/GPUOpen-Effects/pbrt-v4
Internal Format

Triangle pairing (preprocess)
- Pairing triangles in the same warp
  - A single kernel launch
  - Reduces the input for further passes about 30%

Conversion BVH2 to BVH4 (postprocess)
- Iterative top-down pass
  - One kernel launch per level
Instance Bounding Boxes

- We need bounding boxes of the instantiated bottom-level geometries
- Transforming the root bounding box is too conservative
- Transforming geometric primitives themselves is too costly
- Transforming grandchildren or children is a good compromise
Batch Construction

- Multiple HIP streams allow to build multiple BVH concurrently
- HIP kernel launch and allocation is expensive
- Batch construction allows to build multiple small BVH in a single kernel launch
  - The size of a BVH is limited by the block size
  - All data in shared memory (no additional global buffers)

One hair strand = One BLAS
4M BLAS’s
Trace Times – Pre-transformed

- Averaged normalized trace times per wave
  - Normalized by PLOC
  - Averaged over all scenes

- LBVH deviated by an outlier
  - 32-bit Morton codes not sufficient in Opera House

- HIPRT faster than Vulkan

- SBVH is comparable with Embree
Trace Speed – Secondary Bounces

Bistro Interior (Pre-transformed)