HIP RT: A Ray Tracing Library in HIP

Takahiro Harada
2022/7
Self Introduction

- Takahiro Harada
- Advanced Rendering Research Group at AMD
- R&D

https://gpuopen.com/research/
What Do You Think about Ray Tracing APIs?

• DirectX® Ray Tracing
• Vulkan® Ray Tracing
• Metal Performance Shaders
• NVIDIA OptiX
Another Ray Tracing Library?

- Necessity
  - AMD RDNA™ 2 GPUs (Navi2x) have hardware ray tracing unit “Ray Accelerator” which cannot be accessible from developers
  - Needed to add ray tracing API for HIP (Heterogeneous-Computing Interface for Portability*)

- How should we design?
  - Take existing API?
  - Design something new?

- Looked at different APIs, decided to design our own
  - Slightly different from others in a few ways

Design

• Designed to be minimum
• No need to learn different shaders (kernels)
• Easier to add to existing applications
  • Adding RT to existing app. shouldn’t be that difficult
History

- Released v.1 in 2022/4
- v.1.1 is soon
Overview of HIP RT

- Ray tracing using BVH

- Use hardware ray tracing on AMD GPUs (Navi 2x)
  - Runs on Vega or Navi1x, Navi2x GPUs
  - No support on Ellesmere and older (HIP)

- Make it run as many platforms as possible
  - HIP and CUDA implementation (AMD and NV)
    - Built on top of Orochi (https://gpuopen.com/orochi/)
Orochi

• Developers need to maintain HIP and CUDA host code
  • They are mostly the same :(  
  • It doesn’t sound right

• Write once using Orochi APIs, then your application
  • Runs on AMD GPUs using HIP
  • Runs on NVIDIA GPUs using CUDA
  • No static linking thus it doesn’t crash even if there is no GPU
  • Implements driver APIs

#include <hip/hip_runtime.h>
hipInit( 0 );
hipDevice device;
hipDeviceGet( &device, 0 );
hipCtx ctx;
hipCtxCreate( &ctx, 0, device );

#include <Orochi/Orochi.h>
oroInitialize( ORO_API_HIP, 0 );
oroInit( 0 );
oroDevice device;
orDeviceGet( &device, 0 );
orCtx ctx;
orCtxCreate( &ctx, 0, device );

https://gpuopen.com/orochi/
Overview of HIP RT

• Ray tracing using BVH

• Use hardware ray tracing on AMD GPUs (Navi 21)
  • Runs on Vega or Navi10, Navi20 GPUs
  • No support on Ellesmere and older (HIP)

• Make it run as many platforms as possible
  • HIP and CUDA implementation (AMD and NV)
    • Built on top of Orochi (https://gpuopen.com/orochi/)
  • Hardware ray tracing works only on HIP
  • Windows and Linux OSes

• Intersection against triangles
  • Can extend to any primitives by writing a custom intersection functions
Overview of HIP RT

• What are the essential things we need to define?
  • Camera?
  • Acceleration structure?
Overview of HIP RT

- User need to understand these objects
  - hiprtGeometry
    - This is an instance in other APIs
    - Collection of primitives (triangles)
Overview of HIP RT

- User need to understand these objects
  - hiprtGeometry
    - This is an instance in other APIs
    - Collection of primitives (triangles)
  - hiprtScene
    - Collection of hiprtGeometries to make a scene
Overview of HIP RT

- Hit types
  - `hiprtTraversalsTerminateAtAnyHit = 1`,
  - `hiprtTraversalsTerminateAtClosestHit = 2`,

- Multiple BVH types depending on your needs
  - `hiprtBuildFlagBitPreferFastBuild = 1`,
  - `hiprtBuildFlagBitPreferHighQualityBuild = 2`,
  - `hiprtBuildFlagBitPreferBalancedBuild = 3`,
  - You can even build a BVH by yourself, pass it to HIP RT
Using HIPRT

- Get the latest driver package from AMD website
- Agree the license, download the SDK package from project page
  - [https://gpuopen.com/hiprt/](https://gpuopen.com/hiprt/)
- Get the latest tutorials at (Optional)
  - [https://github.com/gpuopen-LibrariesAndSDKs/hiprtsdk](https://github.com/gpuopen-LibrariesAndSDKs/hiprtsdk)
- Link your HIP/CUDA program with HIPRT
Using HIPRT

- Get the latest driver package from AMD website

- Agree the license, download the SDK package from project page
  - https://gpuopen.com/hiprt/

- Get the latest tutorials at (Optional)
  - https://github.com/gpuopen-LibrariesAndSDKs/hiprtsdk

- Link your HIP/CUDA program with HIPRT

https://github.com/GPUOpen-LibrariesAndSDKs/HIPRTSDK/tree/main/tutorials
[Tutorial] Geometry Intersection

• 4 steps
  • Context creation
  • Geometry (or Scene) construction
  • Kernel compilation
  • Kernel execution
[Tutorial] Geometry Intersection

- 4 steps
  - Context creation
  - Geometry (or Scene) construction
  - Kernel compilation
  - Kernel execution

```c
hiprtContext ctxt;
hiprtCreateContext( HIPRT_API_VERSION, m_ctxtInput, &ctxt );
```
[Tutorial] Geometry Intersection

- 4 steps
  - Context creation
  - Geometry (or Scene) construction
  - Kernel compilation
  - Kernel execution

```c
hiprtGeometryBuildInput geomInput;
geomInput.type = hiprtPrimitiveTypeTriangleMesh;
geomInput.triangleMesh.primitive = &mesh;

size_t geomTempSize;
hiprtDevicePtr geomTemp;
hiprtBuildOptions options;
options.buildFlags = hiprtBuildFlagBitPreferFastBuild;
hiprtGetGeometryBuildTemporaryBufferSize( ctxt, &geomInput, &options, &geomTempSize );
dMalloc( (u8**)&geomTemp, geomTempSize );

hiprtGeometry geom;
hiprtCreateGeometry( ctxt, &geomInput, &options, &geom );
hiprtBuildGeometry( ctxt, hiprtBuildOperationBuild, &geomInput, &options, geomTemp, 0, geom );
```
[Tutorial] Geometry Intersection

- 4 steps
  - Context generation
  - Geometry (or Scene) construction
  - Kernel compilation
  - Kernel execution

```c
oroFunction func;
buildTraceKernel( ctxt, "../01_geom_intersection/TestKernel.h", "MeshIntersectionKernel", func );

hiprtBuildTraceProgram(hiprtContext context, const char* functionName, const char* src, ...);
```
[Tutorial] Geometry Intersection

• 4 steps
  • Context generation
  • Geometry (or Scene) construction
  • Kernel compilation
  • Kernel execution

oroModuleLaunchKernel( func, nb.x, nb.y, 1, tpb.x, tpb.y, 1, sharedMemBytes, 0, (void**)args, 0 );
**[Tutorial] Geometry Intersection**

```c
extern "C" __global__ void MeshIntersectionKernel(unsigned char* gDst, int2 cRes) {
    const int gIdx = blockIdx.x * blockDim.x + threadIdx.x;
    const int gIdy = blockIdx.y * blockDim.y + threadIdx.y;

    hiprtRay ray;
    float3 o = { gIdx / (float)cRes.x, gIdy / (float)cRes.y, -1.0f};
    float3 d = { 0.0f, 0.0f, 1.0f};
    ray.origin = o;
    ray.direction = d;
    ray.maxT = 1000.f;

    int dstIdx = gIdx + gIdy * cRes.x;
    gDst[ dstIdx * 4 + 0 ] = hasHit ? ((float)gIdx / cRes.x) * 255 : 0;
    gDst[ dstIdx * 4 + 1 ] = hasHit ? ((float)gIdy / cRes.y) * 255 : 0;
    gDst[ dstIdx * 4 + 2 ] = 0;
    gDst[ dstIdx * 4 + 3 ] = 255;
}
```
extern "C" __global__ void MeshIntersectionKernel(hiprtGeometry geom, unsigned char* gDst, int2 cRes) {
    const int gIdx = blockIdx.x * blockDim.x + threadIdx.x;
    const int gIdy = blockIdx.y * blockDim.y + threadIdx.y;

    hiprtRay ray;
    float3 o = { gIdx / (float)cRes.x, gIdy / (float)cRes.y, -1.0f};
    float3 d = { 0.0f, 0.0f, 1.0f};
    ray.origin = o;
    ray.direction = d;
    ray.maxT = 1000.f;

    hiprtGeomTraversalClosest tr(geom, ray);
    hiprtHit hit = tr.getNextHit();
    bool hasHit = hit.primID != hiprtInvalidValue;

    int dstIdx = gIdx + gIdy * cRes.x;
    gDst[ dstIdx * 4 + 0 ] = hasHit ? ((float)gIdx / cRes.x) * 255 : 0;
    gDst[ dstIdx * 4 + 1 ] = hasHit ? ((float)gIdy / cRes.y) * 255 : 0;
    gDst[ dstIdx * 4 + 2 ] = 0;
    gDst[ dstIdx * 4 + 3 ] = 255;
}

extern "C" __global__ void MeshIntersectionKernel(hiprtGeometry geom, unsigned char* gDst, int2 cRes) {
    const int gIdx = blockIdx.x * blockDim.x + threadIdx.x;
    const int gIdy = blockIdx.y * blockDim.y + threadIdx.y;

    hiprtRay ray;
    float3 o = { gIdx / (float)cRes.x, gIdy / (float)cRes.y, -1.0f};
    float3 d = { 0.0f, 0.0f, 1.0f};
    ray.origin = o;
    ray.direction = d;
    ray.maxT = 1000.0f;

    hiprtGeomTraversalClosest tr(geom, ray);
    hiprtHit hit = tr.getNextHit();
    bool hasHit = hit.primID != hiprtInvalidValue;

    int dstIdx = gIdx + gIdy * cRes.x;
    gDst[ dstIdx * 4 + 0 ] = hasHit ? ((float)gIdx / cRes.x) * 255 : 0;
    gDst[ dstIdx * 4 + 1 ] = hasHit ? ((float)gIdy / cRes.y) * 255 : 0;
    gDst[ dstIdx * 4 + 2 ] = 0;
    gDst[ dstIdx * 4 + 3 ] = 255;
}

How much code do you need to write to render a single triangle in another API?
## HIPRT Tutorials

<table>
<thead>
<tr>
<th>List of tutorials</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>01_geom_intersection</td>
<td>Intersection using hiprtGeometry.</td>
</tr>
<tr>
<td>02_scene_intersection</td>
<td>Intersection using hiprtScene.</td>
</tr>
<tr>
<td>03_custom_intersection</td>
<td>Using a custom intersection function.</td>
</tr>
<tr>
<td>04_shared_stack</td>
<td>Using shared stack for traversal which is essential to get a good performance.</td>
</tr>
<tr>
<td>05_custom_bvh</td>
<td>Loading a BVH a user provides.</td>
</tr>
<tr>
<td>06_obj_AO</td>
<td>Loading obj file and rendering AO.</td>
</tr>
<tr>
<td>07_motion_blur</td>
<td>Rendering objects under motion.</td>
</tr>
</tbody>
</table>
Closing

- Next Release
  - At SIGGRAPH2022
  - Bounding box program
  - More optimization

- Thanks for HIPRT development team in ARR, David McAllistor, Bruno Stefanizzi

- Project page:
  - [https://gpuopen.com/hiprt/](https://gpuopen.com/hiprt/)

- Blog:
  - [https://gpuopen.com/learn/introducing-hiprt/](https://gpuopen.com/learn/introducing-hiprt/)

- Github repository:
  - [https://github.com/gpuopen-LibrariesAndSDKs/hiprtsdk](https://github.com/gpuopen-LibrariesAndSDKs/hiprtsdk)

- Documentation
  - [https://radeon-pro.github.io/RadeonProRenderDocs/en/hiprt/about.html](https://radeon-pro.github.io/RadeonProRenderDocs/en/hiprt/about.html)
Disclaimer

The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions, and typographical errors. The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. Any computer system has risks of security vulnerabilities that cannot be completely prevented or mitigated. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes.

THIS INFORMATION IS PROVIDED ‘AS IS.’ AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS, OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION. AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY RELIANCE, DIRECT, INDIRECT, SPECIAL, OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF AMD IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

2022, Advanced Micro Devices, Inc. All rights reserved.

AMD, the AMD Arrow logo, and combinations thereof are trademarks of Advanced Micro Devices, Inc. Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.

Vulkan and the Vulkan logo are registered trademarks of the Khronos Group Inc.

DirectX is either registered trademark or trademark of Microsoft Corporation in the US and/or other countries.