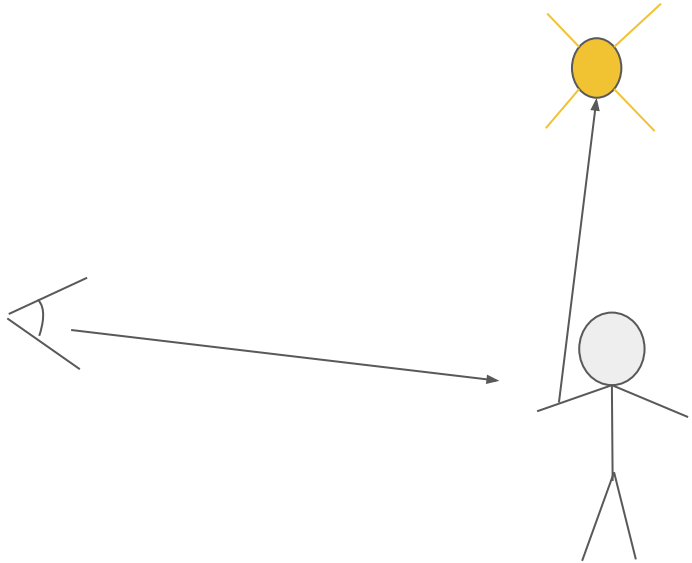
World’s slowest raytracer

Bas Nieuwenhuizen, XDC 2022

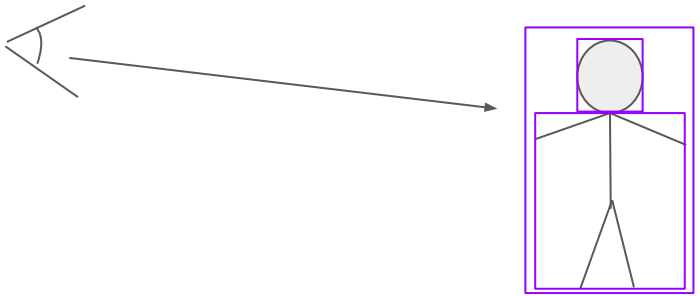
Ray Tracing, what is it?



How does one trace a ray?

- Checking all triangles is slow

- Bounding Volume Hierarchies
  - Of course there is a tree structure
  - Every node has a bounding box



Tracing Rays

Hardware RT acceleration on RDNA 2

```
uvec4 image_bvh_intersect_ray(uvec4    descriptor,
                              uint64_t node_pointer,
                              float    extent,
                              vec3     origin,
                              vec3     direction,
                              vec3     inv_direction);
```

Returns:

- Internal node: id of intersecting children or -1
- Triangle node: distance and barycentric coordinates of intersection

BVH nodes on RDNA 2

Triangle node (64 bytes)

```
vec3 vertices[5];
uint flags;
```

- Allows up to 4 triangles using pointer tags

Internal fp32 node (128 bytes)

```
uint child_id[4];
struct {
   vec3 min;
   vec3 max;
} child_bound[4];
```

Internal fp16 node (64 bytes)

```
Above with fp16 bounds
```

How to trace a ray on RDNA2?

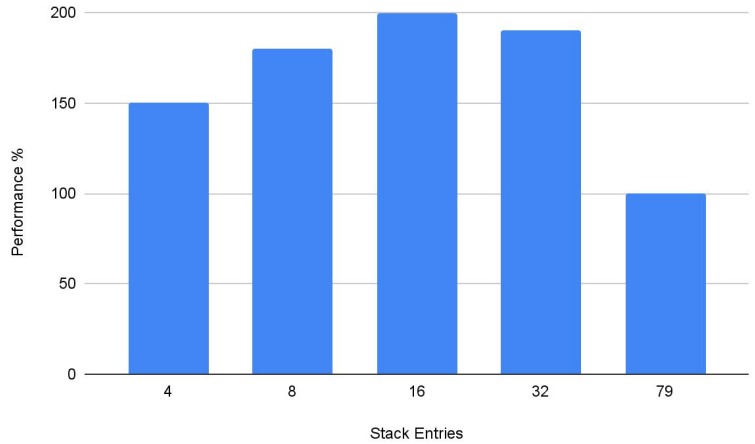
- Using shader code!
- Depth First Search

Occupancy woes

- You typically need a stack for DFS
- Backtracking means more box node intersections

Options for stack:

- VRAM
- LDS (shared memory)



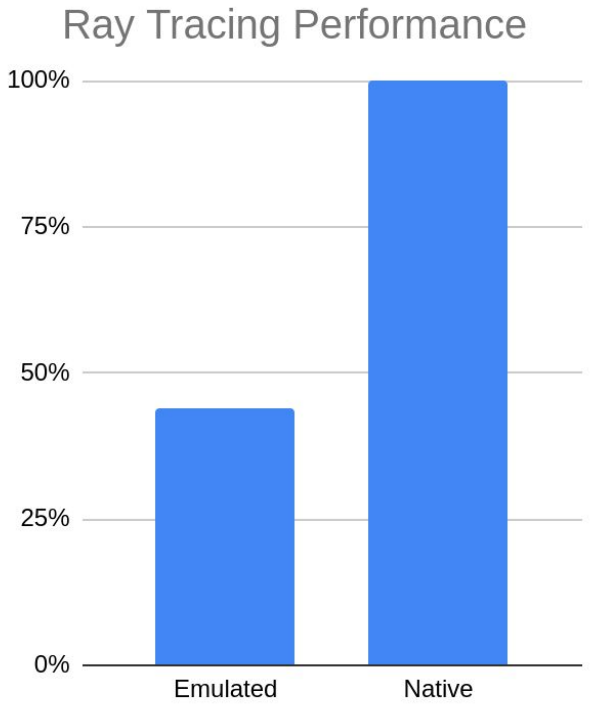
Short stack + backtracking

- 16 entry stack in LDS
- Backtrack if stack is empty
  - Less than 1% of iterations has a lane that is backtracking.

Bonus: No depth limits on BVH

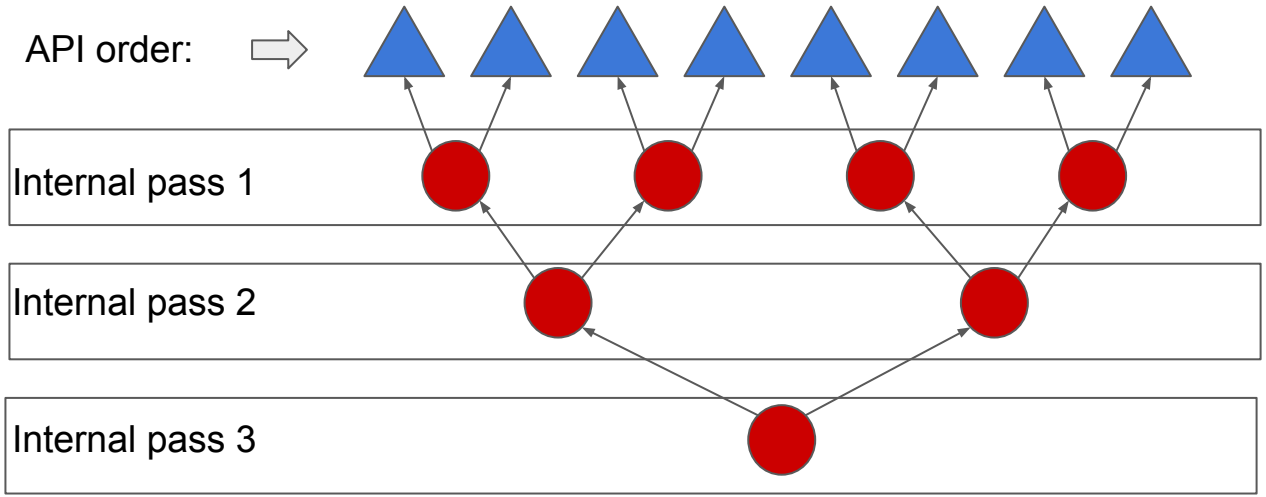
Support on older GPUs

- Implement the single instruction in software
- Works for all supported GPUs



Building a BVH

Naive BVH Construction



CTS Fails

- This is very dependent on triangle order
- CTS had a test that hits worst-case

Idea: Sort triangles first

- E.g. on center of bounding box.

Morton Codes

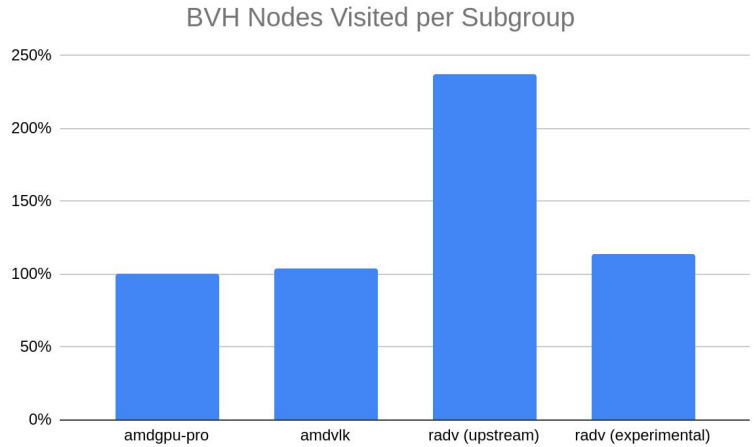
- X: $x_6x_5x_4x_3x_2x_1x_0$
- Y: $y_6y_5y_4y_3y_2y_1y_0$
- Z: $z_6z_5z_4z_3z_2z_1z_0$

->

$z_6y_6x_6z_5y_5x_5z_4y_4x_4z_3y_3x_3z_2y_2x_2z_1y_1x_1z_0y_0x_0$

Going further

Still significantly worse than what is possible.



Further experiments:

- Top-down build using SAH with binning
- Parallel Locally-Ordered Clustering

Giving similar results.

Other BVH builders

- Intel GRL code
  - Heavily dependent on cmdbuffer gymnastics that we can’t do ..
- Gpurt (RT implementation of AMDVLK)
  - Written in HLSL
  - Glslang support for HLSL is incomplete/broken
  - DXC was considered not an appropriate dependency

Walked into the NIH trap pretty easily …

RT Pipelines

How to trace rays: the easy way

Ray Queries

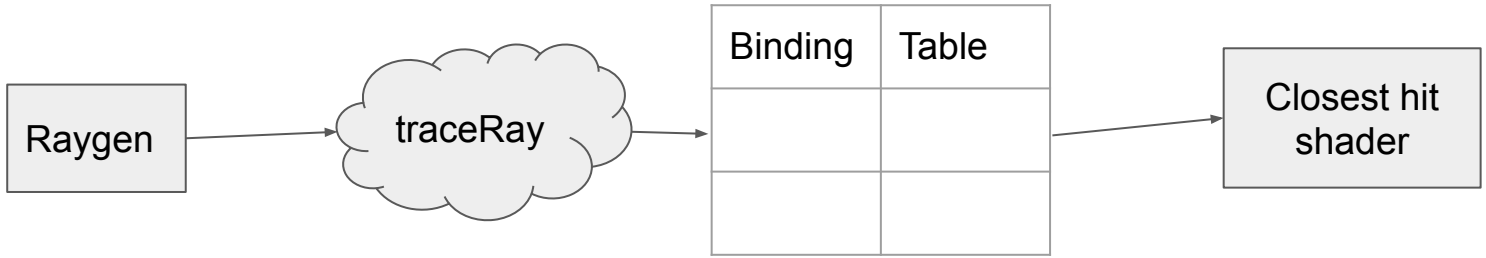
```
rayQueryEXT rayQuery;
rayQueryInitializeEXT(rayQuery, topLevelAS, gl_RayFlagsTerminateOnFirstHitEXT,
                      mask, origin, tmin, direction, tmax);

while (rayQueryProceedEXT(rayQuery)) {
   // process current intersection for e.g. opaqueness
}


if (rayQueryGetIntersectionTypeEXT(rayQuery, true) ==
    gl_RayQueryCommittedIntersectionTriangleEXT) {
   // hit a triangle
}
```

How to trace rays: the complicated way

- Ray Tracing pipelines
  - Callback based
  - Every callback is a new shader stage
  - Many shaders of the same stage possible with a binding table.
  - Callbacks can trace more rays (recursion)



Implementation

- Lower the shader to continuation passing style:
  - Shader ends after traceRay
  - Have a new resume shader for after the traceRay finishes
- Give each shader a unique id

```
void main() {
   // pre-trace stuff
   traceRay(...);
   // post-trace stuff
}
```

```
void main() {
   // pre-trace stuff
   // push all the variables to scratch stack
   // push resume shader id to scratch stack
   next_shader_id = TRACE_RAYS_SHADER_ID;
}

void main() {
   // pop everything from stack
   // post-trace stuff
   next_shader_id = /* value from top of stack */
}
```

Implementation 2

- Tie all this together with a big loop and switch

```
void main() {
   uint next_shader_id= raygen_binding_table[0];
   while (next_shader_id != 0) {
      switch(next_shader_id) {
       …
      }
   }
}
```

Not Meeting Expectations

- Pipeline libraries come with expectations
- Recompiling everything every time does not meet those expectations

```
uniform_next_shader_addr =
get_first_active_lane(next_shader_addr);
// indirect branch to uniform_next_shader_addr
```

- Allows for separate compilation
- But needs ACO changes

Current Status

Using Raytracing in RADV

Ray Queries: Enabled by default

Ray Tracing Pipelines: Use `RADV_PERFTEST=rt`

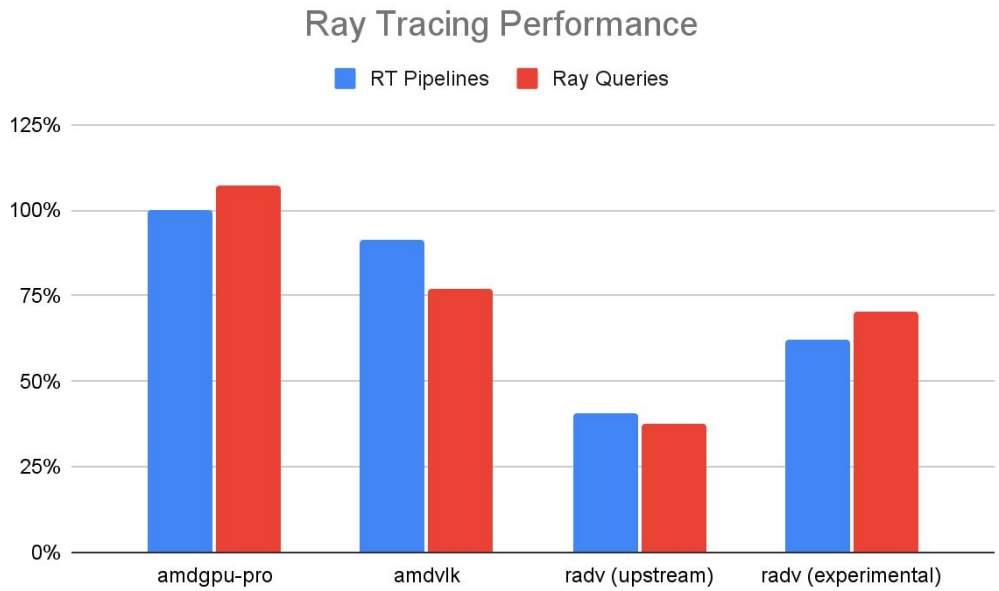
- Main blocker: separate compilation of shaders

Games

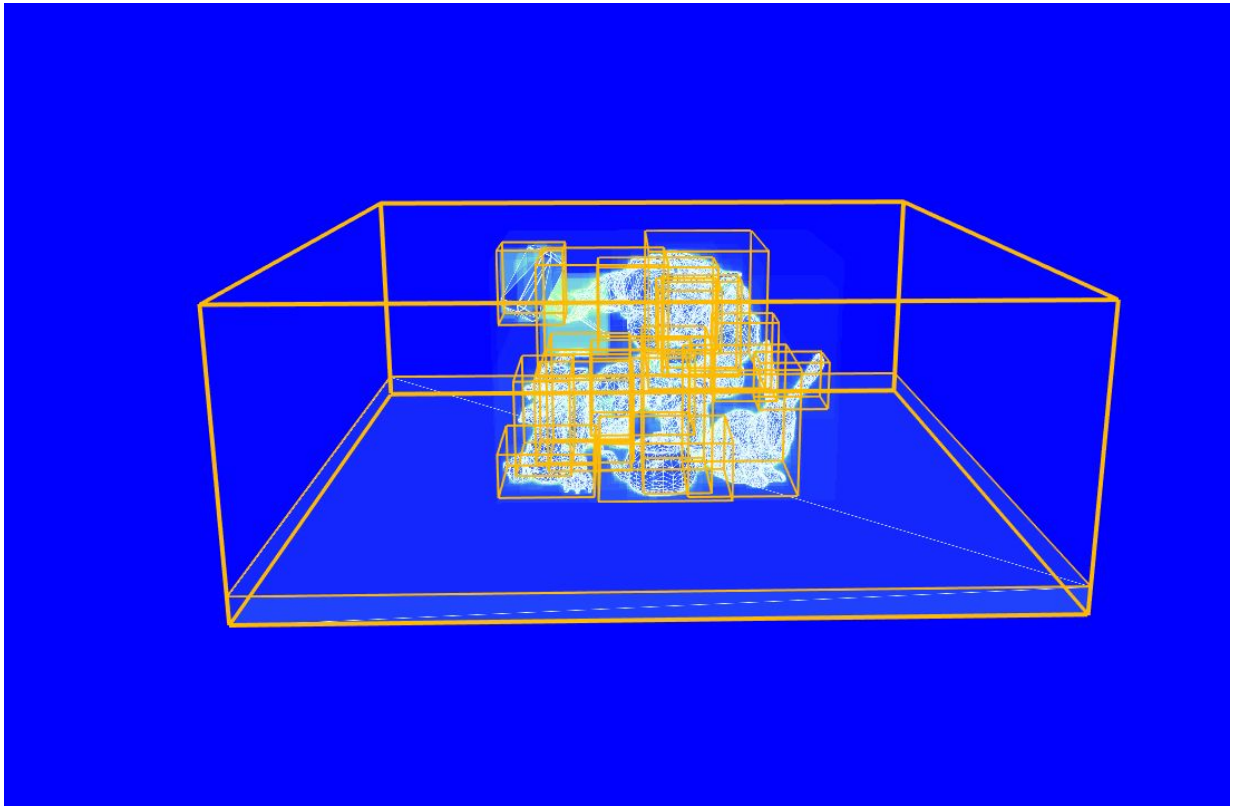
Working RT Effects:

- Quake 2 RTX
- Control
- Deathloop
- Resident Evil Village
- Metro Exodus: Extended Edition

Performance



Radeon Rays Analyzer



New Contributors

- Konstantin Seurer
- Friedrich Vock

Made some great contributions for raytracing

Next Steps

Next Steps: Features

- Separate shader compilation
  - Enable ray tracing by default
- Indirect BVH builds
  - Needed for DXR 1.1

Next steps: Performance

- Land better BVH building algorithms
- Use multiple triangles per node & fp16 box nodes
- Microoptimize the hell out of the traversal loop
- Optimizations for the main loop: tail calls etc.