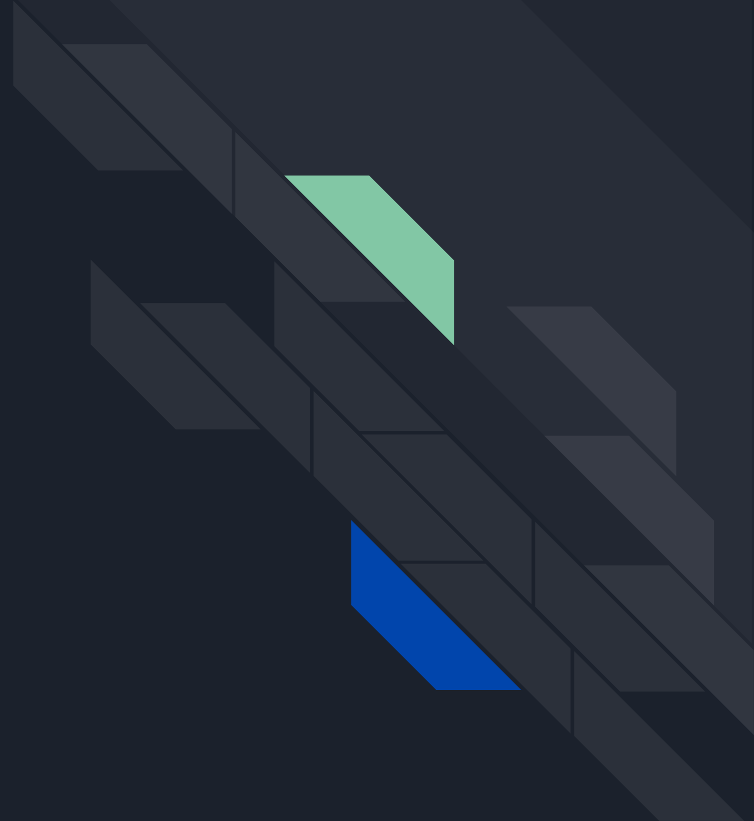
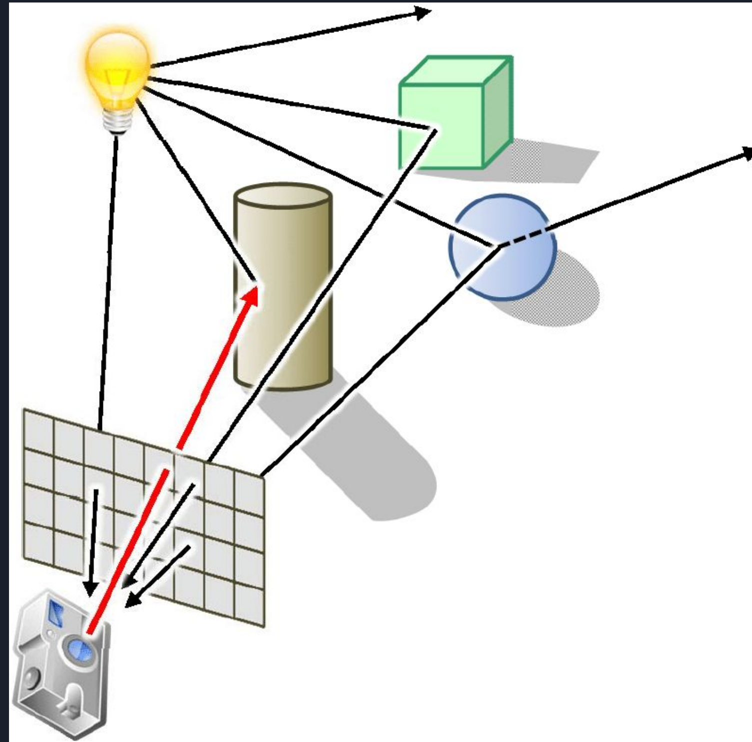
Connor Abbott

What is Ray Tracing?



What is Ray Tracing?



from https://www.researchgate.net/figure/Illustration-of-basic-ray-tracing_fig1_317224001

Naive Ray Tracing

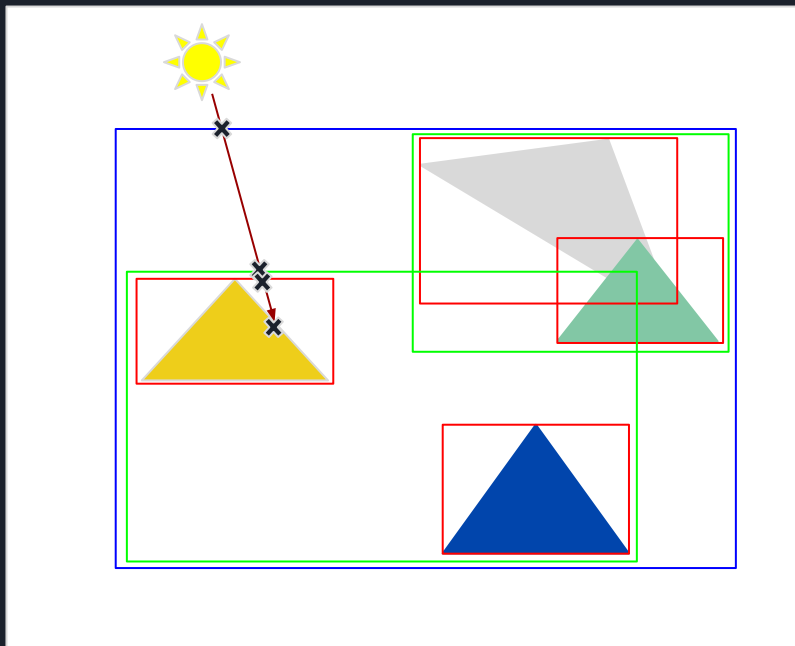
```
for each pixel (i, j):
    ray = ray_from(camera_origin, camera_dir, camera_up, i, j)
    t_min = infinity # distance to the closest intersecting object
    closest_object = null
    # find the closest intersection
    for each object: # triangle, sphere, etc.
        if object.intersects(ray, &t) and t < t_min:
            t_min = t
            closest_object = object

    # calculate the light emitted, casts
    # secondary rays again loop over all objects!
    intersection_point = ray.origin + t_min * ray.dir
    image[i,j] = closest_object.color(intersection_point, ray.dir)
```

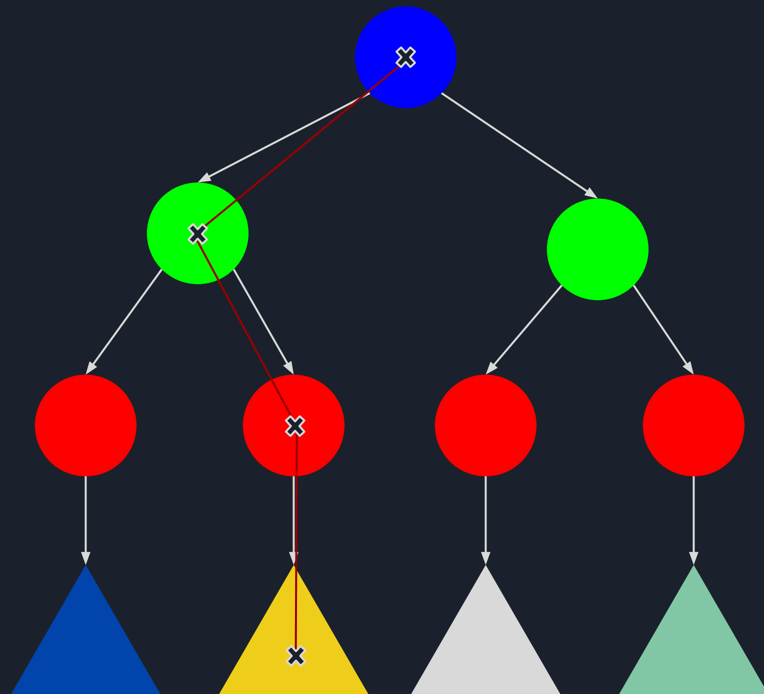
- Problem #1: Time complexity
  - O(objects * pixels * bounces) is impractically slow!
- An *acceleration structure* quickly skips irrelevant parts of the scene

```
as = build_acceleration_structure(objects)
for each pixel (i, j):
     ray = ray_from(camera_origin, camera_dir, camera_up, i, j)
     t_min, closest_object = as.intersect(ray)

     # calculate the light emitted, casts secondary rays
     # secondary rays also use acceleration structure
     intersection_point = ray.origin + t_min * ray.dir
     image[i,j] = closest_object.color(as, intersection_point,
ray.dir)
```

- Most common acceleration structure: *BVH (Bounding Volume Hierarchy) Trees*

BVH Trees

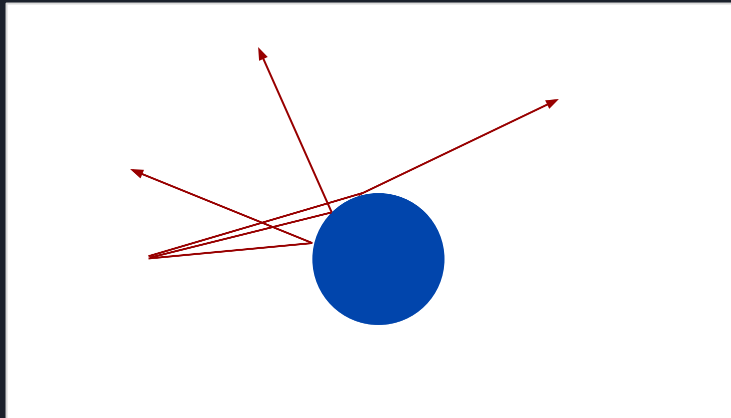




- Scenes contain both static + dynamic geometry
- Rebuilding the whole acceleration structure each frame is expensive
- Split into *top-level* (TLAS) and *bottom-level* (BLAS) acceleration structures
  - Reuse most BLASs across frames
- TLAS contains *instances* of objects
  - Each instance has a BLAS pointer + transformation matrix
- BLAS contains *primitives* (triangles, or programmable objects defined by shader code with a given bounding box)
- Each ray intersection walks the TLAS and then BLAS

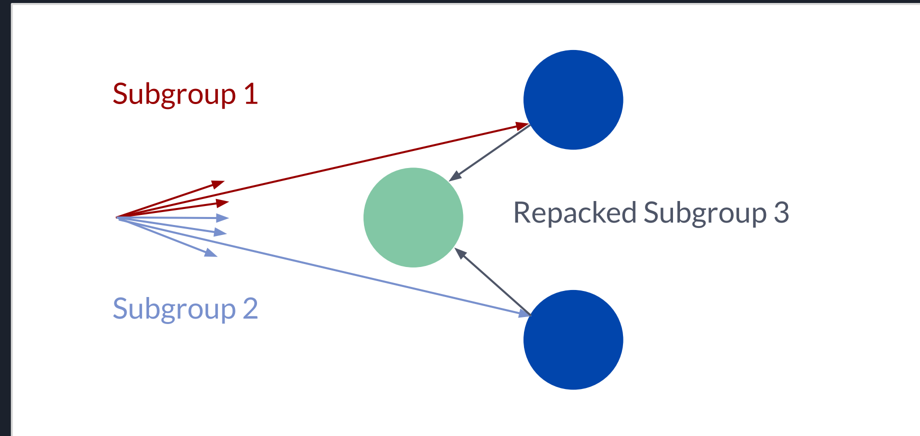
Invocation Repacking

- Problem #2: *Ray coherence*
- "Coherent" rays bounce in different directions and become non-coherent
- Worse for complex scenes with many bounces



Invocation Repacking

- Separate shader for each material being hit
- Batch up and reorder execution of these shaders
- Shaders must be split into *main shader* and *continuation shaders*



Invocation Repacking

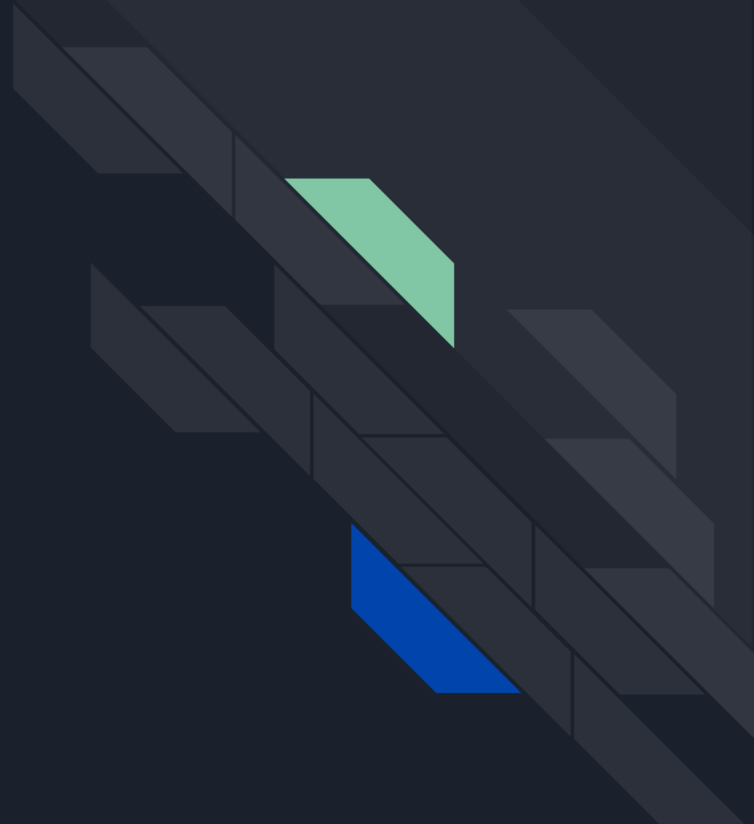
```
color1 = trace_ray(...)
color2 = trace_ray(...)
return (color1 + color2) / 2
```

```
main:
trace_ray(..., continuation1 )

continuation1:
color1 = trace_ray_result
save(color1) # write to stack
trace_ray(..., continuation2 )

continuation2:
color1 = restore() # read stack
color2 = trace_ray_result
return (color1 + color2) / 2
```

`nir_lower_shader_calls` in Mesa

- `VK_KHR_ray_query`
  - Intersect a ray with an acceleration structure in any shader
  - Can be used with compute shaders or as part of the classic pipeline for secondary rays (better shadows, reflections etc.)
- `VK_KHR_ray_tracing_pipeline`
  - Separate per-material shaders (instead of giant compute ubershader)
  - Allows implementations to do invocation repacking
- `VK_KHR_acceleration_structure`
  - Build an opaque acceleration structure on the CPU or GPU
  - Used by both extensions

Ray Tracing in Adreno



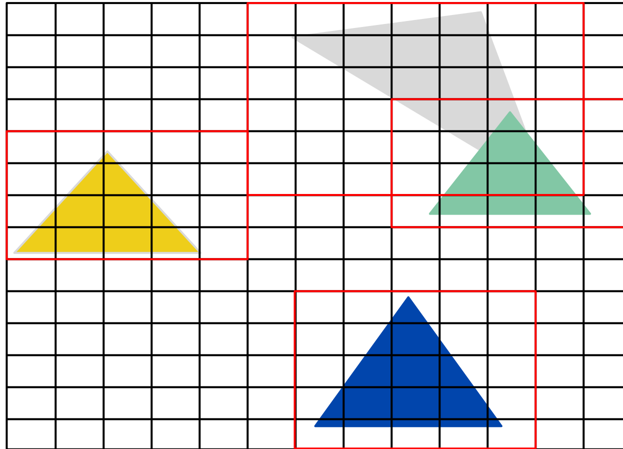
- a740+, x1e laptops: Ray Tracing Unit (RTU)
  - One shader instruction: `ray_intersection`
  - Intersects with one BVH node at a time
  - Shader keeps track of stack of nodes
  - Meant for VK_KHR_`ray_query`
- a750+: Application QRisc Engine (AQE)
  - Coprocessor for dynamic work generation
  - Implements RT pipelines and invocation repacking
  - Experimental, not exposed by default with the blob driver
  - Not implemented yet in turnip

- Each node in the tree is 64 bytes
- Two main types of nodes:
  - Internal node: Compressed *axis-aligned bounding box* (AABB) for up to 8 children
  - Leaf nodes

Internal Nodes

- Grid scale (aka "shared exponent") for x, y, z axes
- Origin point for the grid
- 8-bit grid offset (aka "mantissa") for 6 AABB bounds for each child

Internal Nodes



grid origin: (..., ..., ...)
grid scale (..., ..., ...)
4 children:
- For each child: $(x_{min}, y_{min}, z_{min})$, $(x_{max}, y_{max}, z_{max})$
Children base offset

...

child 0

child 1

child 2

child 3

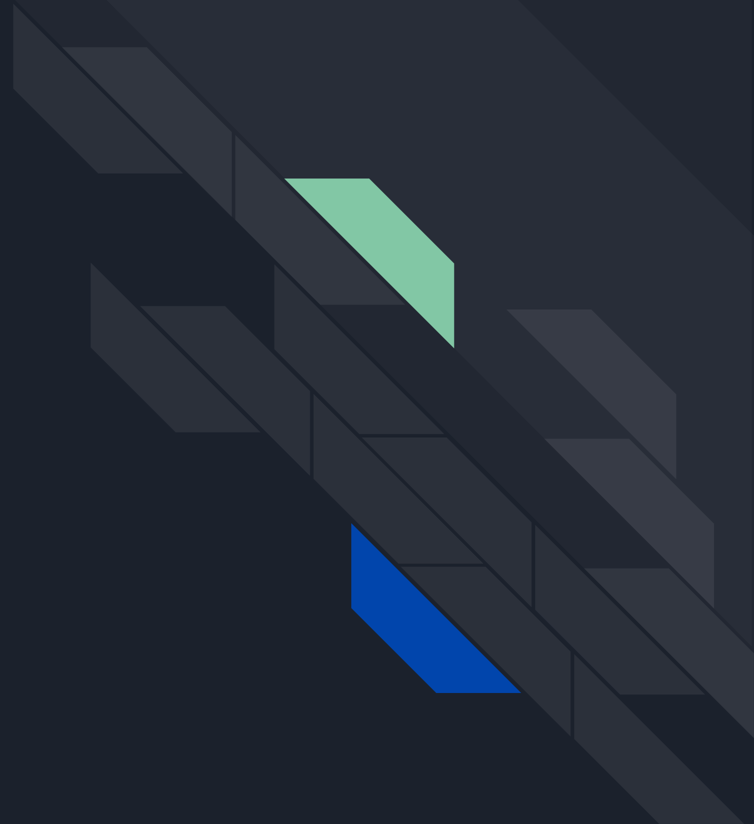
...

- Single base offset for children: children must be contiguous!
- `ray_intersection` returns a bitmask of hit children, sorted by distance
    - 3 bits per child times 8 children, plus count of hit children
    - Plus the base children offset

Leaf nodes

- Simple uncompressed encoding
  - Triangle nodes: list of 32-bit floating point vertices
  - AABB nodes: bounding box min and max coordinates
  - Instance nodes: same as AABB but with extra instance culling information
- *Very* complex compressed encoding for triangle & AABB nodes
  - Reverse engineered but not implemented yet in turnip!
  - Even on the blob, only implemented when building on CPU
  - Can store up to 4 triangles in a single 64-byte descriptor

Ray Tracing in Turnip



Prior Art in Mesa

- anv was the first driver to gain RT support
- BVH traversal is handled entirely in hardware
- BVH building is very Intel-specific, complicated, designed to be shared with Windows driver
  - Not a great base for other drivers
- radv was the second driver
  - BVH building uses a generic "IR" to support different construction methods
  - Mostly driver-independent!
  - BVH traversal is implemented in the driver, like Qualcomm

Turnip BVH Building

- Can it be done? 🤔 Yes!
  - Fork radv, remove encoding part
  - `s/radv_ir/vk_ir/`
  - Add support for different subgroup sizes
  - Use generic Vulkan meta framework for compiling kernels
    - Not done yet for radix sort
  - Add driver callbacks:
    - Get the maximum size of a final encoded AS
    - Encode an AS
    - Update an AS (optional, not used by turnip yet)

- Building is split into multiple *passes*
  - Each pass is run in parallel for all BVH trees being built
  - Improves parallelism and reduces pipeline bubbles when building multiple BVHs at once
  - However this complicates the encoding callback
- Driver may choose different encoding tradeoffs (size, build speed, traversal speed) based on user flags
  - This is exposed through different *encoding keys* chosen by the driver
  - BVHs are sorted based on encoding key by the runtime
  - The driver may bind different pipelines based on the key

Generic BVH Building

- The IR resides in user-allocated scratch space
- Consists of:
  - The header (`vk_ir_header`)
  - An array of leaf nodes
  - An array of internal nodes (directly after the leaf nodes)
- Each leaf node has data taken directly from the user primitive
- Internal nodes (`vk_ir_box_node`) have an array of two children
  - The encoder collapses internal nodes for BVH formats with more children

Generic BVH Building

- Requires a few "generic" callbacks for functionality not in Vulkan for convenience:
  - `vkCmdFillBuffer` with device address
  - Write immediate data
    - Used to fill `vk_ir_header`
    - Expected to only be used with a small amount of data
    - `CP_MEM_WRITE` on AMD/Qualcomm
  - "non-aligned" dispatches similar to OpenCL
- All of these could/should be Vulkan extensions

- Various algorithms require forward progress guarantees because workgroups wait on results of earlier workgroups
    - Could be weakened if necessary by e.g. assigning workgroups by incrementing an atomic
- This is something else that could/should be a Vulkan extension
- In practice, just Assume It Works (tm)

BVH Encoding in Turnip

- Again, heavily based on radv
- `s/radv_/tu_/`
- Top-down algorithm for folding children into internal nodes
  - Each node allocates space for its children and encodes any leaf children
  - Modified in turnip to always allocate children contiguously
- Added support for compressing internal nodes

Ray Traversal in Turnip

Ray Traversal in Turnip

- Again adapted from radv
- Terrifying pile of `nir_builder` to implement the traversal loop
  - Future project: compile from CLC instead
- Lower opaque `rayQueryEXT` struct to concrete Turnip-specific struct
- Keep track of a limited number of ancestors
  - When pushing a node onto the stack, overwrite the oldest ancestor
  - When we run out of space, re-intersect it
  - This hopefully happens rarely
- Different stack layout due to different format of HW instruction output

Future Work

- Getting it merged
- More performance tuning?
- Support for accelerated updating of BVH trees
- Support compressed triangle/AABB nodes
- AQE and `VK_KHR_ray_tracing_pipeline`

- Generic BVH building: https://gitlab.freedesktop.org/mesa/mesa/-/merge_requests/28446
- Turnip VK_KHR_`ray_query`: https://gitlab.freedesktop.org/mesa/mesa/-/merge_requests/28447
- RTU documentation: https://gitlab.freedesktop.org/freedreno/freedreno/-/wikis/a7xx-ray-tracing
- AQE documentation: https://gitlab.freedesktop.org/freedreno/freedreno/-/wikis/AQE#ray-tracing
- radv: Can someone else please port it over to common BVH building 🙏