BASICS OF OPTIMIZATION

Before jumping to GPU, take some baby steps

BEFORE DIVING INTO PARALLELIZATION…

- Need to talk about some performance basics

BEFORE DIVING INTO PARALLELIZATION…

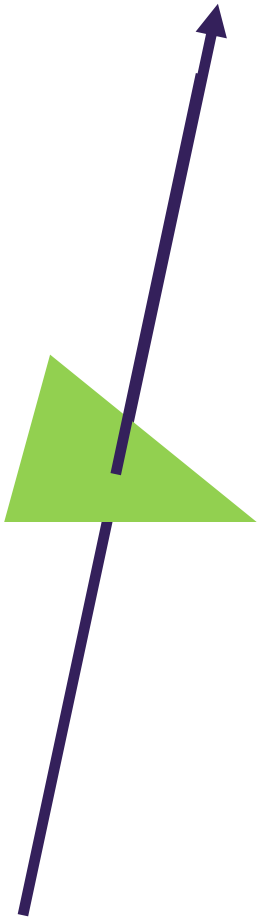
- Need to talk about some performance basics
  - Why is tracing rays slow at all?

BEFORE DIVING INTO PARALLELIZATION…

- Need to talk about some performance basics
  - Why is tracing rays slow at all?
- Consider basic ray tracing algorithm:
  - Take a ray through your scene

BEFORE DIVING INTO PARALLELIZATION…

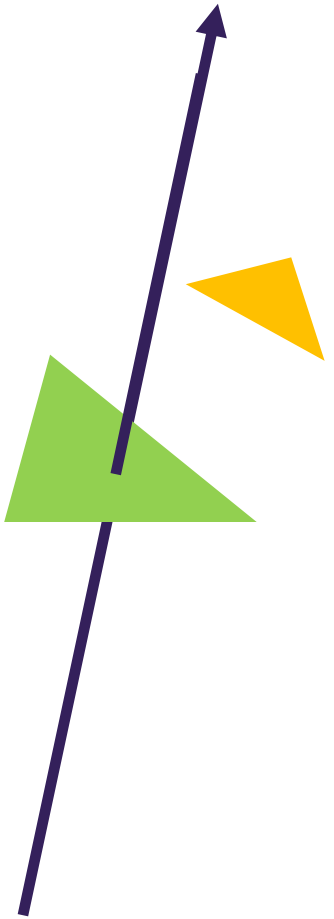
- Need to talk about some performance basics
  - Why is tracing rays slow at all?
- Consider basic ray tracing algorithm:
  - Take a ray through your scene
  - Test triangle to find intersection



BEFORE DIVING INTO PARALLELIZATION…

- Need to talk about some performance basics
  - Why is tracing rays slow at all?

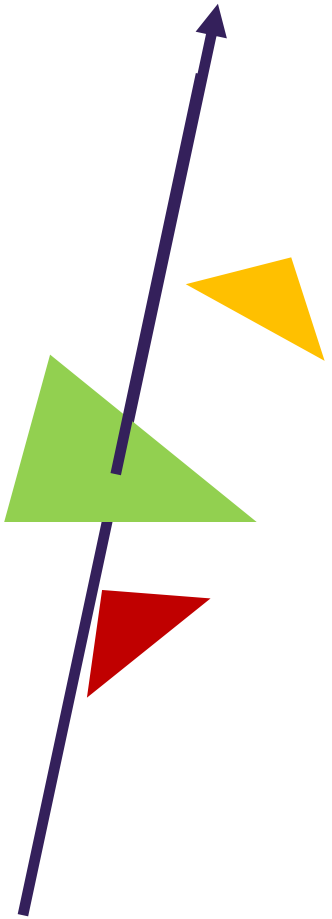
- Consider basic ray tracing algorithm:
  - Take a ray through your scene
  - Test triangle to find intersection
    - Repeat



BEFORE DIVING INTO PARALLELIZATION…

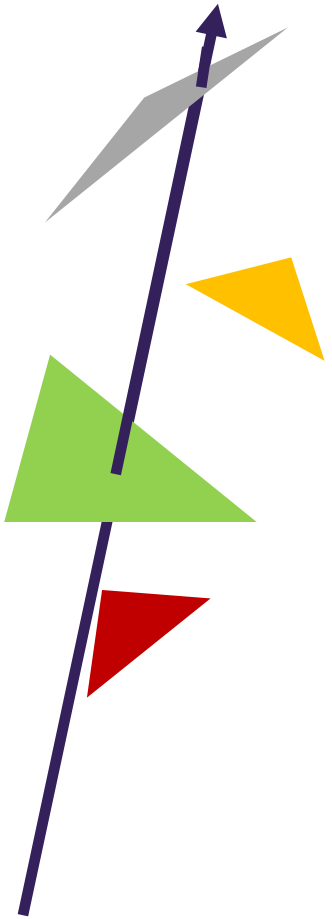
- Need to talk about some performance basics
  - Why is tracing rays slow at all?

- Consider basic ray tracing algorithm:
  - Take a ray through your scene
  - Test triangle to find intersection
    - Repeat



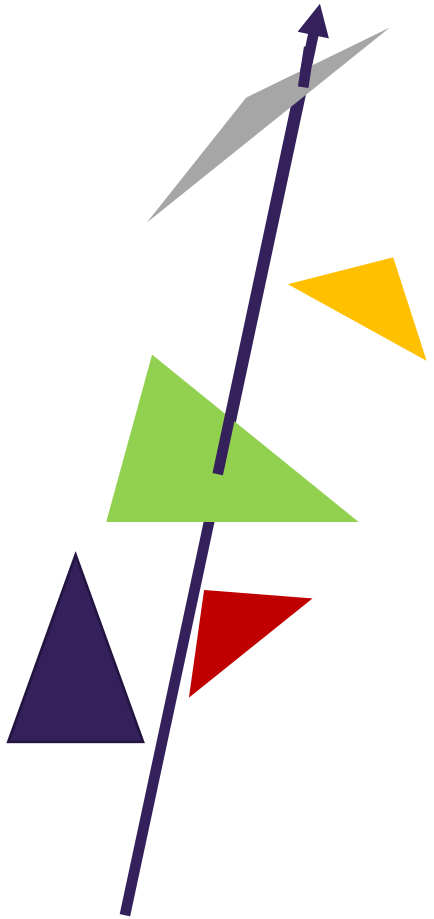
BEFORE DIVING INTO PARALLELIZATION…

- Need to talk about some performance basics
  - Why is tracing rays slow at all?
- Consider basic ray tracing algorithm:
  - Take a ray through your scene
  - Test triangle to find intersection
    - Repeat



BEFORE DIVING INTO PARALLELIZATION…

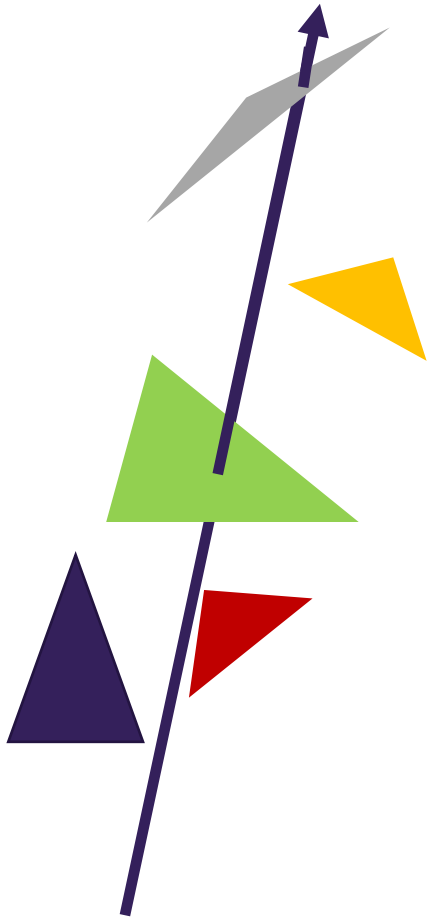
- Need to talk about some performance basics
  - Why is tracing rays slow at all?
- Consider basic ray tracing algorithm:
  - Take a ray through your scene
  - Test triangle to find intersection
    - Repeat



BEFORE DIVING INTO PARALLELIZATION…

- Need to talk about some performance basics
  - Why is tracing rays slow at all?

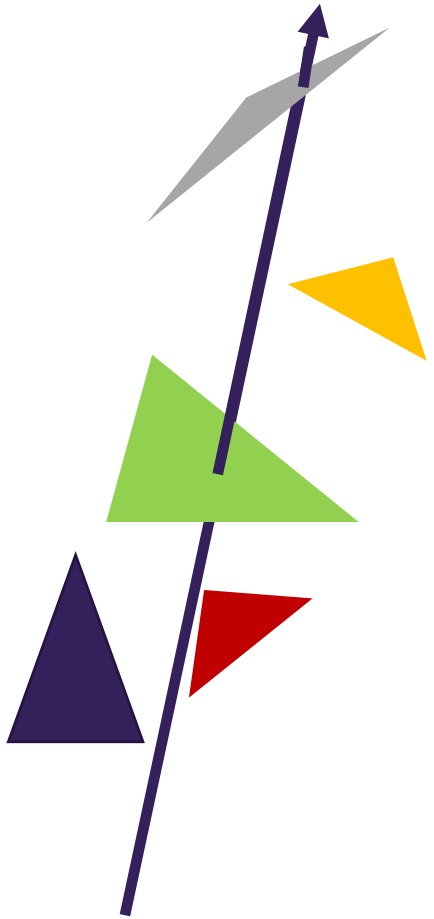
- Consider basic ray tracing algorithm:
  - Take a ray through your scene
  - Test triangle to find intersection
    - Repeat
  - How do you know when you're done?



BEFORE DIVING INTO PARALLELIZATION…

- Need to talk about some performance basics
  - Why is tracing rays slow at all?

- Consider basic ray tracing algorithm:
  - Take a ray through your scene
  - Test triangle to find intersection
    - Repeat
  - How do you know when you're done?
    - When you've tested every triangle?



BEFORE DIVING INTO PARALLELIZATION…

- Need to talk about some performance basics
  - Why is tracing rays slow at all?

- Consider basic ray tracing algorithm:
  - Take a ray through your scene
  - Test triangle to find intersection
    - Repeat
  - How do you know when you're done?
    - When you've tested every triangle?
    - Very expensive…
    - Every ray could test, 1 million (or more) triangles

WHAT'S OUR COMPUTATION BUDGET?

WHAT'S OUR COMPUTATION BUDGET?

- Let's be easy on ourselves:
  - Target just 1920 x 1080 at 60 fps

WHAT'S OUR COMPUTATION BUDGET?

- Let's be easy on ourselves:
  - Target just 1920 x 1080 at 60 fps
  - We need 125 million pixels per second!

WHAT'S OUR COMPUTATION BUDGET?

- Let's be easy on ourselves:
  - Target just 1920 x 1080 at 60 fps
  - We need 125 million pixels per second!

- With a ~10 TFLOP state-of-the-art GPU
  - If tracing one ray per pixel…
  - About 80,000 flops per ray

WHAT'S OUR COMPUTATION BUDGET?

- Let's be easy on ourselves:
  - Target just 1920 x 1080 at 60 fps
  - We need 125 million pixels per second!

- With a ~10 TFLOP state-of-the-art GPU
  - If tracing one ray per pixel…
  - About 80,000 flops per ray

- An optimized triangle intersection: ~10 flops
  - Can afford ***at most*** 8,000 intersections per ray

WHAT'S OUR COMPUTATION BUDGET?

- Let's be easy on ourselves:
  - Target just 1920 x 1080 at 60 fps
  - We need 125 million pixels per second!
- With a ~10 TFLOP state-of-the-art GPU
  - If tracing one ray per pixel…
  - About 80,000 flops per ray
- An optimized triangle intersection: ~10 flops
  - Can afford ***at most*** 8,000 intersections per ray
- Conclusion: ***Don't test every triangle!***

KEY PRINCIPAL TO OPTIMIZATION:

- Make the ***common*** case fast

KEY PRINCIPAL TO OPTIMIZATION:

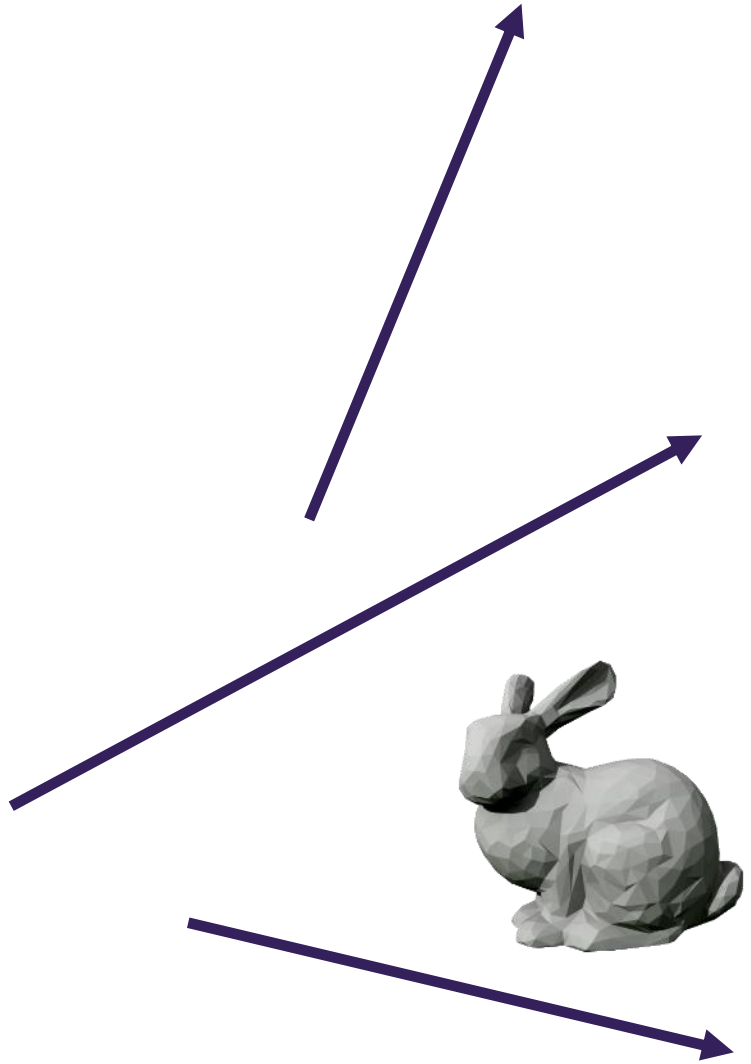
- Make the ***common*** case fast

- Common case in ray tracing?
  - Ray does not intersect a triangle

KEY PRINCIPAL TO OPTIMIZATION:

- Make the ***common*** case fast

- Common case in ray tracing?
  - Ray does not intersect a triangle
  - For any mesh, ray typically misses mesh

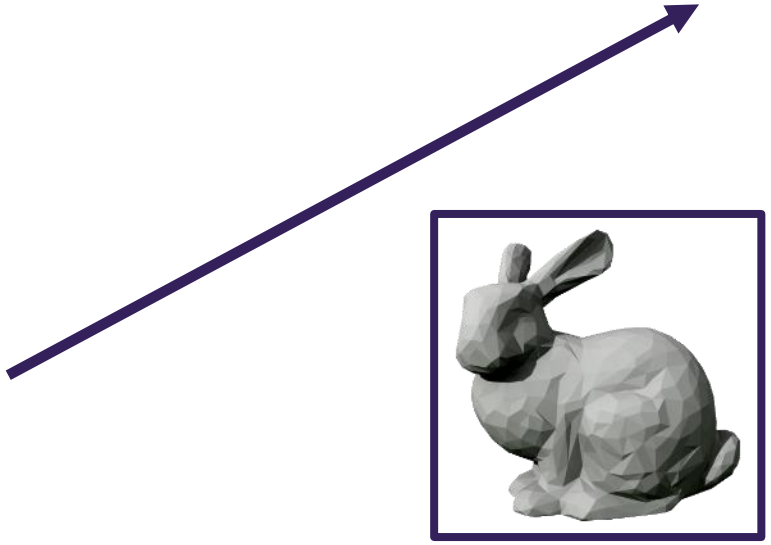


KEY PRINCIPAL TO OPTIMIZATION:

- Make the ***common*** case fast

- Common case in ray tracing?
  - Ray does not intersect a triangle
  - For any mesh, ray typically misses mesh

- Perhaps:
  - First intersect a mesh bounding box

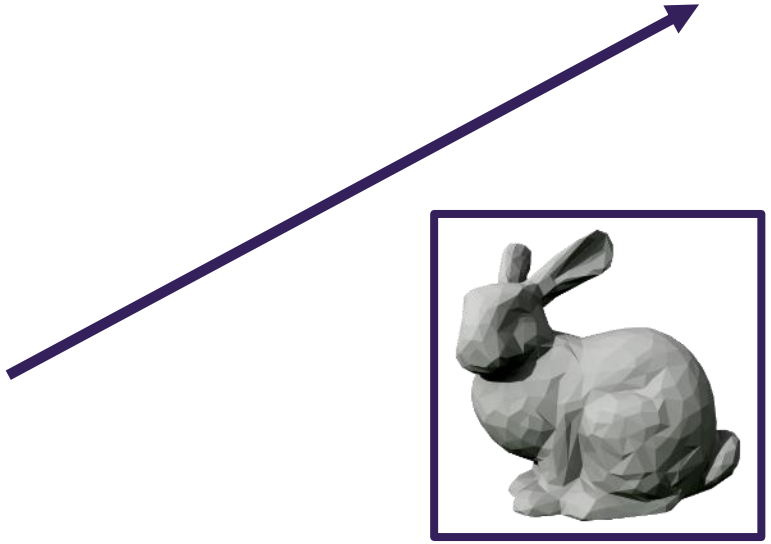


KEY PRINCIPAL TO OPTIMIZATION:

- Make the ***common*** case fast

- Common case in ray tracing?
  - Ray does not intersect a triangle
  - For any mesh, ray typically misses mesh

- Perhaps:
  - First intersect a mesh bounding box
  - Most rays avoid testing thousands of triangles
  - But, extra box test when hit bunny

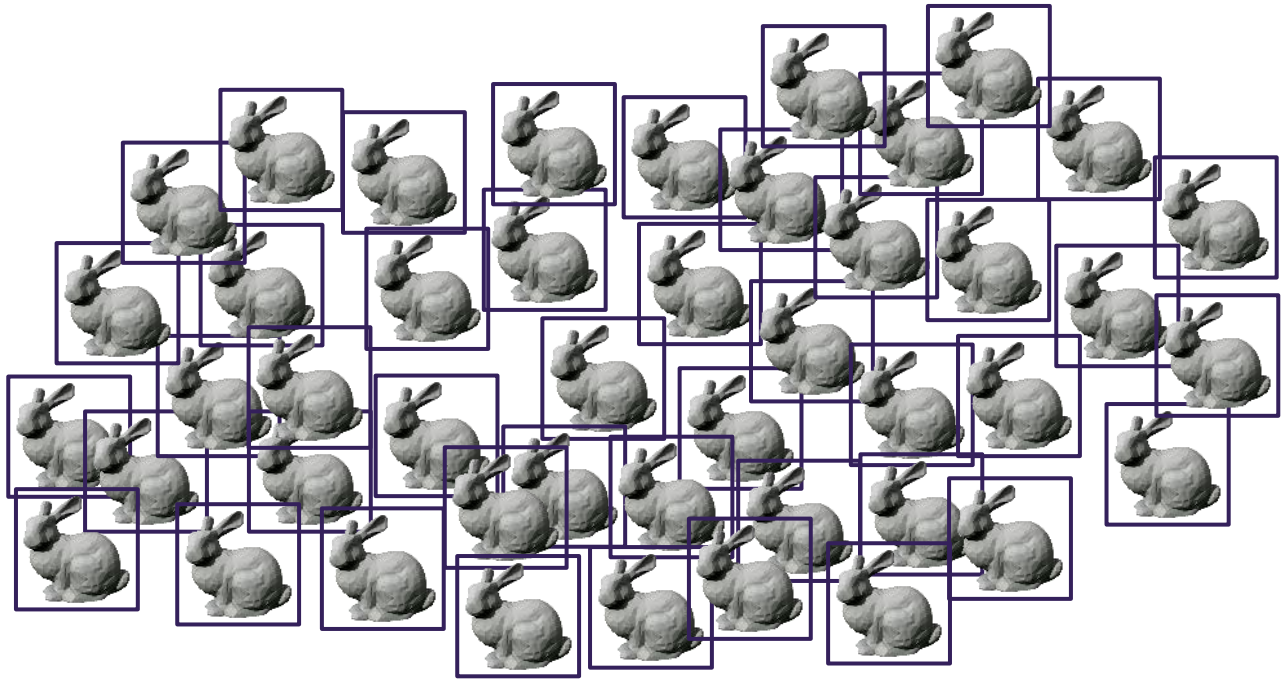


KEY PRINCIPAL TO OPTIMIZATION:

- What if you have thousands of bunnies?

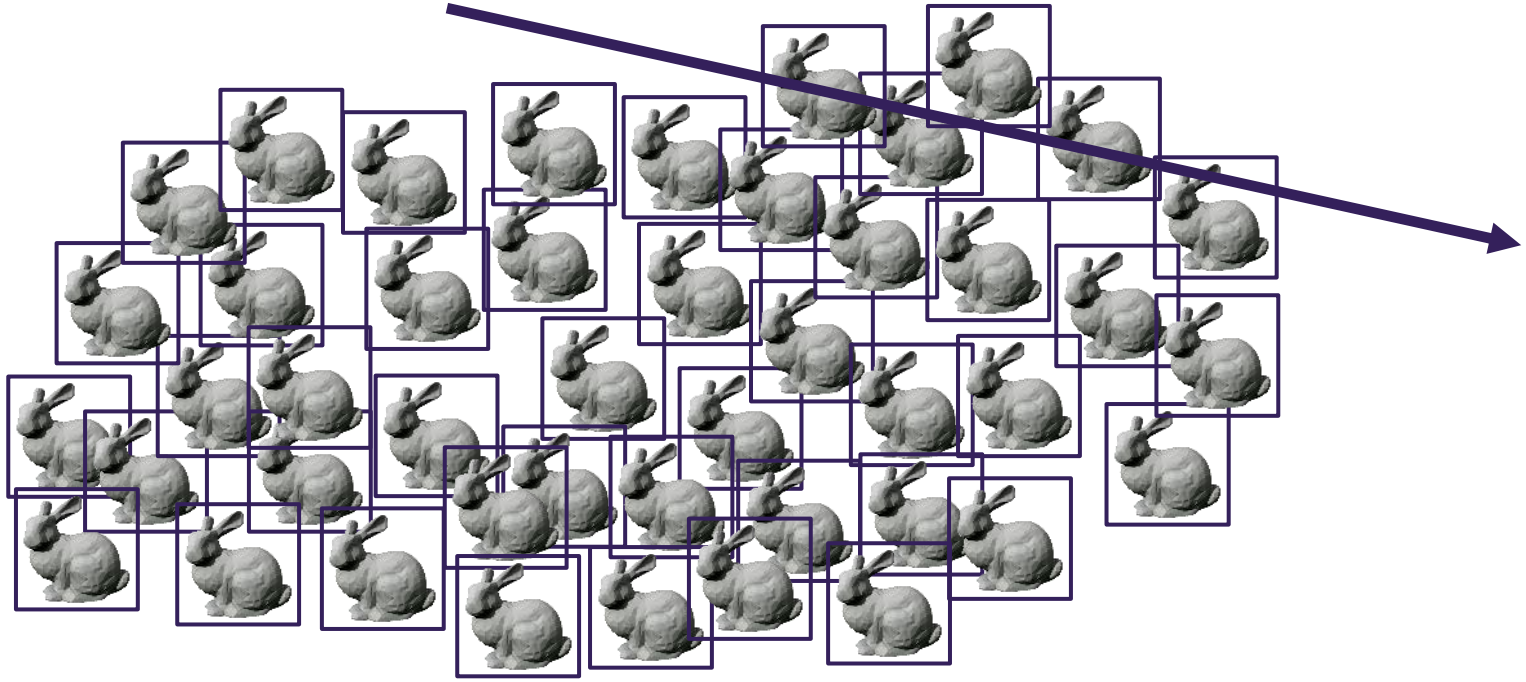
KEY PRINCIPAL TO OPTIMIZATION:

- What if you have thousands of bunnies?



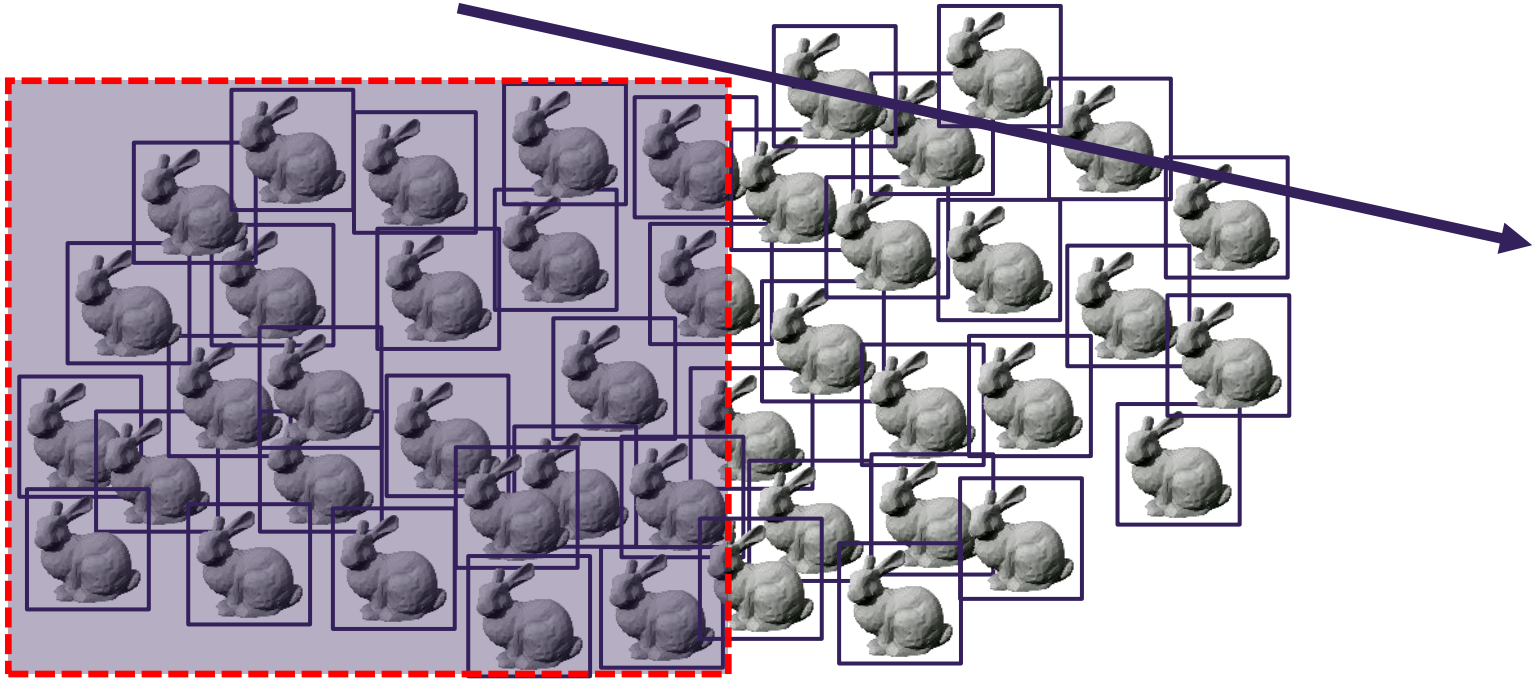
KEY PRINCIPAL TO OPTIMIZATION:

- What if you have thousands of bunnies?
  - Common case: Ray misses most bunnies



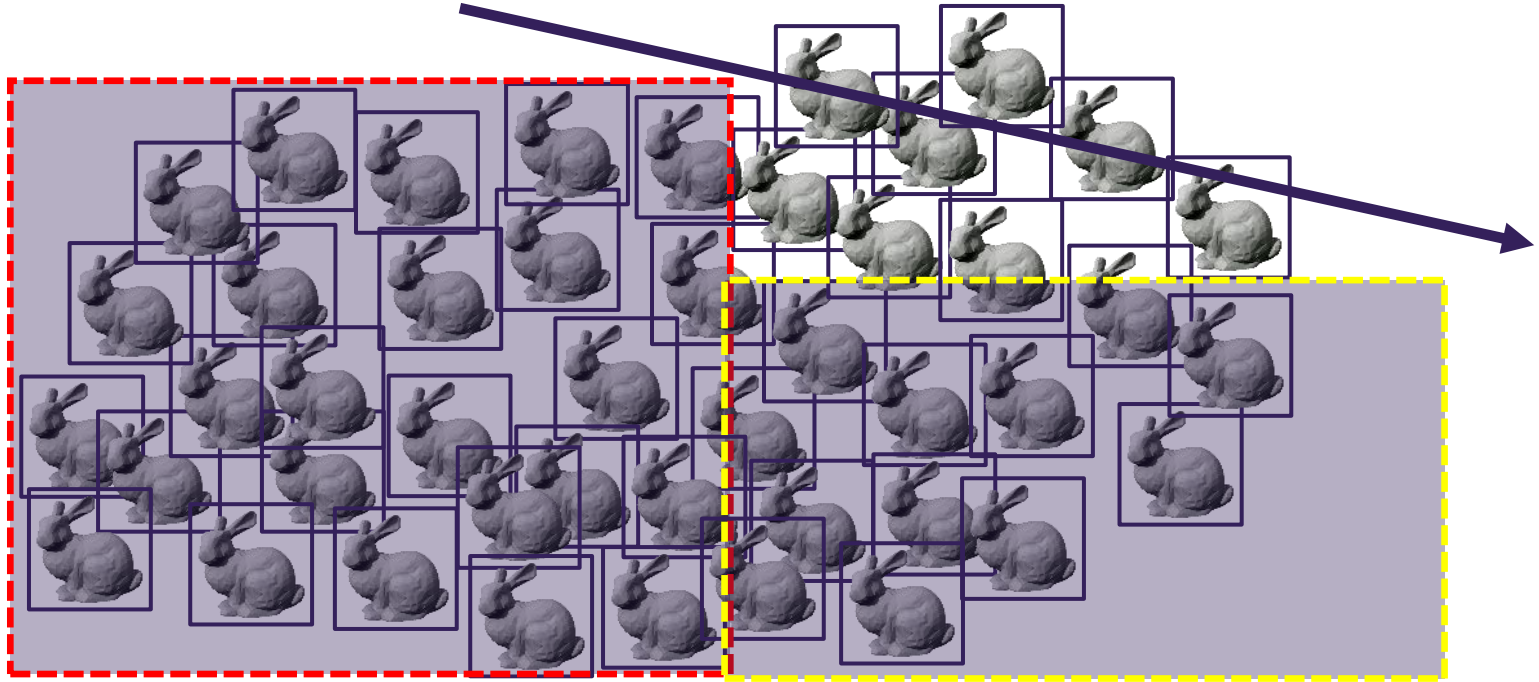
KEY PRINCIPAL TO OPTIMIZATION:

- What if you have thousands of bunnies?
  - Common case: Ray misses most bunnies
  - Can skip testing this half…



KEY PRINCIPAL TO OPTIMIZATION:

- What if you have thousands of bunnies?
  - Common case: Ray misses most bunnies
  - Can skip testing this half… and this quarter… with a few more boxes



KEY PRINCIPAL TO OPTIMIZATION:

- Build a tree of bounding boxes
  - Known as a “bounding volume hierarchy” or BVH

KEY PRINCIPAL TO OPTIMIZATION:

- Build a tree of bounding boxes
  - Known as a “bounding volume hierarchy” or BVH
- When using a principled tree build
  - Reduces number of required intersections
  - From O(N) to O(log N)

KEY PRINCIPAL TO OPTIMIZATION:

- Build a tree of bounding boxes
  - Known as a “bounding volume hierarchy” or BVH
- When using a principled tree build
  - Reduces number of required intersections
  - From O(N) to O(log N)
- With a binary tree, 1 million ray-triangle tests becomes:
  - Around 20 ray-box tests
  - A few ray-triangle tests in leaf nodes

KEY PRINCIPAL TO OPTIMIZATION:

- Production ray tracers ***always*** use some acceleration structure

KEY PRINCIPAL TO OPTIMIZATION:

- Production ray tracers ***always*** use some acceleration structure

- But, ***which*** structure? How do you best build it?
  - Literally decades of research

KEY PRINCIPAL TO OPTIMIZATION:

- Production ray tracers ***always*** use some acceleration structure

- But, ***which*** structure? How do you best build it?
  - Literally decades of research
  - Continuing to today (e.g., “Wide BVH Traversal with a Short Stack,” Vaidyanathan et al. 2019)

KEY PRINCIPAL TO OPTIMIZATION:

- Production ray tracers ***always*** use some acceleration structure

- But, ***which*** structure? How do you best build it?
  - Literally decades of research
  - Continuing to today (e.g., “Wide BVH Traversal with a Short Stack,” Vaidyanathan et al. 2019)

- When starting real-time ray tracing, best bet:
  - Use someone else’s code
  - Quality of your BVH easily affects performance by 2x, 3x, or >10x
    - Varies per scene!
  - Luckily most APIs will build structure