Accelerating Shadow Rays Using Volumetric Occluders and Modified kd-Tree Traversal

Peter Djeu*, Sean Keely*, and Warren Hunt‡

* University of Texas at Austin ‡ Intel Labs

hpg09

Shadow Rays

- Shadow rays are often time consuming
- Intersection (isect) required with lots of triangles

vs

Triangles

Baseline

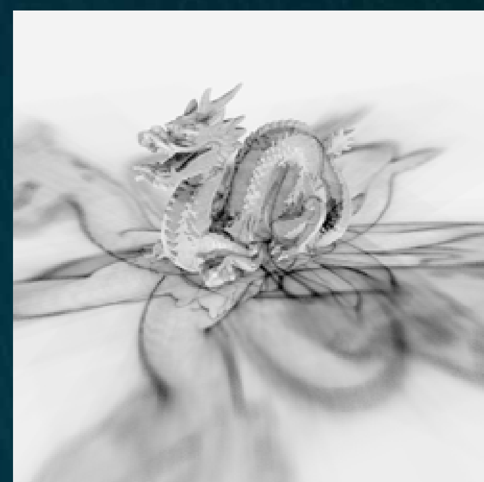
Our approach

vs

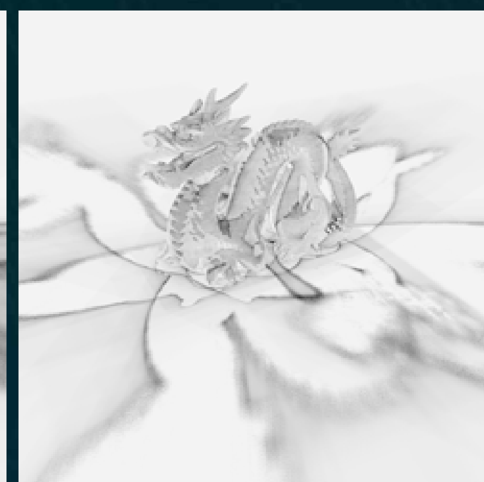
Volumetric Occluders and Triangles

Ray Tracing and Shadows

- Goal: faster shadows without modifying result
  - 50% reduction in time spent on shadow rays
  - Identical images



Baseline

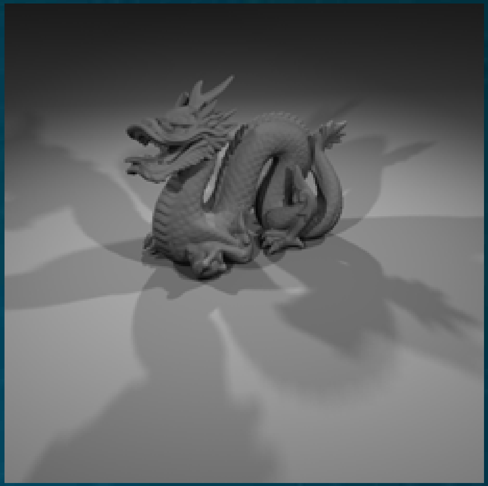


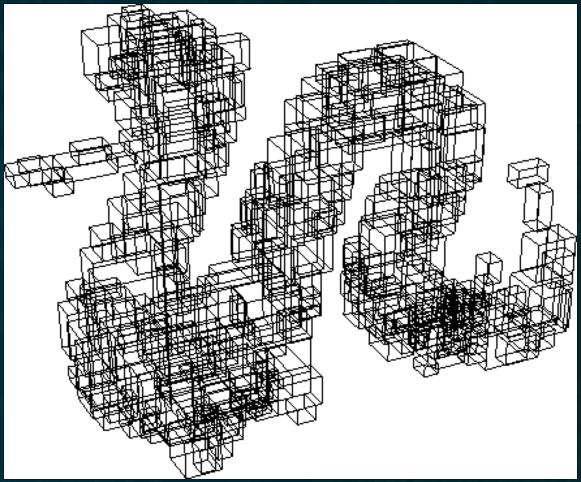
Our approach

What is a Volumetric Occluder?

- An axis aligned bounding box within the mesh interior

- Kd-tree nodes used to represent volumetric occluders





Outline

1. Create volumetric occluders

2. Modify kd-tree traversal to use volumetric occluders
    - Two novel algorithms

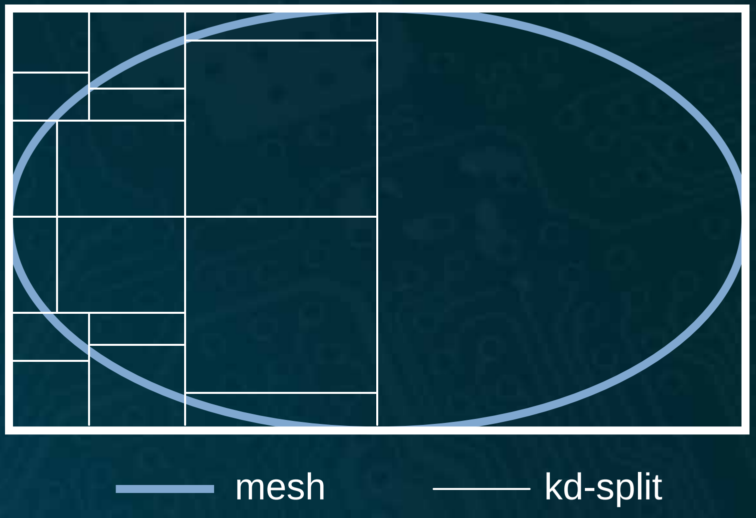
3. Reuse volumetric occluders

4. Results
    - Monte Carlo soft shadows

One prerequisite: Manifold mesh

- Input mesh must be manifold
  - Watertight
  - Consistent face orientation
  - No self intersections

Example: Oval mesh

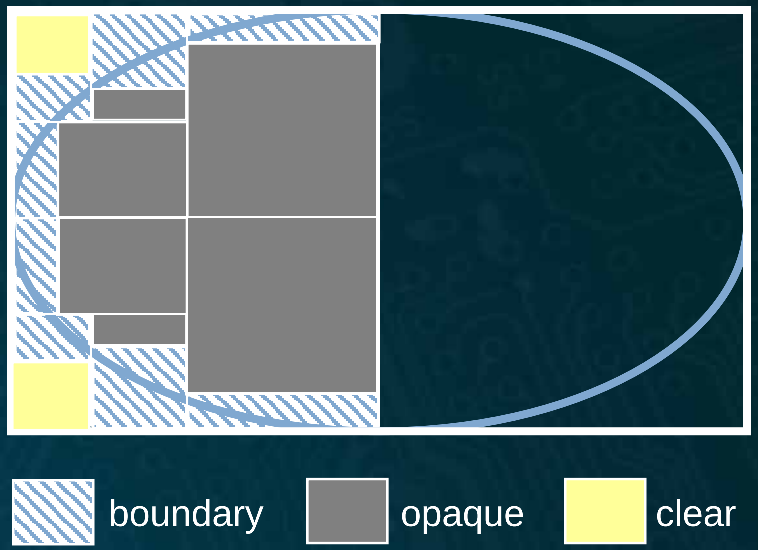


- Normal build – SAH

Classification

- For each leaf node in kd-tree, classify as:
  - Boundary node – non-empty
  - Opaque node – empty and inside
  - Clear node – empty and outside

Classifying the oval mesh



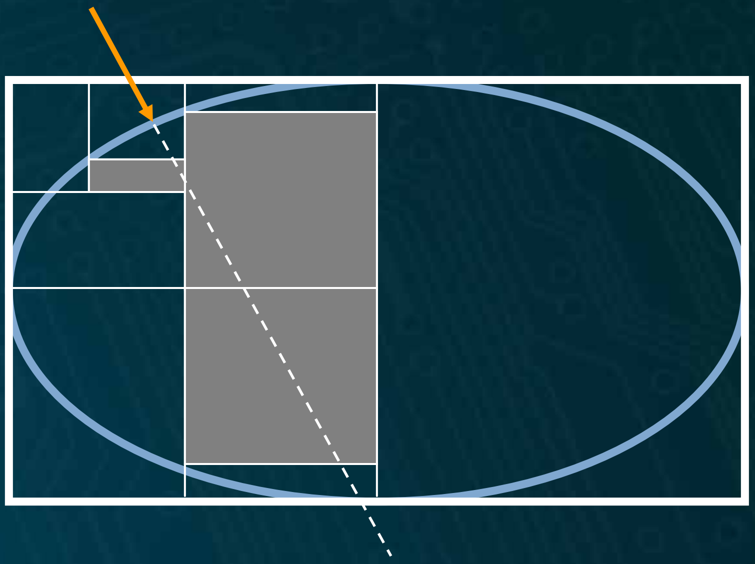
- Classification is easy on manifold mesh
- Opaque nodes are volumetric occluders

But are they useful?

- Intersection with a volumetric occluder implies intersection with mesh geometry provided:
  - At least one ray endpoint is outside of the mesh
- Volumetric occluders accelerate shadow rays
  - Cheap to isect
  - Often larger – better occlusion

One major problem

- Volumetric occluders are inaccessible under normal kd-tree traversal order!



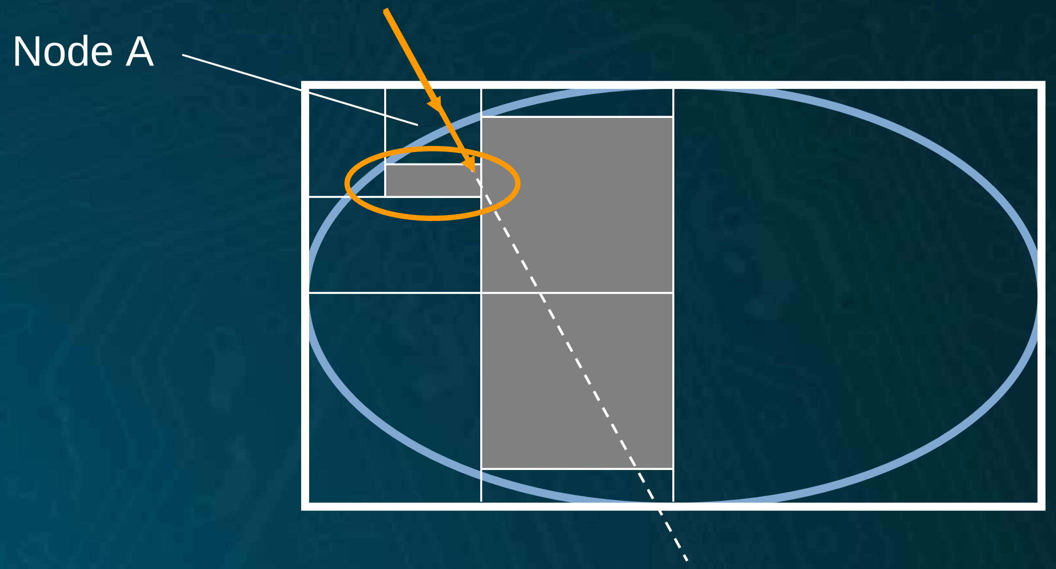
Standard Ray Order

Modifying kd-tree traversal to use volumetric occluders

- We present two ways to modify kd-tree traversal order
  - Goal: encounter volumetric occluders during traversal
- Both solutions perform the same task at different cost
- Both solutions enable encountering volumetric occluders during traversal
  - Intersection becomes a bit-mask and compare

Traversal Mod 1: Extended Ray Order

- Extend the ray past boundary nodes
    - Defer geometry isect using Deferment List



Deferment List

Node A

Extended Ray Order

Extended Ray Order Summary

- Speculatively take extra traversal steps
- If speculation pays off, we encounter a volumetric occluder

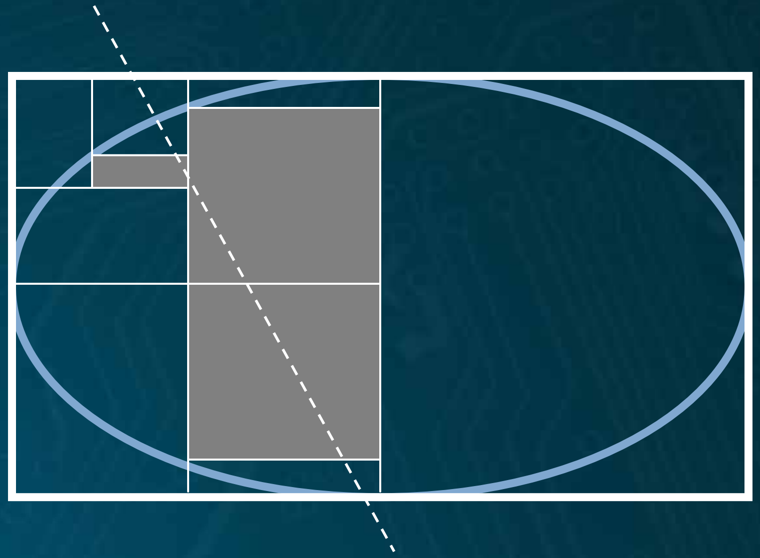
Extended Ray Order Expected Behavior

+ High chance geometry isects decrease

– Traversal steps guaranteed to increase

Intro to Traversal Mod 2: BFS

- Breadth-first search (BFS)
  - Shallow nodes are visited before deep ones
  - Code change: Stack → Queue

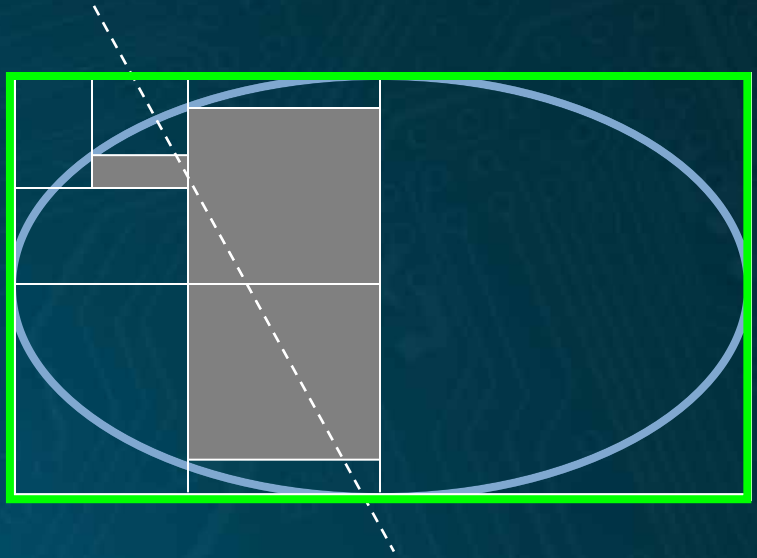


Queue

BFS Order

Intro to Traversal Mod 2: BFS

- Breadth-first search (BFS)
  - Shallow nodes are visited before deep ones
  - Code change: Stack → Queue



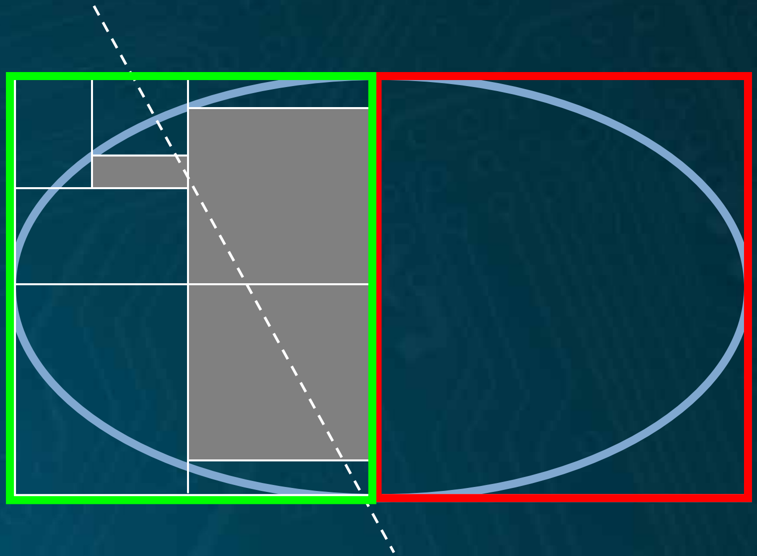
Queue

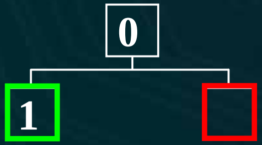
0

BFS Order

Intro to Traversal Mod 2: BFS

- Breadth-first search (BFS)
  - Shallow nodes are visited before deep ones
  - Code change: Stack → Queue





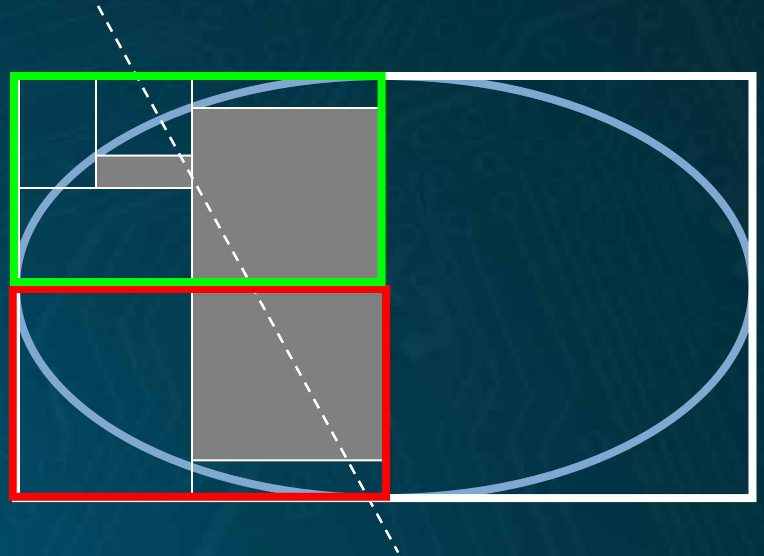
Queue

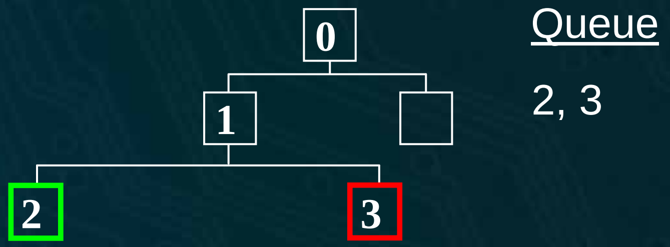
1

BFS Order

Intro to Traversal Mod 2: BFS

- Breadth-first search (BFS)
  - Shallow nodes are visited before deep ones
  - Code change: Stack → Queue

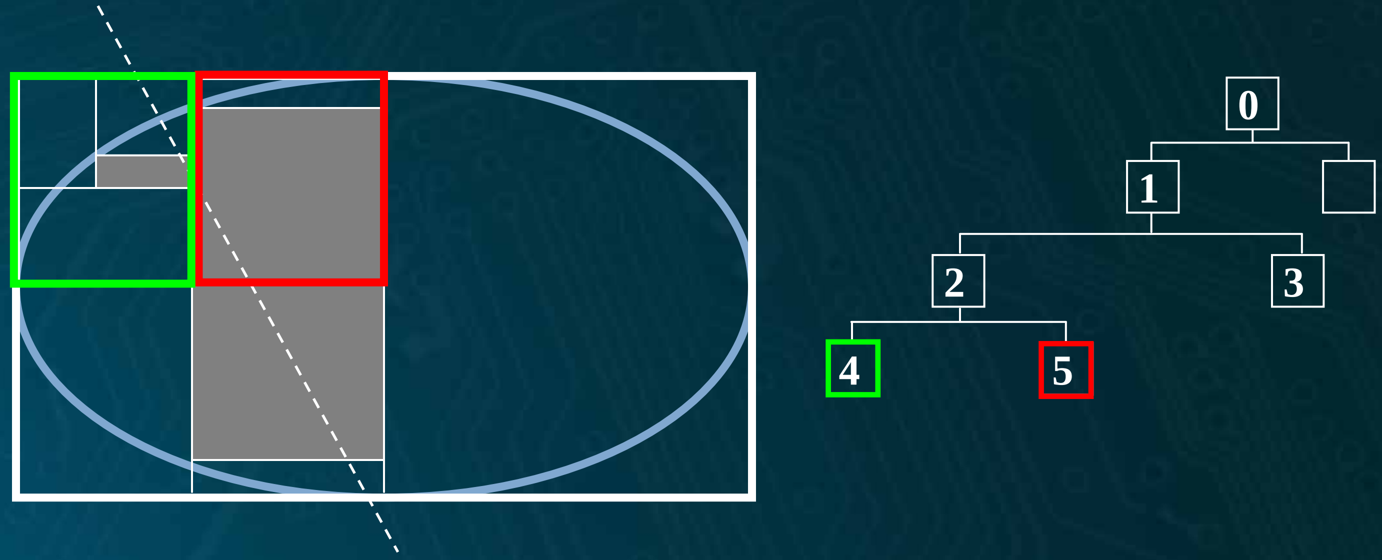




BFS Order

Intro to Traversal Mod 2: BFS

- Breadth-first search (BFS)
  - Shallow nodes are visited before deep ones
  - Code change: Stack → Queue



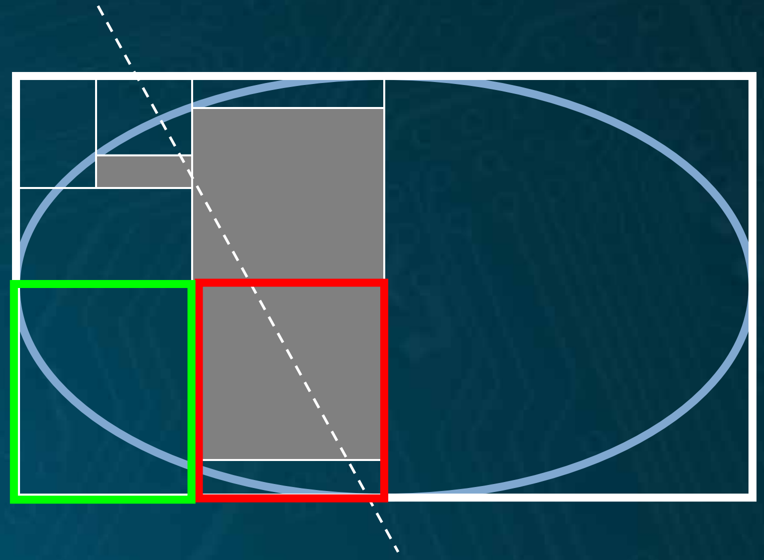
Queue

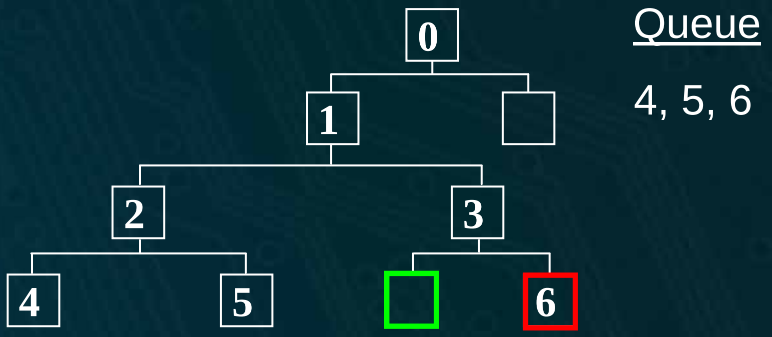
3, 4, 5

BFS Order

Intro to Traversal Mod 2: BFS

- Breadth-first search (BFS)
  - Shallow nodes are visited before deep ones
  - Code change: Stack → Queue

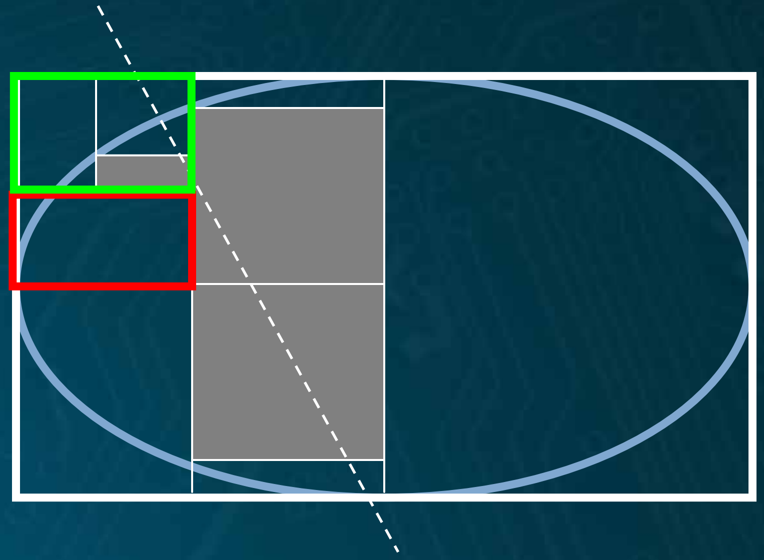


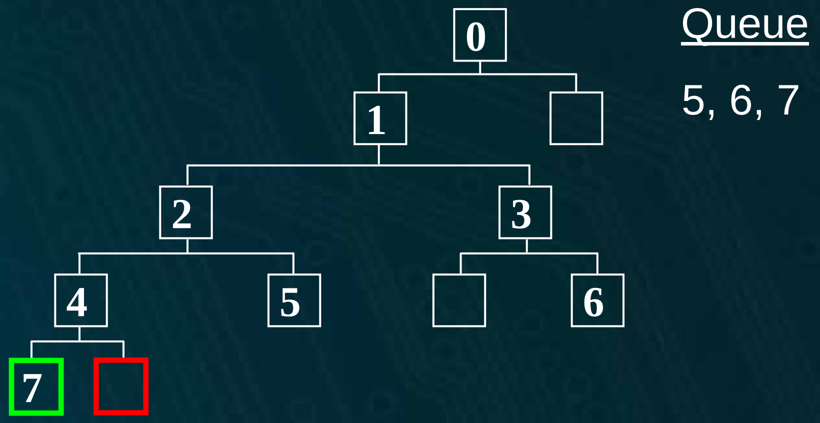


BFS Order

Intro to Traversal Mod 2: BFS

- Breadth-first search (BFS)
  - Shallow nodes are visited before deep ones
  - Code change: Stack → Queue

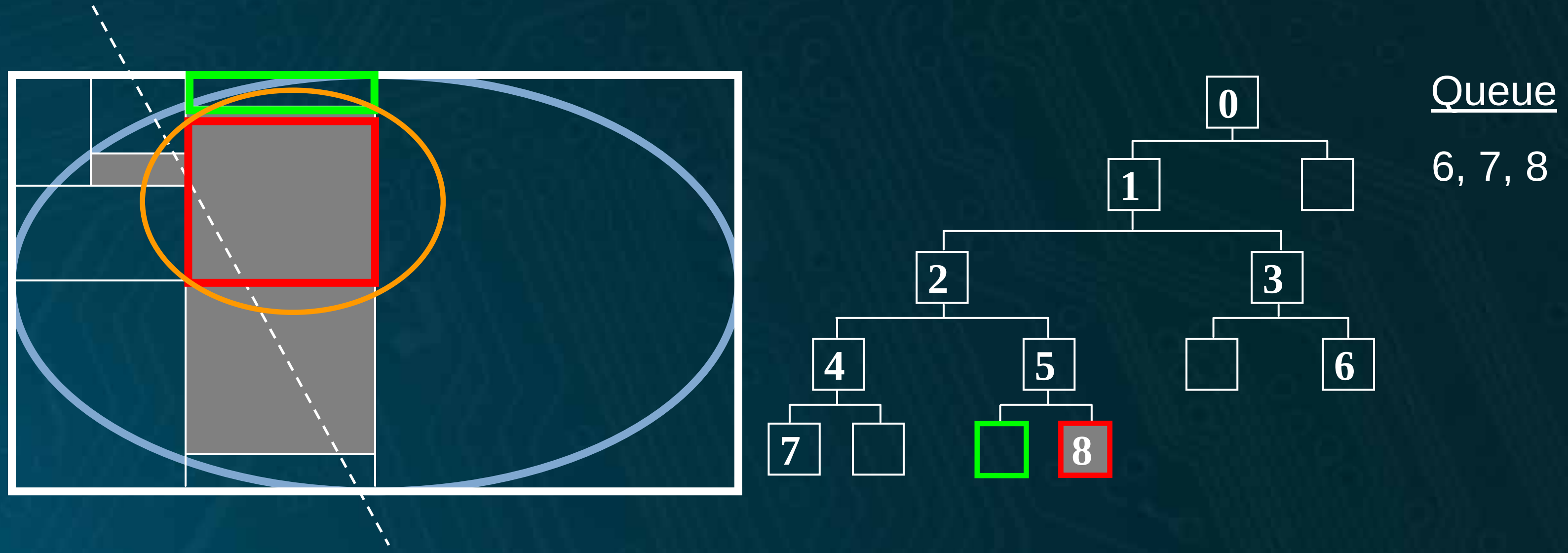




BFS Order

Intro to Traversal Mod 2: BFS

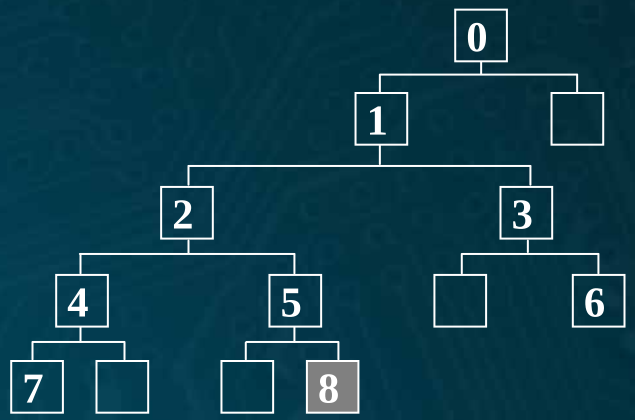
- Breadth-first search (BFS)
  - Shallow nodes are visited before deep ones
  - Code change: Stack → Queue



BFS Order

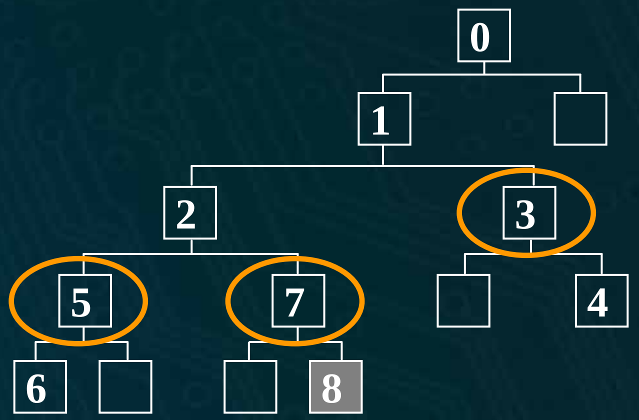
Traversal Mod 2: QDBFS

- Quick descent bread-first search (QDBFS)
  - Immediately descend in one child case



BFS Order

Nodes visited: 9
Pushes: 10
Pops: 8

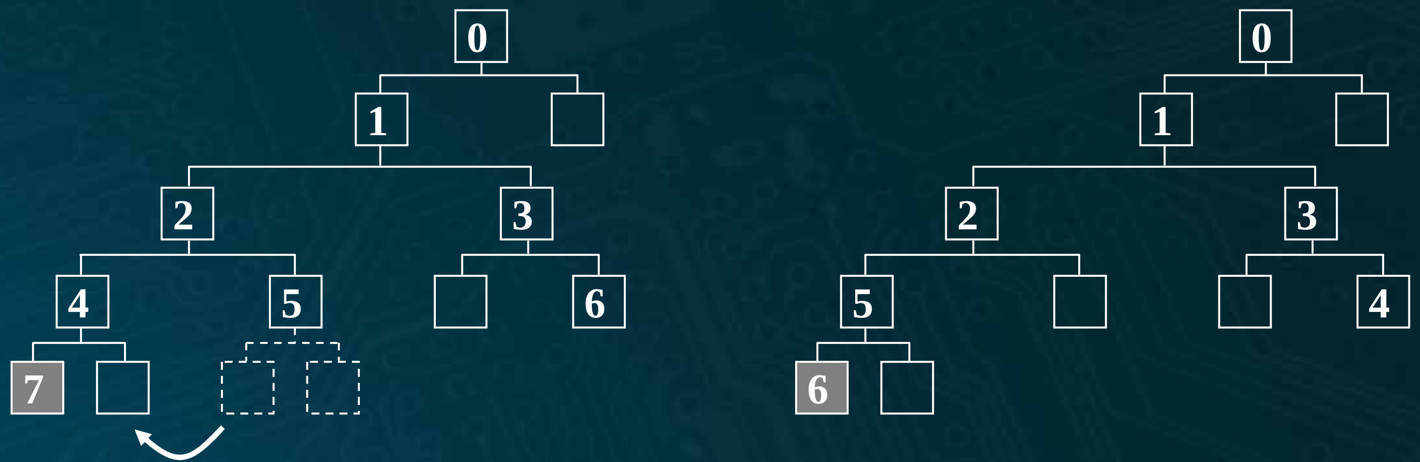


QDBFS Order

Nodes visited: 9
Pushes: 8
Pops: 4

Quick descent

- Can discover volumetric occluders sooner



BFS Order

Nodes visited: 8
Pushes: 10
Pops: 7

QDBFS Order

Nodes visited: 7
Pushes: 6
Pops: 3

BFS and QDBFS Summary

- BFS
  - Finds large volumetric occluders
- QDBFS
  - Less queue traffic
  - 15% - 20% faster than BFS

- QDBFS will be used for rest of talk

QDBFS Expected Behavior

+ High chance geometry isects decrease

+ Traversal steps may decrease

– Traversal steps may increase

+ Preference for large occluders

Volumetric occluder intersection

- Very, very cheap
- Volumetric occ. tag stored in flag bits in the kd-node
- Intersection leverages all work from kd-tree traversal

Reusing volumetric occluders

- The volumetric occluder cache (VC)
  - Software managed
  - Stores most recently used occluders
- Populate
- Lookup

Volumetric occluder cache tradeoffs

+ Traversal steps very likely to decrease

- On a cache hit, traversal steps drops to 0

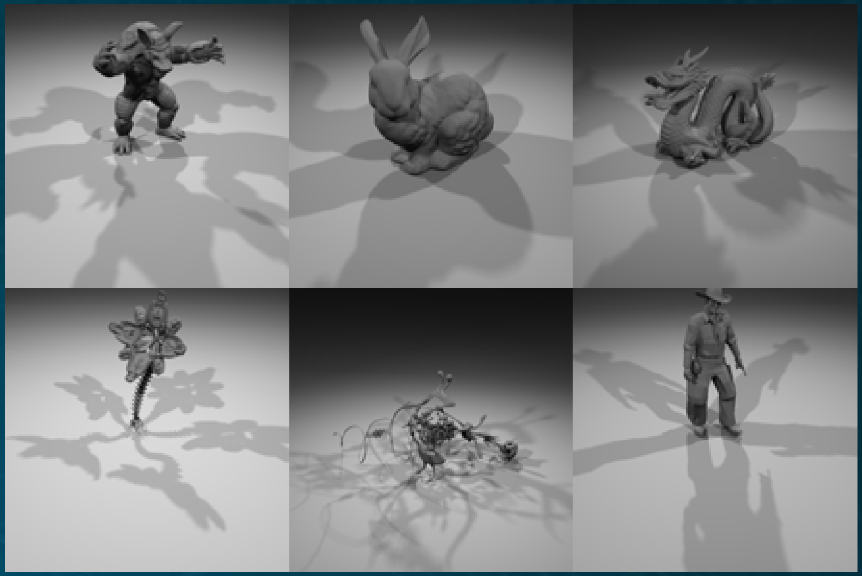
– May perform unnecessary bounding box isects

- If cache is size n, each cache miss costs n isects

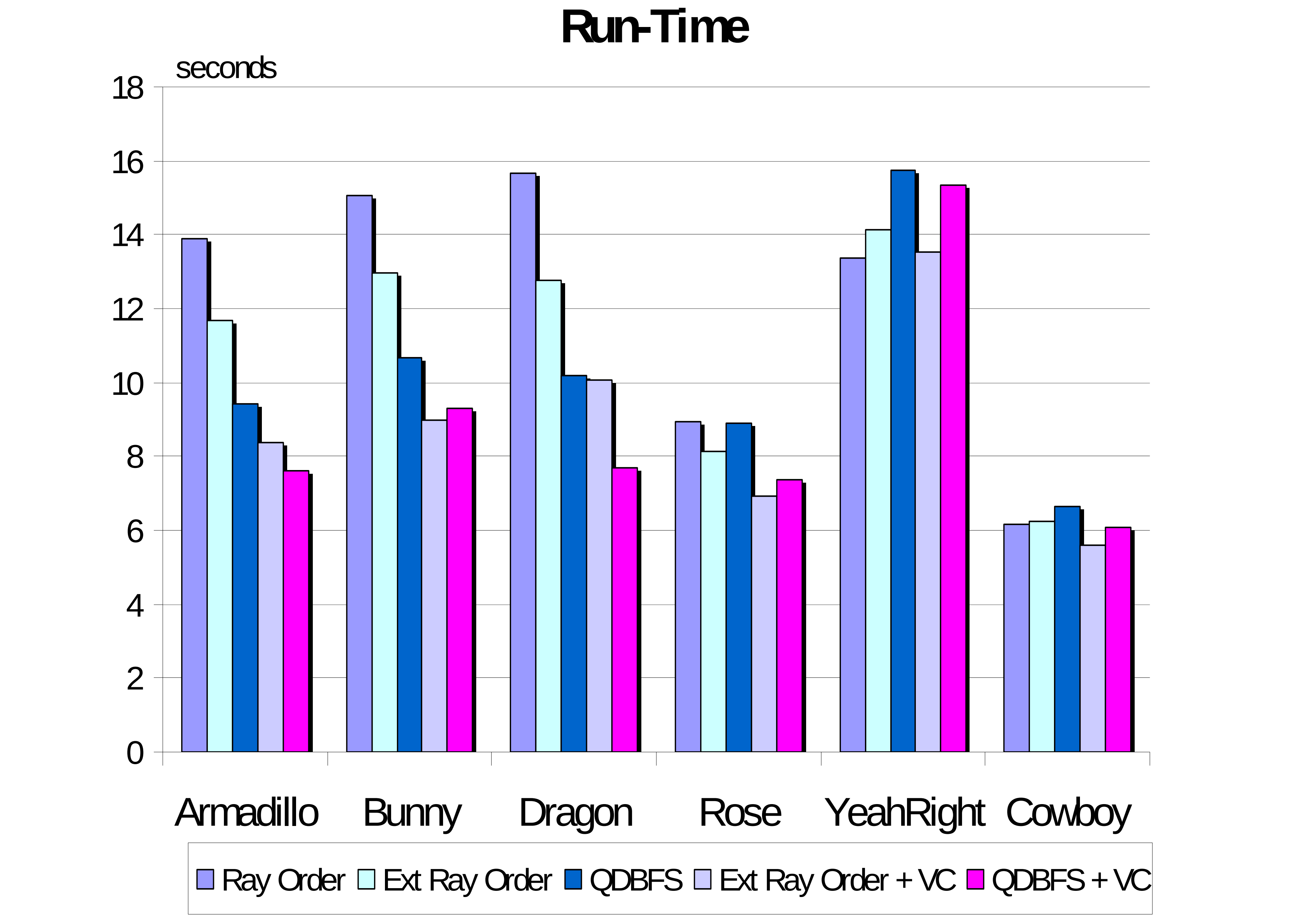
- In our experience, the VC always improved run-times

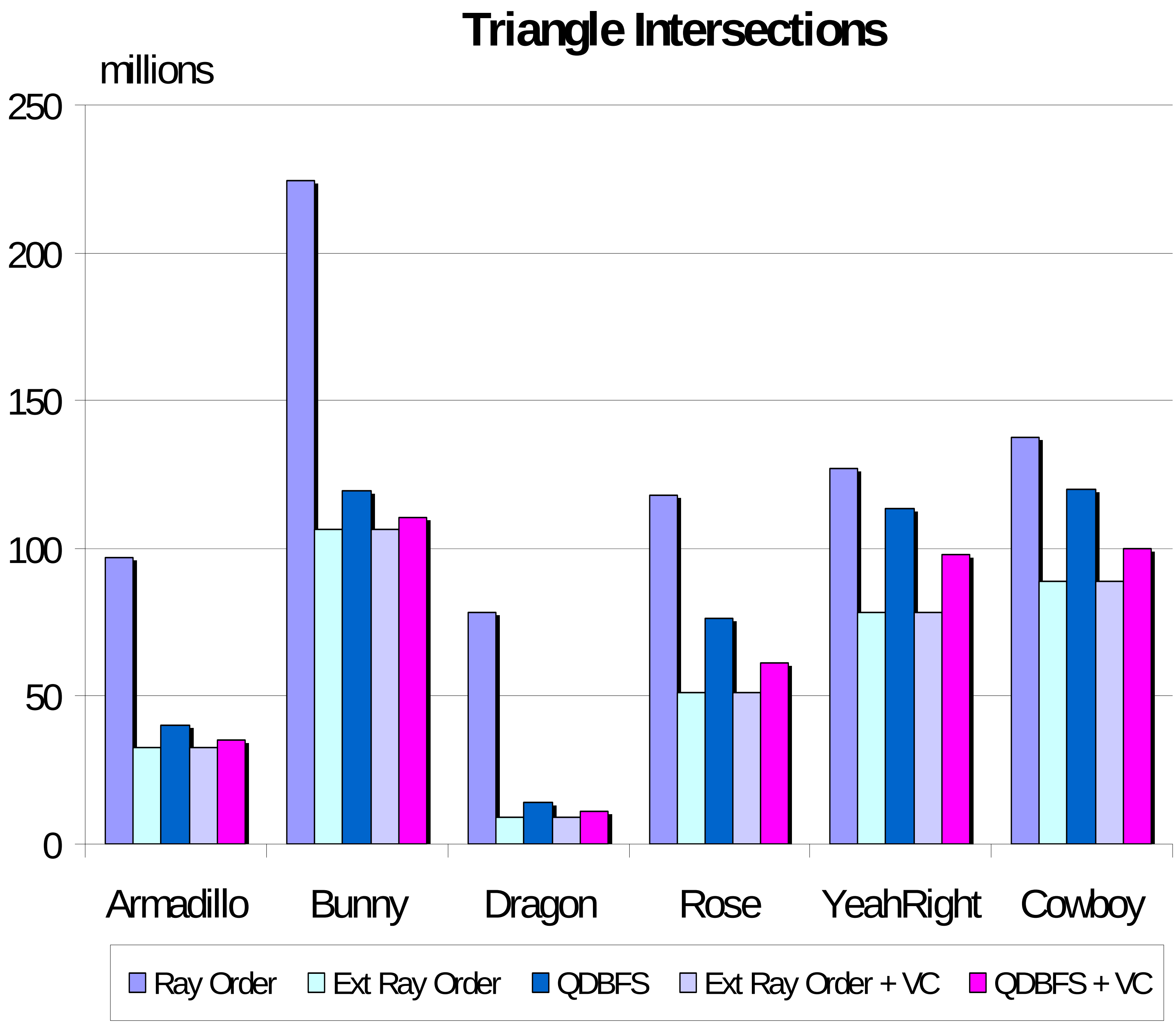
Results

- 4-wide packet tracer
- 1024 x 1024, 5 area lights, ~100 mil shadow rays

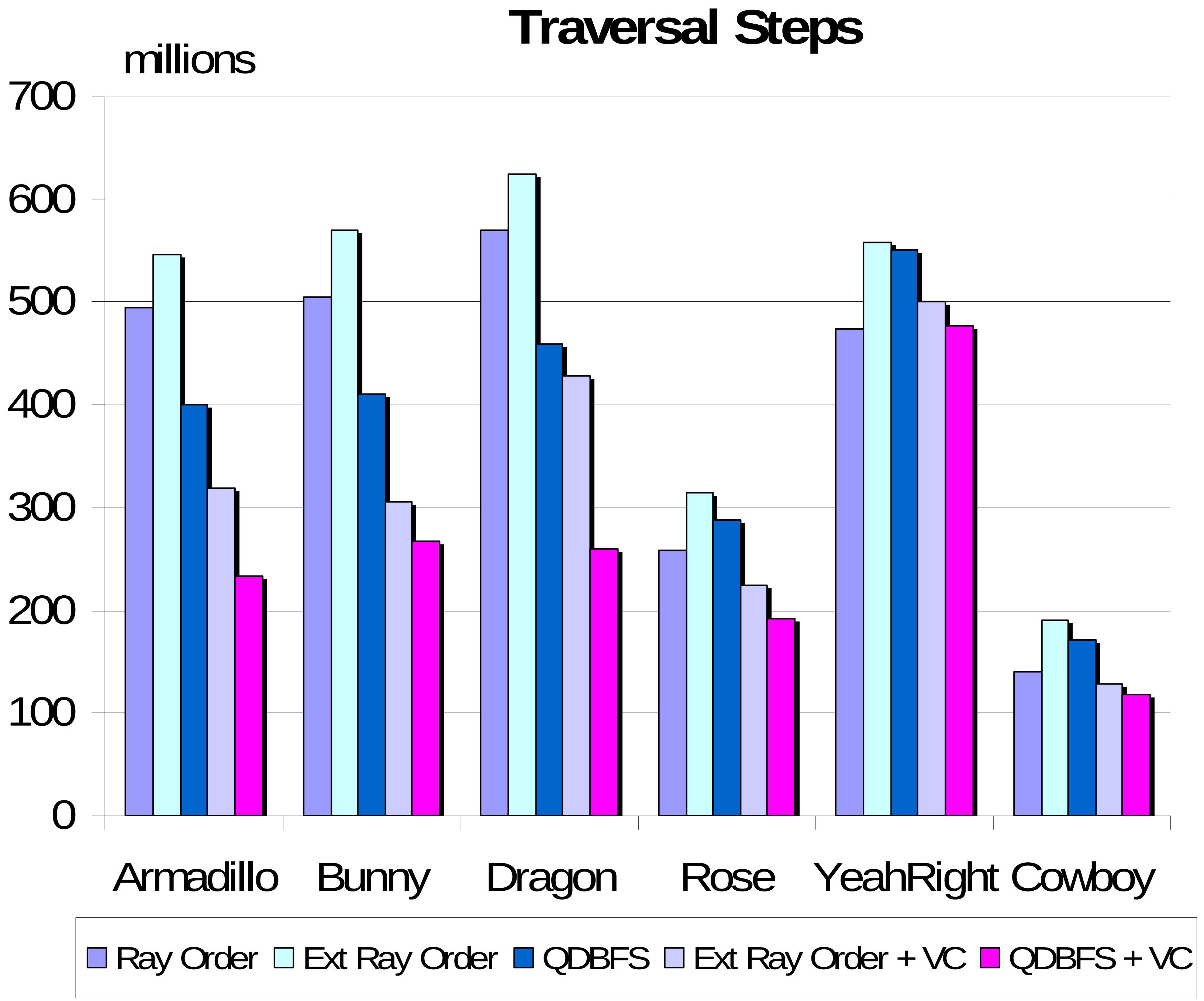


Armadillo	346 k
Bunny	70 k
Dragon	871 k
Rose	326 k
Yeah Right	119 k
Cowboy	7 k


Run-Time
seconds
18
16
14
12
10
8
6
4
2
0
Armadillo
Bunny
Dragon
Rose
YeahRight
Cowboy
Ray Order
Ext Ray Order
QDBFS
Ext Ray Order + VC
QDBFS + VC

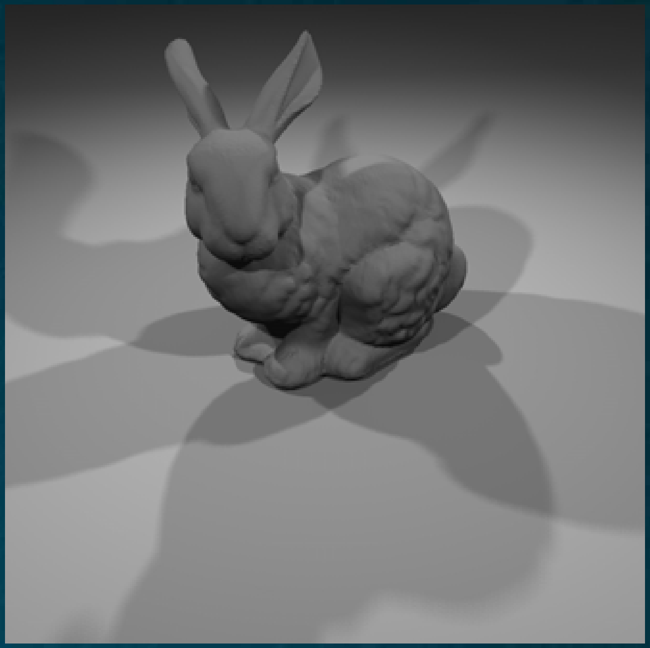


Triangle Intersections
millions
250
200
150
100
50
0
Armadillo
Bunny
Dragon
Rose
YeahRight
Cowboy
Ray Order
Ext Ray Order
QDBFS
Ext Ray Order + VC
QDBFS + VC



Traversal Steps
millions
700
600
500
400
300
200
100
0
Armadillo
Bunny
Dragon
Rose
YeahRight
Cowboy
Ray Order
Ext Ray Order
QDBFS
Ext Ray Order + VC
QDBFS + VC

Simple shadows should be cheap



Simple

Complex

- Our approach: compute cheaper shadows for models that have simple (coherent) shadows

Yeah Right failure case

Baseline

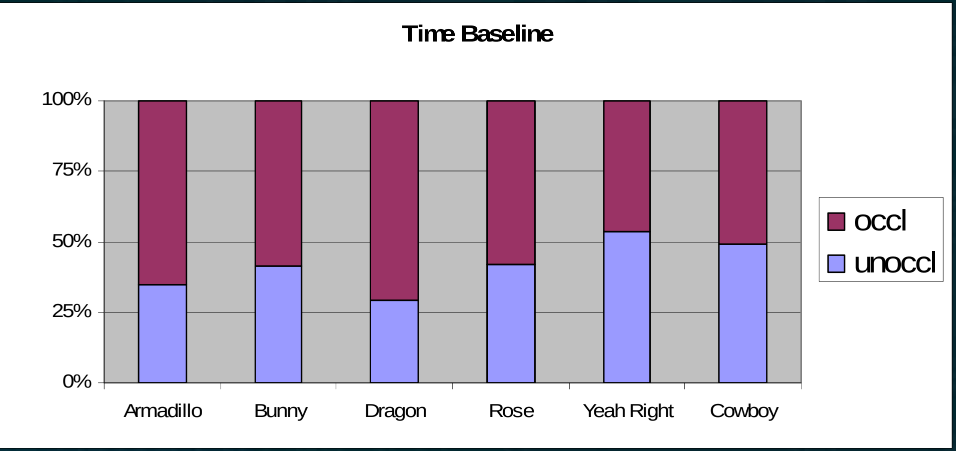
Our approach

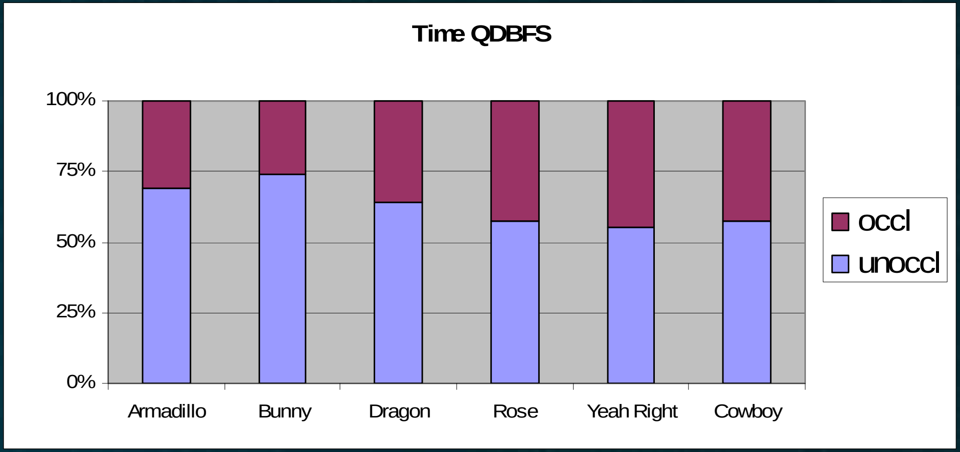
- Shadow has little coherency
- Shadow plagued by silhouette

New bottleneck: Unoccluded shadow rays

Two types of shadow rays

1. occluded - hits some object
2. unoccluded - hits no objects





Extensions

- This work: controlled environment

- Future work
  - Realistic object configurations
  - Realistic lighting configurations

Summary

- Volumetric occluders provide an opportunity to accel. shadow rays
  - Two new kd-tree traversal algorithms
- Up to 50% reduction in time spent on shadow rays
  - In failure cases, performance degradation is graceful
- Produces identical images
- Applies to *any* type of query on binary visibility

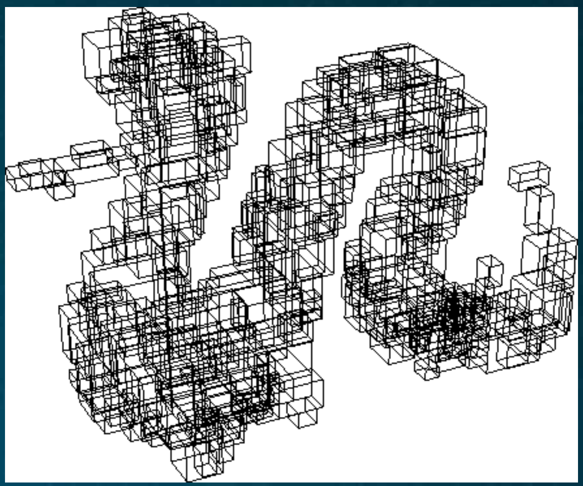
Acknowledgements

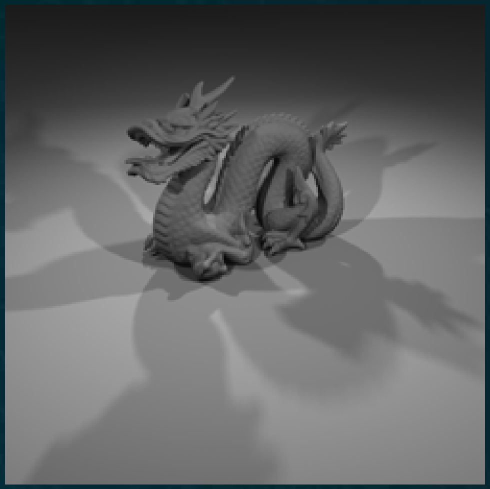
- Bill Mark
- Graphics and Parallel Systems Lab (UT Austin)
- Modelers, especially Techland
- Intel Corporation

Finally… free stuff!

SpiderMind System Library available

http://triangle.csres.utexas.edu/gps/downloads/





Thank You!

Previous Work

Create volumetric occluders using kd-tree nodes

- Inexpensive proxy in an accel. structure
  - Wald et al. 2004 – massive models
  - Yoon et al. 2006 – R-LOD
  - Lacewell et al. 2008 – transmissive aggregate geom.

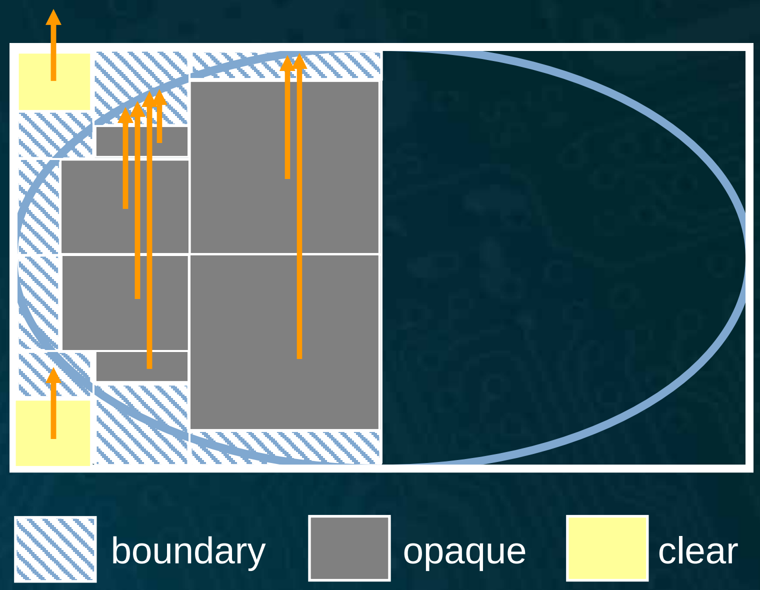
- Exact volumetric data (closely related)
  - Woo and Amanatides 1990 – voxel occlusion testing
  - Schaufler et al. 2000 – occluder fusion
  - Reshetov et al. 2005 – MLRTA

Classifying kd-nodes

For a given leaf node In the kd-tree:

- If the node has geometry, we are done, it is a BOUNDARY node
- If the node is empty:
    - Cast a test ray, origin at node center, any direction
    - If the test ray hits a back-facing polygon
        - Node is inside mesh, node is OPAQUE
    - If the test ray hits a front-facing polygon or nothing
        - Node is outside mesh, node is CLEAR

Illustration of classification



- Manifold mesh guarantees results are well-defined

Size of the deferment list

- Increase in size usually led to increased performance
- Best performance at size 512 (!)
- Why is this the best policy?
  - Theory: it maximizes # of volumetric occlusions
- VC no longer affects # of triangle isects

Another look: shadow ray rate

- In units of megarays / sec
- Up to a 2x improvement
- Worst case still OK
- Single-ray within 2%

Scene	base	vol occ
Armadillo	7.2	13.1
Bunny	6.4	10.7
Dragon	6.0	12.2
Rose	11.6	15.0
Yeah Right	6.8	6.8
Cowboy	16.2	17.9

kd-Node to bounding box lookup table

- Volumetric occluders are added to the cache when they are hit during traversal
- Problem: the bounds of the volumetric occluder are not known at traversal time
- Solution: precompute the lookup table as a preprocess, use it to lookup the bounds at run-time

Cowboy failure case

- Volumetric occluders perform well when:
  - Internal volume is large and expansive
  - Mesh tessellation rate is high
- For Cowboy, the ratio is poor
  - Internal volume is too small compared to tessellation rate

Scene	v.o. SA / geom SA
Armadillo	45
Bunny	18
Dragon	36
Rose	11
Yeah Right	9.6
Cowboy	9.6