GPU Ray-tracing using Irregular Grids

Arsène Pérard-Gayot, Javor Kalojanov, Philipp Slusallek

April 23, 2017

Universität des Saarlandes
Intel Visual Computing Institute
Deutsches Forschungszentrum für Künstliche Intelligenz GmbH

Introduction
- Ray Tracing with Grids
- Challenges

Irregular Grids
- Construction (Part I)
- Traversal
- Construction (Part II)

Results

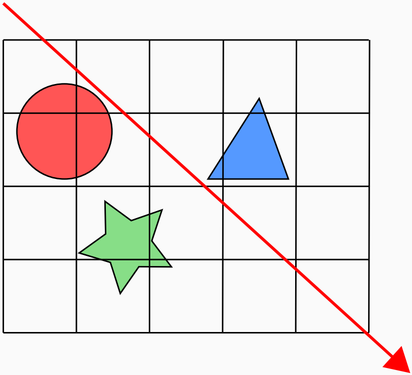
Introduction

Pros

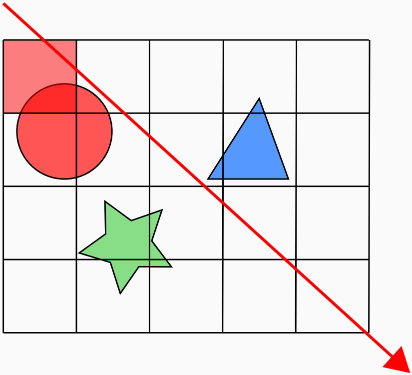
- Very fast parallel construction
- Ordered traversal, early exit
- Stackless traversal

Cons

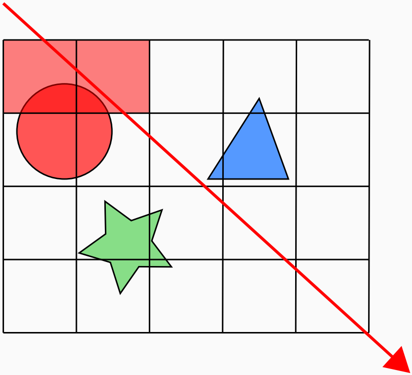
- Empty space skipping: *Teapot in the Stadium*
- Cannot minimize both intersections and traversal steps

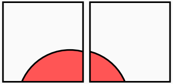


Introduction: Uniform Grid

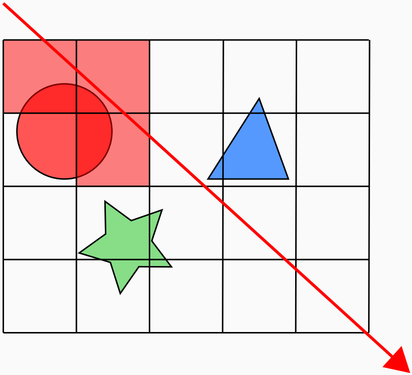


Introduction: Uniform Grid

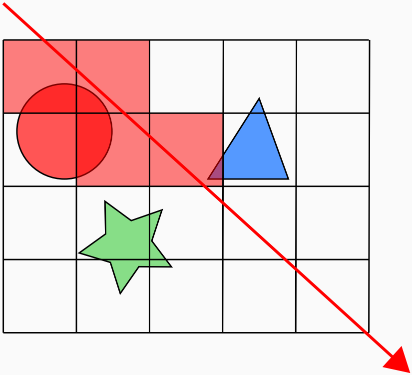


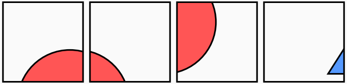


Introduction: Uniform Grid

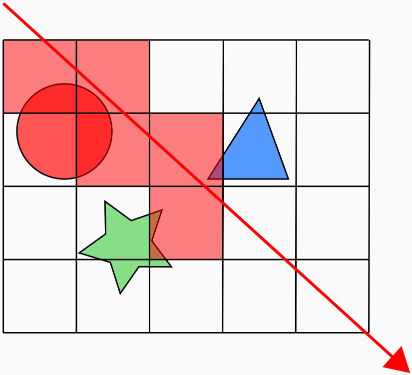


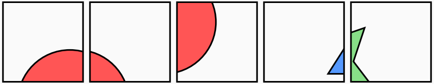
Introduction: Uniform Grid



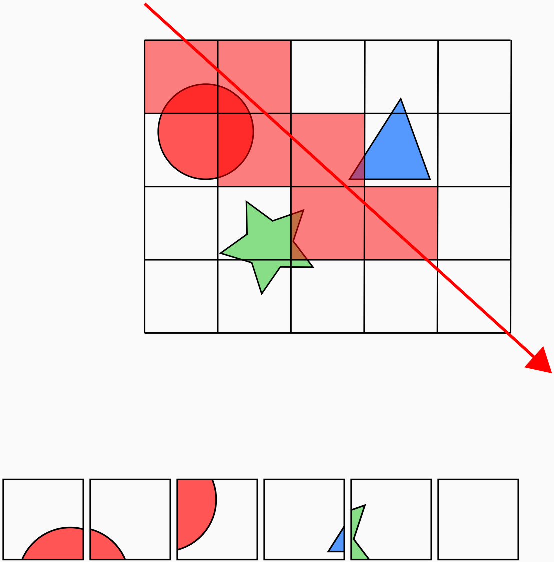


Introduction: Uniform Grid

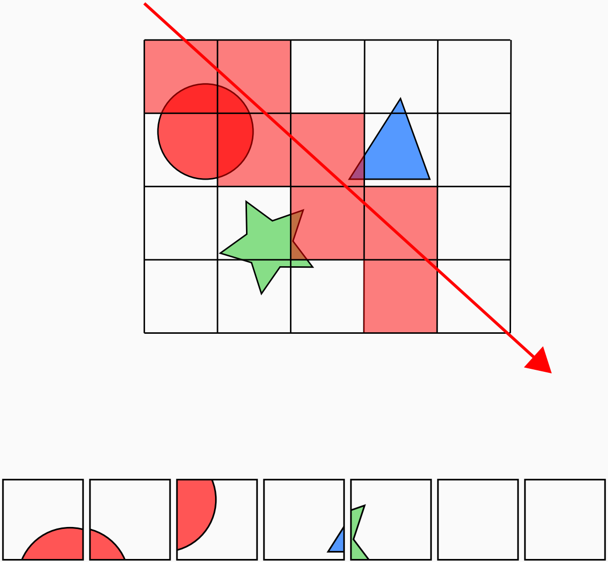




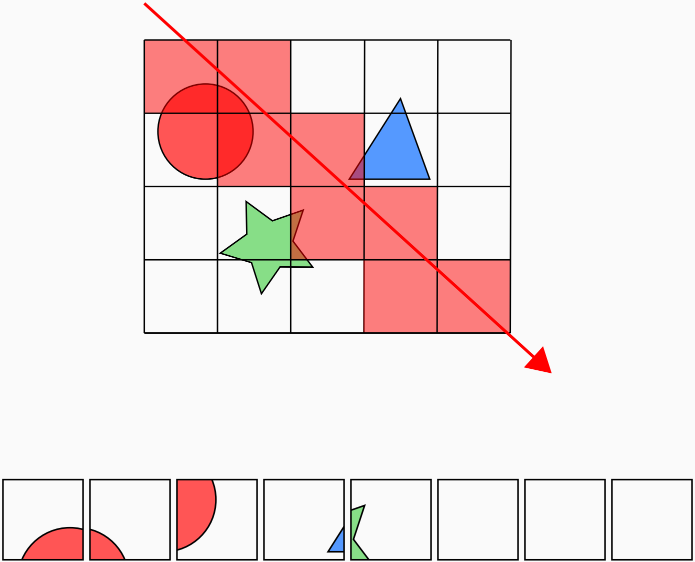
Introduction: Uniform Grid



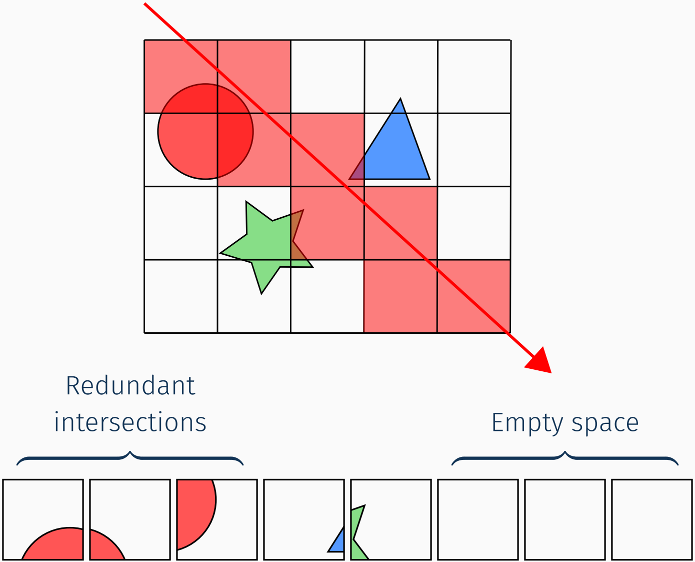
Introduction: Uniform Grid



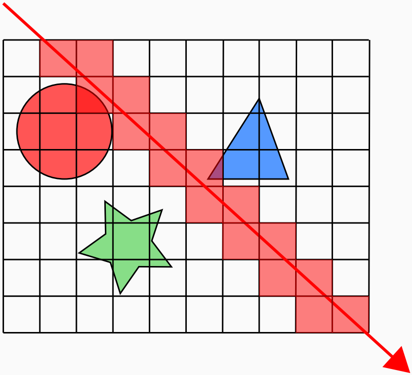
Introduction: Uniform Grid



Introduction: Uniform Grid

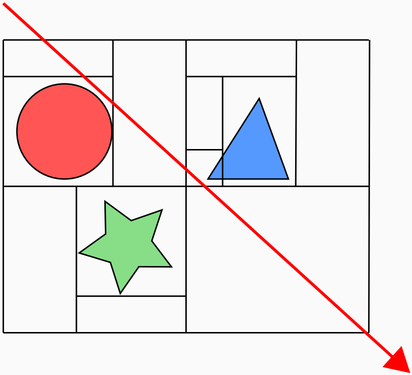


Introduction: Uniform Grid



Increasing resolution

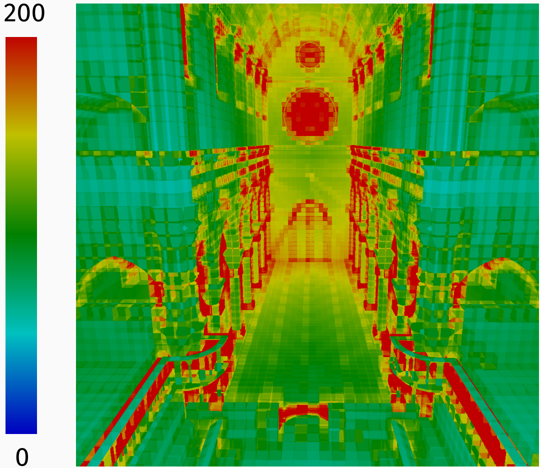
- Fewer intersections
- More traversal steps



Idea: Remove regularity

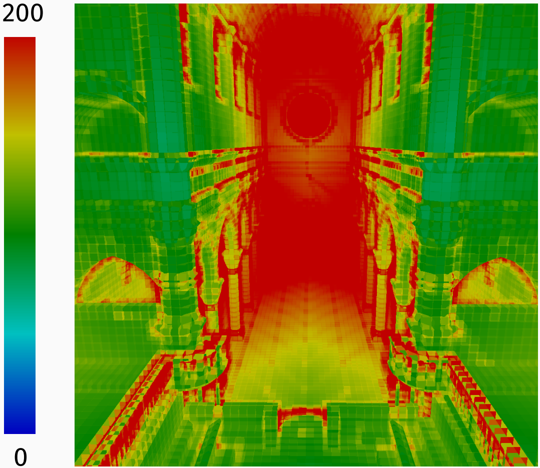
- Start with a dense subdivision
- Optimize cell shape to minimize traversal cost

Uniform Grid: Low Resolution



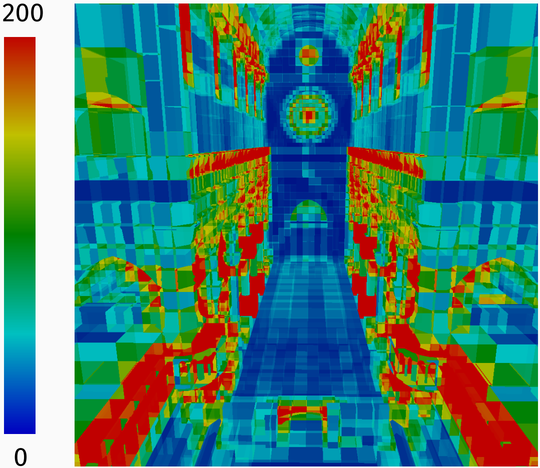
Traversal steps + Intersections

Uniform Grid: Medium Resolution



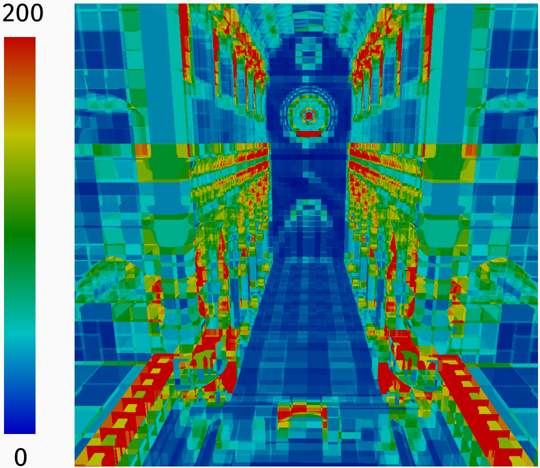
Traversal steps + Intersections

Irregular Grid: Low Resolution



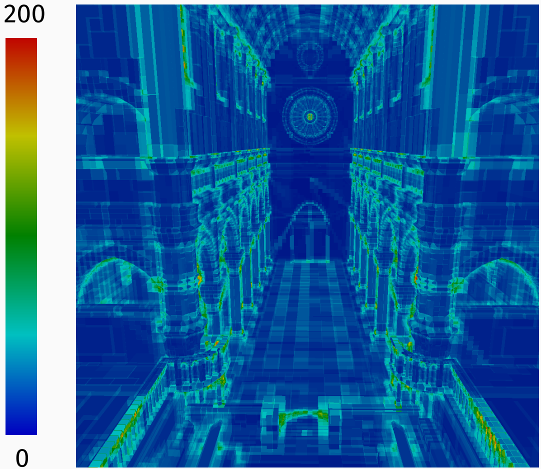
Traversal steps + Intersections

Irregular Grid: Medium Resolution



Traversal steps + Intersections

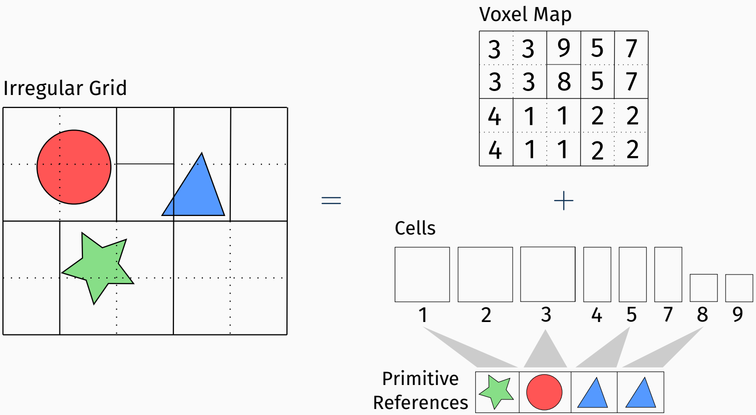
Irregular Grid: High Resolution



Traversal steps + Intersections

Irregular Grids

Data Structure

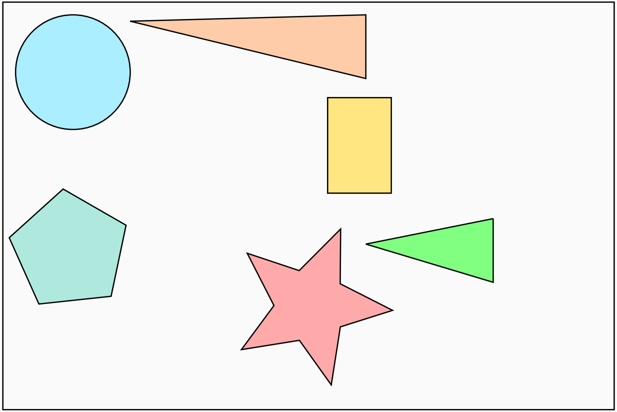


Construction (Part I)

Initialization

- Initial grid
- Two-level construction:
  1. A *coarse* uniform grid
  2. An octree in each of the grid cells
- Adaptive: More effort where the geometry is complex
- Dense: Up to 2^{15} resolution in each second-level cell

Initialization



Construction (Part I)

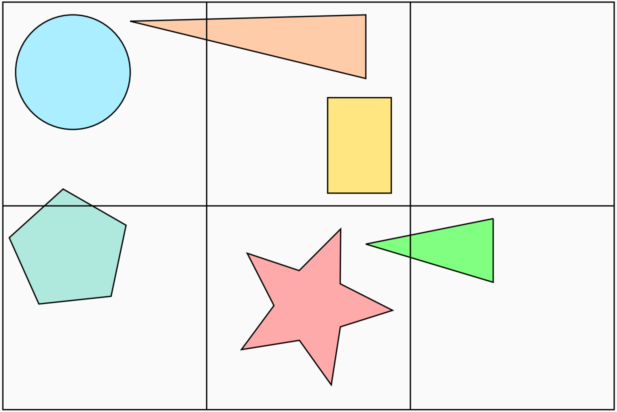
Initialization

- User-defined λ_1 controls top-level resolution
- With scene volume V and number of objects N [Cle+83]:

$$R_{\{x,y,z\}} = d_{\{x,y,z\}} \sqrt[3]{\frac{\lambda_1 N}{V}}$$

- Tries to make cells cubic

Initialization



Construction (Part I)

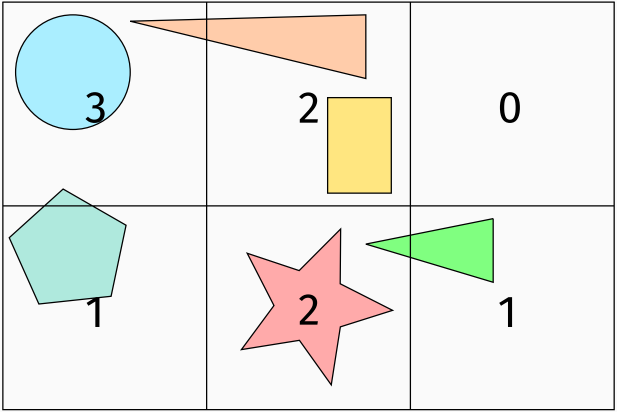
Initialization

- Octree depth computed independently in each cell
- Same formula, but: λ_2, local number of objects & volume
- Clamp resolution to a power of two:

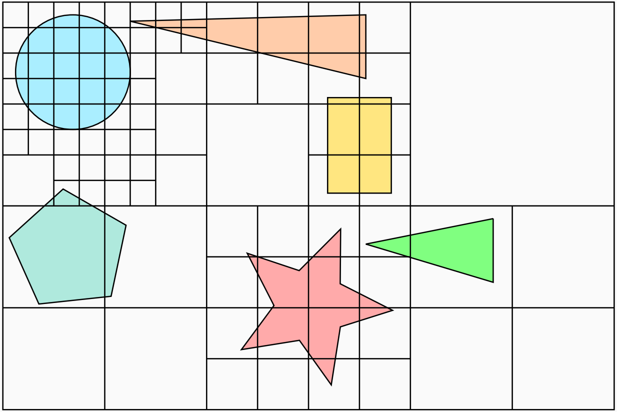
$$D = \lceil log_2(max(R_x, R_y, R_z)) \rceil$$

- Compact: only $log_2(log_2(R_{max}))$ bits needed
  - 4 bits = max. resolution of $2^{15} \times 2^{15} \times 2^{15}$

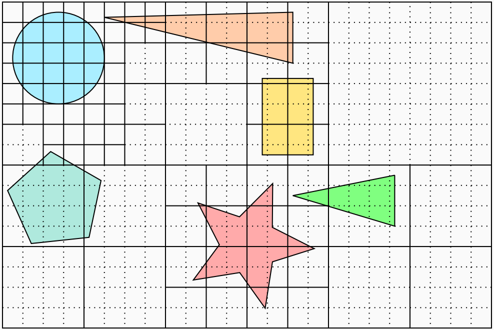
Initialization



Initialization



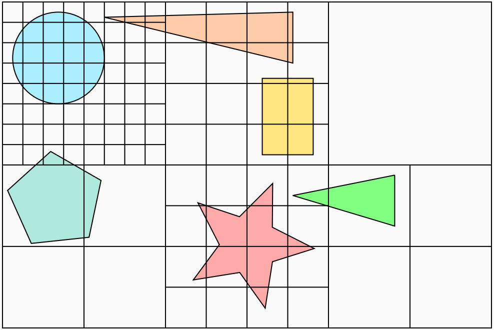
Construction (Part I): Virtual Grid



Property

Cells are aligned on a virtual grid of resolution $R_{x,y,z}\,2^{D}$

Construction (Part I): Voxel Map



Voxel map as a two level grid

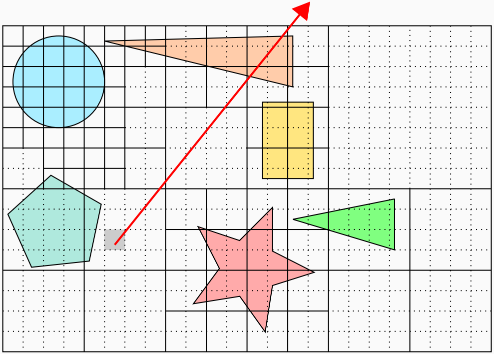
Memory efficient/Fast lookup

Traversal

- The data structure is not optimal
- But it can already be used for traversal

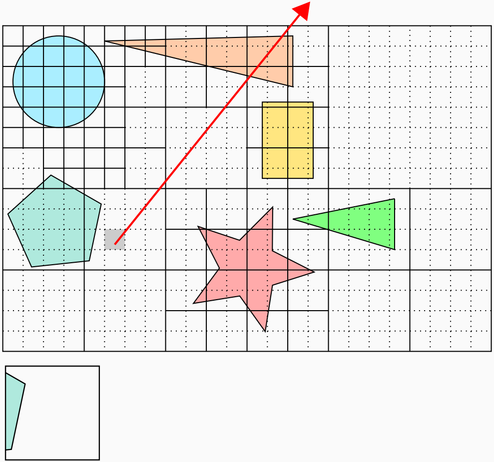
Ideas

- Maintain position on the virtual grid
- Recompute increment along the ray at each step



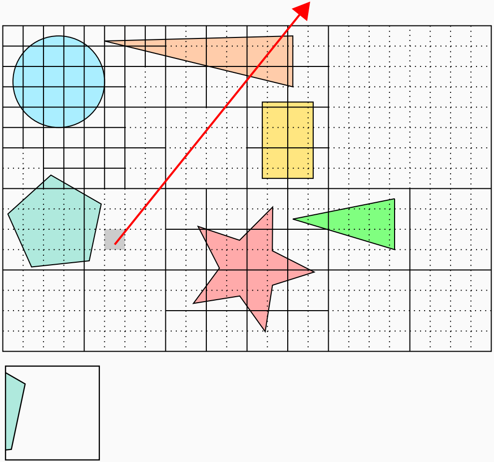
1. Locate ray origin
2. Loop
   - 2.1 Intersect primitives
   - 2.2 Exit if hit is within cell
   - 2.3 Locate exit point
   - 2.4 Move to next cell

Interlude: Traversal



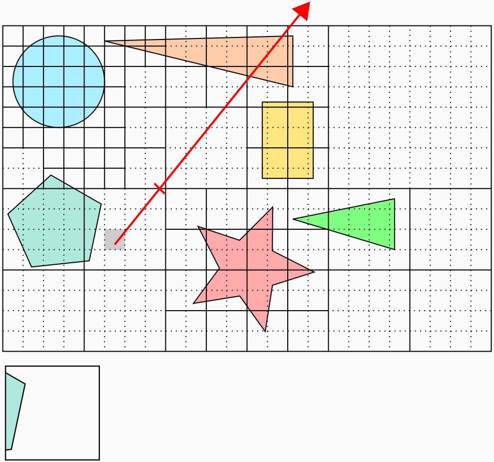
1. Locate ray origin
2. Loop
   - 2.1 Intersect primitives
   - 2.2 Exit if hit is within cell
   - 2.3 Locate exit point
   - 2.4 Move to next cell

Interlude: Traversal

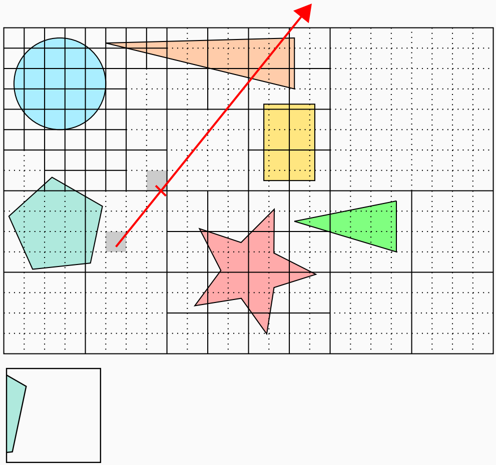


1. Locate ray origin
2. Loop
   - 2.1 Intersect primitives
   - 2.2 Exit if hit is within cell
   - 2.3 Locate exit point
   - 2.4 Move to next cell

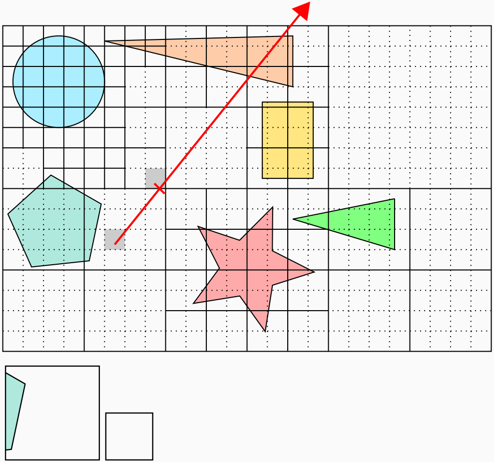
Interlude: Traversal



1. Locate ray origin
2. Loop
   - 2.1 Intersect primitives
   - 2.2 Exit if hit is within cell
   - 2.3 Locate exit point
   - 2.4 Move to next cell

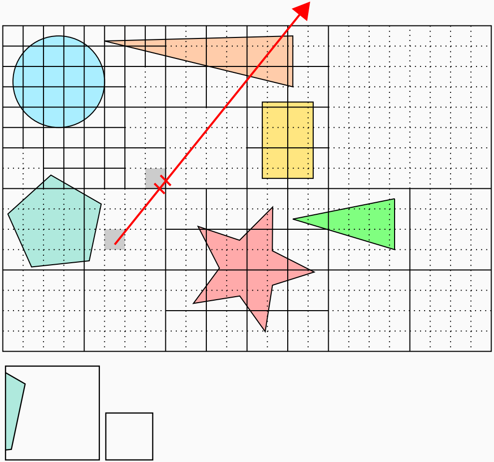


1. Locate ray origin
2. Loop
   2.1 Intersect primitives
   2.2 Exit if hit is within cell
   2.3 Locate exit point
   2.4 Move to next cell



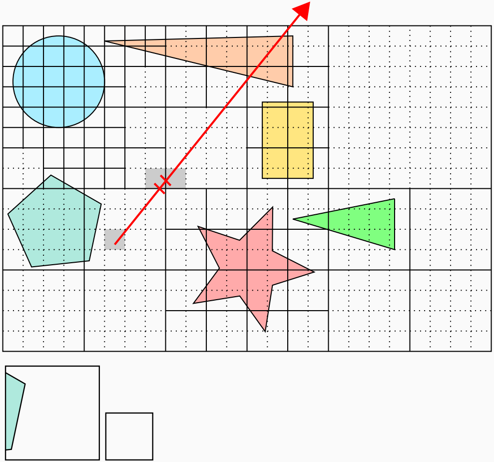
1. Locate ray origin
2. Loop
    2.1 Intersect primitives
    2.2 Exit if hit is within cell
    2.3 Locate exit point
    2.4 Move to next cell

Interlude: Traversal



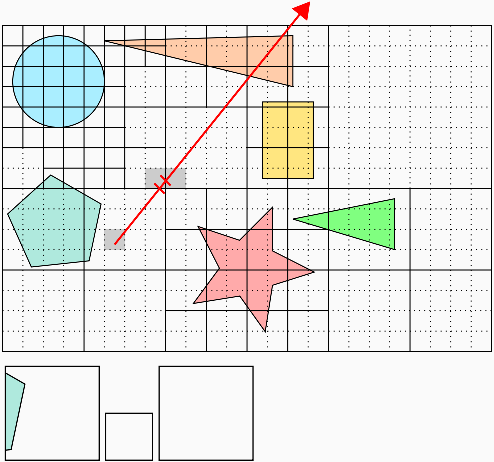
1. Locate ray origin
2. Loop
   2.1 Intersect primitives
   2.2 Exit if hit is within cell
   2.3 Locate exit point
   2.4 Move to next cell

Interlude: Traversal



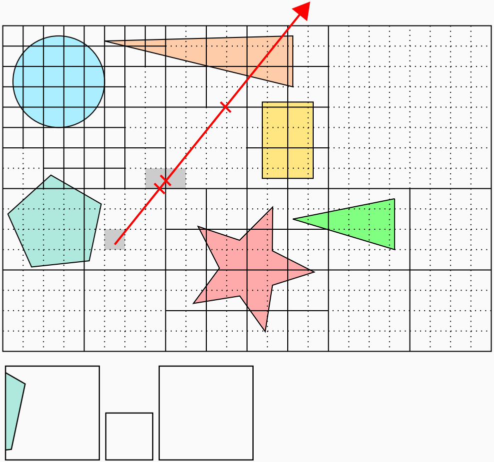
1. Locate ray origin
2. Loop
   2.1 Intersect primitives
   2.2 Exit if hit is within cell
   2.3 Locate exit point
   2.4 Move to next cell

Interlude: Traversal



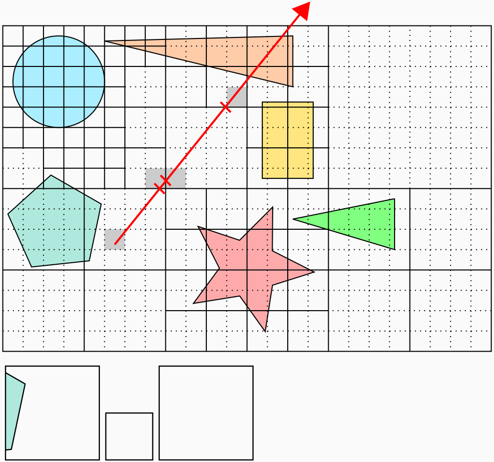
1. Locate ray origin
2. Loop
   - 2.1 Intersect primitives
   - 2.2 Exit if hit is within cell
   - 2.3 Locate exit point
   - 2.4 Move to next cell

Interlude: Traversal

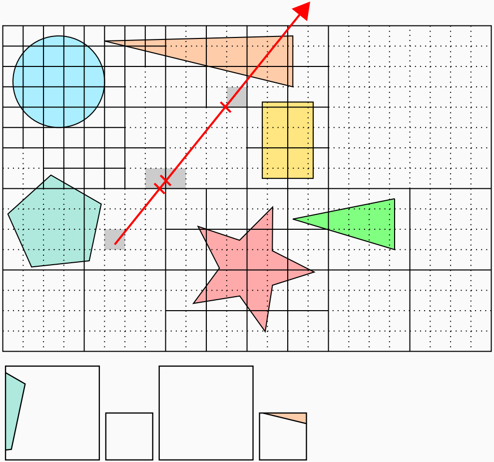


1. Locate ray origin
2. Loop
   2.1 Intersect primitives
   2.2 Exit if hit is within cell
   2.3 Locate exit point
   2.4 Move to next cell

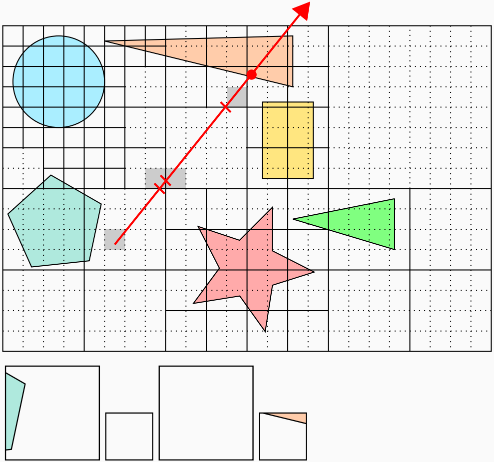
Interlude: Traversal



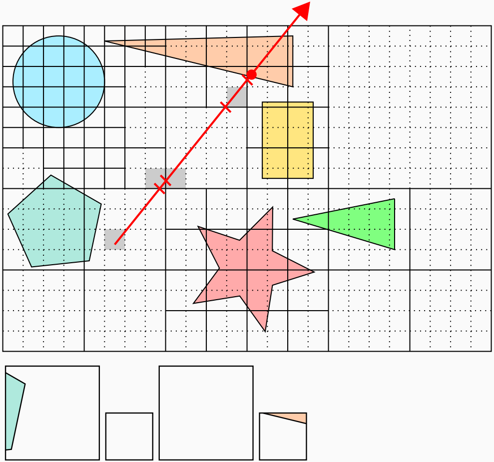
1. Locate ray origin
2. Loop
   2.1 Intersect primitives
   2.2 Exit if hit is within cell
   2.3 Locate exit point
   2.4 Move to next cell



1. Locate ray origin
2. Loop
   - 2.1 Intersect primitives
   - 2.2 Exit if hit is within cell
   - 2.3 Locate exit point
   - 2.4 Move to next cell

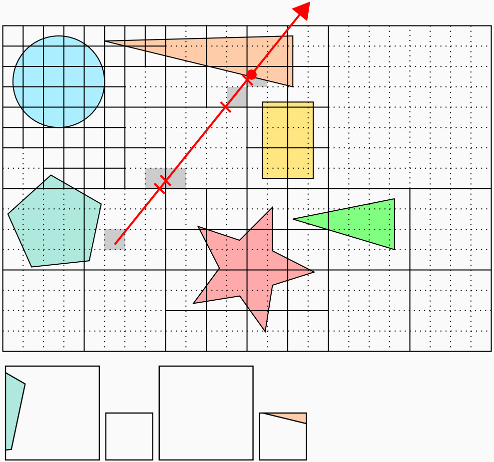


1. Locate ray origin
2. Loop
   2.1 Intersect primitives
   2.2 Exit if hit is within cell
   2.3 Locate exit point
   2.4 Move to next cell

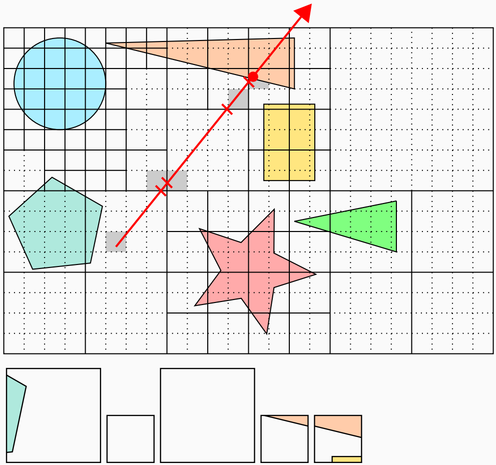


1. Locate ray origin
2. Loop
   2.1 Intersect primitives
   2.2 Exit if hit is within cell
   2.3 Locate exit point
   2.4 Move to next cell

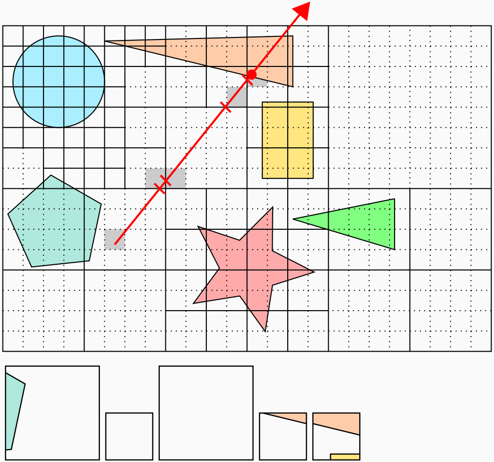
Interlude: Traversal



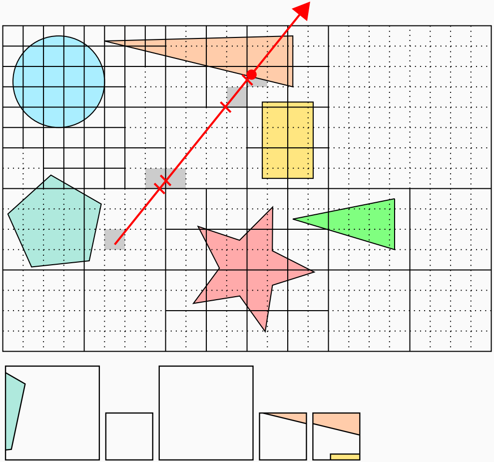
1. Locate ray origin
2. Loop
   - 2.1 Intersect primitives
   - 2.2 Exit if hit is within cell
   - 2.3 Locate exit point
   - 2.4 Move to next cell



1. Locate ray origin
2. Loop
    - 2.1 Intersect primitives
    - 2.2 Exit if hit is within cell
    - 2.3 Locate exit point
    - 2.4 Move to next cell



1. Locate ray origin
2. Loop
   2.1 Intersect primitives
   2.2 Exit if hit is within cell
   2.3 Locate exit point
   2.4 Move to next cell



1. Locate ray origin
2. Loop
    - 2.1 Intersect primitives
    - 2.2 Exit if hit is within cell
    - 2.3 Locate exit point
    - 2.4 Move to next cell

Contruction (Part II)

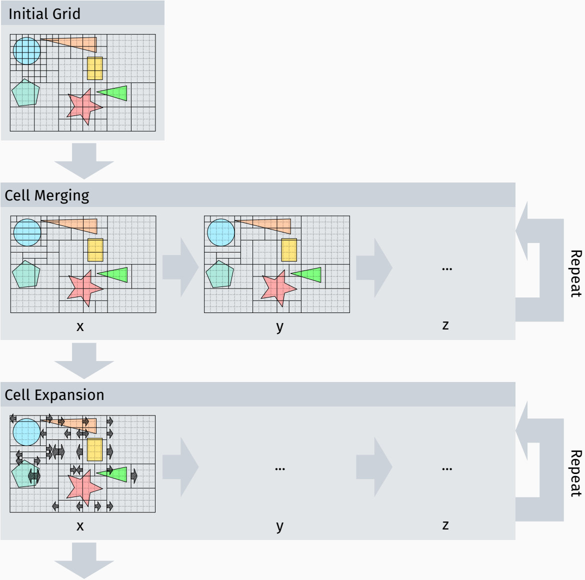
Traversal Performance

- Poor empty space skipping $\Longrightarrow$ memory latency
- Redundant intersections $\Longrightarrow$ instr./memory latency

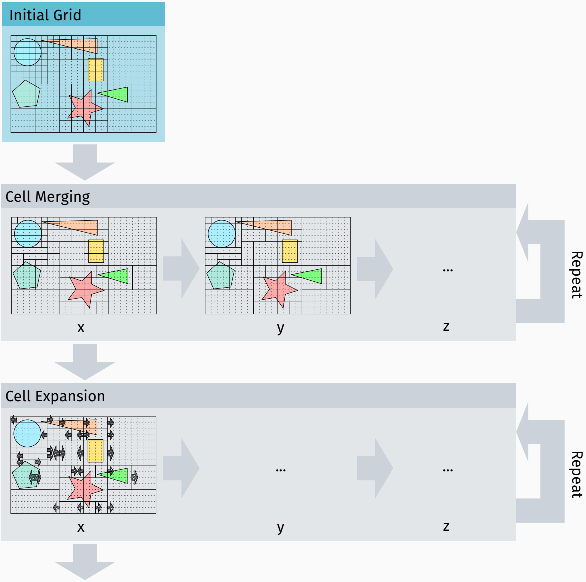
Cell Merging and Expansion

- Local (greedy) optimizations
- Examine cells and their neighborhoods
- Keep optimizations simple and parallelizable

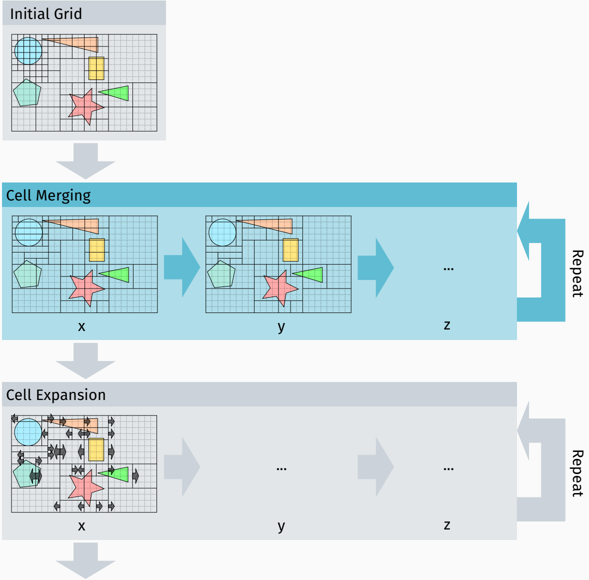
CONTRUCTION (PART II): OPTIMIZATION PASSES



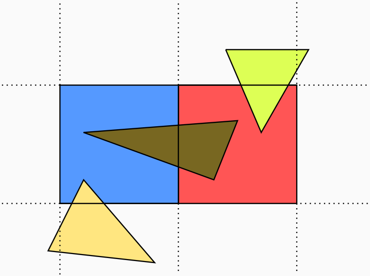
Contruction (Part II): Optimization Passes



Contruction (Part II): Optimization Passes



Contruction (Part II): Cell Merging



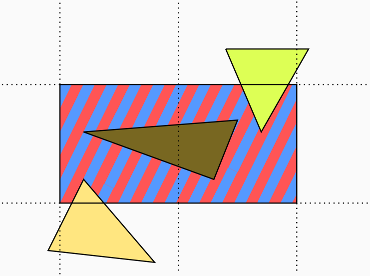
Cell Merging

- Merge each cell with its neighbor if the SAH decreases:

$$|\mathcal{R}(A)|\,\mathcal{SA}(A) + |\mathcal{R}(B)|\,\mathcal{SA}(B) \geq |\mathcal{R}(A \cup B)|\,\mathcal{SA}(A \cup B) - \mathcal{C}_t$$

- For empty and non-empty cells

Contruction (Part II): Cell Merging



Limitations

- Only consider the union of 2 aligned cells
- Union must be a box

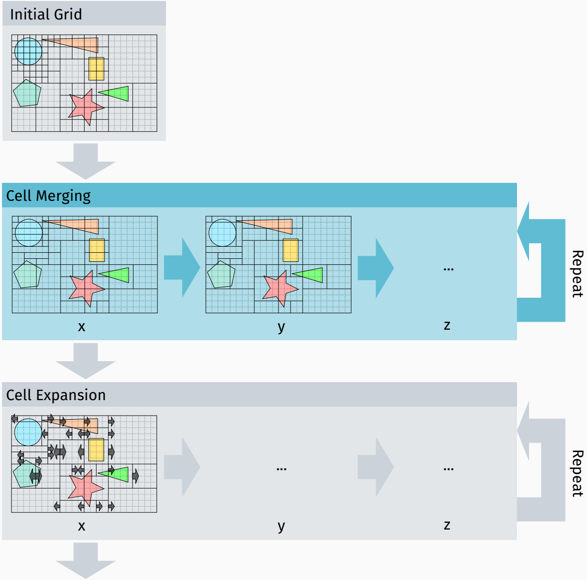
Stopping criterion

- Keep merging until:

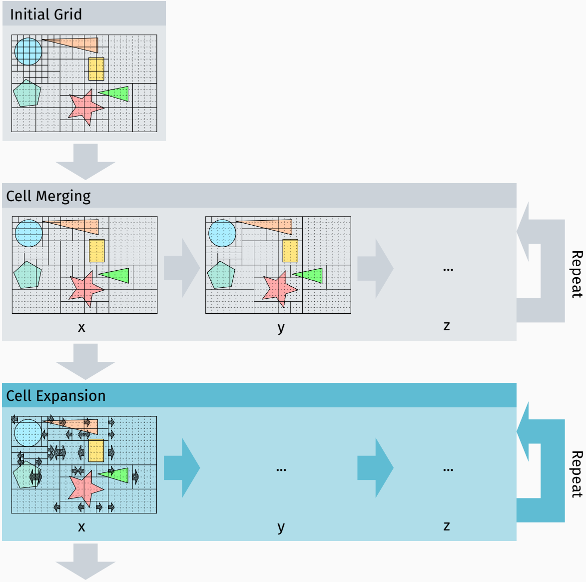
$$N_{after} \geq \alpha N_{before}$$

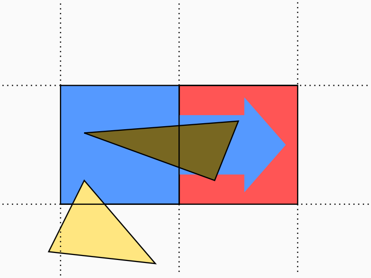
  - N_{after}/N_{before}: number of cells after/before merging
  - $\alpha = 0.995$

Contruction (Part II): Optimization Passes



Contruction (Part II): Optimization Passes



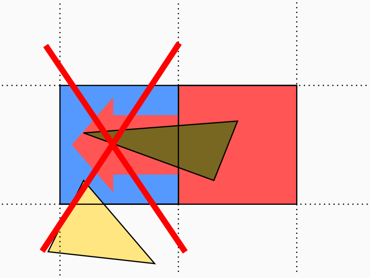


Cell Expansion

- Expand the exit boundaries of the cells
- Must maintain correctness of traversal:

$$\mathcal{R}(B) \subset \mathcal{R}(A)$$

Contruction (Part II): Cell Expansion

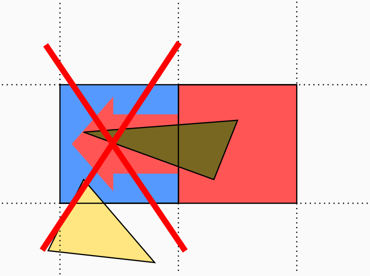


Cell Expansion

- Expand the exit boundaries of the cells
- Must maintain correctness of traversal:

$$\mathcal{R}(A) \not\subset \mathcal{R}(B)$$

Contruction (Part II): Cell Expansion

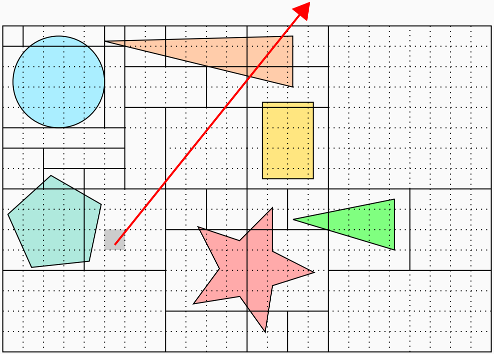


Limitations

- Must examine every neighbor on the box face
- Binary decision, no partial expansion

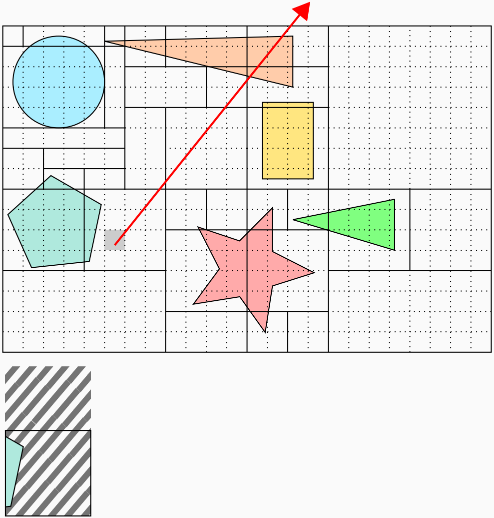
Stopping criterion

- Fixed number of expansion passes:
  - 3 for static scenes,
  - 1 for dynamic scenes.



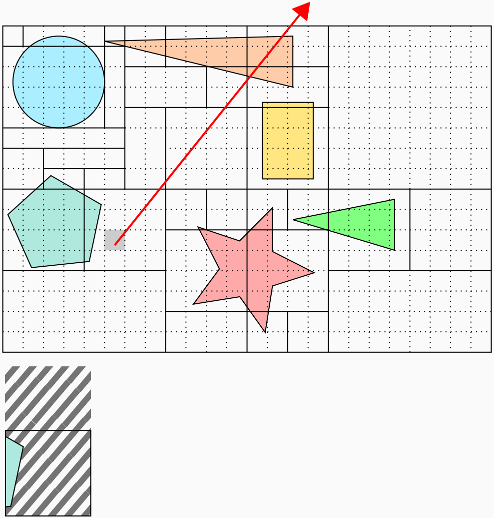
1. Locate ray origin
2. Loop
   - 2.1 Intersect primitives
   - 2.2 Exit if hit is within cell
   - 2.3 Locate exit point
   - 2.4 Move to next cell

Contruction (Part II): Impact on Traversal



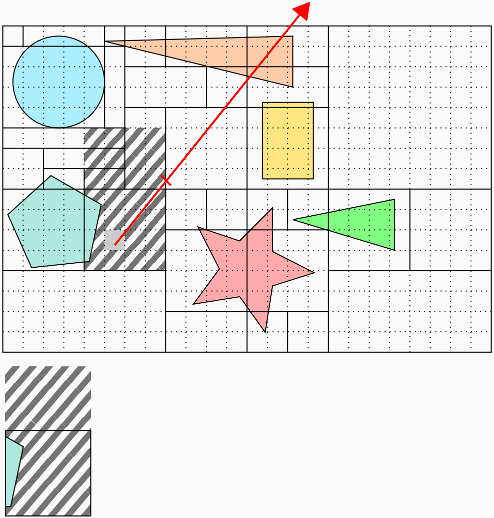
1. Locate ray origin
2. Loop
   - 2.1 Intersect primitives
   - 2.2 Exit if hit is within cell
   - 2.3 Locate exit point
   - 2.4 Move to next cell

CONTRUCTION (PART II): IMPACT ON TRAVERSAL



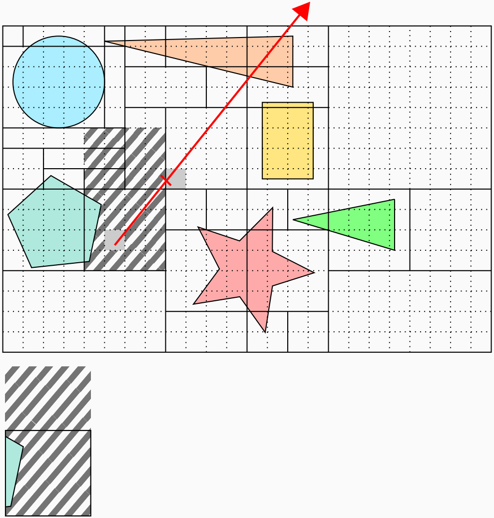
1. Locate ray origin
2. Loop
   - 2.1 Intersect primitives
   - 2.2 Exit if hit is within cell
   - 2.3 Locate exit point
   - 2.4 Move to next cell

Contruction (Part II): Impact on Traversal



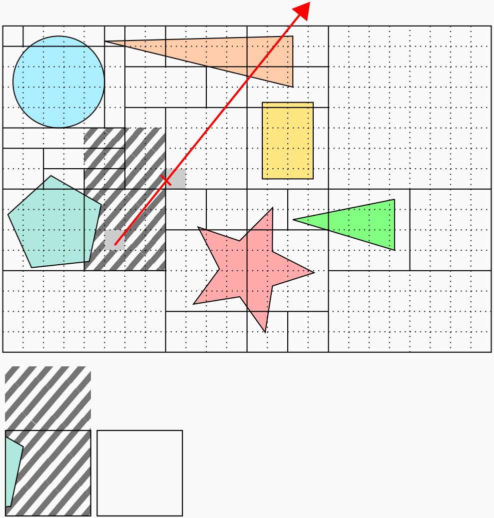
1. Locate ray origin
2. Loop
    2.1 Intersect primitives
    2.2 Exit if hit is within cell
    2.3 Locate exit point
    2.4 Move to next cell

Contruction (Part II): Impact on Traversal



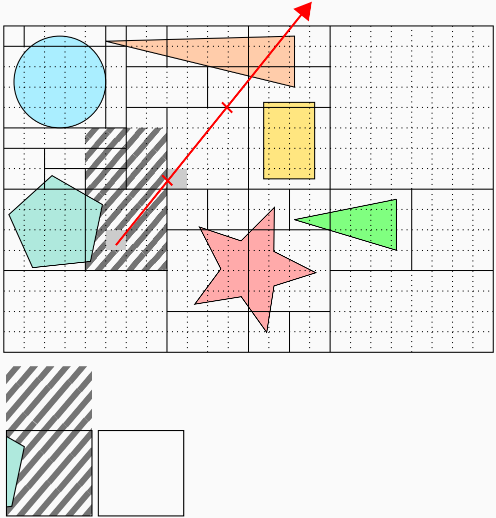
1. Locate ray origin
2. Loop
   2.1 Intersect primitives
   2.2 Exit if hit is within cell
   2.3 Locate exit point
   2.4 Move to next cell

Contruction (Part II): Impact on Traversal



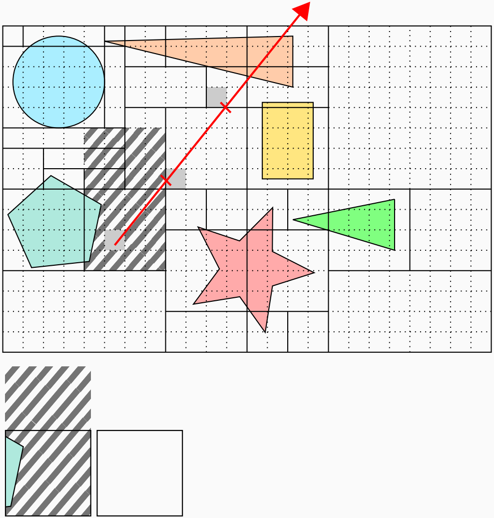
1. Locate ray origin
2. Loop
   2.1 Intersect primitives
   2.2 Exit if hit is within cell
   2.3 Locate exit point
   2.4 Move to next cell

Contruction (Part II): Impact on Traversal



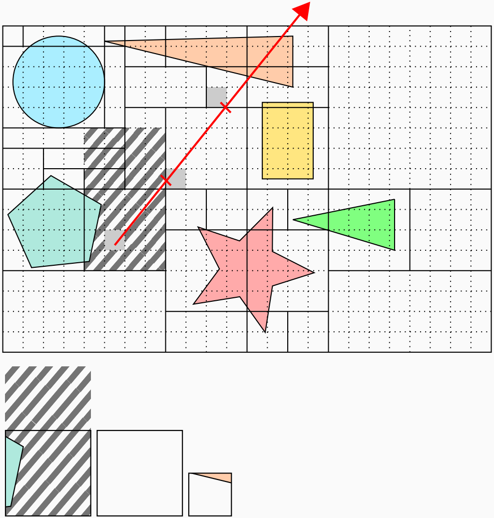
1. Locate ray origin
2. Loop
   2.1 Intersect primitives
   2.2 Exit if hit is within cell
   2.3 Locate exit point
   2.4 Move to next cell

Contruction (Part II): Impact on Traversal



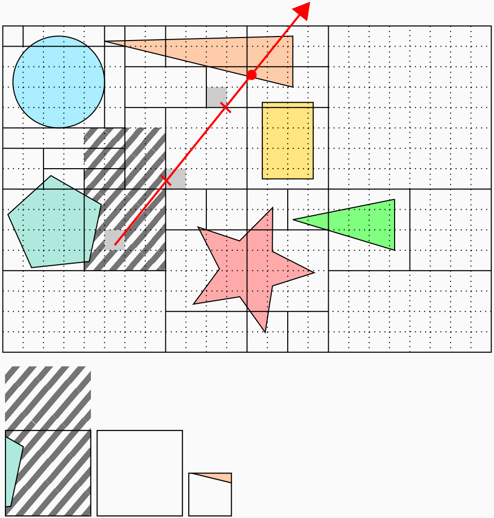
1. Locate ray origin
2. Loop
   - 2.1 Intersect primitives
   - 2.2 Exit if hit is within cell
   - 2.3 Locate exit point
   - 2.4 Move to next cell

Contruction (Part II): Impact on Traversal



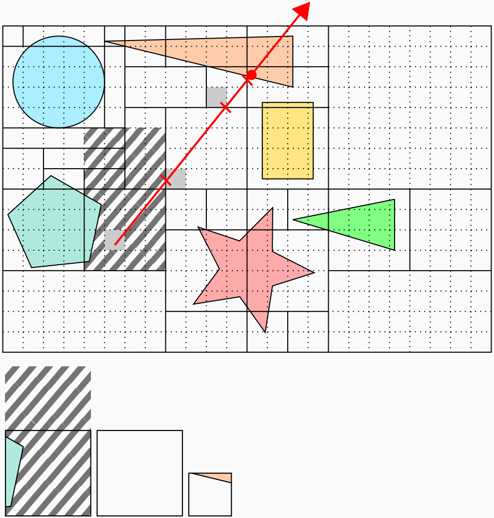
1. Locate ray origin
2. Loop
   - 2.1 Intersect primitives
   - 2.2 Exit if hit is within cell
   - 2.3 Locate exit point
   - 2.4 Move to next cell

Contruction (Part II): Impact on Traversal



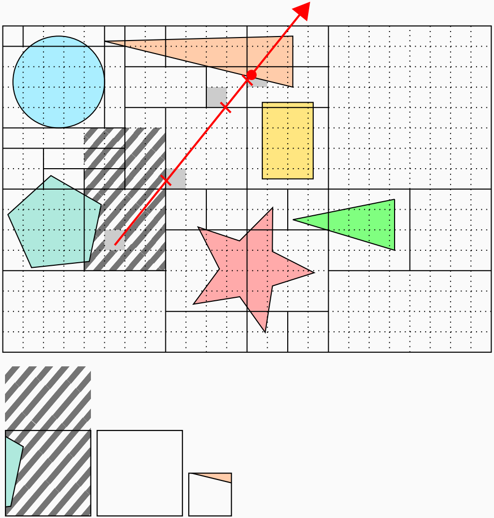
1. Locate ray origin
2. Loop
   2.1 Intersect primitives
   2.2 Exit if hit is within cell
   2.3 Locate exit point
   2.4 Move to next cell

Contruction (Part II): Impact on Traversal



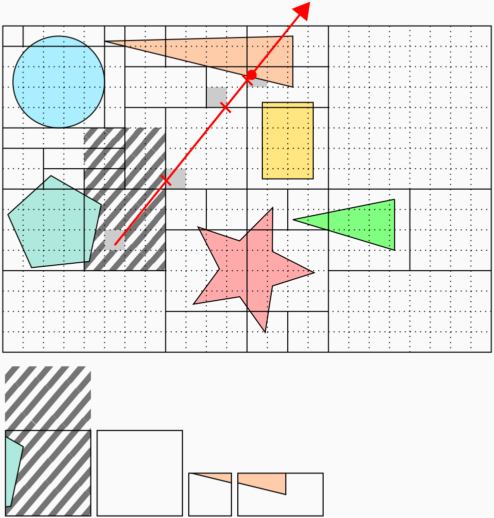
1. Locate ray origin
2. Loop
   2.1 Intersect primitives
   2.2 Exit if hit is within cell
   2.3 Locate exit point
   2.4 Move to next cell

Contruction (Part II): Impact on Traversal



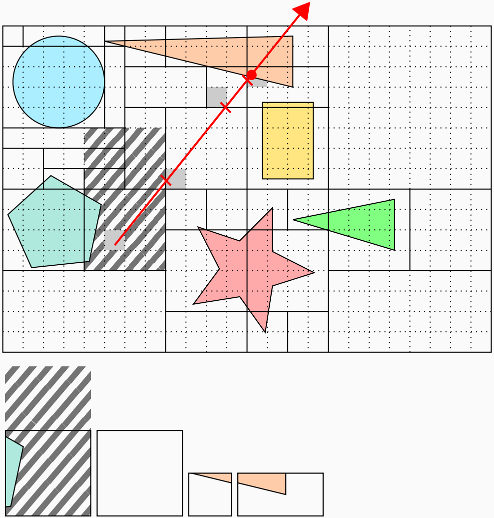
1. Locate ray origin
2. Loop
   - 2.1 Intersect primitives
   - 2.2 Exit if hit is within cell
   - 2.3 Locate exit point
   - 2.4 Move to next cell

Contruction (Part II): Impact on Traversal



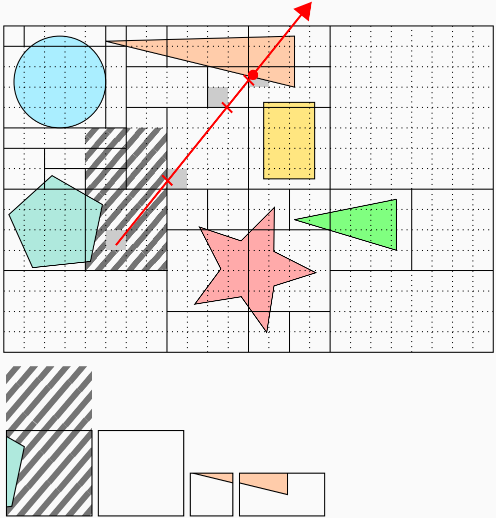
1. Locate ray origin
2. Loop
    2.1 Intersect primitives
    2.2 Exit if hit is within cell
    2.3 Locate exit point
    2.4 Move to next cell

Contruction (Part II): Impact on Traversal



1. Locate ray origin
2. Loop
   - 2.1 Intersect primitives
   - 2.2 Exit if hit is within cell
   - 2.3 Locate exit point
   - 2.4 Move to next cell

Contruction (Part II): Impact on Traversal



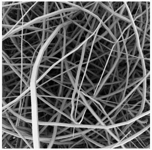
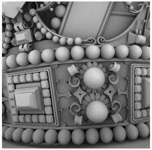
1. Locate ray origin
2. Loop
   - 2.1 Intersect primitives
   - 2.2 Exit if hit is within cell
   - 2.3 Locate exit point
   - 2.4 Move to next cell

Results

Results: Source Code

GPU implementation

- `https://github.com/madmann91/hagrid`
- Parallel construction & traversal
- CUDA implementation
- MIT license

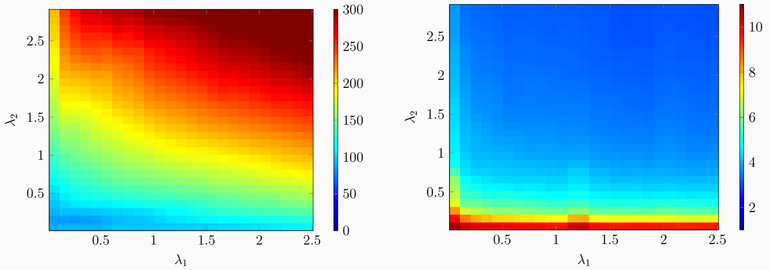
Parameters

- $(\lambda_1, \lambda_2) = (0.12, 2.4)$ for every scene
- Memory footprint $\approx$ SBVH [SFD09]
- Different viewpoints

Results: Static Scenes

Scene	#Tris	Build times (ms)	Primary (MRays/s) SBVH	Primary (MRays/s) Ours	AO (MRays/s) SBVH	AO (MRays/s) Ours	Random (MRays/s) SBVH	Random (MRays/s) Ours
Sponza	262K	26	409 265	653 +60% 473 +78%	270 187	386 +43% 234 +25%	166	274 +65%
Conference	283K	22	583 523	597 +2% 526 +1%	303 326	332 +10% 338 +4%	295	312 +6%
Hairball	2.9M	893	100 79	148 +48% 93 +18%	53 63	69 +30% 61 -3%	19	26 +37%
Crown	3.5M	203	232 181	296 +28% 191 +6%	108 112	120 +11% 125 +12%	221	238 +8%
San Miguel	7.9M	492	227 157	291 +28% 180 +15%	119 125	119 +0% 115 -8%	119	160 +34%

Results: Build Times vs. Traversal Performance



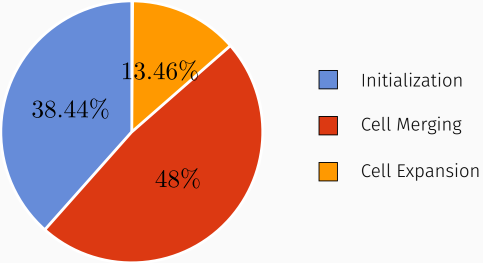
Build Times (ms) Traversal Times (ms)

Lower = Better

Varying parameters for *Crown*

- No local optimum $\neq$ two-level grid
- Increasing density $\implies$ increasing performance

Results: Construction Steps Performance



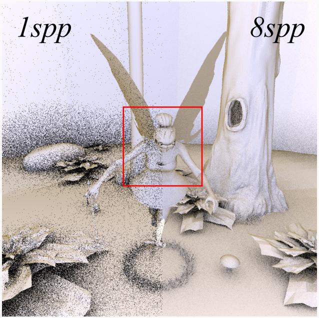
Time spent during construction

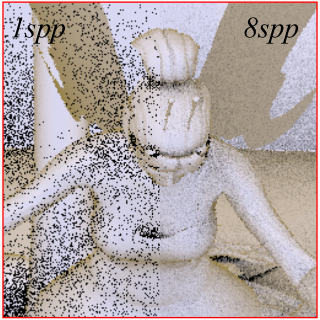
- Average over all static scenes
- Dominated by initialization & merging

Methodology

- Comparison with two-level grids [KBS11]
- Fixed time budget
- Two-level grids: choose optimal resolution
- Irregular grid:
    - Fixed ratio: $\lambda_1 : \lambda_2 = 1 : 8$.
    - Range: $\lambda_1 \in [0.01, 0.3]$, $\lambda_2 \in [0.08, 2.4]$
    - Start at minimum, increase until $T_{build} = 0.5\, T_{budget}$

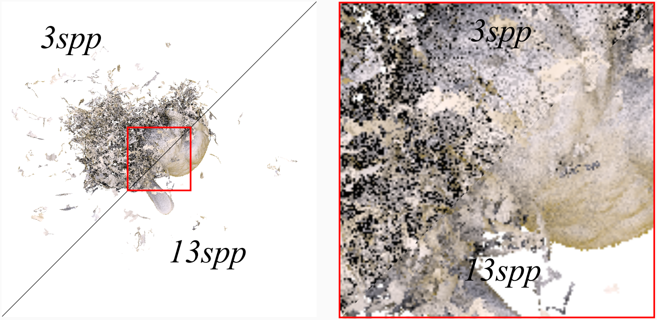
Results: Dynamic Scenes





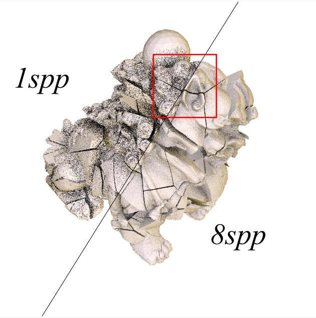
	10FPS (100ms)		20FPS (50ms)		30FPS (33ms)	
	2L Grid	Ours	2L Grid	Ours	2L Grid	Ours
λ_1, λ_2	0.2, 2.0	0.3, 2.4	0.2, 2.0	0.3, 2.4	0.2, 2.0	0.3, 2.4
AO spp	2	20	1	8	0	3

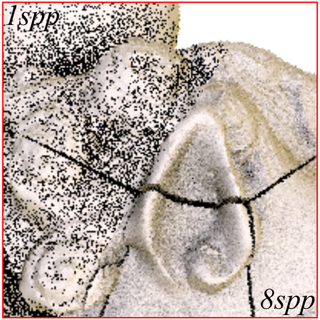
Results: Dynamic Scenes



	10FPS (100ms)		20FPS (50ms)		30FPS (33ms)	
	2L Grid	Ours	2L Grid	Ours	2L Grid	Ours
λ_1, λ_2	0.2, 2.0	0.3, 2.4	0.2, 2.0	0.3, 2.4	0.2, 2.0	0.3, 2.4
AO spp	21	57	8	24	3	13

Results: Dynamic Scenes





	10FPS (100ms)		20FPS (50ms)		30FPS (33ms)	
	2L Grid	Ours	2L Grid	Ours	2L Grid	Ours
λ_1, λ_2	0.03, 0.6	0.3, 2.4	0.03, 0.6	0.02, 0.16	0.03, 0.6	0.01, 0.08
AO spp	1	8	0	1	0	0

Results: Conclusion

Irregular grid properties

- Ordered, stackless traversal
- Same construction/traversal algorithm for:
  - Static scenes
  - Dynamic scenes
- Performance similar/superior to state-of-the-art

Future directions

- Exploring initial subdivision schemes
- Different voxel map structure
- More aggressive optimizations

Questions?

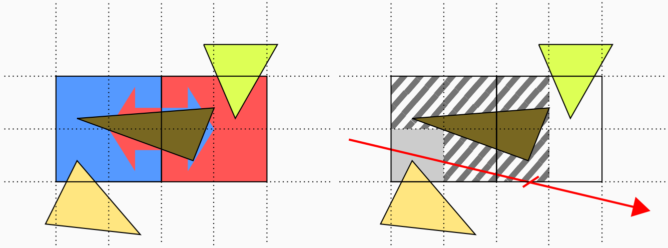
Backup: Related Work

Macro regions

Irregular grid
(uniform initialization)

Macro Regions [Dev89]

- Limited to empty space
- Based on uniform grids



Partial expansion

- Expand cells partially over their neighbors
- Test primitives inside neighbor for intersection
- Implemented in GitHub version
- Additional +10-20% over merge + basic expansion

References

J. G. Cleary et al. “Design and analysis of a parallel ray tracing computer”. In: *Graphics Interface ’83*. 1983, pp. 33–38.

Olivier Devillers. “The Macro-Regions: An Efficient Space Subdivision Structure for Ray Tracing”. In: *EG 1989-Technical Papers*. Eurographics Association, 1989.

Javor Kalojanov, Markus Billeter, and Philipp Slusallek. “Two-Level Grids for Ray Tracing on GPUs”. In: *EG 2011 - Full Papers*. Ed. by Oliver Deussen Min Chen. Llandudno, UK: Eurographics Association, 2011, pp. 307–314.

Martin Stich, Heiko Friedrich, and Andreas Dietrich. “Spatial splits in bounding volume hierarchies”. In: *In Proc. of High-Performance Graphics*. 2009, pp. 7–13.