GPU Ray-tracing using Irregular Grids

Arsène Pérard-Gayot, Javor Kalojanov, Philipp Slusallek
April 23, 2017
Introduction
  Ray Tracing with Grids
  Challenges

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

Results
INTRODUCTION
Introduction: Ray Tracing with Grids

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

Redundant intersections

Empty space
Introduction: Uniform Grid

Redundant intersections
Empty space

\[
\begin{array}{|c|c|}
\hline
\begin{array}{|c|c|}
\hline
z & z \\
\hline
\end{array} \\
\hline
\end{array}
\]
INTRODUCTION: UNIFORM GRID
INTRODUCTION: UNIFORM GRID
Introduction: Uniform Grid

Redundant intersections
Empty space
Introduction: Uniform Grid
Introduction: Uniform Grid
Introduction: Uniform Grid

[Diagram of a uniform grid with various shapes and a red arrow indicating a path through the grid.]
Introduction: Uniform Grid

- Redundant intersections
- Empty space
Introduction: Uniform Grid

Increasing resolution

- Fewer intersections
- More traversal steps
Introduction: Our solution

Idea: Remove regularity

- Start with a dense subdivision
- Optimize cell shape to minimize traversal cost
Introduction: Our solution

Uniform Grid: Low Resolution

Traversal steps + Intersections
Uniform Grid: Medium Resolution

Traversal steps + Intersections
Introduction: Our solution

Irregular Grid: Low Resolution

Traversal steps + Intersections
Introduction: Our solution

Irregular Grid: Medium Resolution

Traversal steps + Intersections
Introduction: Our solution

Irregular Grid: High Resolution

Traversal steps + Intersections
IRREGULAR GRIDS
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
Initialization

- User-defined $\lambda_1$ 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: $\lambda_2$, 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

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

- **3**: Circle
- **2**: Hexagon
- **2**: Star
- **1**: Triangle
- **0**: Triangle

---

**Construction (Part I)**
Initialization
Construction (Part I): Virtual Grid

Property
Cells are aligned on a virtual grid of resolution $R_{x,y,z} 2^D$
Voxel map as a two level grid
Memory efficient/Fast lookup
Interlude: Traversal

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
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
## Traversal Performance

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

## Cell Merging and Expansion

- Local (greedy) optimizations
- Examine cells and their neighborhoods
- Keep optimizations simple and parallelizable
CONSTRUCTION (PART II): OPTIMIZATION PASSES

Initial Grid

Cell Merging

Cell Expansion

Repeat

Repeat
CONSTRUCTION (PART II): OPTIMIZATION PASSES

**Initial Grid**

**Cell Merging**

**Cell Expansion**

Repeat...
CONSTRUCTION (PART II): OPTIMIZATION PASSES

Initial Grid

Cell Merging

Cell Expansion

Repeat

Repeat

Repeat
Cell Merging

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

\[ |\mathcal{R}(A)| SA(A) + |\mathcal{R}(B)| SA(B) \geq |\mathcal{R}(A \cup B)| SA(A \cup B) - C_t \]

- For empty and non-empty cells
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
- \( \alpha = 0.995 \)
CONSTRUCTION (PART II): OPTIMIZATION PASSES

Initial Grid

Cell Merging

Repeat

Cell Expansion

Repeat

Repeat

...
CONSTRUCTION (PART II): OPTIMIZATION PASSES

Initial Grid

Cell Merging

Cell Expansion

Repeat

Repeat
Cell Expansion

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

\[ R(B) \subseteq R(A) \]
Cell Expansion

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

\[ \mathcal{R}(A) \not\subseteq \mathcal{R}(B) \]
**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
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
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
CONSTRUCTION (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
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
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
RESULTS
## GPU implementation

- [https://github.com/madmann91/hagrid](https://github.com/madmann91/hagrid)
- Parallel construction & traversal
- CUDA implementation
- MIT license
Results: Static Scenes

Parameters

- \((\lambda_1, \lambda_2) = (0.12, 2.4)\) for every scene
- Memory footprint \(\approx\) SBVH [SFD09]
- Different viewpoints
## Results: Static Scenes

<table>
<thead>
<tr>
<th>Scene</th>
<th>#Tris</th>
<th>Build times (ms)</th>
<th>Primary (MRays/s)</th>
<th>AO (MRays/s)</th>
<th>Random (MRays/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>SBVH</td>
<td>Ours</td>
<td>SBVH</td>
</tr>
<tr>
<td>Sponza</td>
<td>262K</td>
<td>26</td>
<td>409</td>
<td>653</td>
<td>+60%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>265</td>
<td>473</td>
<td>+78%</td>
</tr>
<tr>
<td>Conference</td>
<td>283K</td>
<td>22</td>
<td>583</td>
<td>597</td>
<td>+2%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>523</td>
<td>526</td>
<td>+1%</td>
</tr>
<tr>
<td>Hairball</td>
<td>2.9M</td>
<td>893</td>
<td>100</td>
<td>148</td>
<td>+48%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>79</td>
<td>93</td>
<td>+18%</td>
</tr>
<tr>
<td>Crown</td>
<td>3.5M</td>
<td>203</td>
<td>232</td>
<td>296</td>
<td>+28%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>181</td>
<td>191</td>
<td>+6%</td>
</tr>
<tr>
<td>San Miguel</td>
<td>7.9M</td>
<td>492</td>
<td>227</td>
<td>291</td>
<td>+28%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>157</td>
<td>180</td>
<td>+15%</td>
</tr>
</tbody>
</table>

|       |       |                  | 166  | 274  | +65% |
|       |       |                  | 295  | 312  | +6%  |
|       |       |                  | 19   | 26   | +37% |
|       |       |                  | 221  | 238  | +8%  |
|       |       |                  | 119  | 160  | +34% |
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
Results: Dynamic Scenes

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 $T_{\text{build}} = 0.5 T_{\text{budget}}$
**Results: Dynamic Scenes**

<table>
<thead>
<tr>
<th>λ₁, λ₂</th>
<th>10FPS (100ms)</th>
<th>20FPS (50ms)</th>
<th>30FPS (33ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AO spp</td>
<td>1spp</td>
<td>1spp</td>
<td>1spp</td>
</tr>
<tr>
<td></td>
<td>2L Grid</td>
<td>2L Grid</td>
<td>2L Grid</td>
</tr>
<tr>
<td></td>
<td>0.2, 2.0</td>
<td>0.2, 2.0</td>
<td>0.2, 2.0</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>20</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0.3, 2.4</td>
<td>0.3, 2.4</td>
<td>0.3, 2.4</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>8</td>
<td>3</td>
</tr>
</tbody>
</table>
Results: Dynamic Scenes

<table>
<thead>
<tr>
<th>λ₁, λ₂</th>
<th>10FPS (100ms)</th>
<th>20FPS (50ms)</th>
<th>30FPS (33ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2L Grid</td>
<td>Ours</td>
<td>2L Grid</td>
</tr>
<tr>
<td>AO spp</td>
<td>0.2, 2.0</td>
<td>0.3, 2.4</td>
<td>0.2, 2.0</td>
</tr>
<tr>
<td></td>
<td>21</td>
<td>57</td>
<td>8</td>
</tr>
</tbody>
</table>
# Results: Dynamic Scenes

<table>
<thead>
<tr>
<th>λ₁, λ₂</th>
<th>AO spp</th>
<th>10FPS (100ms)</th>
<th>20FPS (50ms)</th>
<th>30FPS (33ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>2L Grid</td>
<td>Ours</td>
<td>2L Grid</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.03, 0.6</td>
<td>0.3, 2.4</td>
<td>0.03, 0.6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>8</td>
<td>0</td>
</tr>
</tbody>
</table>
## 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?
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


