Taranis: Ray Tracing Radiative Transfer in SPH

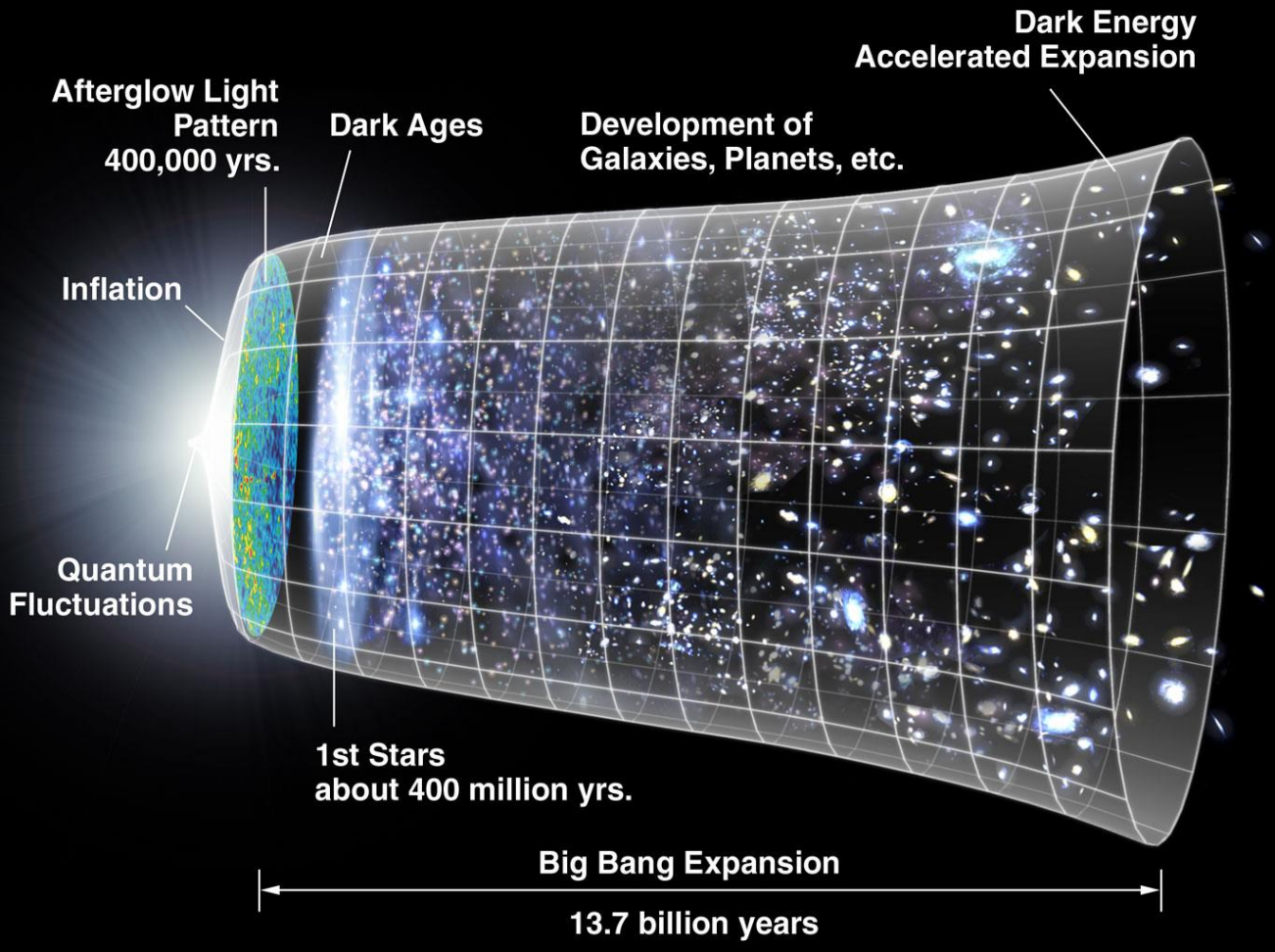
Sam Thomson spth@roe.ac.uk
Eric Tittley, Martin Rüfenacht, Alex Bush

Institute for Astronomy, University of Edinburgh

Introduction

- **GRACE: GPU-Accelerated Ray-Tracing for Astrophysics**

- Taranis: GRACE + Radiative Transfer (CPU and GPU, in progress)


Dark Energy
Accelerated Expansion
Afterglow Light
Pattern
400,000 yrs.
Dark Ages
Development of
Galaxies, Planets, etc.
Inflation
Quantum
Fluctuations
1st Stars
about 400 million yrs.
Big Bang Expansion
13.7 billion years

Motivation

- Currently, radiative transfer is treated by:
  - Ignoring it
  - Diffusion approximation
  - Higher-order moments of the radiative transfer equation
  - Ray tracing
- Usually done by post-processing
- Ray tracing is the **most accurate**, but **slowest**, solution: naively need $N_{\mathrm{particles}} (\sim 128^3 - 512^3)$ rays per source

Aside: Cosmological Simulations

Grid-based (Eulerian)

- Grid is fixed, fluid flow determined from neighbouring cells
- Cell determines the fluid properties at its location

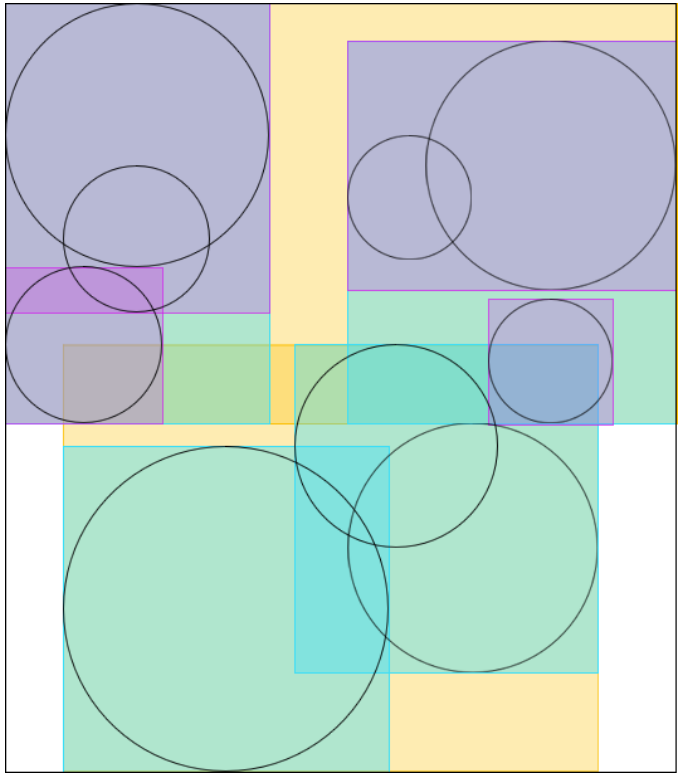
Smoothed Particle Hydrodynamics (Lagrangian)

- SPH particles move with the flow of the fluid
- Fluid properties at a point depends (formally) on **all** particles

Acceleration Structures

- Naively scales as $N_{\text{rays}} \times N_{\text{particles}}$

- Acceleration structure: $N_{\text{rays}} \times \log N_{\text{particles}}$ scaling
  - *k*-d Tree
  - **Bounding Volume Hierarchy (BVH)**

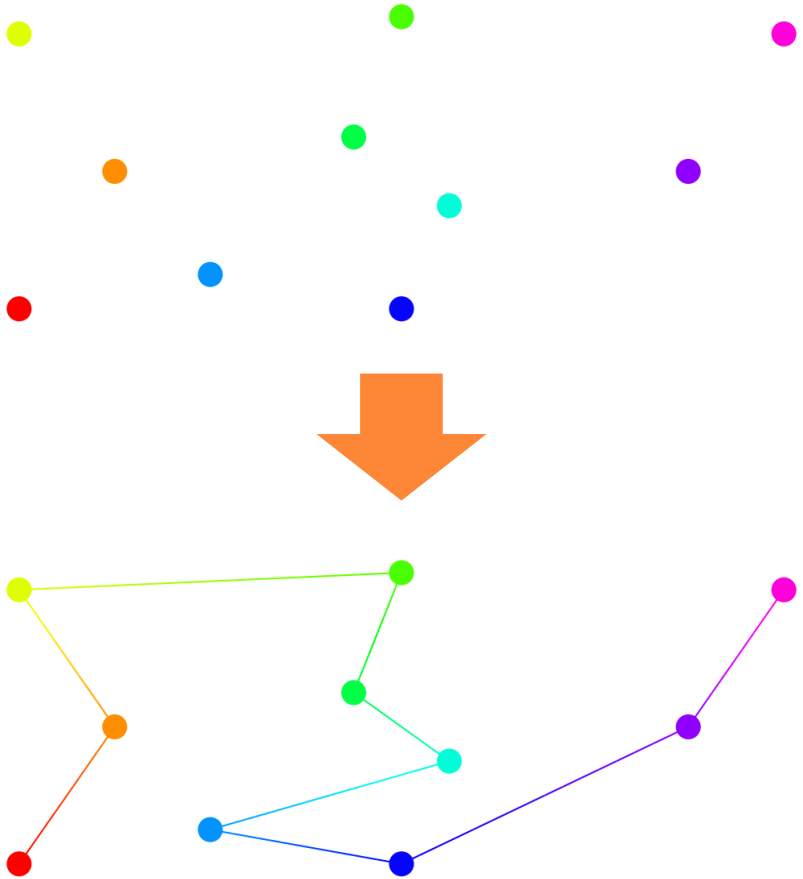


Tree Construction With A Space-filling Curve

1. Order all particles along a 1D curve

2. Place particles into **nodes** according to their position along the line

3. Assign axis-aligned bounding boxes (**AABBs**) to all nodes, starting at the **leaves**



Lauterbach et al. (2009)
Warren & Salmon (1993)

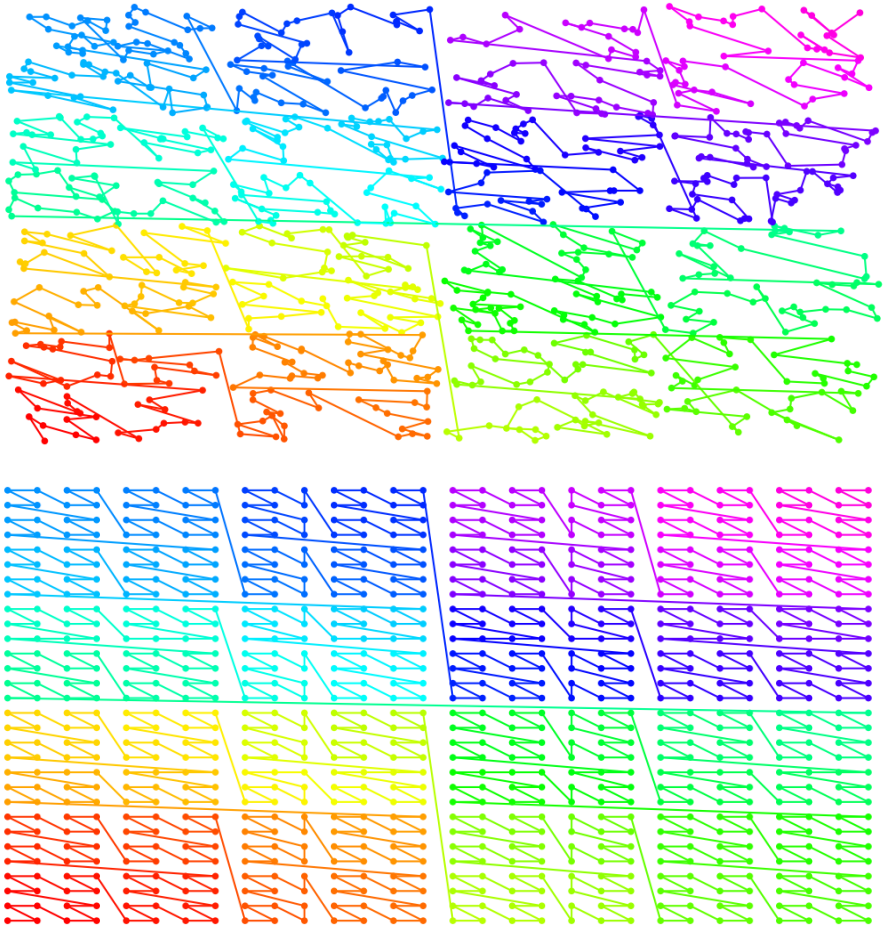
The Morton Curve

- Map floats $(x, y) \in [0, 1)$ to integers $(x', y') \in [0, 2^E)$ and interleave the bits:

1. $(x, y) = (0.25, 0.60)$

$$\xrightarrow{\text{int} : [0,2^5)} (x', y') = (7, 18) = (00111, 10010)$$

2. key $= 0100101110 = 302$

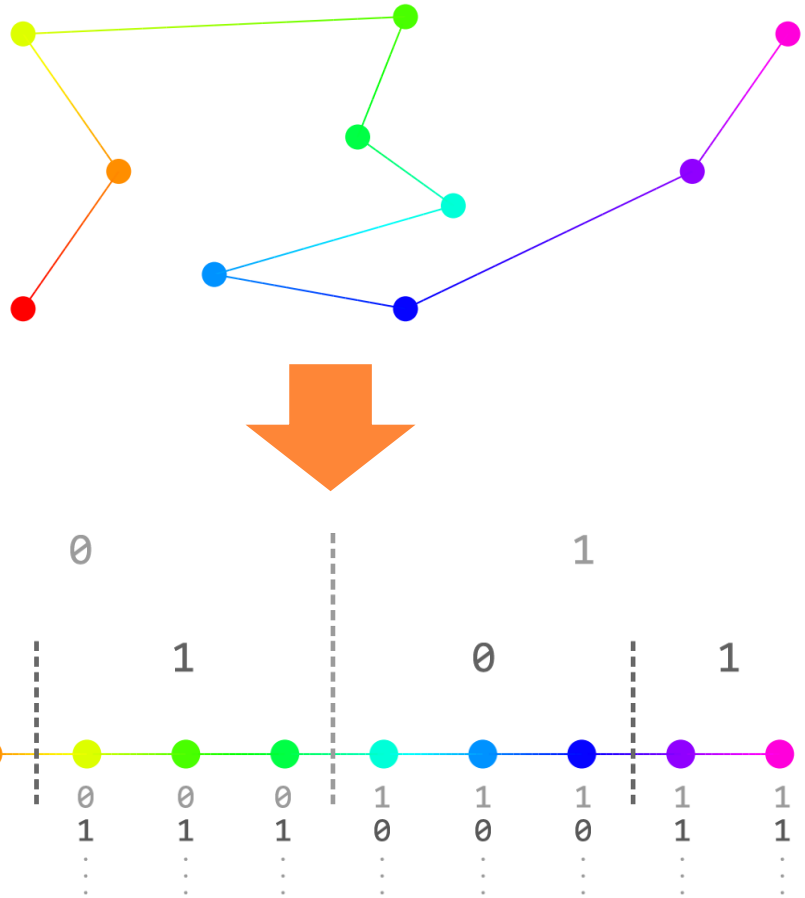


Tree Construction With A Space-filling Curve

1. Order all particles along a 1D curve

2. Place particles into **nodes** according to their position along the line

3. Assign axis-aligned bounding boxes (**AABBs**) to all nodes, starting at the **leaves**

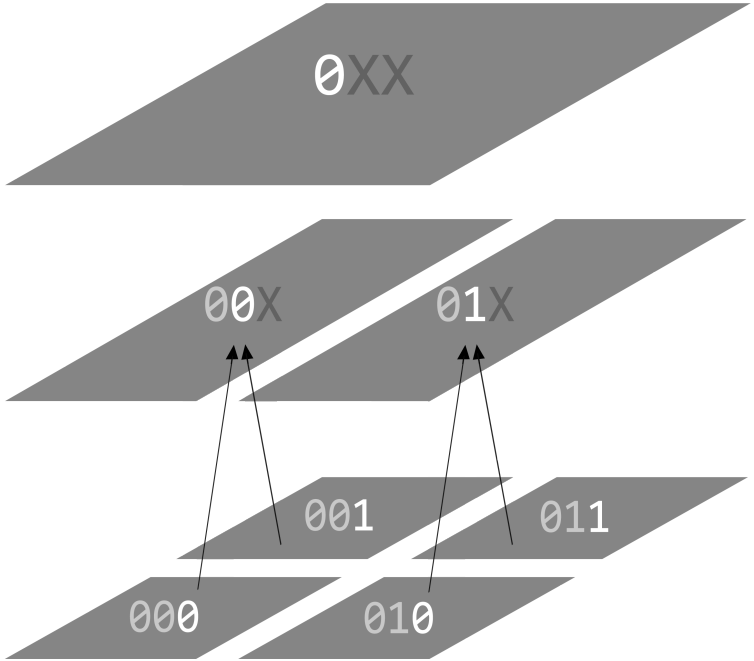


Tree Construction With A Space-filling Curve

1. Order all particles along a 1D curve

2. Place particles into **nodes** according to their position along the line

3. Assign axis-aligned bounding boxes (**AABBs**) to all nodes, starting at the **leaves**



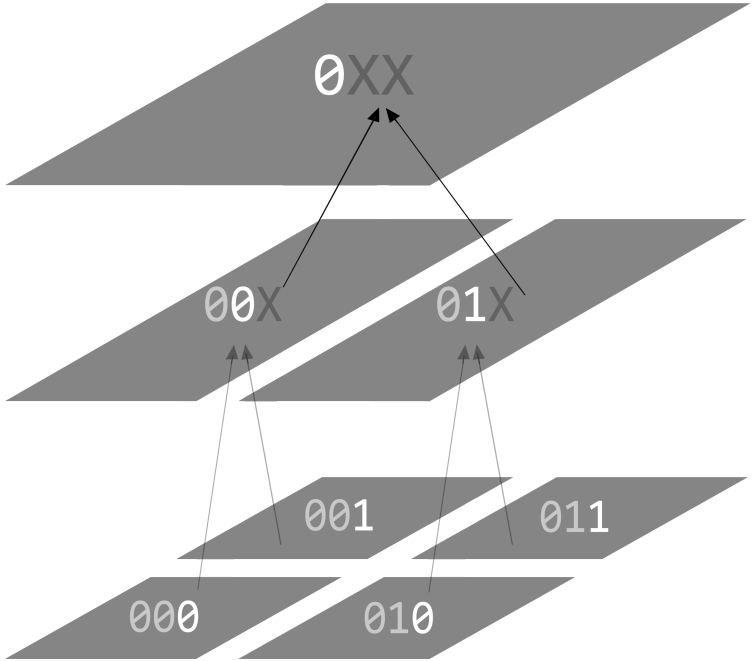
Karras (2012)

Tree Construction With A Space-filling Curve

1. Order all particles along a 1D curve

2. Place particles into **nodes** according to their position along the line

3. Assign axis-aligned bounding boxes (**AABBs**) to all nodes, starting at the **leaves**

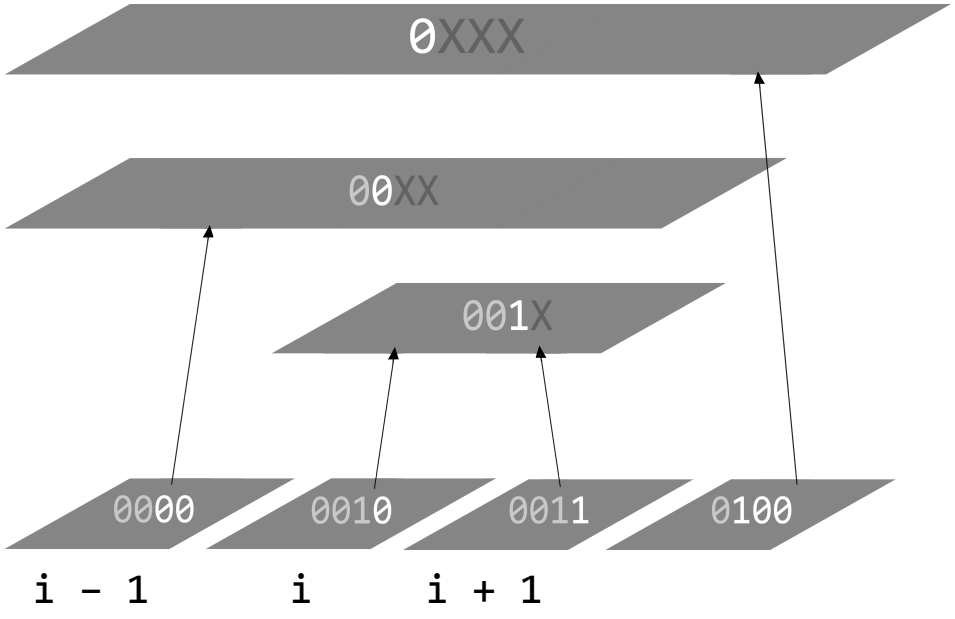


Karras (2012)

Tree Construction With A Space-filling Curve

- In our implementation, tree hierarchy and AABB finding occur simultaneously

- The tree climb is iterative; each thread block covers an (overlapping) range of leaves
- Each block independently processes a contiguous subset of the input nodes
- For 128^3 particles, we can build a tree in ~20 (40) ms



Apetrei (2014)

Tree Construction With A Space-filling Curve

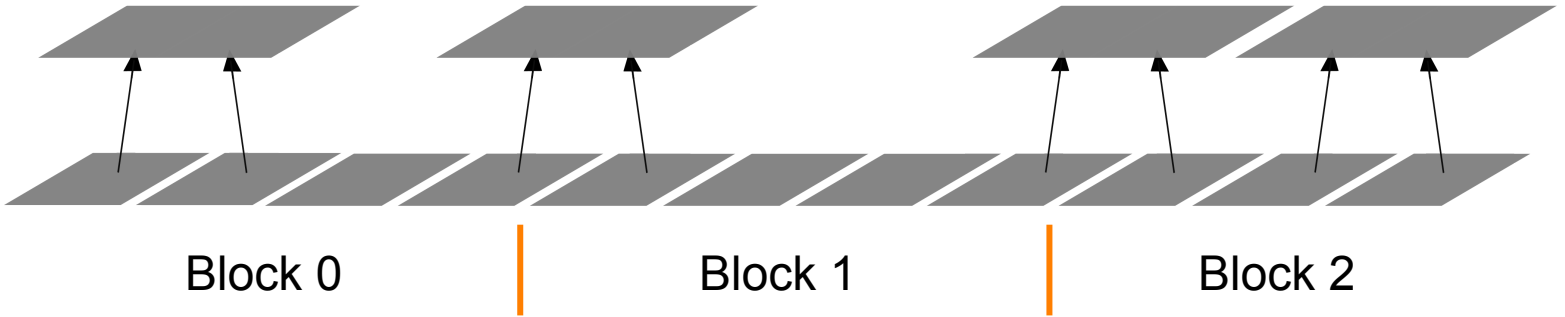
- In our implementation, tree hierarchy and AABB finding occur simultaneously

- The tree climb is iterative; each iteration adds a layer of nodes on top of the last

- Each block independently processes a contiguous subset of the input nodes

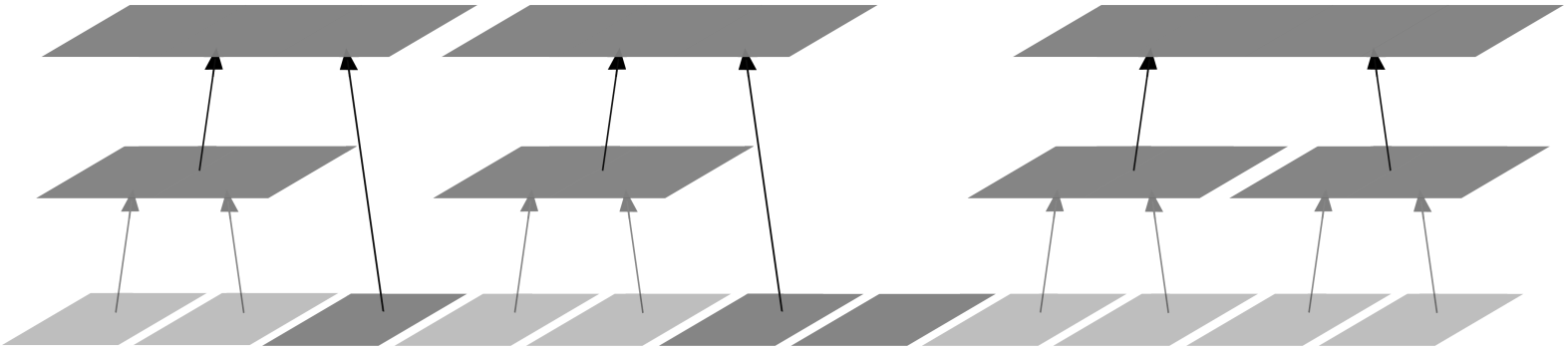
- For 128^3 particles, we can build a tree in $\sim 20\ (40)$ ms


Block 0
Block 1
Block 2


Block 0
Block 1
Block 2

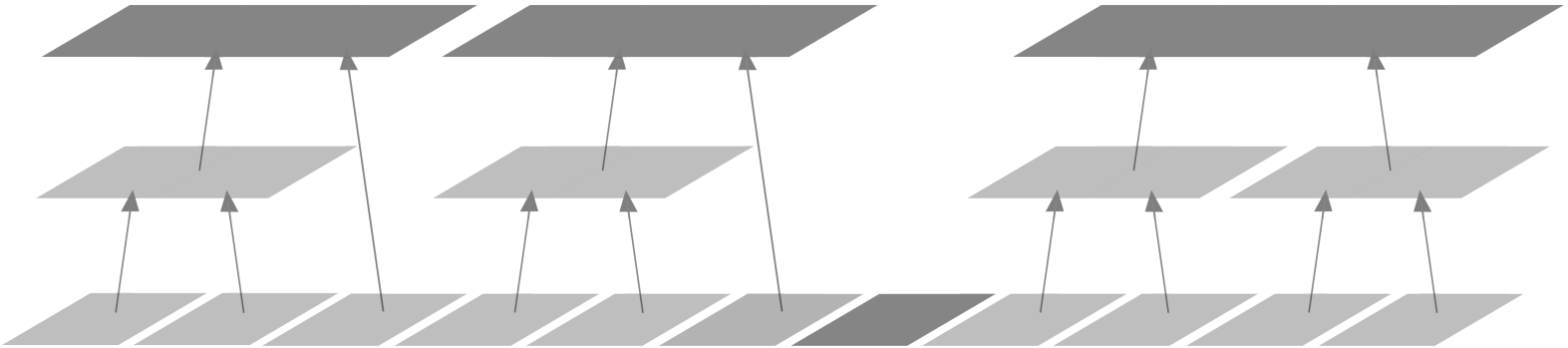
Block 0

Block 1

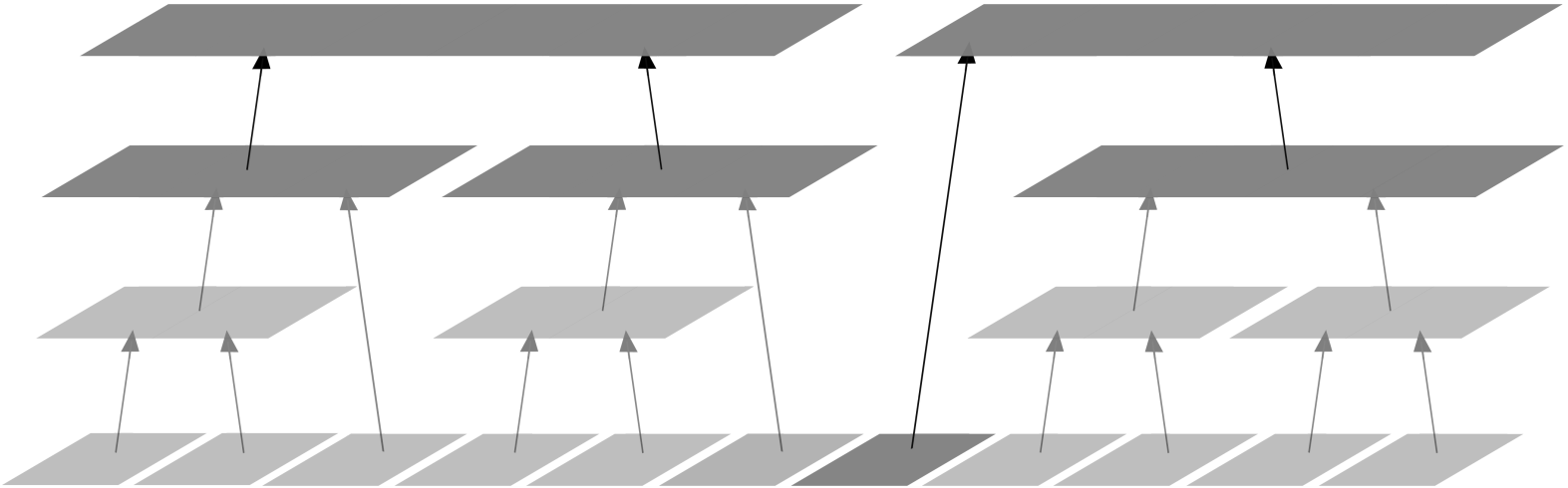


Block 0

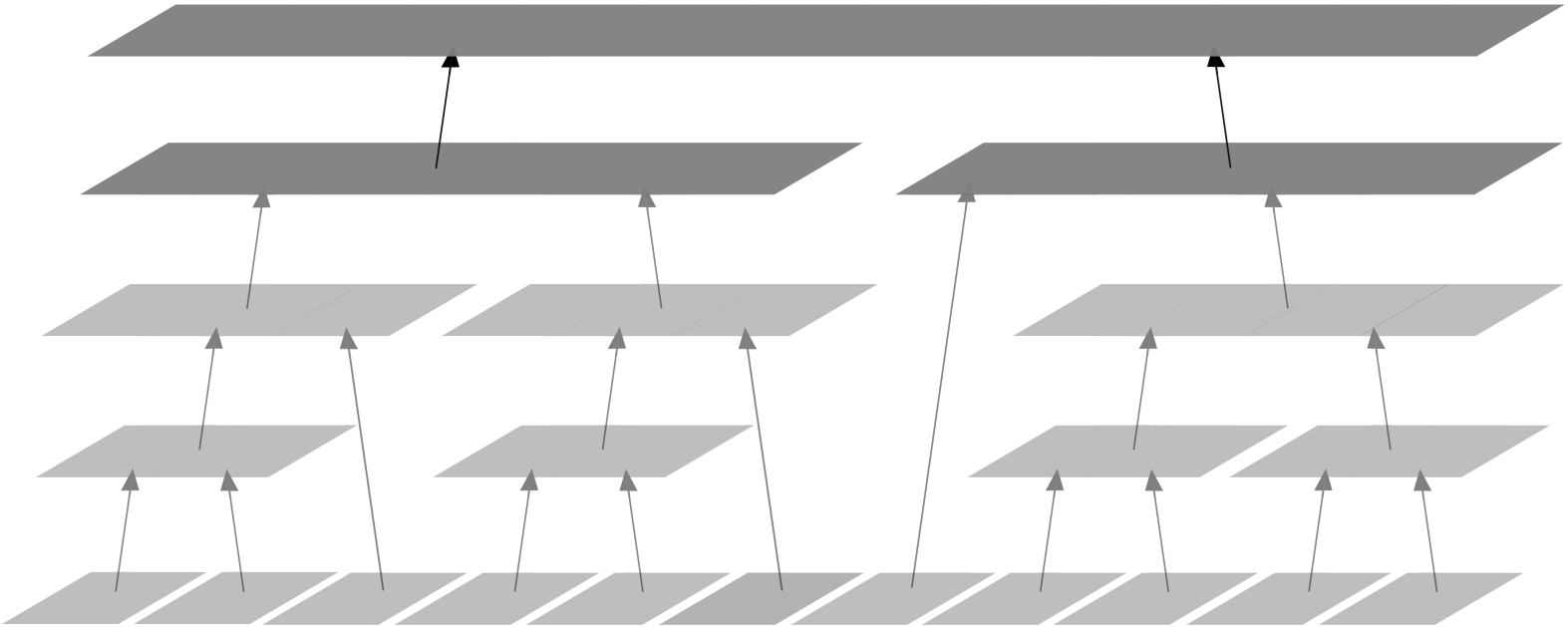
Block 1



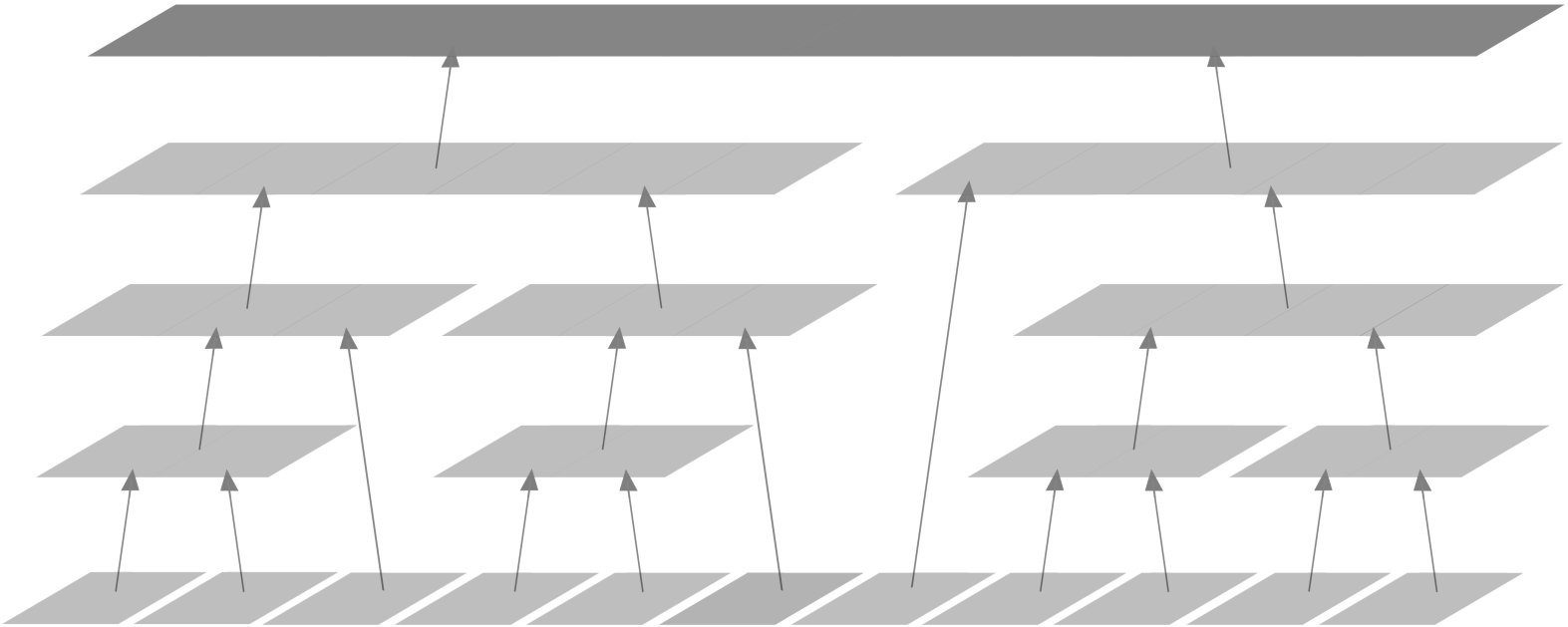
Block 0



Block 0



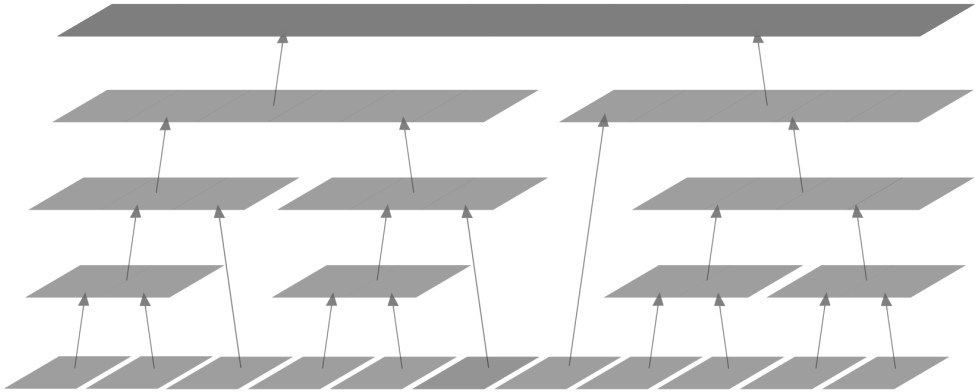
Block 0



Block 0

Tree Construction With A Space-filling Curve

- In our implementation, tree hierarchy and AABB finding occur simultaneously
- The tree climb is iterative; each iteration adds a layer of nodes on top of the last
- Each block independently processes a contiguous subset of the input nodes

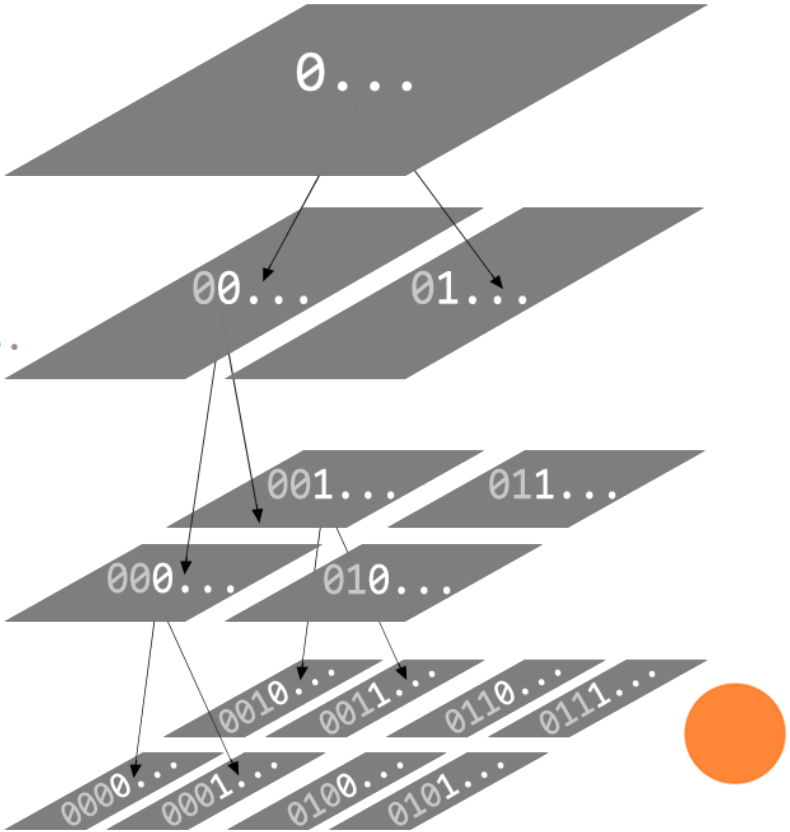


- For 128^3 particles, we can build a tree in $\sim 20\ (40)$ ms

BVH Traversal

- Typical traversal loop:

```
1  def traverse(ray, node):
2      if intersect(ray, node.AABB):
3          if node.is_leaf():
4              # Hit a leaf: intersect spheres.
5              for sphere in node.spheres:
6                  if intersect(ray, sphere):
7                      # Do work/output.
8          else:
9              # Node has children.
10             traverse(ray, node.get_left())
11             traverse(ray, node.get_right())
12
13
14 traverse(ray, tree.root())
```



GPU BVH Traversal

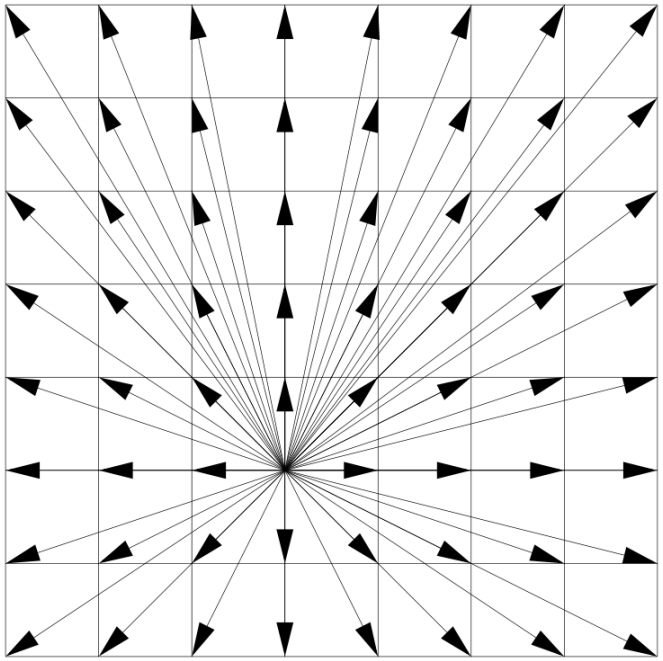
- Traversal with a stack:

```
1  def traverse(ray, tree):
2      stack.push(tree.root)
3      while not stack.empty():
4          node = stack.pop()
5          if intersect(ray, node.AABB):
6              if node.is_leaf():
7                  # Hit a leaf: intersect spheres.
8                  for sphere in node.spheres:
9                      if intersect(ray, sphere):
10                         # Do work/output.
11             else:
12                 # Node has children.
13                 stack.push(node.get_right())
14                 stack.push(node.get_left())
15
16 traverse(ray, tree)
```

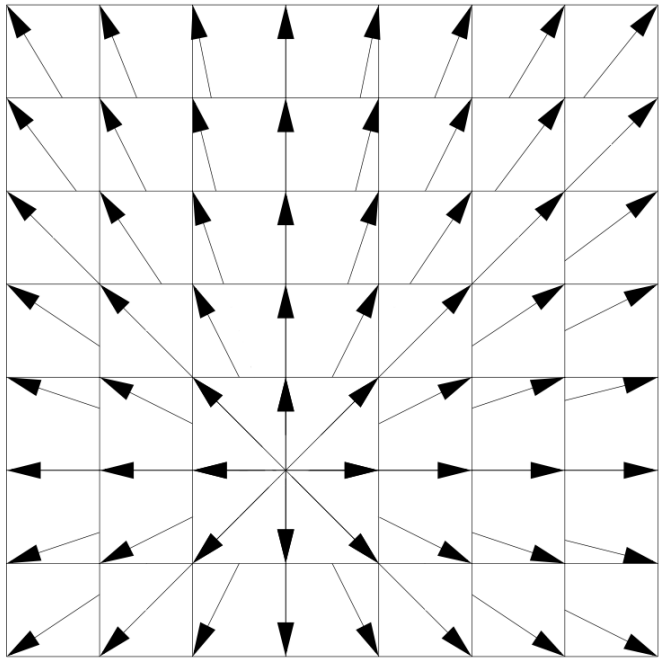
- Optimizations:
  - **Multiple spheres in a leaf** ($\sim 2\times$)
  - **Packet tracing** ($\sim 2\times$)
  - Packed nodes structs (64 bytes: hierarchy and child AABBs) ($\sim 1.3\times$)
  - Shared memory sphere caching ($\sim 1.2\times$)
  - Texture fetches of node and sphere data ($\sim 1.1\times$)

Aside: Ray Tracing in Astrophysics

- Long characteristics

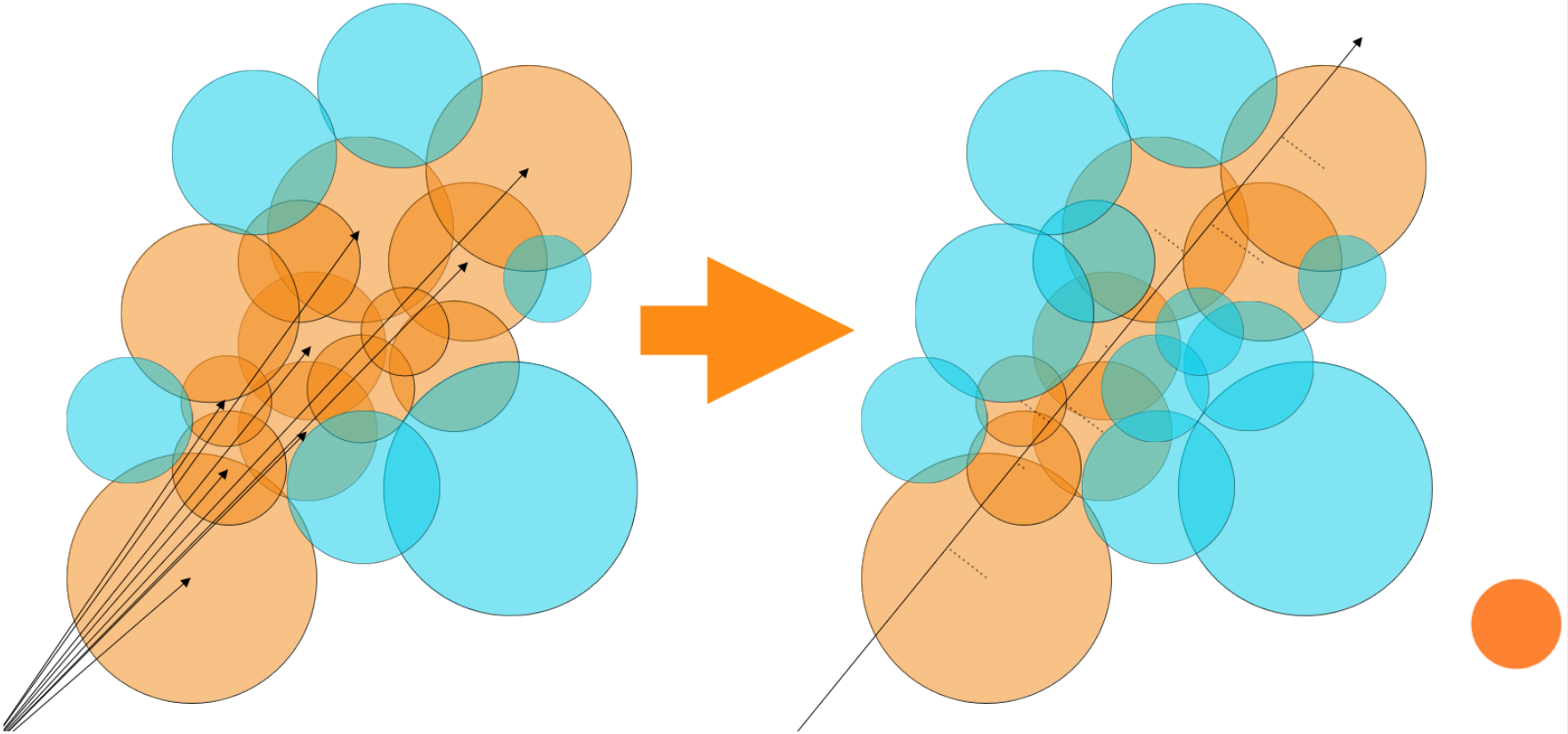


- Short characteristics



Rijkhorst et al. (2006), A&A, **452**, 907

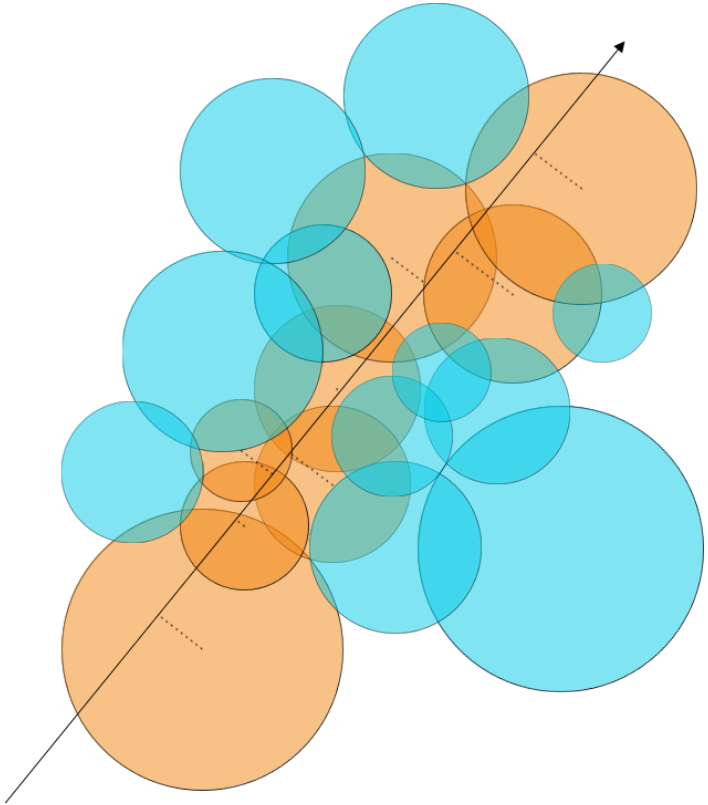
GRACE TRACE ALGORITHM



GRACE+TARANIS TRACE ALGORITHM

1. Output data for every intersection:
   I. Trace: count per-**ray** hits
   II. Scan sum hit counts
   III. Trace: output per-**hit** column densities
   IV. Sort per-**ray** outputs by distance
   V. Scan sum per-**ray** outputs

2. Result is cumulative column density **up to** each intersected particle for each ray

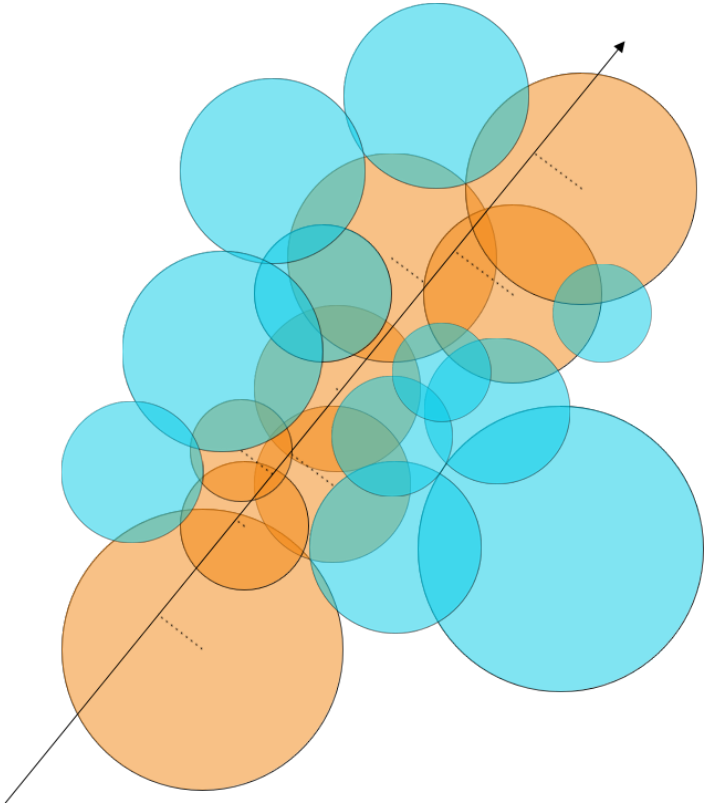


GRACE+Taranis Trace Algorithm

- Source-to-particle column densities sufficient for radiative transfer:

1. Accumulate ionization and heating **rates** for each particle (in parallel with atomics)

2. Update particles' ionization and temperature variables (independently and in parallel)



PERFORMANCE

- 128^3 particles in a $(10\ \mathrm{Mpc})^3$ box at the end of hydrogen reionization ($z \sim 6$); comparing to an optimized CPU code: OpenMP, SIMD ray packets and SAH-optimized BVH
- **'CPU/GPU':** projected down the *z*-axis through the simulation volume, point-to-point cumulative (512^2 rays)
- **'All intersections':** traced out from centre, all intersection data output (145,024 rays)
- **'+ sort':** sorts all-intersections data by distance along the ray

Metric	CPU (2x 16-core AMD Opteron 6276 @ 2.3 GHz)	GPU (1x Tesla M2090)	GPU all intersections (1x Tesla M2090)	GPU all intersections + sort (1x Tesla M2090)
Rays / second	3.0×10^5	1.2×10^6	4.0×10^5	2.1×10^5
Rays / second / £	~50	~160	~55	~30
Rays / J @ TDP	~1300	~5300	~1800	~960

PERFORMANCE

- This work: peak performance for all intersections, rays traced from centre
- **'CPU'**: cumulative projection/point-to-point (as in previous slide)
- **'OptiX'**: intersection counts **only**

Metric	CPU (2x 16-core AMD Opteron 6276 @ 2.3 GHz)	OptiX (1x GTX 670)	M2090 (ECC)	GTX 670	K20 (ECC)	GTX 970
Rays / second	3.0×10^5	4.8×10^5	4.0×10^5	4.2×10^5	6.3×10^5	9.6×10^5
Rays / second (inc. sort)	N/A	N/A	2.1×10^5	2.5×10^5	3.3×10^5	4.5×10^5

Outlook

- Combined GRACE with CPU radiative transfer code
- Will be combined with existing GPU port
- GRACE API will remain separate for use in other projects
- GRACE released under GPL within ~two months (sooner on request – just e-mail me)

Thank You

Contact:

- Sam Thomson, University of Edinburgh, UK
- spth@roe.ac.uk

References

- Lauterbach, C., Garland, M., Sengupta, S., Luebke, D., & Manocha, D. (2009). “Fast BVH Construction on GPUs”. *Computer Graphics Forum*, **28**(2), 375–384.
- Warren, M., & Salmon, J. (1993). “A parallel hashed oct-tree n-body algorithm.” In *Proceedings of the 1993 ACM/IEEE Conference on Supercomputing*, 12–21. New York, NY, USA: ACM.
- Karras, T. (2012). “Maximizing Parallelism in the Construction of BVHs, Octrees, and K-d Trees.” In *Proceedings of the Fourth ACM SIGGRAPH / Eurographics Conference on High-Performance Graphics*, 33-37.
- Apetrei, C. (2014) “Fast and Simple Agglomerative LBVH Construction.” In *Computer Graphics and Visual Computing (CGVC).*