A PARALLEL SPH IMPLEMENTATION ON MULTI-CORE CPUS

(2010)

Ihmsen, M., Nadir, A., Becker, N., Teschner, M.

presented by Steffen Haug

A PARALLEL SPH IMPLEMENTATION ON MULTI-CORE CPUS

Applies to particle simulations in general!

(And raytracing, rigid body collision, ...)

Smoothed-particle Hydrodynamics

(Briefly)

$$A(x) = \int_{\Omega} A(V)\delta(x - V)\,dV$$

Start with the convolution definition of δ.

$$A(x) = \int_\Omega A(V)\, W(x - V)\, dV$$

Replace δ-function with smoothing kernel that "works like" δ.

Smoothed-particle Hydrodynamics

$$A(x) = \int_{\Omega} A(V)\, W(x - V)\, dV$$

Key points: W has *compact support* of radius h

$\int_{\Omega} W = 1$ (Normalization)

$W \longrightarrow \delta$ as $h \longrightarrow 0$

$$A(x) = \int_{\Omega} A(V)\, W(x - V)\, dV$$

Tldr: W is a "bell-like" curve.

(Convolution by bell curve "smoothes" signal, hence the name SPH)

$$A(x_i) = \sum_j A(x_j)\, W(x_i - x_j)\, V_j$$

Replace integral over Ω with sum over samples at N "particles".

$$i, j \in \{1 \ldots N\}$$

$$A(x_i) = \sum_j A(x_j)\, W(x_i - x_j)\, V_j$$

Volume element dV is now the volume V_j of particle j.

Smoothed-particle Hydrodynamics

$$A_i = \sum_j A_j W_{ij} V_j$$

Standard compact notation.

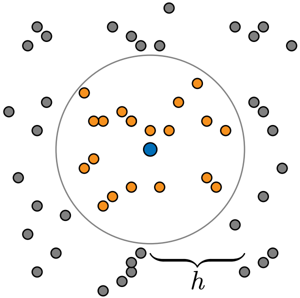
$$A_i = \sum_j A_j W_{ij} V_j$$

From this you can derive discretizations for ∇A, $\nabla^2 A$, $\nabla \cdot A$, and so on...

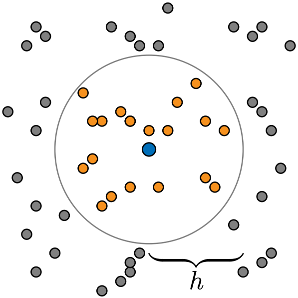
$$A_i = \sum_j A_j W_{ij} V_j$$

Zero outside the ball $B_h(x_i)$!

(Compact support of W)



We need to discard the *vast majority* of particles.



Fast access to particles in $B_h(x_i)$ is a major optimization!

"Fixed-radius near neighbors"

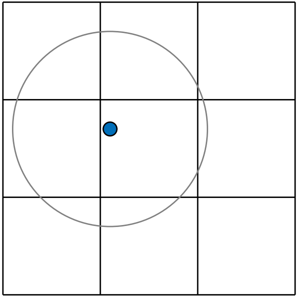
(Classic computational geometry problem)

The authors present two strategies

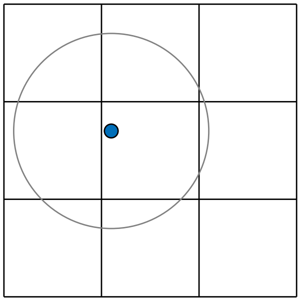
INDEX SORTING and SPATIAL HASHING

I will present some criticism, especially in the context of GPU.

Both methods use a tiling of size h



$B_h(x_i)$ is completely contained in 3×3 tile region.



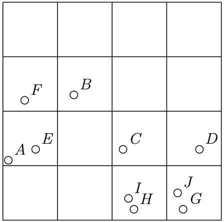
Trivially constructed by rounding the coordinates:

$$i = \left\lfloor \frac{x}{h} \right\rfloor, \quad j = \left\lfloor \frac{y}{h} \right\rfloor$$

Index sorting

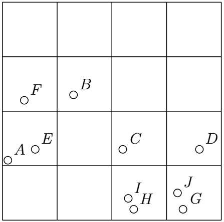
Index sorting

1. Create a *finite* regular grid



Index sorting

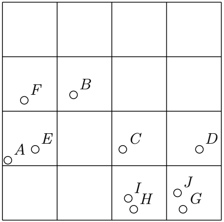
1. Create a *finite* regular grid
2. Calculate the strided grid cell index for each particle



4	9	6	7	4	8	3	2	2	3
A	*B*	*C*	*D*	*E*	*F*	*G*	*H*	*I*	*J*

Index sorting

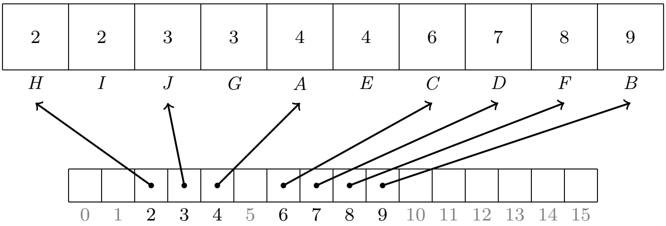
1. Create a *finite* regular grid
2. Calculate the strided grid cell index for each particle
3. Sort particles by grid index



2	2	3	3	4	4	6	7	8	9
H	I	J	G	A	E	C	D	F	B

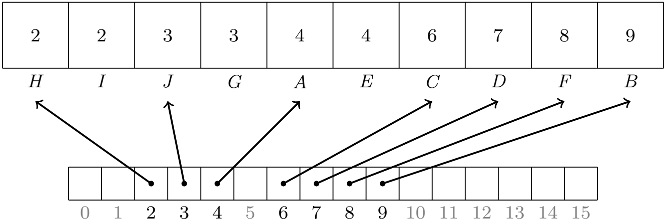
Index sorting

1. Create a *finite* regular grid
2. Calculate the strided grid cell index for each particle
3. Sort particles by grid index
4. Identify where the edges of each grid cell is in the particle buffer



Index sorting

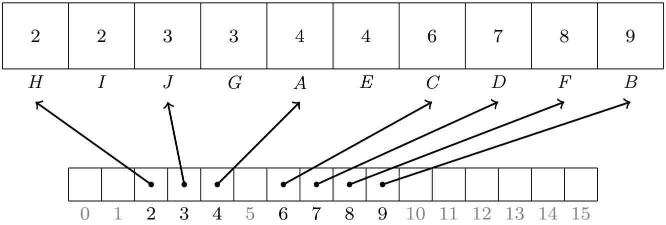
1. Create a *finite* regular grid
2. Calculate the strided grid cell index for each particle
3. Sort particles by grid index
4. Identify where the edges of each grid cell is in the particle buffer



Domain must be finite: Unique index maps to finite set of buckets.

Index sorting

1. Create a *finite* regular grid
2. Calculate the strided grid cell index for each particle
3. Sort particles by grid index
4. Identify where the edges of each grid cell is in the particle buffer



Possible optimization: Order grid cells by space filling curve.

Criticism

1. **Finite domain is an annoying restriction**
2. High number of vacant grid cells
3. Full sort can be very expensive

Criticism

1. **Finite domain is an annoying restriction**
2. High number of vacant grid cells
3. Full sort can be very expensive

Advantages

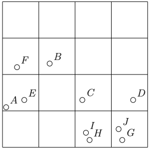
1. **No dynamic memory allocation required**
2. Minimal synchronization between threads required

Verdict: *Reasonably well-suited for GPU*

(Compact) spatial hashing

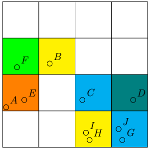
Spatial hashing

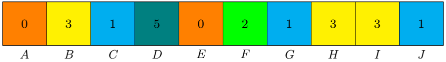
1. Create a (possibly infinite) regular tiling



Spatial hashing

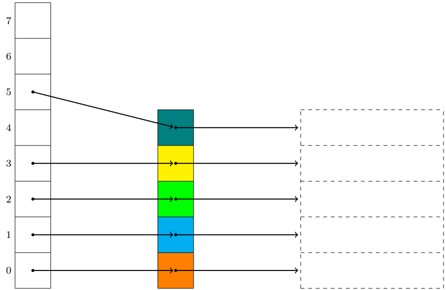
1. Create a (possibly infinite) regular tiling
2. *Hash* the grid cell index for each particle
   (Muliple grid cells can hash to the same bucket!)





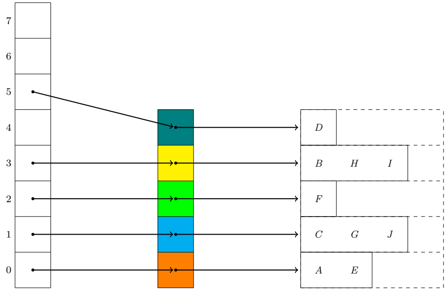
Spatial hashing

1. Create a (possibly infinite) regular tiling
2. *Hash* the grid cell index for each particle
   (Muliple grid cells can hash to the same bucket!)
3. For each populated bucket, allocate a particle buffer in a secondary structure
   (Initial capacity k, grow with amortized constant time)



Spatial hashing

1. Create a (possibly infinite) regular tiling
2. *Hash* the grid cell index for each particle
   (Muliple grid cells can hash to the same bucket!)
3. For each populated bucket, allocate a particle buffer in a secondary structure
   (Initial capacity k, grow with amortized constant time)
4. Copy each particle into its respective particle buffer



Criticism

1. **Uses dynamic memory allocation in multiple places**
2. **Forced reallocation of vectors in multple critical places**
3. **Contention between threads on particle buffers**
   (Imagine 10K threads mutating the same vectors...)
4. Hash collisions cause performance degradation
5. Particle buffers live in separate (non-coherent) heap allocations
   (The bane of dynamic vectors is that it fragments the heap)

Criticism

1. **Uses dynamic memory allocation in multiple places**
2. **Forced reallocation of vectors in multple critical places**
3. **Contention between threads on particle buffers**
   (Imagine 10K threads mutating the same vectors...)
4. Hash collisions cause performance degradation
5. Particle buffers live in separate (non-coherent) heap allocations
   (The bane of dynamic vectors is that it fragments the heap)

Advantages

1. **No restriction on domain size**
2. Avoids complete sort
3. Can more easily be incrementally updated

🤮

Verdict: *Absolutely useless on GPU*

method	construction	query	total
basic uniform grid	25.7 (27.5)	38.1 (105.6)	**63.8** (133.1)
index sort [Gre08]	35.8 (38.2)	29.1 (29.9)	**64.9** (77.3)
Z-index sort	16.5 (20.5)	26.6 (29.7)	**43.1** (50.2)
spatial hashing	41.9 (44.1)	86.0 (89.9)	**127.9** (134.0)
compact hashing	8.2 (9.4)	32.1 (55.2)	**40.3** (64.6)

Table 2: *Performance analysis of different spatial acceleration methods with and without (in brackets) reordering of particles. Timings are given in milliseconds for CBD 130K and include storing of pairs.*