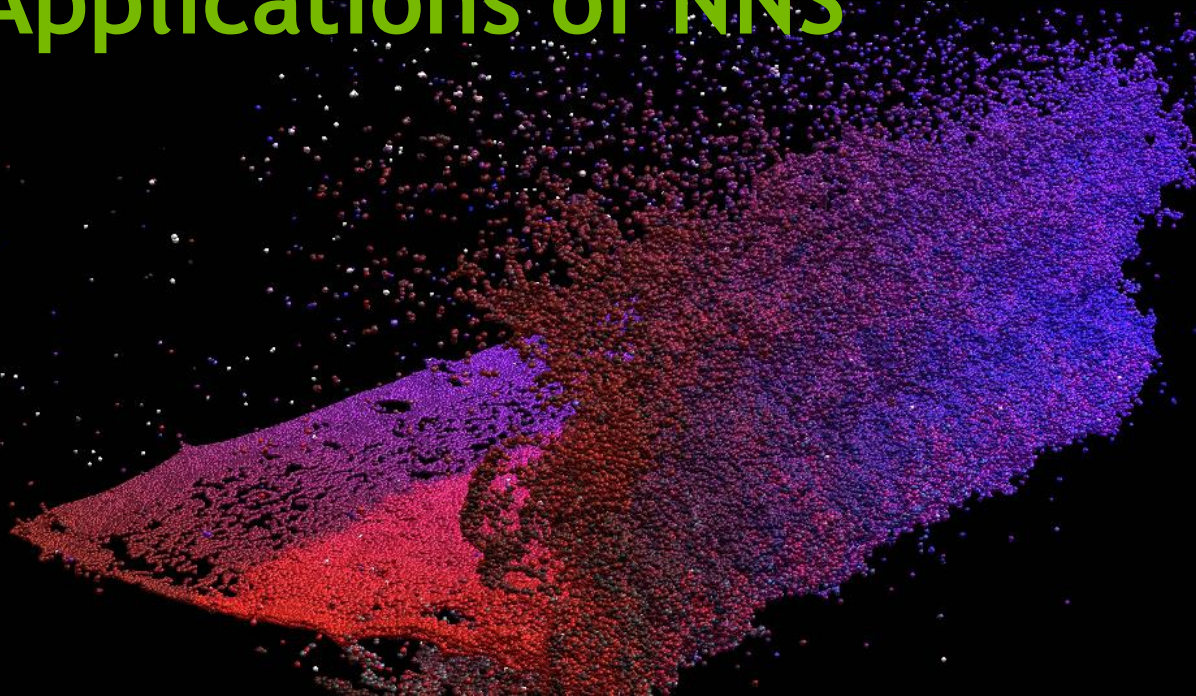


FAST FIXED-RADIUS NEAREST NEIGHBORS: INTERACTIVE MILLION-PARTICLE FLUIDS

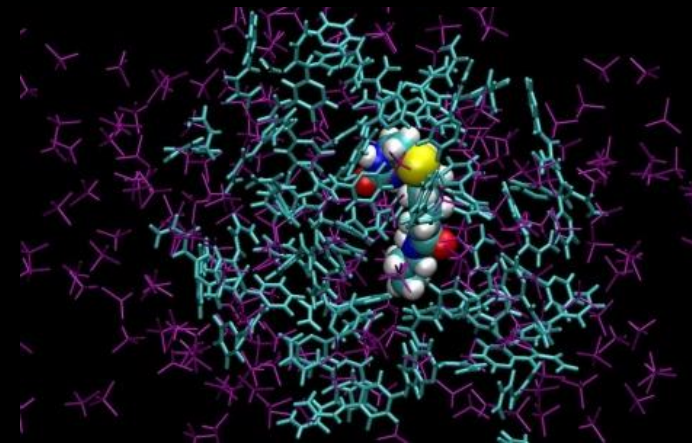
Rama C. Hoetzlein, Graphics Devtech, NVIDIA



Applications of NNS



Fluid Simulation



Molecular Modeling

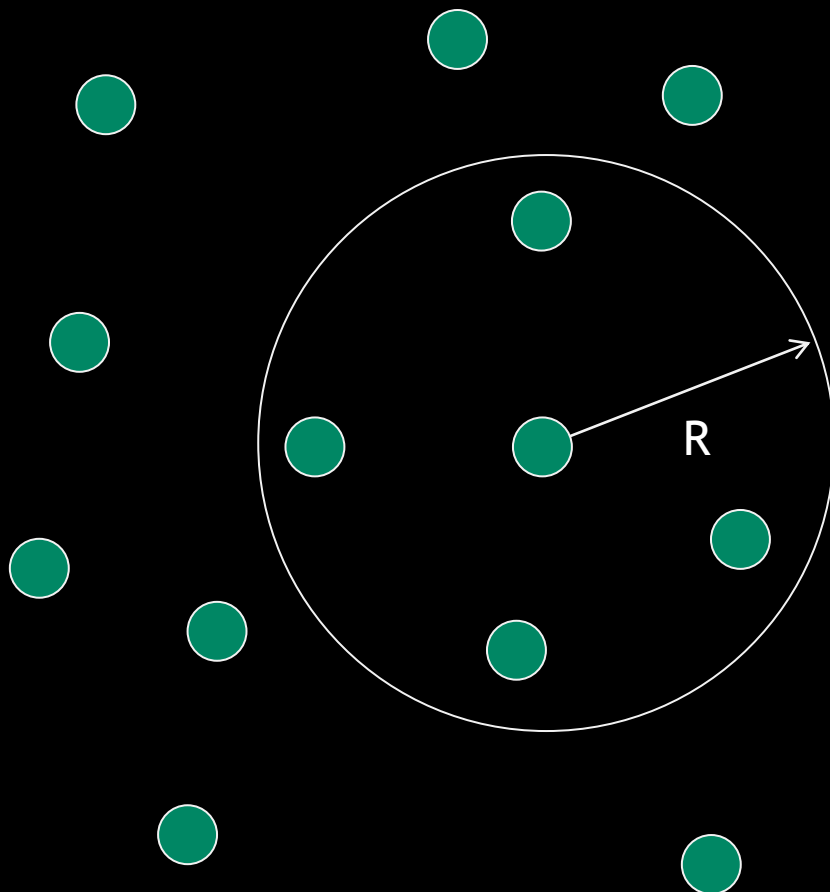


Cognitive Science - Behavioral simulation



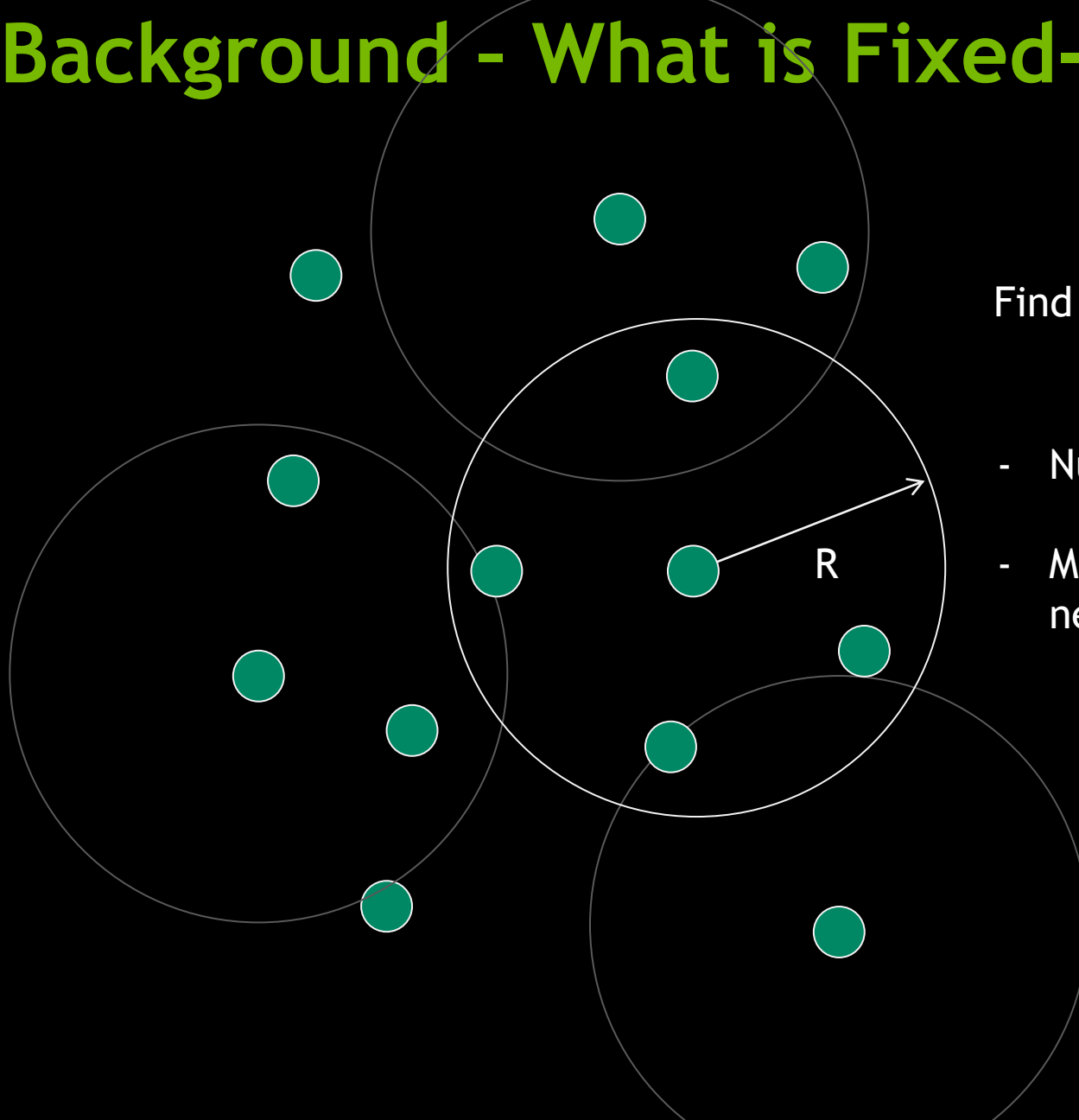
3D Scanning - Point Cloud Reconstruction

Background - What is Fixed-Radius NNS?



Find all neighbors in a fixed radius R

Background - What is Fixed-Radius NNS?

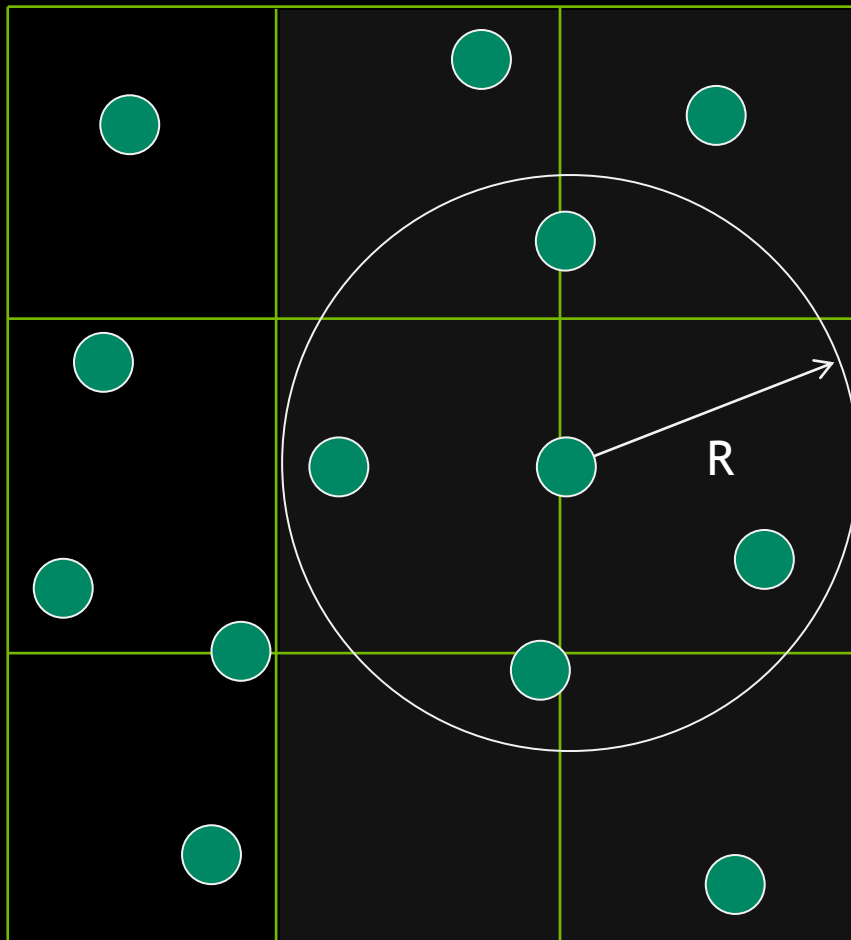


Find all neighbors in a fixed radius R

- Number of neighbors may vary
- May need to find all fixed-radius neighbors of all particles

$O(n^2)$ brute force

Overall Strategy

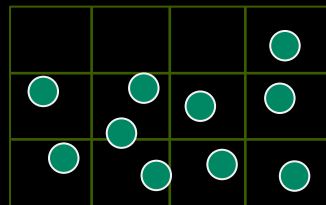


Spatial Partitioning:

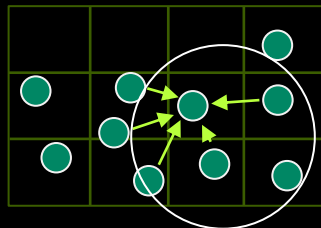
1. Partition space equally into bins
2. Insert each particle into bins
3. Only need to search particles found in neighboring bins

$$O(N k)$$

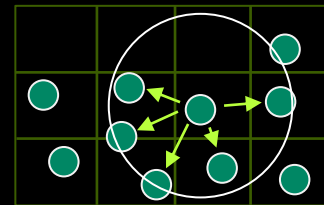
Use in Simulation



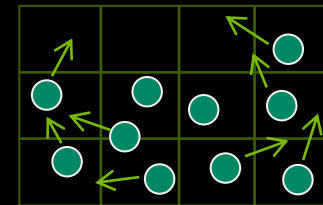
Insert Particles



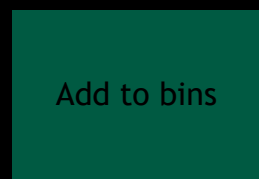
Compute Pressures



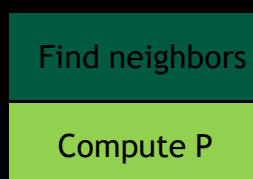
Compute Forces



Integration (Advance)



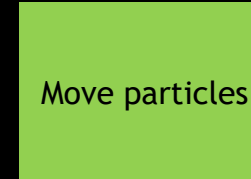
$O(N)$



$O(N(k+D_1))$



$O(N(k+D_2))$



$O(N * D_3)$



Nearest Neighbor Search



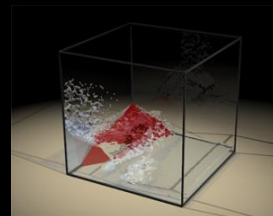
Domain Specific Cost

Research Background

Fluid Simulation



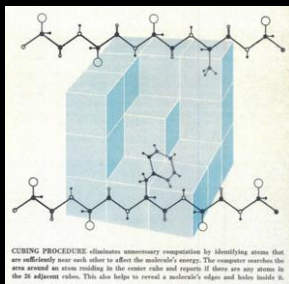
2003 Muller
Particle-Based Fluid Simulation
for Interactive Apps (SPH)



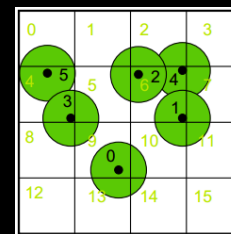
2009 Solenthaler
Predictor-Corrector (PCISPH)



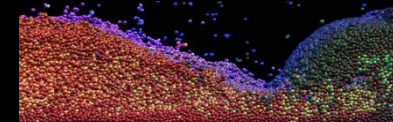
2013 Mathias & Muller
Position Based Fluids



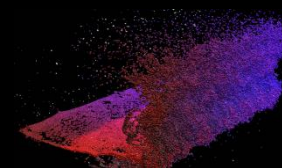
1966 Levinthal
Molecular-Model Building by Computer
"Cubing" method



2010 Simon Green
Particle Simulation using CUDA
Parallel Radix Sort

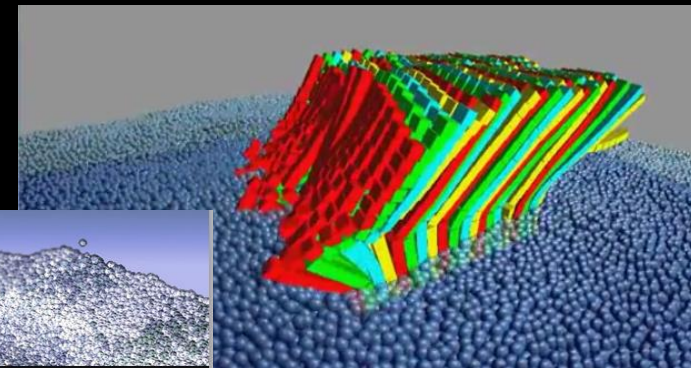
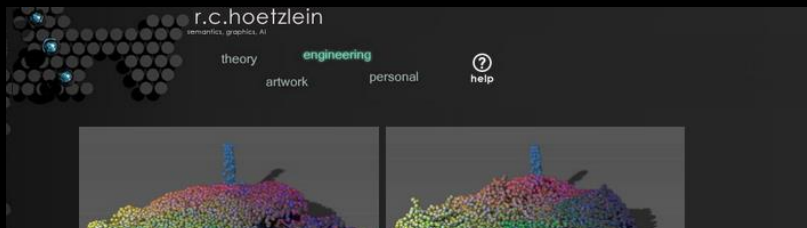


2013 Hoetzlein
Parallel Counting Sort NNS



Fluids v.3 (SPH)

Nearest Neighbor Search



FLUIDS v.2 - A Fast, Open Source, Fluid

Zlib license. CPU & GPU simulator.
 - CPU. Requires basic graphic card (GeForce 256MB or better).
 - GPU. Requires CUDA capable card (GeForce 7000 or better).
 Visual Studio 2008.
 Windows download: [fluids_v2.zip](#)
 Linux download: [fluids_v2.tar.gz](#)
 (Thanks to Fariza Dian Prasetyo, Institute of Technology Sepuluh Nopember)

***NEW* FLUIDS v.3 - A Large-Scale, Open Source Fluid Simulator**
 Fluids v.3, released Dec 2012, is now available. The latest version features up to 8,000,000 particles.

With a history in astrophysics, there is a general perception that FLUIDS is difficult or complex. FLUIDS v.1 was designed to be readable, efficient, and easy to understand. It is currently available. FLUIDS v.1 was designed for research.

The reader is referred to the following instructions for more details.

2006. Micky Kelager (DIKU, Copenhagen).
 2004. Marcus Vesterlund (Umea Univ, Sweden).

Surface Reconstruction:

I've received several emails about surface reconstruction. I'm interested in this research. I've started a web page for discussion. Check it out here: [Surface Reconstruction of 3D Data](#)

Notes for the Novice:

- You've probably implemented a simple, fast fluid simulation. If you have the right inter-particle forces, these particles can

Fluids v.3

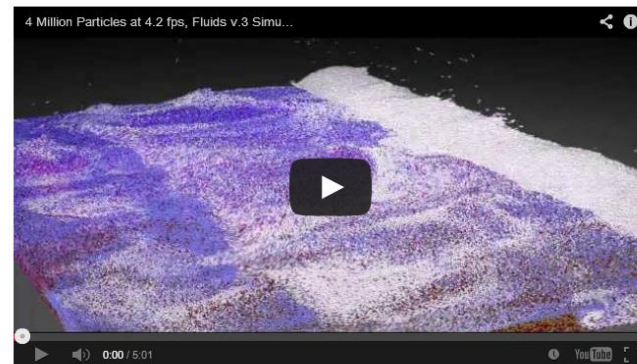
A Large-Scale, Open Source Fluid Simulator

WELCOME DOWNLOAD PERFORMANCE DEVELOPMENT FAQ

Welcome

Fluids v.3 is a large-scale, open source fluid simulator for the CPU and GPU using the smooth particle hydrodynamics method. Fluids is capable of efficiently simulating up to 8 million particles on the GPU (on 1500 MB of ram).

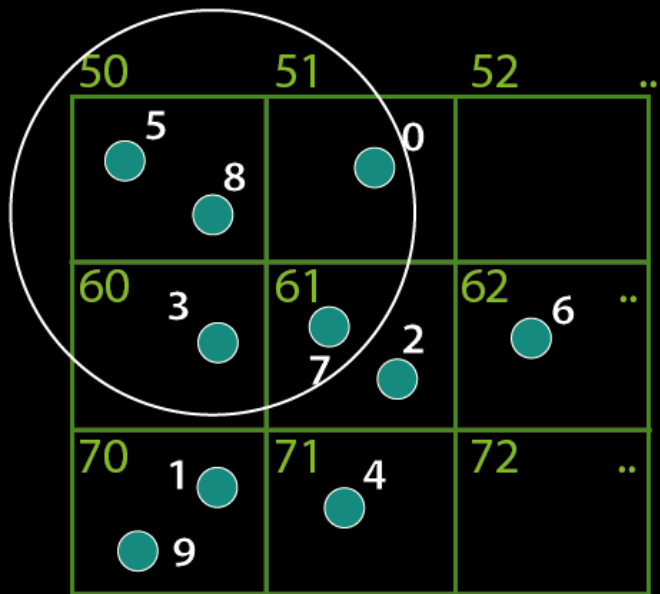
This demo video shows 4 million particles simulated at 1/2 fps. At this rate, 3000 frames are simulated in just 1.5 hours. Published in December 2012, this is the fastest, freely available GPU simulator (for now anyway). See the [Performance](#) page for details.



Fluids v.3 <http://fluids3.com>

High Performance computing for scientific applications.
 “Just the basics”, Zlib license, Research & Education, Solves NNS

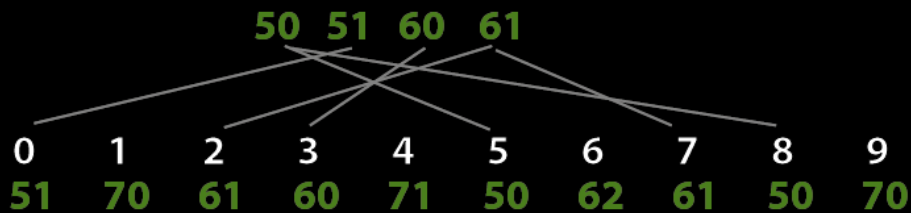
Grid Search



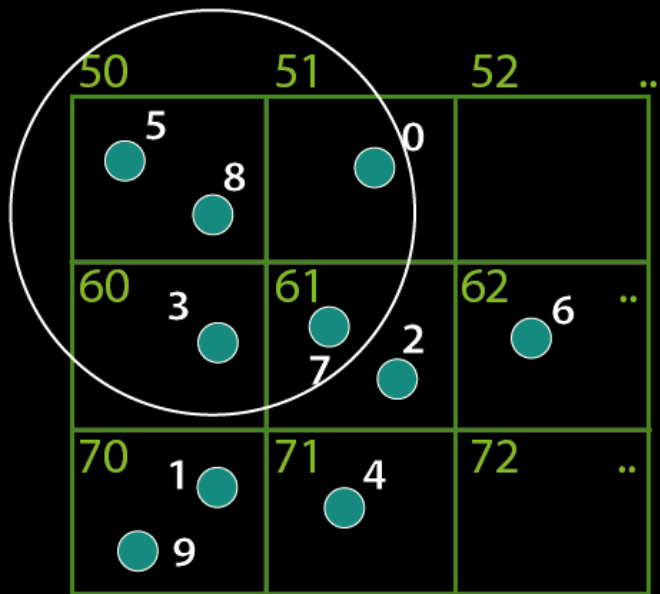
Original Input

0	1	2	3	4	5	6	7	8	9
51	70	61	60	71	50	62	61	50	70

Neighboring Bins



Grid Search

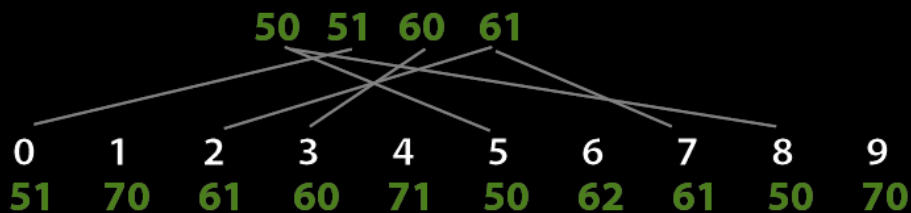


Goal:
Sort on every frame

Original Input

0	1	2	3	4	5	6	7	8	9
51	70	61	60	71	50	62	61	50	70

Neighboring Bins

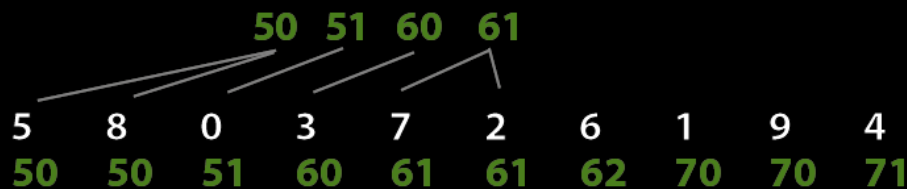


SLOW - SCATTERED READS

Ideal Layout - Sorted by bins

5	8	0	3	7	2	6	1	9	4
50	50	51	60	61	61	62	70	70	71

Neighboring Bins



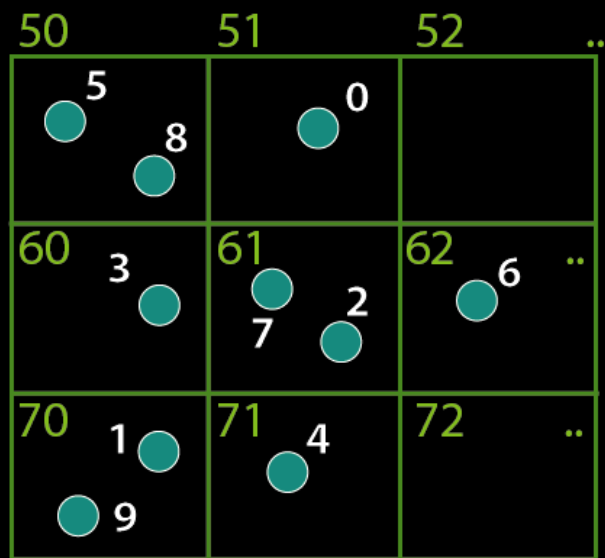
COHERENT READS!

Radix Sort

Previous method (Green 2008)

Input

0	1	2	3	4	5	6	7	8	9
51	70	61	60	71	50	62	61	50	70

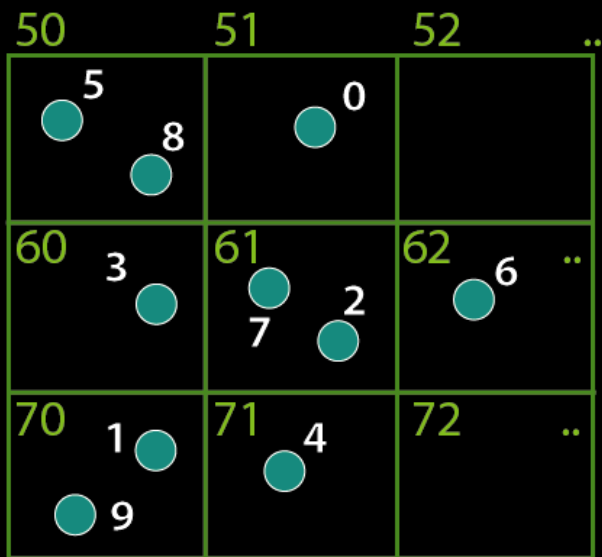


Radix Sort

Previous method (Green 2008)

Input

0	1	2	3	4	5	6	7	8	9
51	70	61	60	71	50	62	61	50	70

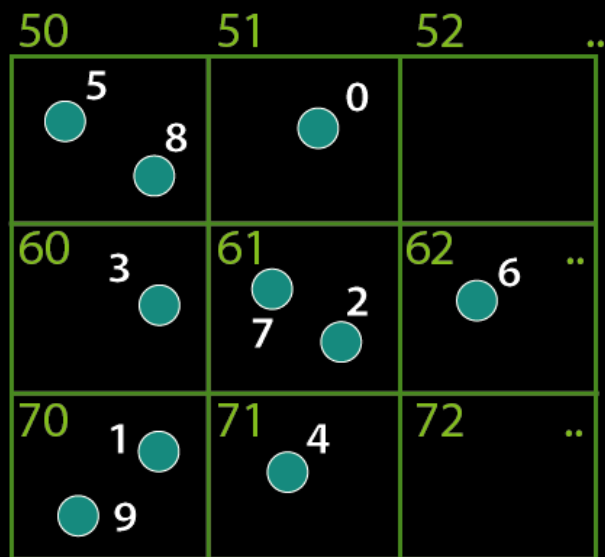


Radix Sort
Digit 0

0	1	2	3	4	5	6	7	8	9
51	70	61	60	71	50	62	61	50	70
	0 = 5			1 = 4			2 = 1		

Radix Sort

Previous method (Green 2008)



Input

0	1	2	3	4	5	6	7	8	9
51	70	61	60	71	50	62	61	50	70

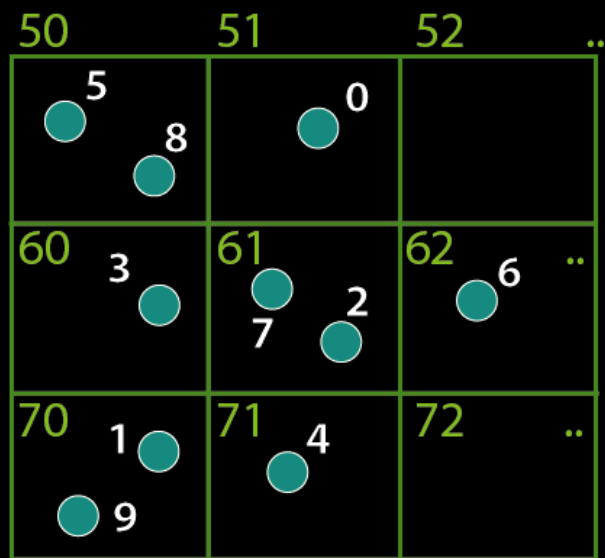
Radix Sort
Digit 0

0	1	2	3	4	5	6	7	8	9
51	70	61	60	71	50	62	61	50	70
	0 = 5			1 = 4			2 = 1		

70	60	50	50	70	51	61	71	61	62
----	----	----	----	----	----	----	----	----	----

Radix Sort

Previous method (Green 2008)



Input

0	1	2	3	4	5	6	7	8	9
51	70	61	60	71	50	62	61	50	70

Radix Sort

Digit 0

0	1	2	3	4	5	6	7	8	9
51	70	61	60	71	50	62	61	50	70
	0 = 5			1 = 4			2 = 1		

70	60	50	50	70	51	61	71	61	62
----	----	----	----	----	----	----	----	----	----

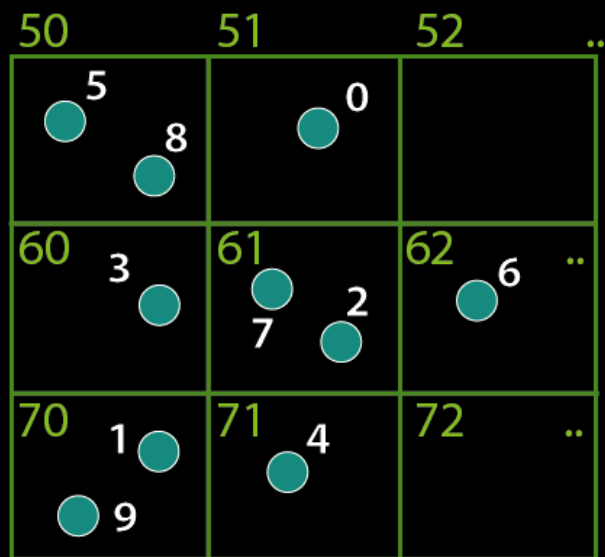
Digit 1

0	1	2	3	4	5	6	7	8	9
70	60	50	50	70	51	61	71	61	62
	5 = 3			6 = 4			7 = 3		

50	50	51	60	61	61	62	70	70	71
----	----	----	----	----	----	----	----	----	----

Radix Sort

Previous method (Green 2008)



Input

0	1	2	3	4	5	6	7	8	9
51	70	61	60	71	50	62	61	50	70

Radix Sort
Digit 0

0	1	2	3	4	5	6	7	8	9
51	70	61	60	71	50	62	61	50	70
	0 = 5			1 = 4			2 = 1		

70	60	50	50	70	51	61	71	61	62
----	----	----	----	----	----	----	----	----	----

Digit 1

0	1	2	3	4	5	6	7	8	9
70	60	50	50	70	51	61	71	61	62
	5 = 3			6 = 4			7 = 3		

50	50	51	60	61	61	62	70	70	71
----	----	----	----	----	----	----	----	----	----

Repeat step for each digit in key

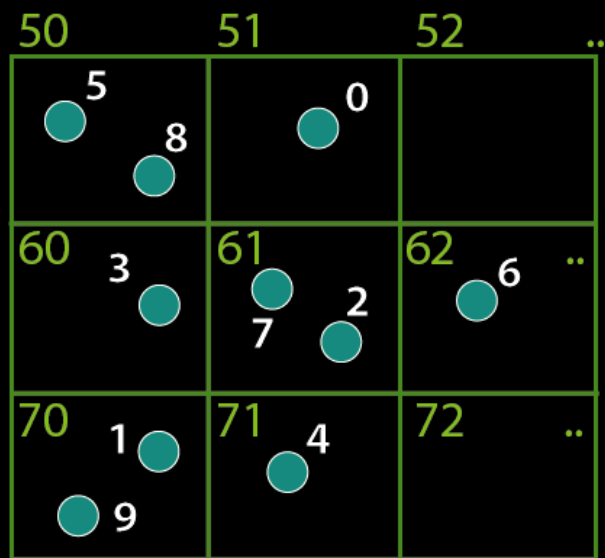
Output

5	8	0	3	7	2	6	1	9	4
50	50	51	60	61	61	62	70	70	71

Counting Sort

Input

0	1	2	3	4	5	6	7	8	9
51	70	61	60	71	50	62	61	50	70



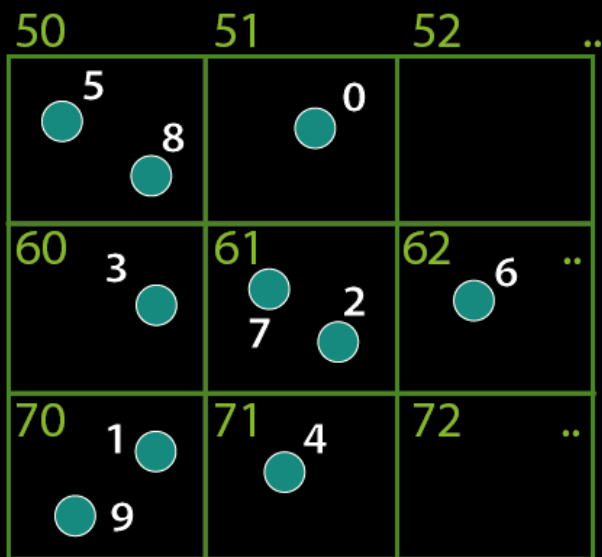
Observations:

A full sort on key is not necessary.
Inside a bin, all keys will be duplicated.
Order in bin not an issue.

Counting Sort

Input

0	1	2	3	4	5	6	7	8	9
51	70	61	60	71	50	62	61	50	70



Insert + Counts

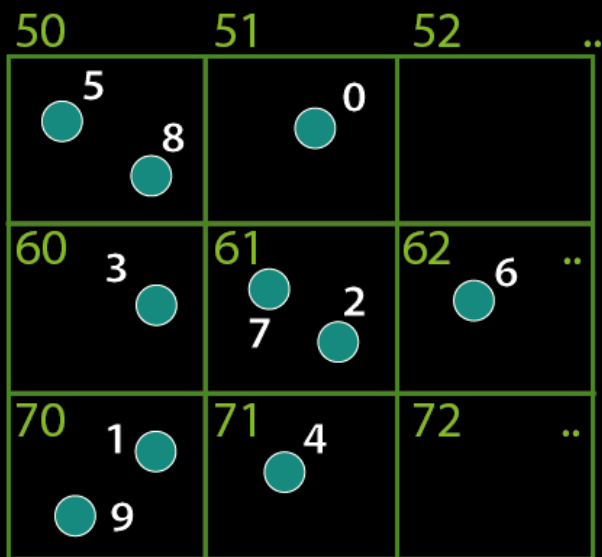
0	1	2	3	4	5	6	7	8	9
51	70	61	60	71	50	62	61	50	70
1	1	1	1	1	1	1	2	2	2

50 = 2	60 = 1	70 = 2
51 = 1	61 = 2	71 = 1
52 = 0	62 = 1	72 = 0

Counting Sort

Input

0	1	2	3	4	5	6	7	8	9
51	70	61	60	71	50	62	61	50	70



Insert + Counts

0	1	2	3	4	5	6	7	8	9
51	70	61	60	71	50	62	61	50	70
1	1	1	1	1	1	1	2	2	2
<p>50 = 2 60 = 1 70 = 2</p> <p>51 = 1 61 = 2 71 = 1</p> <p>52 = 0 62 = 1 72 = 0</p>									

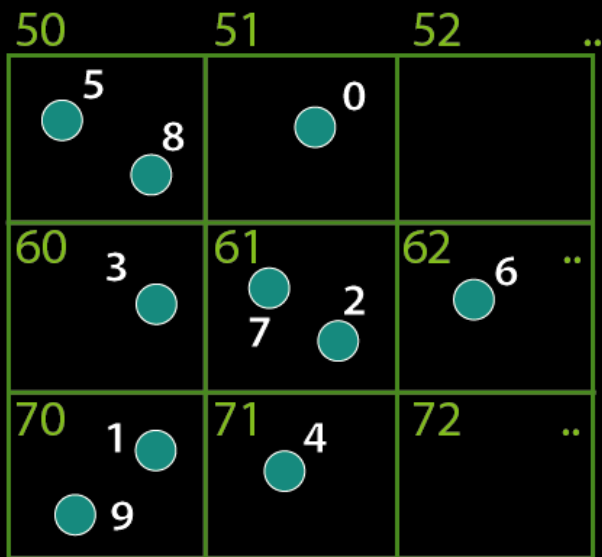
Prefix Sum

50	50	51	60	61	61	62	70	70	71
0		2	3	4		6	7		9

Counting Sort

Input

0	1	2	3	4	5	6	7	8	9
51	70	61	60	71	50	62	61	50	70



Insert + Counts

0	1	2	3	4	5	6	7	8	9
51	70	61	60	71	50	62	61	50	70
1	1	1	1	1	1	1	2	2	2

50 = 2 60 = 1 70 = 2
 51 = 1 61 = 2 71 = 1
 52 = 0 62 = 1 72 = 0

Prefix Sum

50	50	51	60	61	61	62	70	70	71
0	2	3	4	6	7	9			

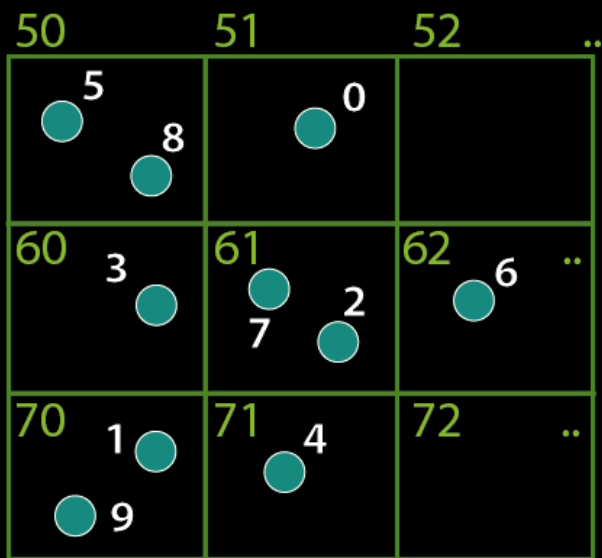
Counting Sort

0	1	2	3	4	5	6	7	8	9
		1					2		
5	8	0	3	2	7	6	1	9	4
50	50	51	60	61	61	62	70	70	71

Counting Sort

Input

0	1	2	3	4	5	6	7	8	9
51	70	61	60	71	50	62	61	50	70



Insert + Counts

0	1	2	3	4	5	6	7	8	9
51	70	61	60	71	50	62	61	50	70
1	1	1	1	1	1	1	2	2	2

50 = 2	60 = 1	70 = 2
51 = 1	61 = 2	71 = 1
52 = 0	62 = 1	72 = 0

Prefix Sum

50	50	51	60	61	61	62	70	70	71
0		2	3	4		6	7		9

Counting Sort

0	1	2	3	4	5	6	7	8	9
5	8	0	3	2	7	6	1	9	4
50	50	51	60	61	61	62	70	70	71

Diagram showing arrows from index 2 to index 4 and from index 7 to index 6, indicating the placement of elements.

Output

5	8	0	3	2	7	6	1	9	4
50	50	51	60	61	61	62	70	70	71

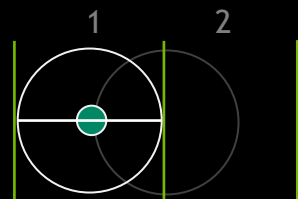
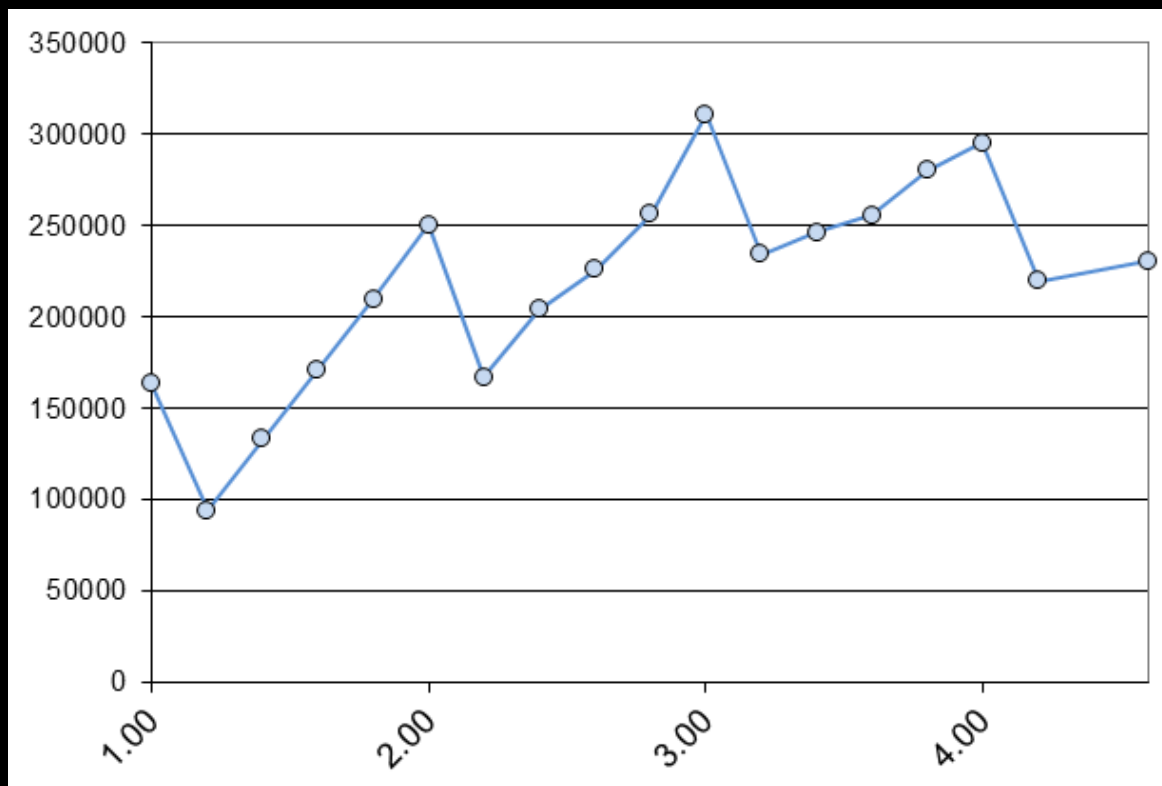
What is a good Grid Cell Size?

Too many particles per cell

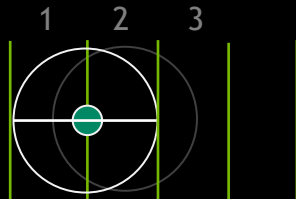
*e.g.
 $p = 200+$ / cell*

Too many grid cells to check

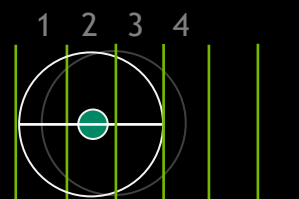
*e.g.
 $5 \times 5 \times 5 = 125$ cells*



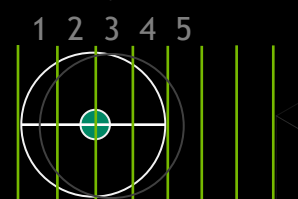
$$h=d/1.00 > 2$$



$$h=d/2.00 > 3$$



$$h=d/3.00 > 4$$



$$h=d/4.00 > 5$$

Performance Comparison

CUDA Particles (Radix Sort)

calcHashD
 assign particle to cell

sortParticles
 thrust:sort_by_key
 example: CUDA RadixSort
 for 1 to 4 (each byte in key)
 RadixSum
 RadixPrefixSum
 RadixAddOffsetAndShuffle

reorderDataAndFindCellStart
 copy particles in sorted order

collided

integrateSystem
 thrust::for_each

Fluids v.3 (Counting Sort)

InsertParticles
 assign particle to cell
 AtomicAdd for bin counts & particle indices

CountingSort
 PrefixSumInit, compute bin offsets

CountingSortIndex
 copy particles in sorted order

Collisions

Advance
 integrate system

Performance Comparison

CUDA Particles (Radix Sort)

- 1 calcHashD
assign particle to cell
- sortParticles
 - thrust:sort_by_key
 - example: CUDA RadixSort
 - for 1 to 4 (each byte in key)
- 4 RadixSum
- 4 RadixPrefixSum
- 4 RadixAddOffsetAndShuffle
- 1 reorderDataAndFindCellStart
copy particles in sorted order
- 1 collided
- 1 integrateSystem
thrust::for_each

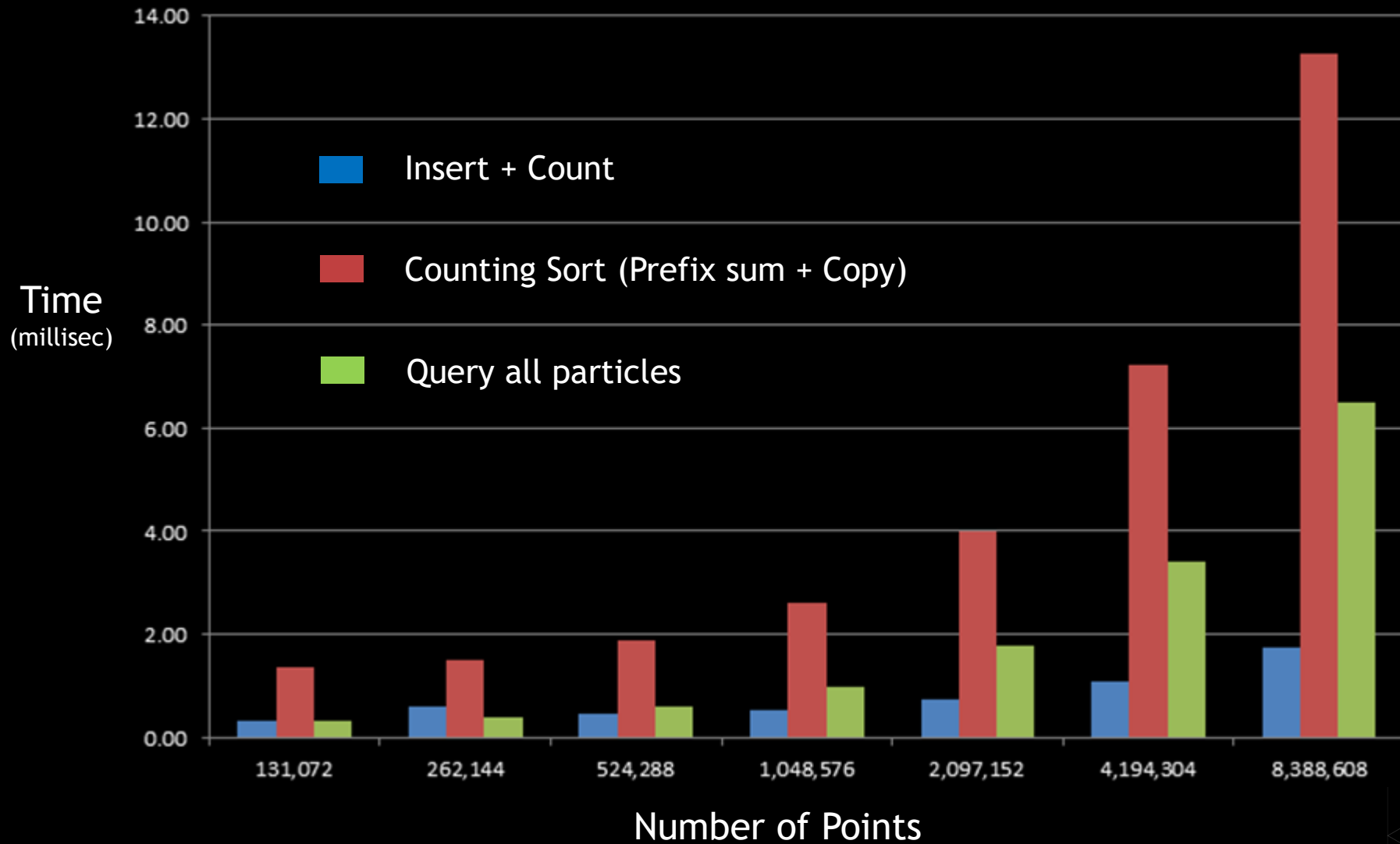
16 Kernel calls / Frame

Fluids v.3 (Counting Sort)

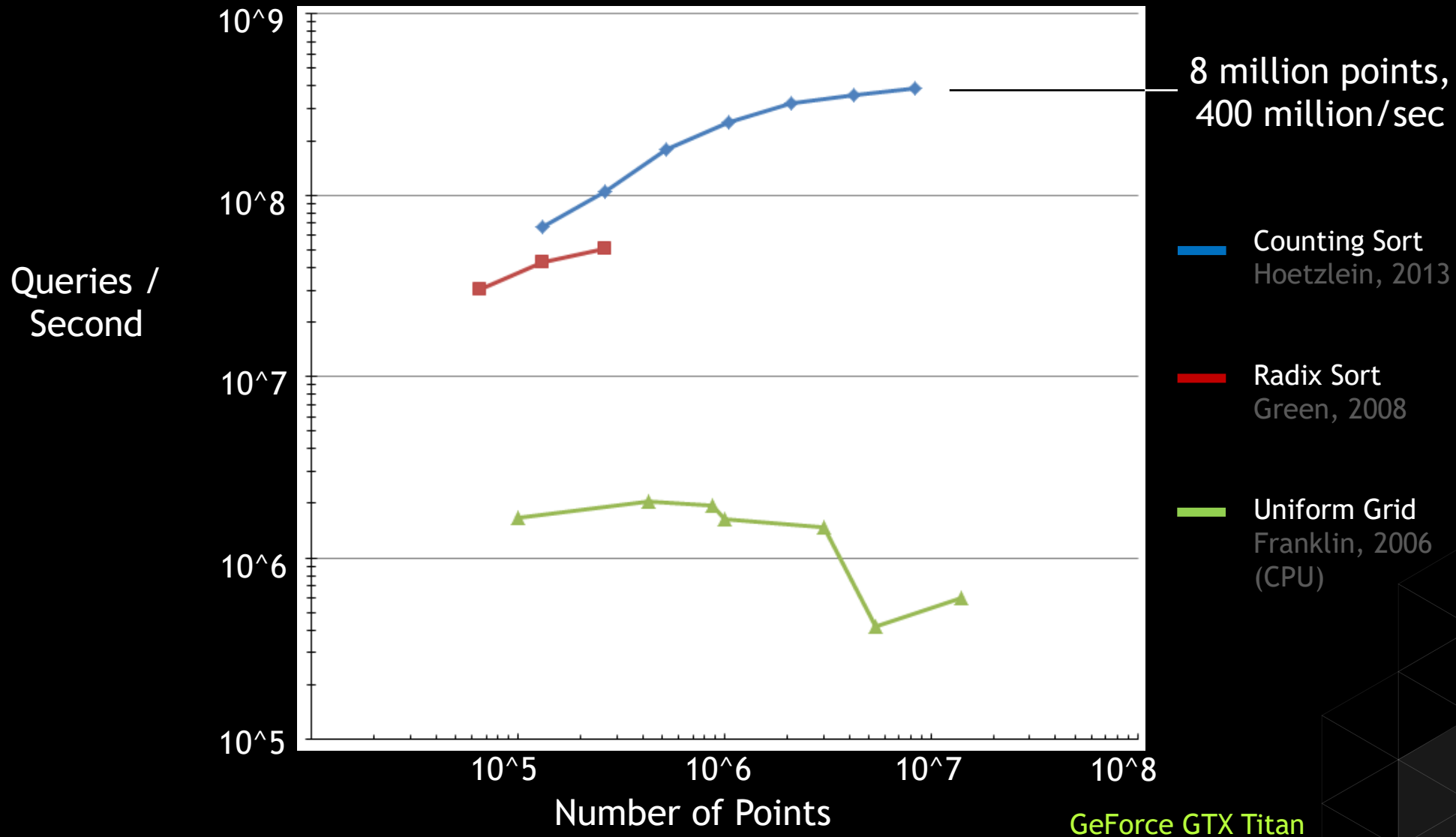
- 1 InsertParticles
assign particle to cell
→ **AtomicAdd for bin counts & particle indices**
- CountingSort
 - 1 PrefixSumInit, compute bin offsets
- CountingSortIndex
 - 1 copy particles in sorted order
- 1 Collisions
- Advance
 - 1 integrate system

5 Kernel calls / Frame

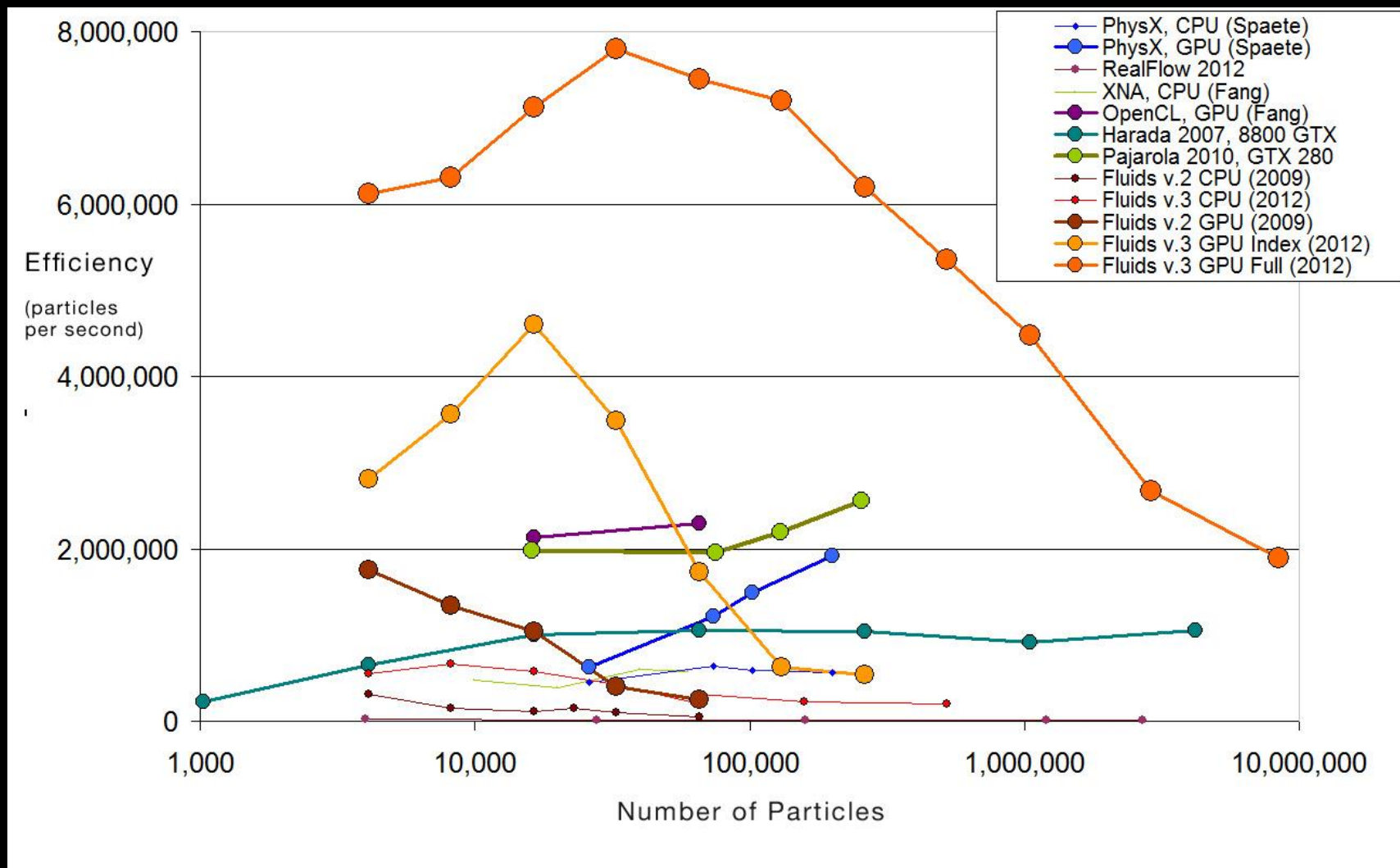
Results (NNS)



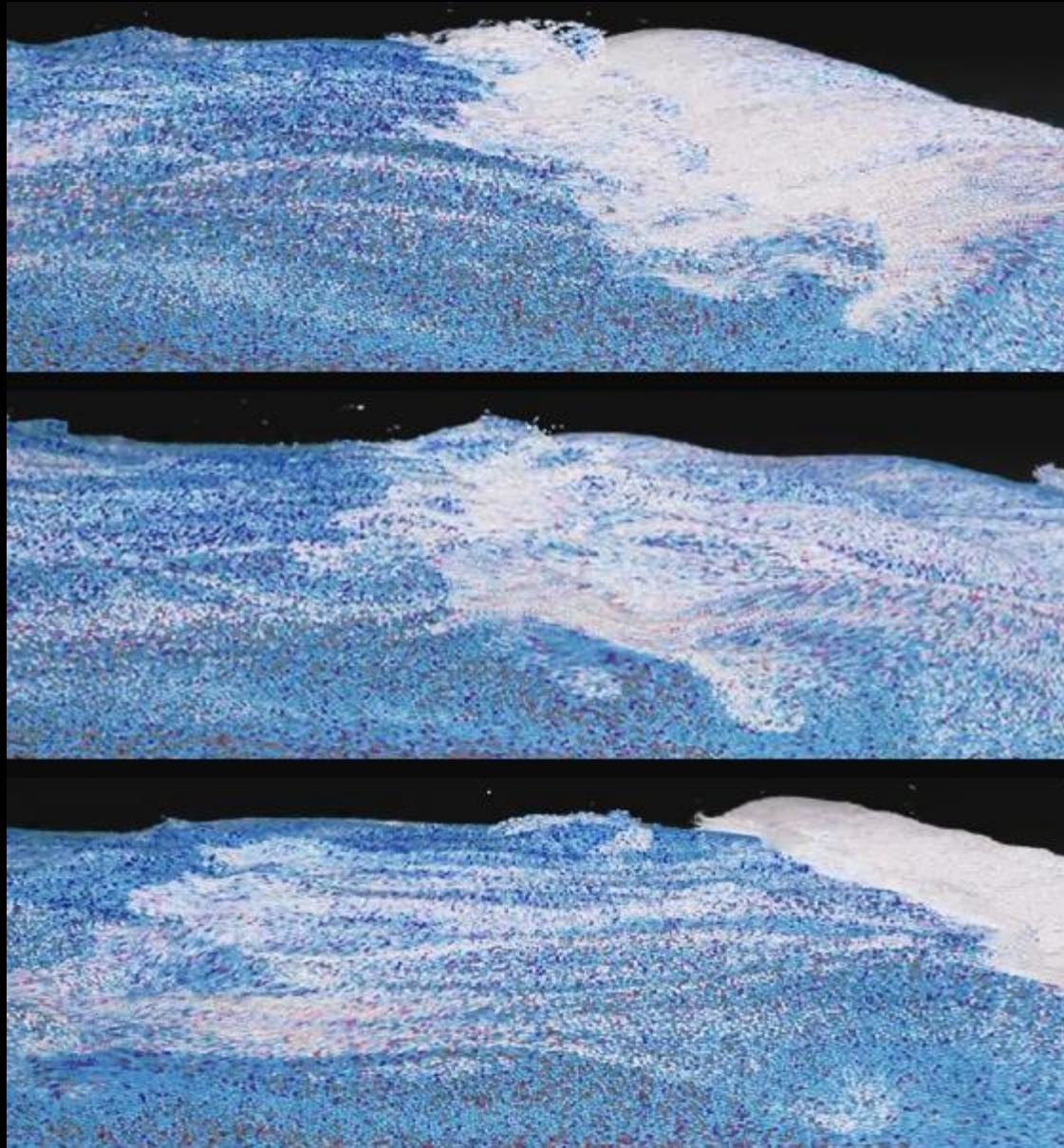
Results (NNS)



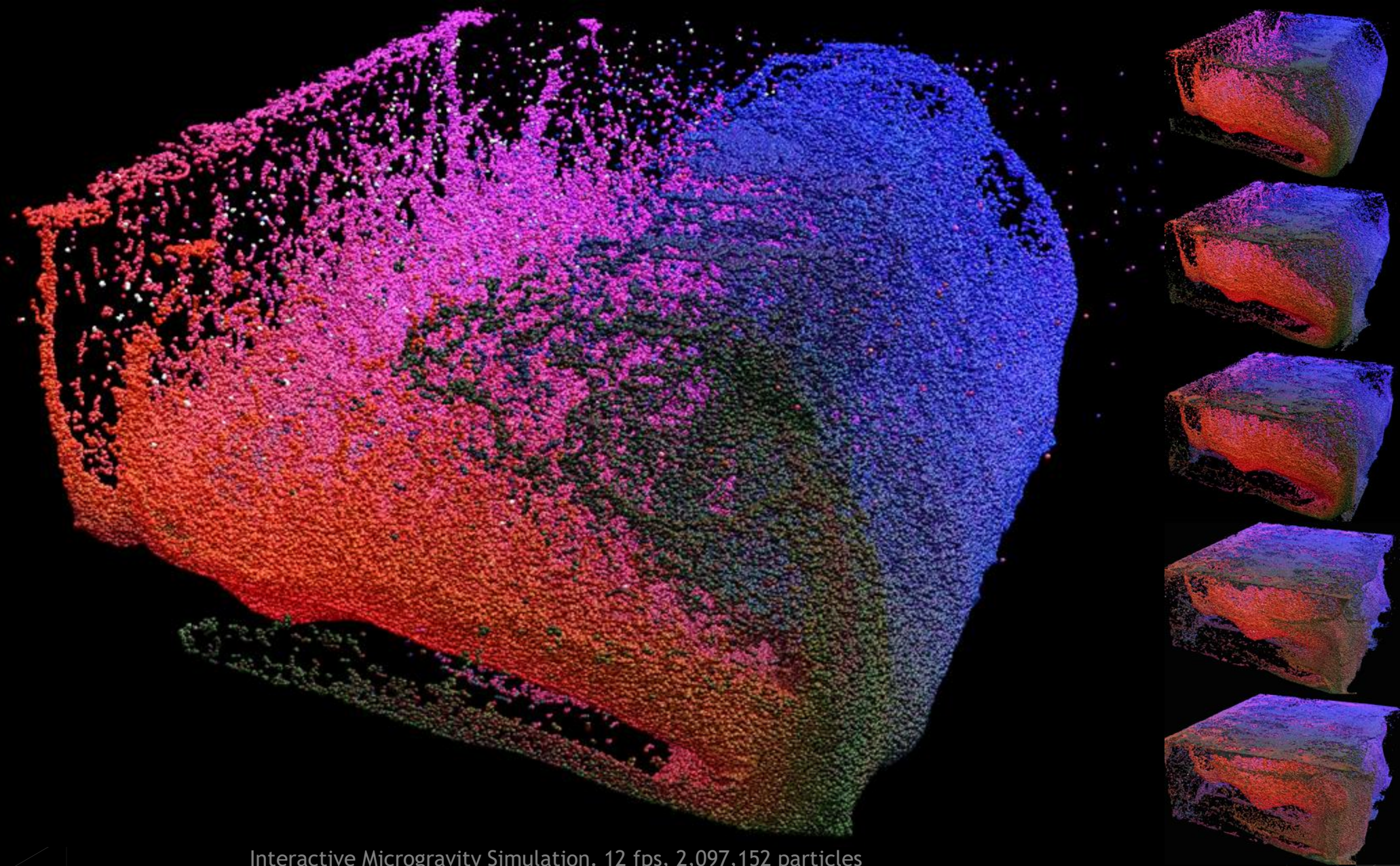
Results (Fluids)



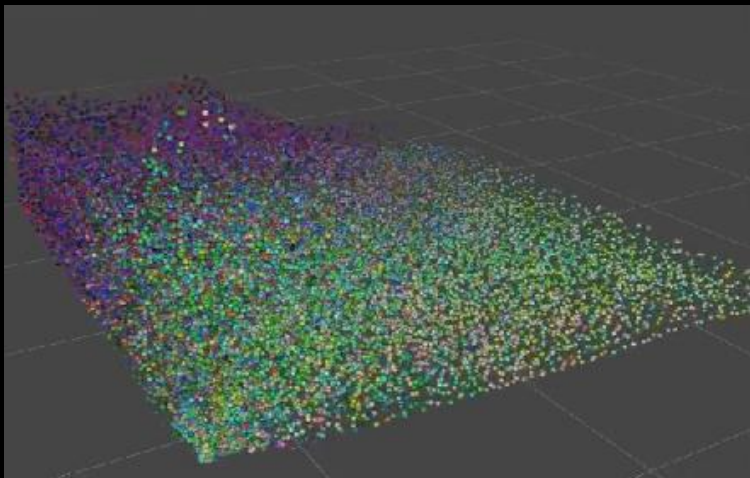
* Results do not show fluid accuracy or time step.



Continuous Ocean Simulation, 4.2 fps, 4,194,304 particles

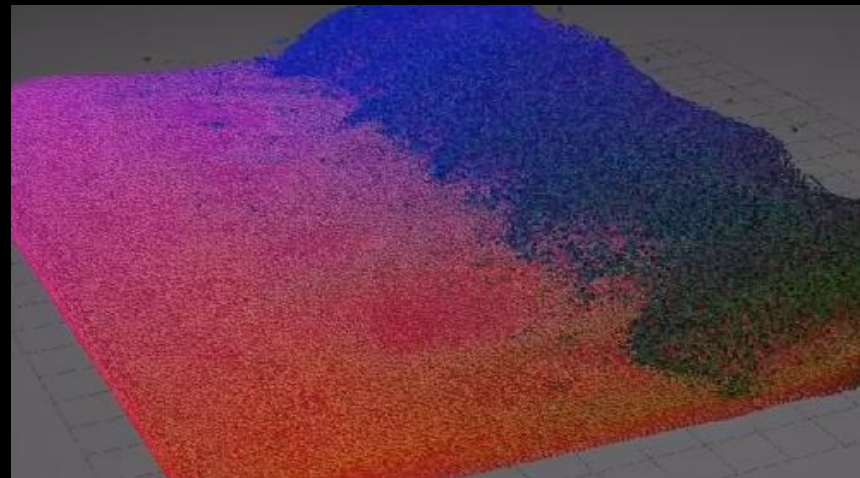


Interactive Microgravity Simulation. 12 fps, 2,097,152 particles



Fluids v.2, 2009
16,384 at 32 fps

same hardware
11x faster



Fluids v.3, 2013
193,487 at 32 fps

GeForce GTX 460M

Thank you!

Rama C. Hoetzlein

<http://ramakarl.com/fluids3>