

GPU TECHNOLOGY
CONFERENCE

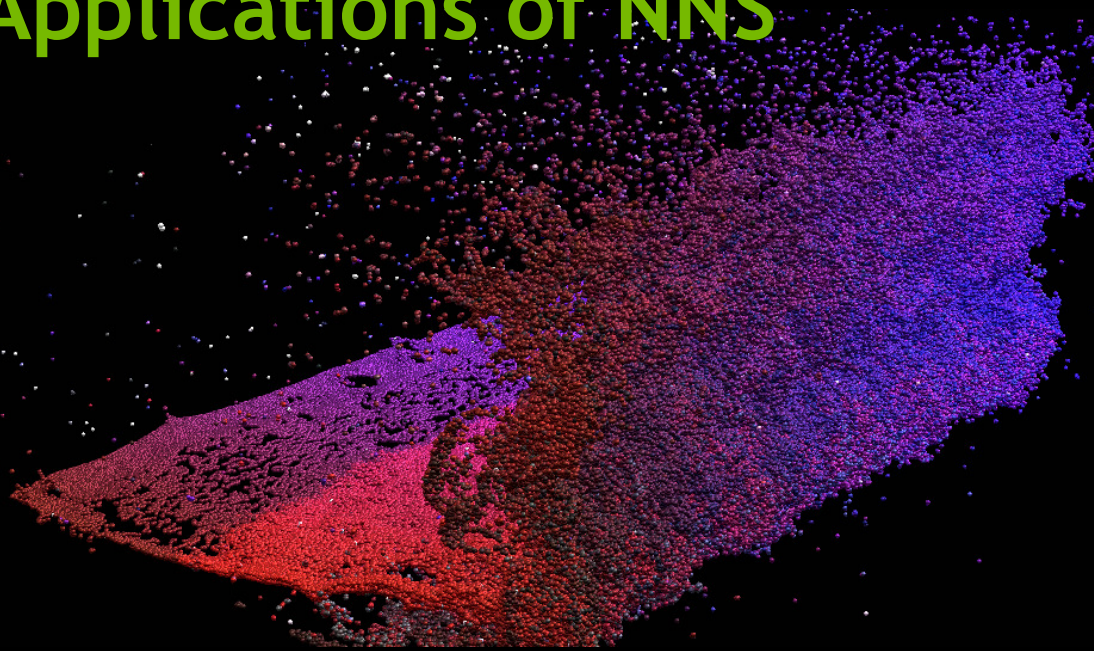


FAST FIXED-RADIUS NEAREST NEIGHBORS: INTERACTIVE MILLION-PARTICLE FLUIDS

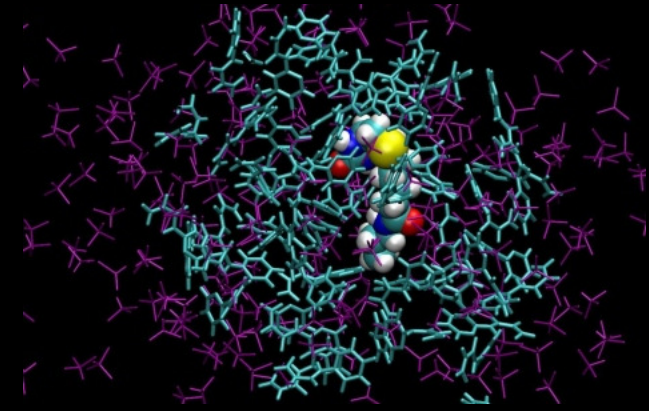
Rama C. Hoetzlein, Graphics Devtech, NVIDIA



Applications of NNS



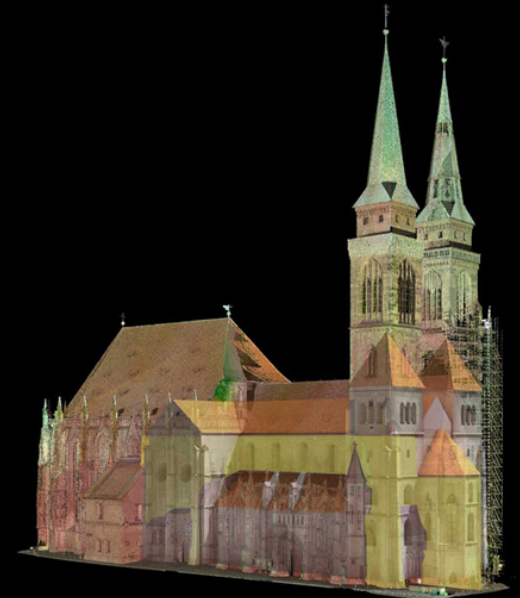
Fluid Simulation



Molecular Modeling

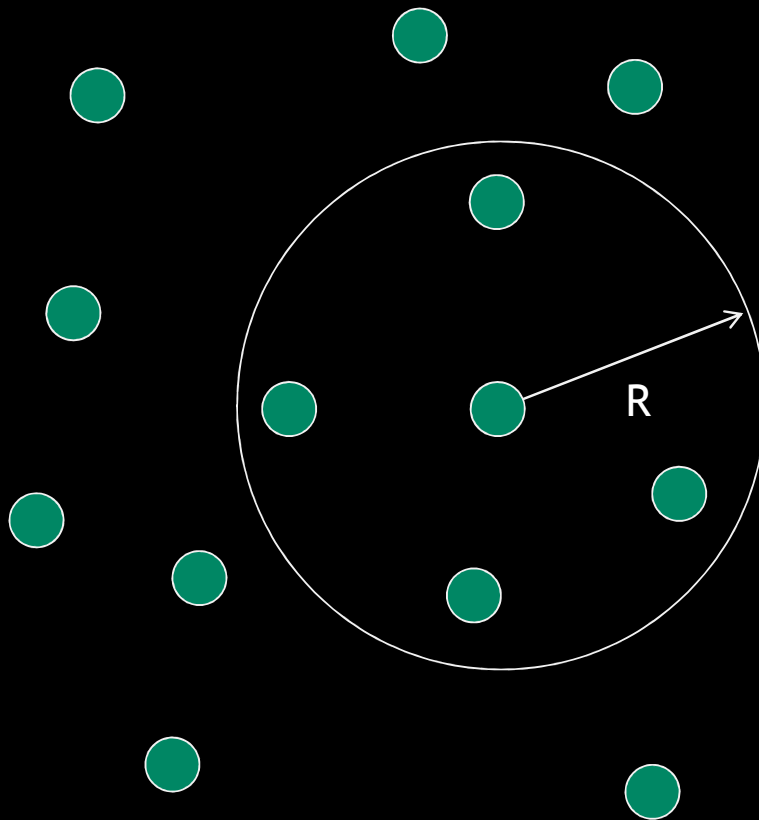


Cognitive Science - Behavioral simulation



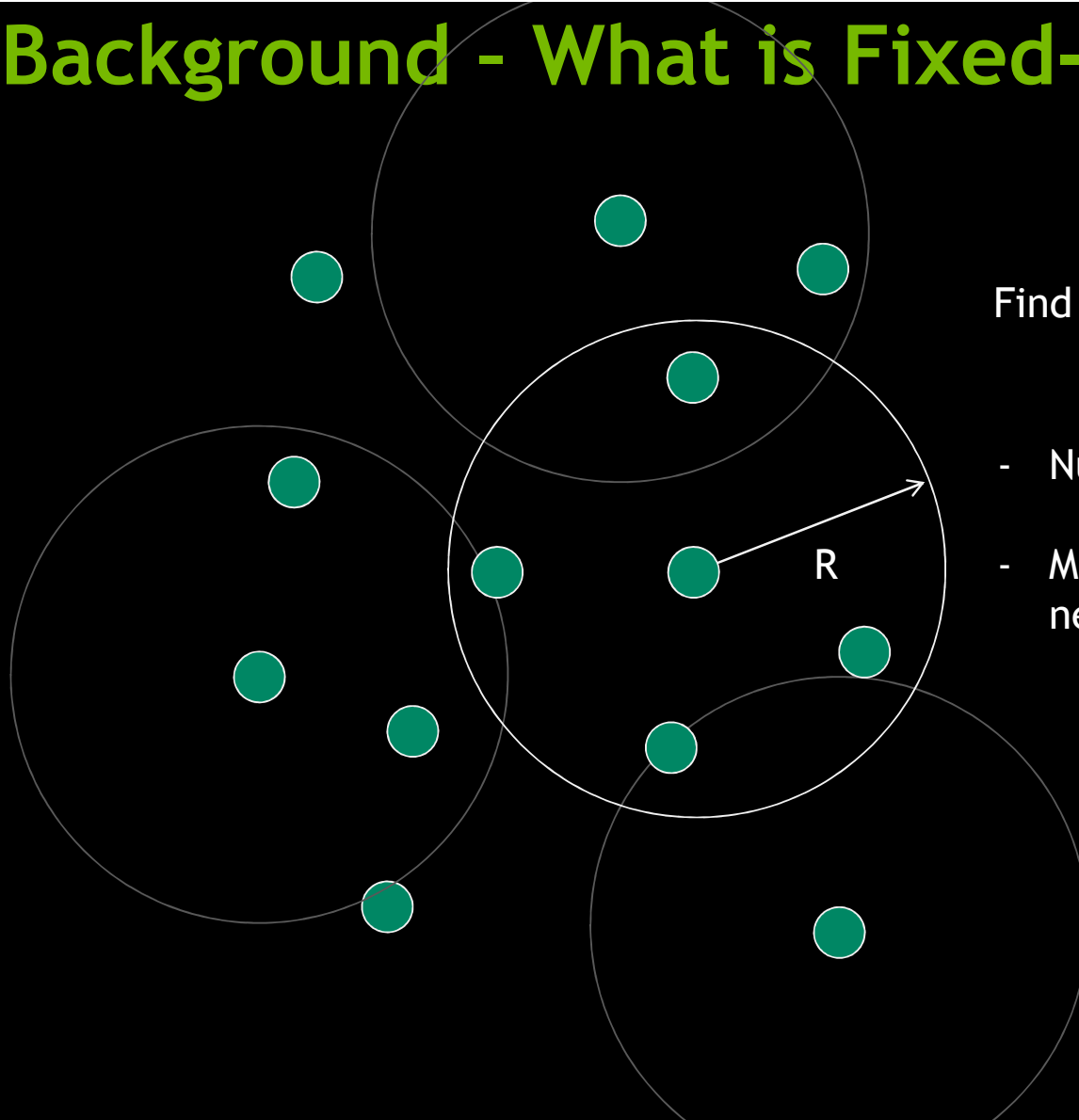
3D Scanning - Point Cloud Reconstruction

Background - What is Fixed-Radius NNS?



Find all neighbors in a fixed radius R

Background - What is Fixed-Radius NNS?

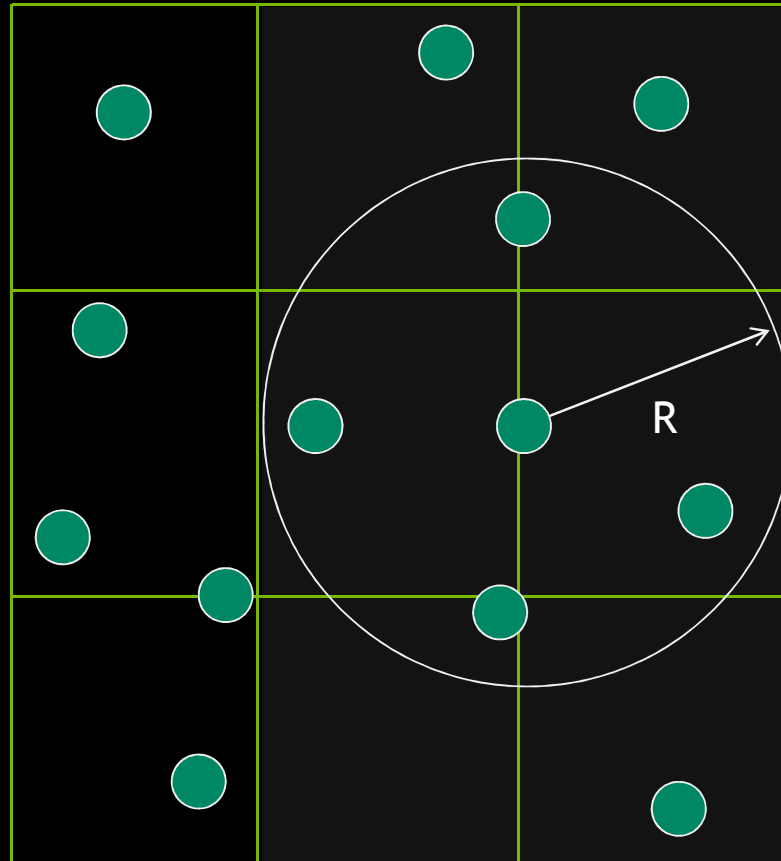


Find all neighbors in a fixed radius R

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

$O(n^2)$ brute force

Overall Strategy

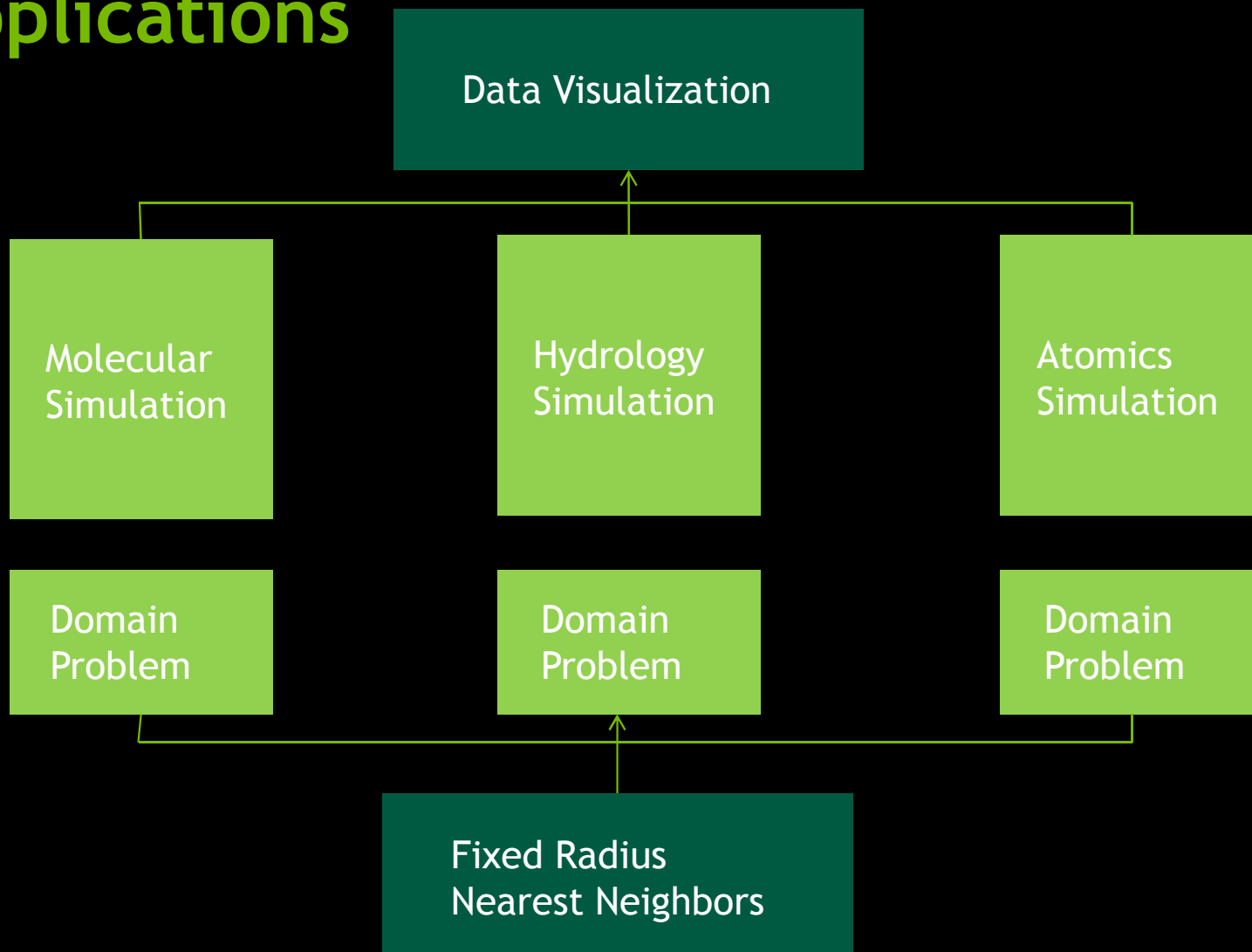


Spatial Partitioning:

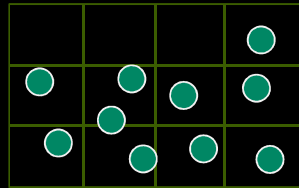
1. Partition space equally into bins
2. Insert each particle into bins
3. Only need to search particles found in neighboring bins

$$O(Nk)$$

Applications



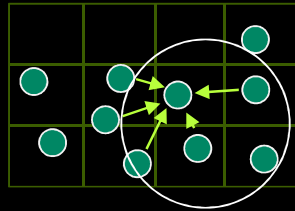
Applications - Example



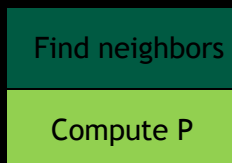
Insert Particles



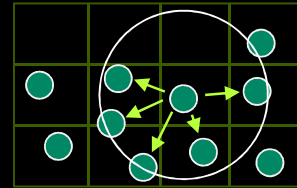
$O(N)$



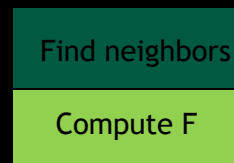
Compute Pressures



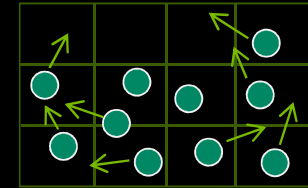
$O(N(k+D_1))$



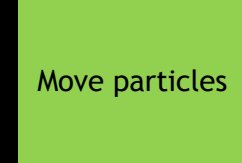
Compute Forces



$O(N(k+D_2))$



Integration (Advance)



$O(N * D_3)$

Nearest Neighbor Search

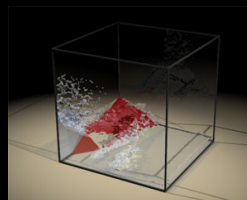
Domain Specific Cost

Research Background

Fluid Simulation



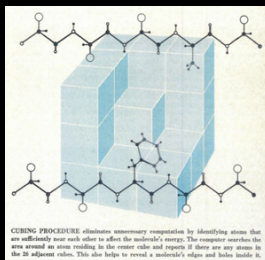
2003 Muller
Particle-Based Fluid Simulation for Interactive Apps (SPH)



2009 Solenthaler
Predictor-Corrector (PCISPH)



2013 Macklin & Muller
Position Based Fluids



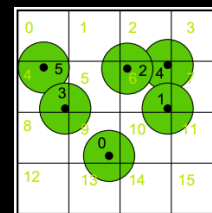
1966 Levinthal
Molecular-Model Building by Computer "Cubing" method

Nearest Point Query on 184,088,599 Points in E^3 with a Uniform Grid
W. Randolph Franklin, Member, IEEE

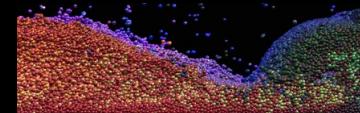
Abstract—NAQT3 is an algorithm and implementation to preprocess more than 10^8 fixed points in E^3 and then perform nearest point queries against them. With fixed and query points drawn from the same distribution, NAQT3's expected preprocessing and query time are $O(N)$, per point, with a very small constant factor. The data structure is a uniform grid in E^3 , typically with the same number of grid cells as points. The storage budget for

implemented with several different data structures, based on kd-trees and box-decomposition trees. Murphy and Skiena's Ranger uses 1-4 trees. [15], [16]. Krivos also has a recent fast implementation with kd-trees. [11]. Finally, if successive queries are close, perhaps one can efficiently traverse the dataset from one answer to the next.

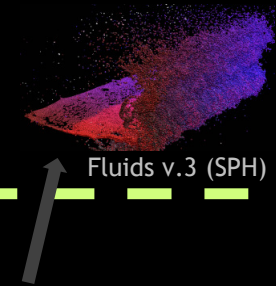
2006 Randolph Franklin
Nearest Point Query on 184,088,599 Points in E^3 with a Uniform Grid. CPU Grid Search



2010 Simon Green
Particle Simulation using CUDA Parallel Radix Sort



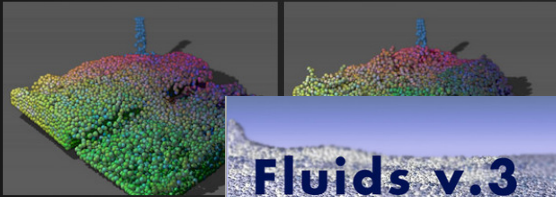
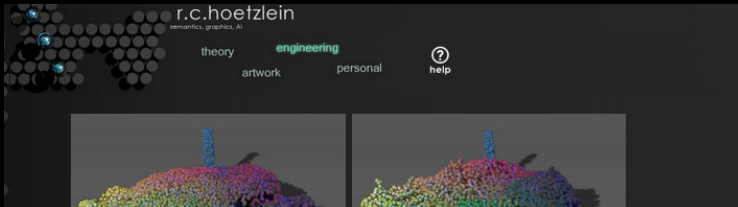
2013 Hoetzlein
Parallel Counting Sort NNS



Fluids v.3 (SPH)

Nearest Neighbor Search

GPU-based NNS



Fluids v.3

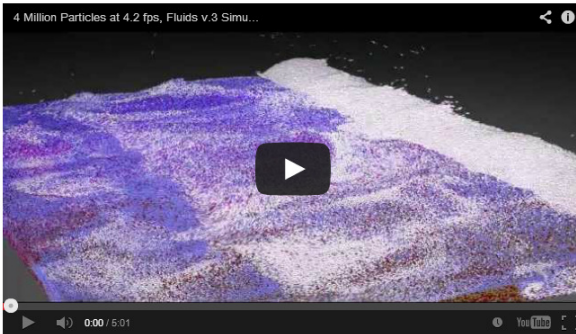
A Large-Scale, Open Source Fluid Simulator

- 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.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 it is difficult or complex. FLUIDS v.1 was developed to be readable, efficient, and easy to understand. It is currently available. FLUIDS v.1 was designed to be easy to use.

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

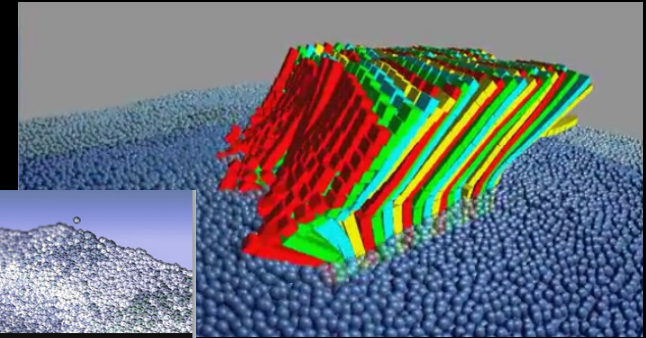
2006. Micky Kelager (DIKU, Copenhagen),
2004. Marcus Vesterlund (Umea Univ, Sweden)

Surface Reconstruction:

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

Notes for the Novice:

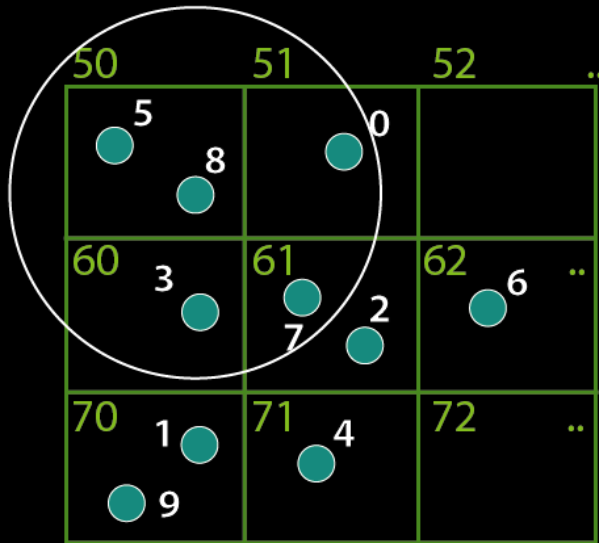
- You've probably implemented a simple, fast fluid simulator. If you have the right inter-particle forces, these particles will



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

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

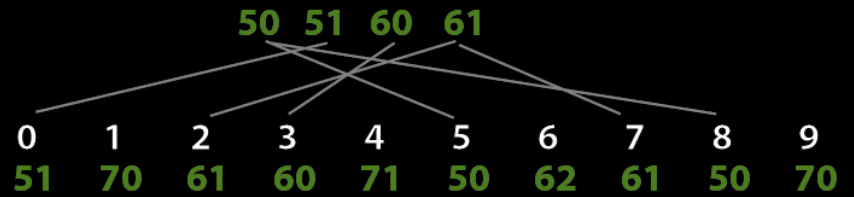
Grid Search



Original Input

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

Neighboring Bins

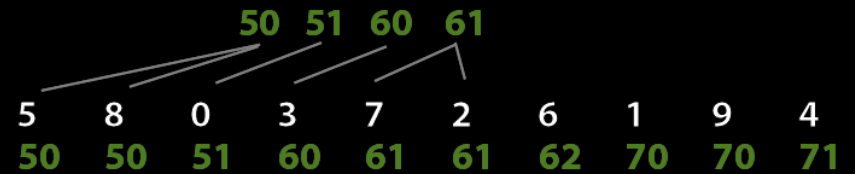


SLOW - SCATTERED READS

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

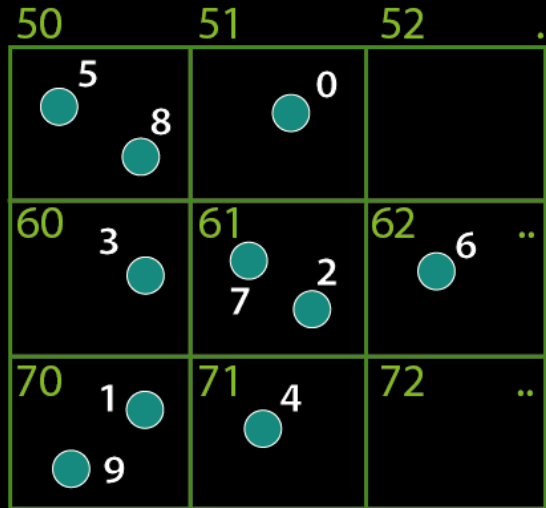
Neighboring Bins



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

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

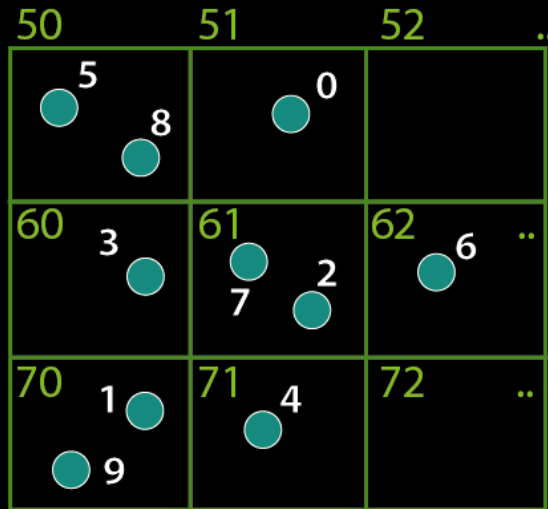
Output

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

Counting Sort

Input

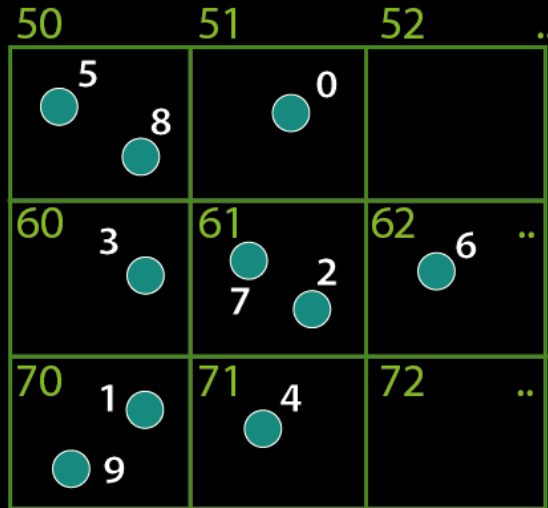
0	1	2	3	4	5	6	7	8	9
51	70	61	60	71	50	62	61	50	70



Observations:

1. Goal is to sort by bin
2. Position inside bin is irrelevant
3. Therefore, many duplicate keys
4. Better to perform 1-radix on exact bins, rather than on digits.

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

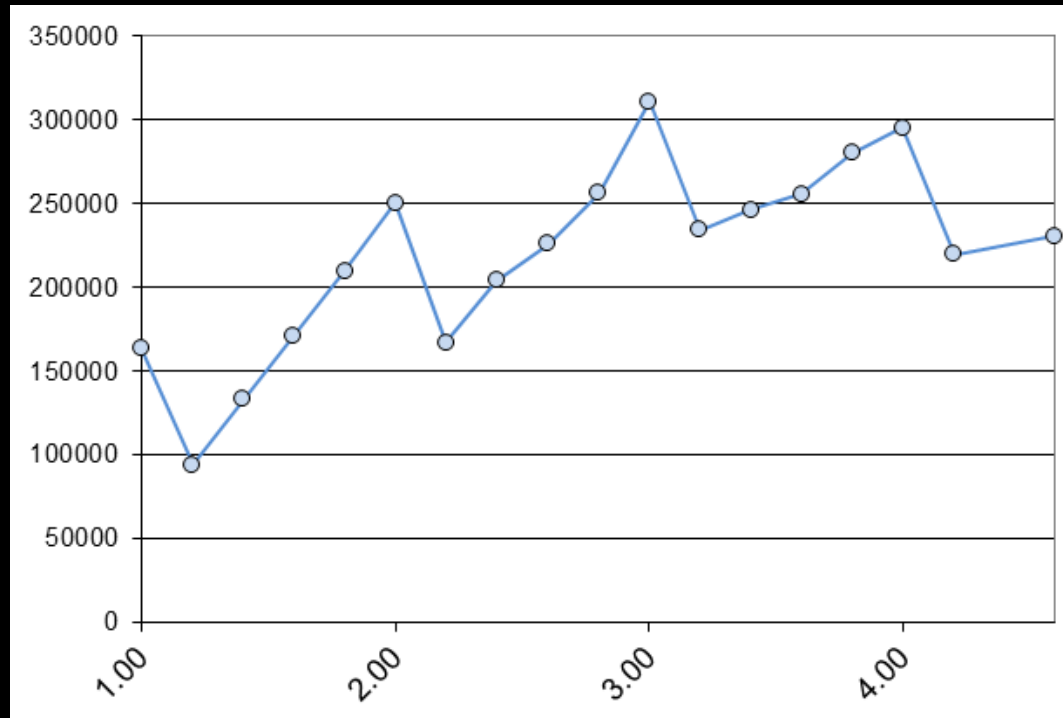
Output

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?

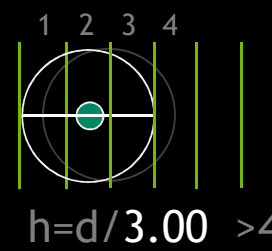
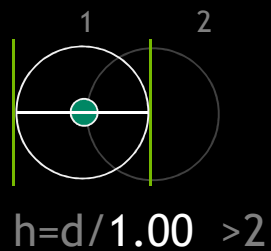
Too many particles per cell

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



Too many grid cells to check

*e.g.
5x5x5 = 125 cells*



Algorithm Comparison

CUDA Particles (Radix Sort)

Insert Particles

assign particle to cell

Sort Particles

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

Bin Counts
Bin Prefix Sum
RadixAddOffsetAndShuffle

Reindex (copy particles in order)

Time integration

Fluids v.3 (Counting Sort)

Insert Particles

assign particle to cell

Sort Particles

Bin Sums
Bin Prefix Sum
~~Radix-Offset-and-Shuffle~~

ReIndex (copy particles in order)

Time integration

Algorithm Comparison

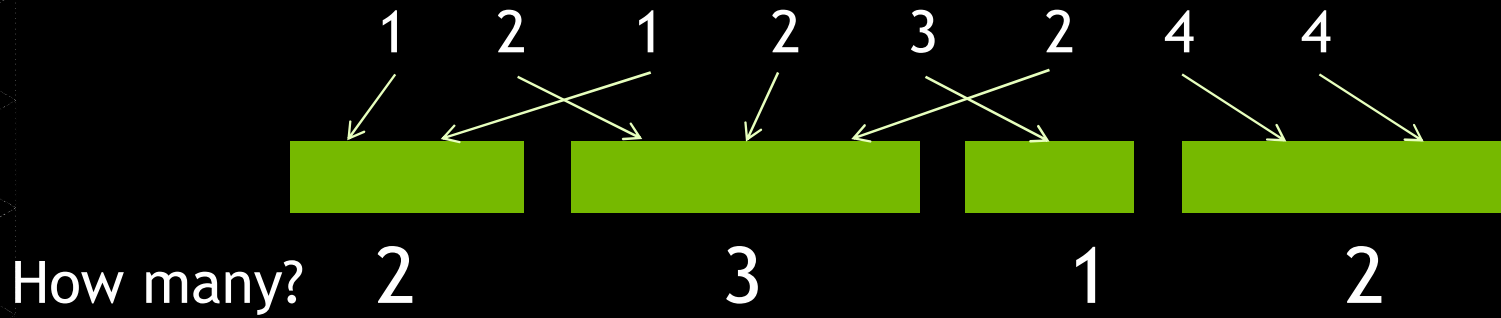
CUDA Particles (Radix Sort)

- 1 Insert Particles
assign particle to cell
- Sort Particles
thrust:sort_by_key
example: CUDA RadixSort
for 1 to 4 (each byte in key)
- 4 Bin Counts
- 4 Bin Prefix Sum
- 4 RadixAddOffsetAndShuffle
- 1 Reindex (copy particles in order)
- 1 Time integration
- 15 Kernel calls / Frame**

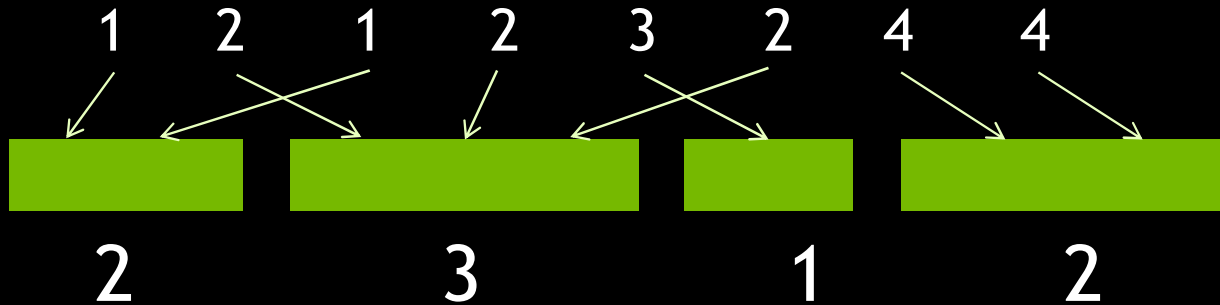
Fluids v.3 (Counting Sort)

- 1 Insert Particles
assign particle to cell
AtomicAdd for Bin Counts & Indices
- Sort Particles
- ~~Bin Sums~~
- 1 Bin Prefix Sum
- ~~Radix-Offset-and-Shuffle~~
- 1 ReIndex (copy particles in order)
- 1 Time integration
- 4 Kernel calls / Frame**

Bin Counting

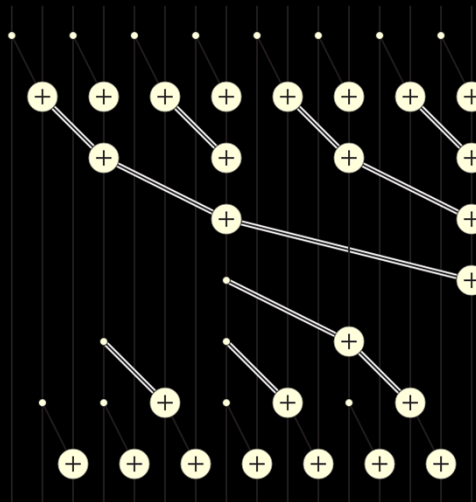


Bin Counting



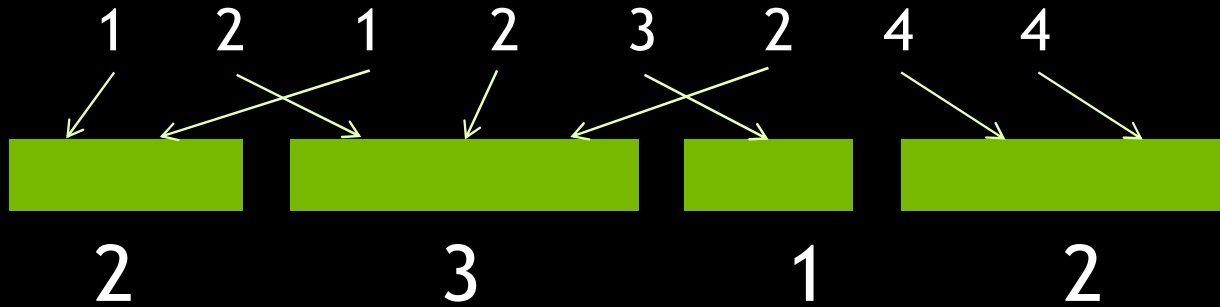
Old Method

Parallel Sums



Each GPU thread computes binary sets of sums.

Bin Counting



New Method

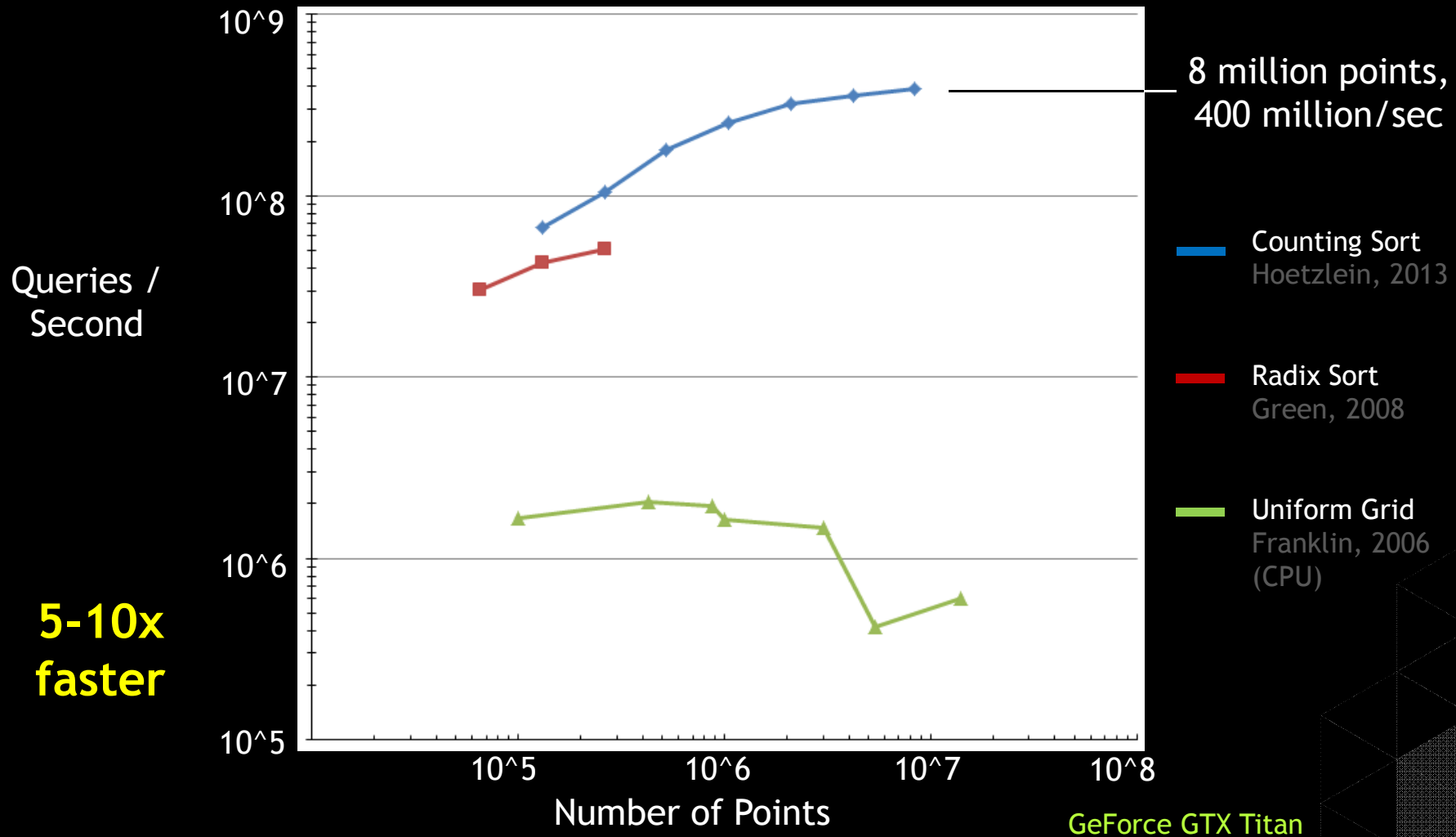


Atomic Adds

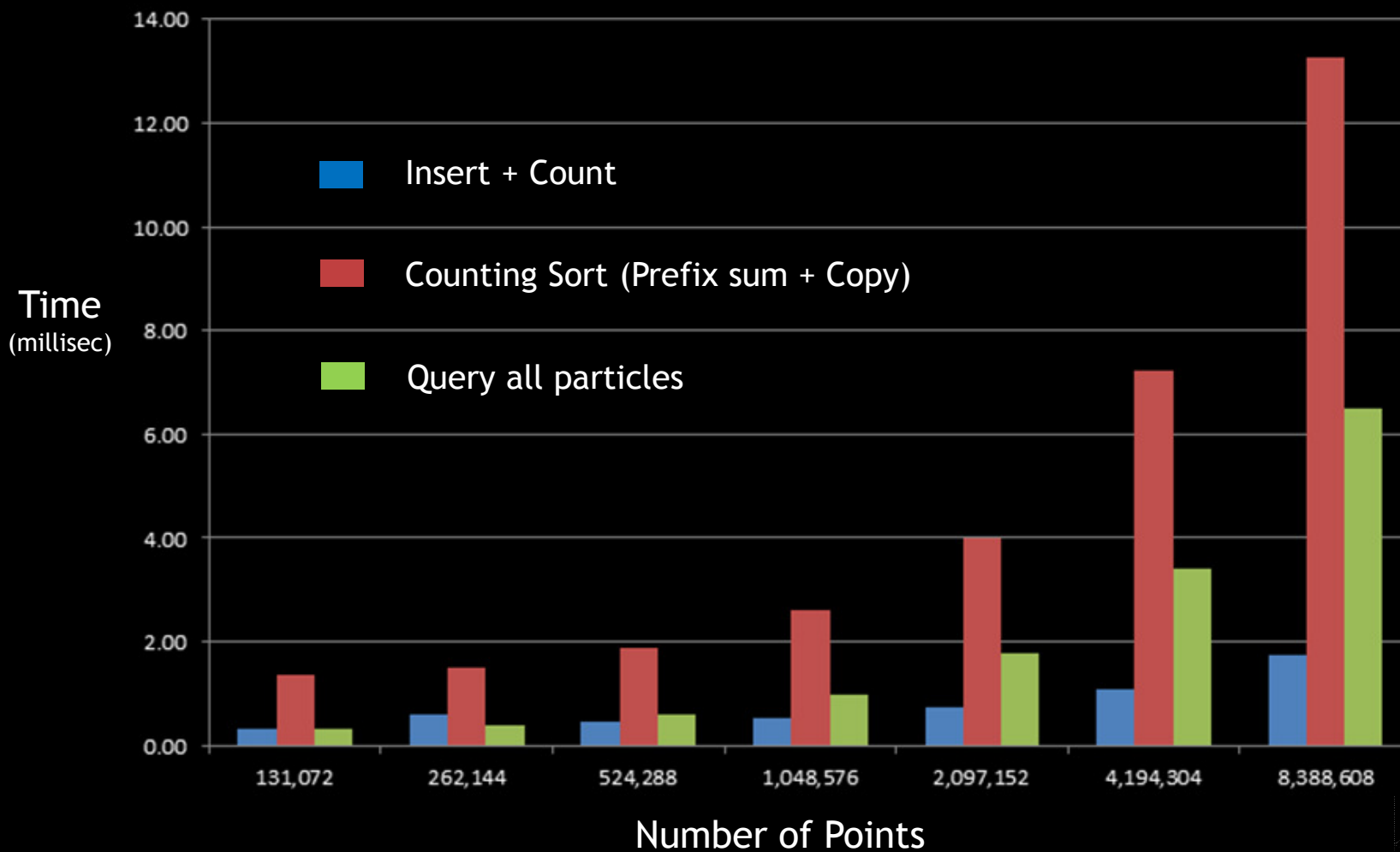
+1	+1	+1	+1
+1	+1		+1
	+1		
2	3	1	2

Each particle atomic-adds into bin *at the same time* bin is determined.

Results - Queries per Second

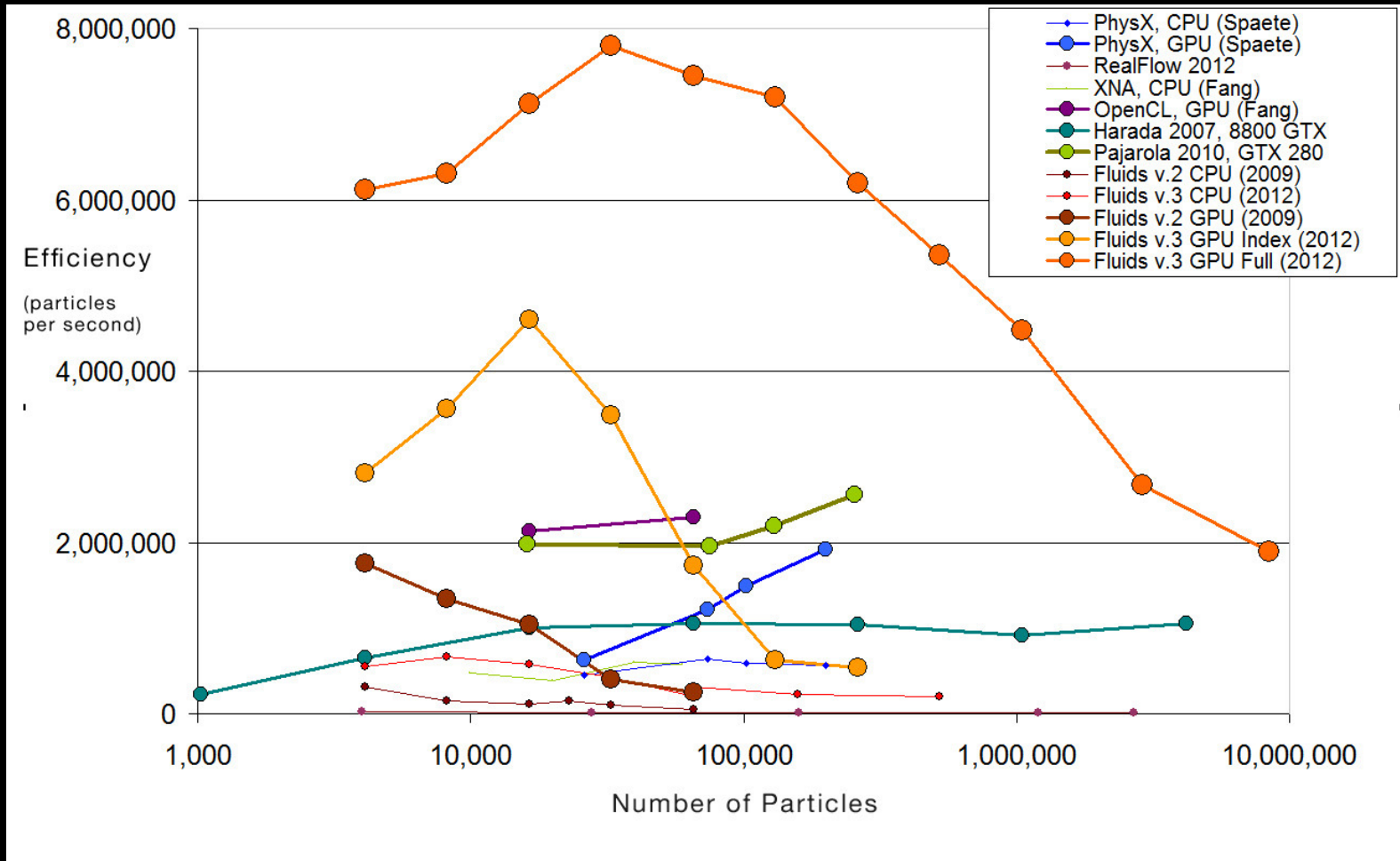


Results - Time for Each Step

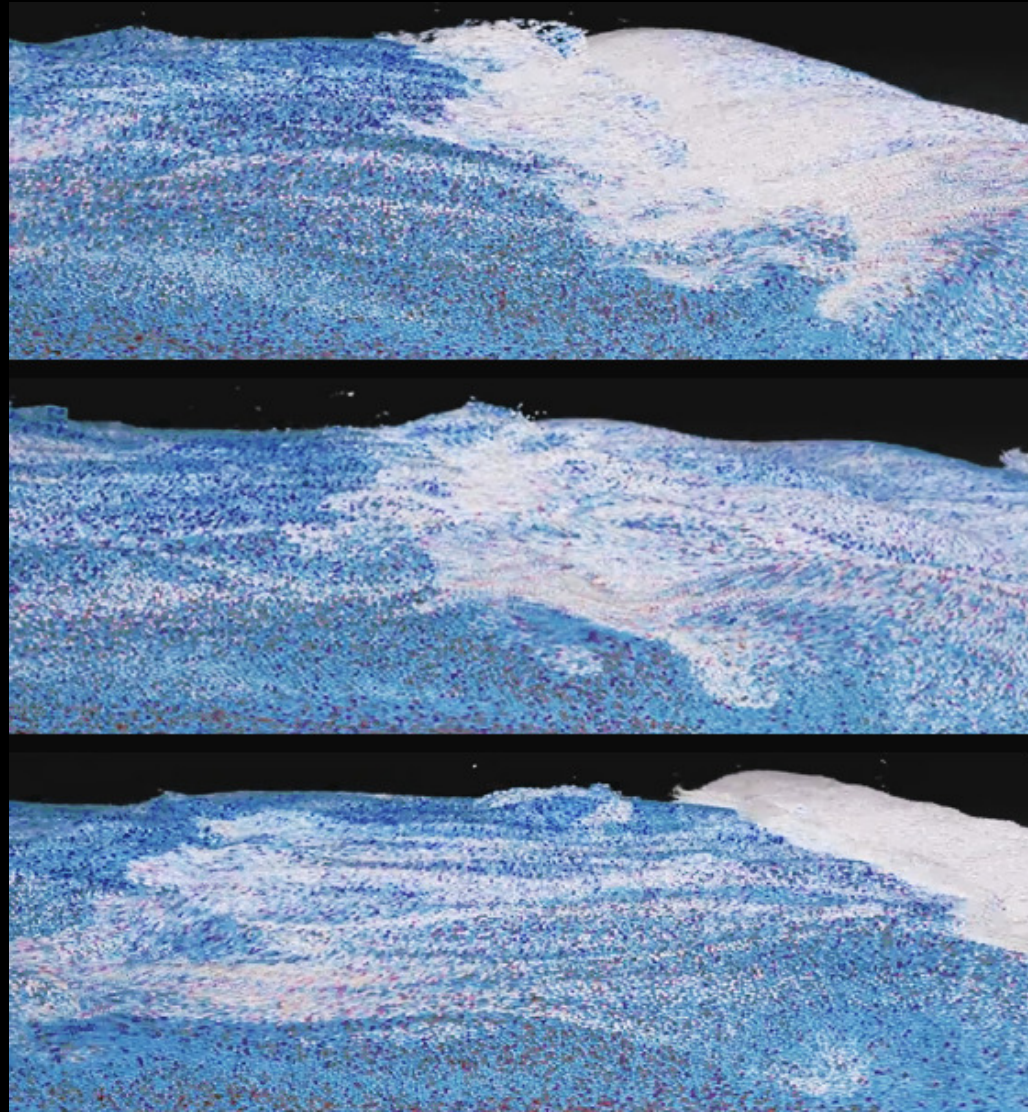


GeForce GTX Titan

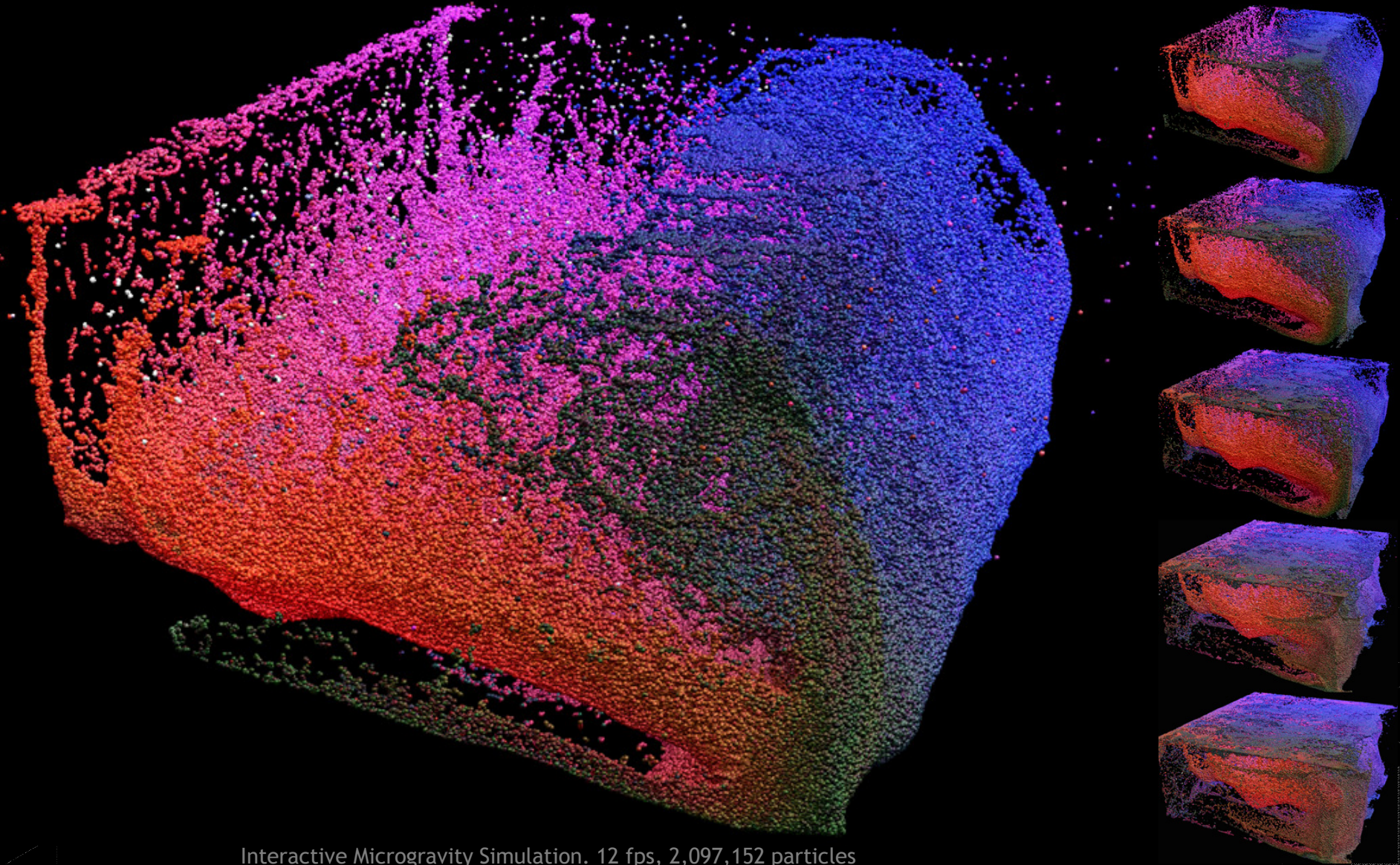
Results - Fluids Example



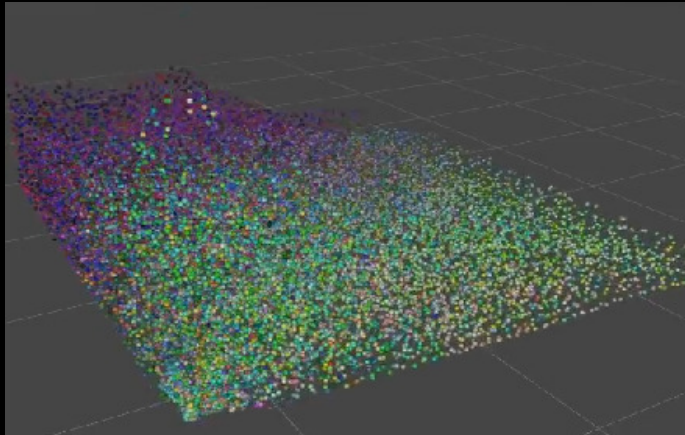
* Results do not show fluid accuracy or time step.



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

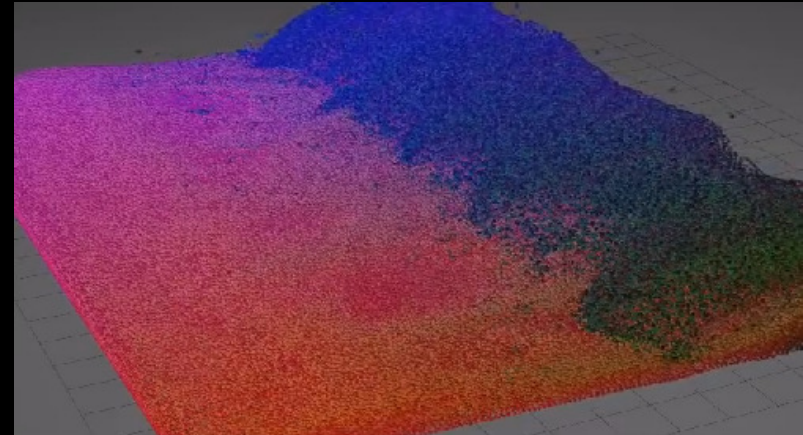


Interactive Microgravity Simulation. 12 fps, 2,097,152 particles



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

same hardware
11x faster



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

Thank you!

- Rama C. Hoetzlein, rama@rchoetzlein.com

<http://fluids3.com>