Interactive Fluid-Particle Simulations using Translating Eulerian Grids

2010 Interactive 3D Graphics and Games

Jonathan Cohen **Sarah Tariq** **Simon Green**

NVIDIA®

Interactive Fluid-Particle Simulations using Translating Eulerian Grids

2010 Interactive 3D Graphics and Games

Jonathan M. Cohen **Sarah Tariq** **Simon Green**

NVIDIA®

Motivation

Goal: Real-time Physics

- Approach 1:
  - Develop off-line serial algorithm today
  - Wait a few years for 20 GHz CPUs
- This will not work!
- Clock speeds not increasing
  - if anything, going down
- Core counts increasing
  - No help if your algorithm isn't parallel

Goal: Real-time Physics

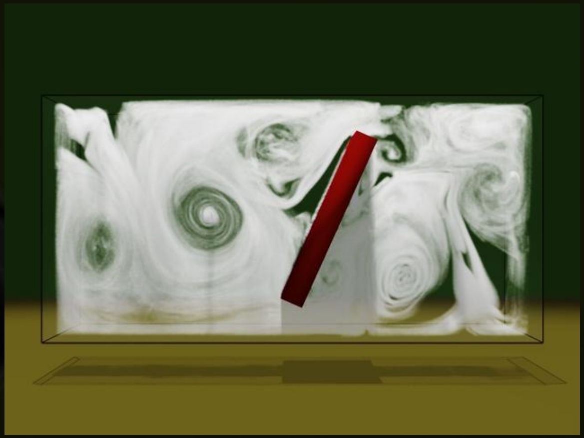
- Approach 2:
  - Develop data parallel method
  - Each generation, core count (roughly) doubles, so wait $\log_2$(Target FLOPS / Current FLOPS) years
- This works, but might not be enough
- Games should look better in the future
  - Do more FLOPS => “better looking” ?
- Asymptotic scaling matters
  - $O(n^2)$ method won’t keep pace with increasing FLOPS as well as $O(n)$ method

A Better Goal: Scalable Physics

- Data Parallel, Linear Time, Scalable Quality
  - Visual fidelity improves as processor throughput increases
- Resolution is obvious quality knob, but must be pared with appropriate method
- low viscosity method + high resolution = high visual quality

Lo-Res, High Viscosity

Lo-Res, Low Viscosity



Hi-Res, Low Viscosity

Outside the box

- “Fluid-in-a-box” is boring, too restrictive
- Particles make ideal rendering proxies
  - geometric complexity scales with GPU throughput
  - Fit into existing game FX pipelines
  - Many rendering options (several at I3D10…)
- Idea: couple particle system with grid-based fluid simulation
  - Track near-field “region of interest” with simulation grid
  - Allow particles to leave simulation grid (far field)
  - Obscure transition point

Interactive Fluid Simulation

i3D 2010

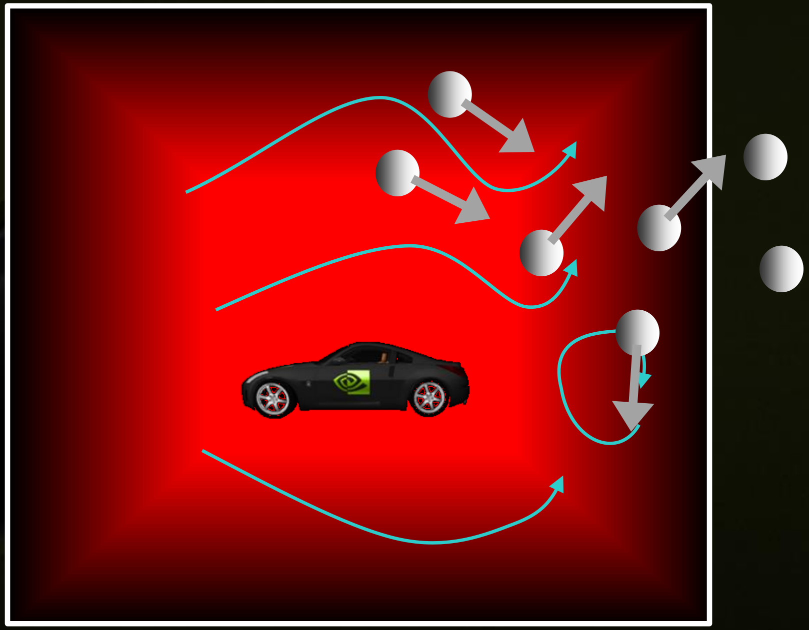
Calculate near-field fluid on grid
Fluid velocities drive particle motions

1. Calculate Fluid Velocities on Regular Grid
   2nd-Order Accurate CUDA Multigrid Solver

2. Interpolate Fluid Velocities onto Particles
   3D Interpolation in CUDA

3. Advance Particles
   CUDA Particle System

4. Render Particles
   CUDA - OpenGL Interop



Key Ingredients

Translating grid via Galilean invariance [Shah 2004]

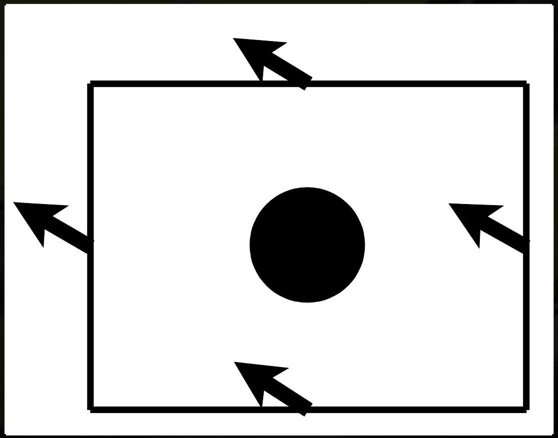
+ 2nd-order semi-Lagrangian advection [Selle 2008]
+ IOP-based pressure solver [Molemaker 2008]
+ Half-angle rendering algorithm [Ikits 2004]
+ Decoupled particle system from fluid sim
+ Many CUDA optimizations
+ Many stability fixes

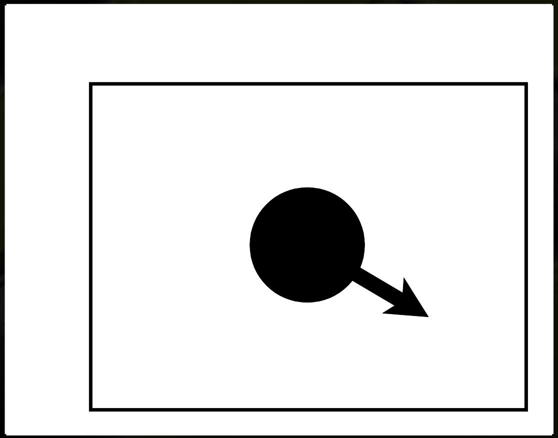
= Interactive fluid-driven effects*

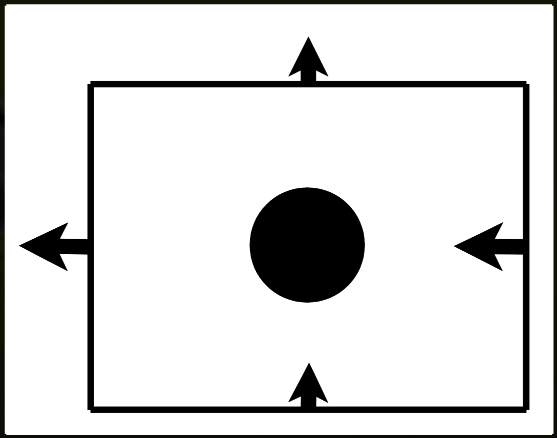
*see paper for details… too much to cover here

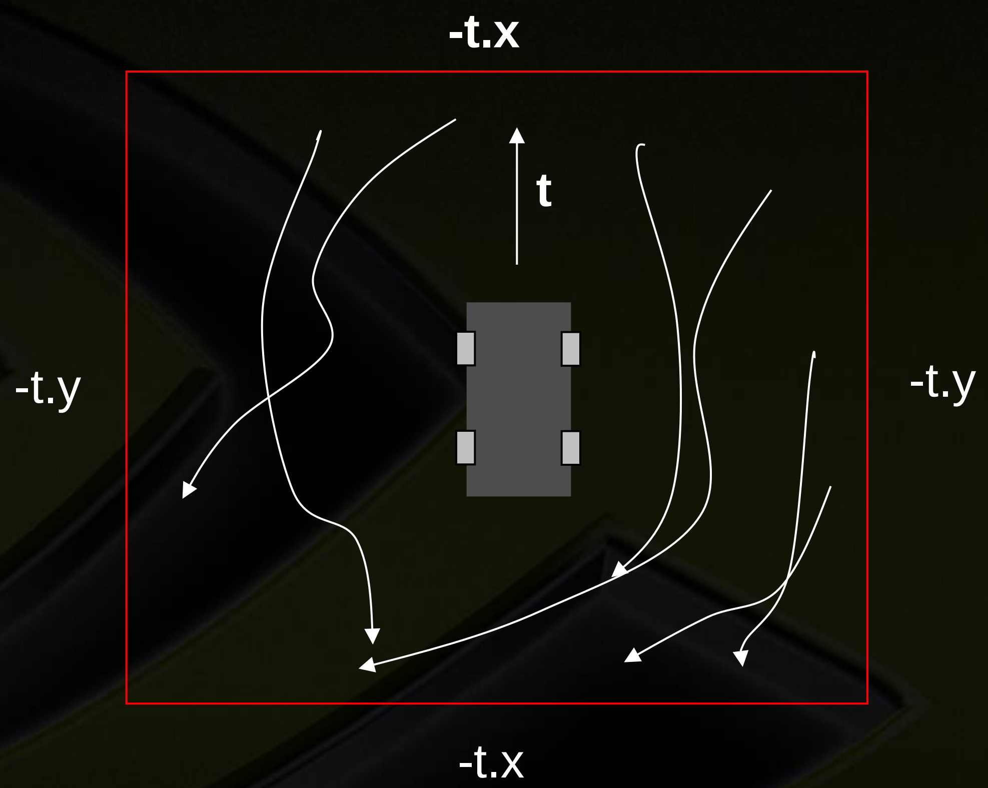
Region Tracking: Galilean Invariance

=








-t.x
t
-t.y
-t.y
-t.x

Pressure Solver

- Enforced flux at boundary must propagate instantaneously via pressure
  - (incompressible fluid = infinite speed of sound)
- => pressure solver must fully converge
- => Cannot use PCG, since it doesn't converge fast enough
- => Must use either FFT or Multigrid
  (FFT requires periodic boundaries)

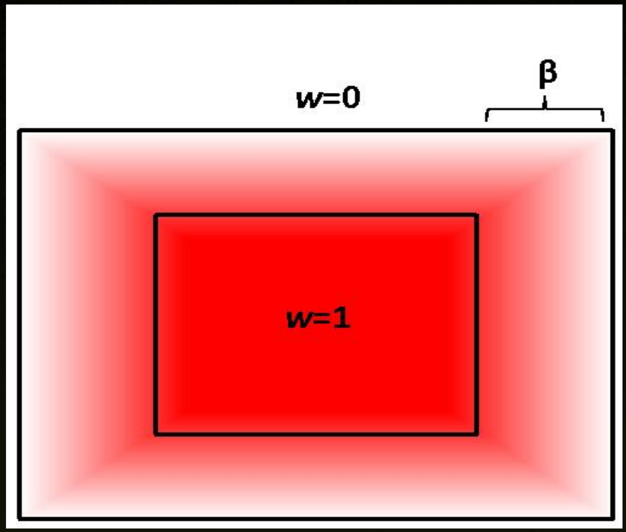
Multigrid Pressure Solver

Grid Size	Mean Time	Std Dev Time	Time / Unknown	L_∞ Error Reduction
64 x 16 x 64	14.8 ms	0.2 ms	22.6 x 10^{-5} ms	8,567 X
128 x 32 x 128	26.4 ms	0.2 ms	5.0 x 10^{-5} ms	11,079 X
256 x 64 x 256	84.6 ms	0.5 ms	2.0 x 10^{-5} ms	11,856 X

Algorithm based on [Molemaker 2008] [Yavneh 1996]
Implementation described in [Cohen 2009]

Particle System

- **Smooth transition from fluid to particles:**



Algorithm 2 Particle System Update

for all i **do**
  $Vel_{inside} \leftarrow$ interpolated world-space velocity at $Pos[i]$
  $Vel_{outside} \leftarrow Vel[i] + \Delta t \cdot Force[i]$
  $w \leftarrow$ blend weight at $Pos[i]$
  $Vel[i] \leftarrow w \cdot Vel_{inside} + (1 - w) \cdot Vel_{outside}$
  $Pos[i] \leftarrow Pos[i] + \Delta t \cdot Vel[i]$
end for

Rendering

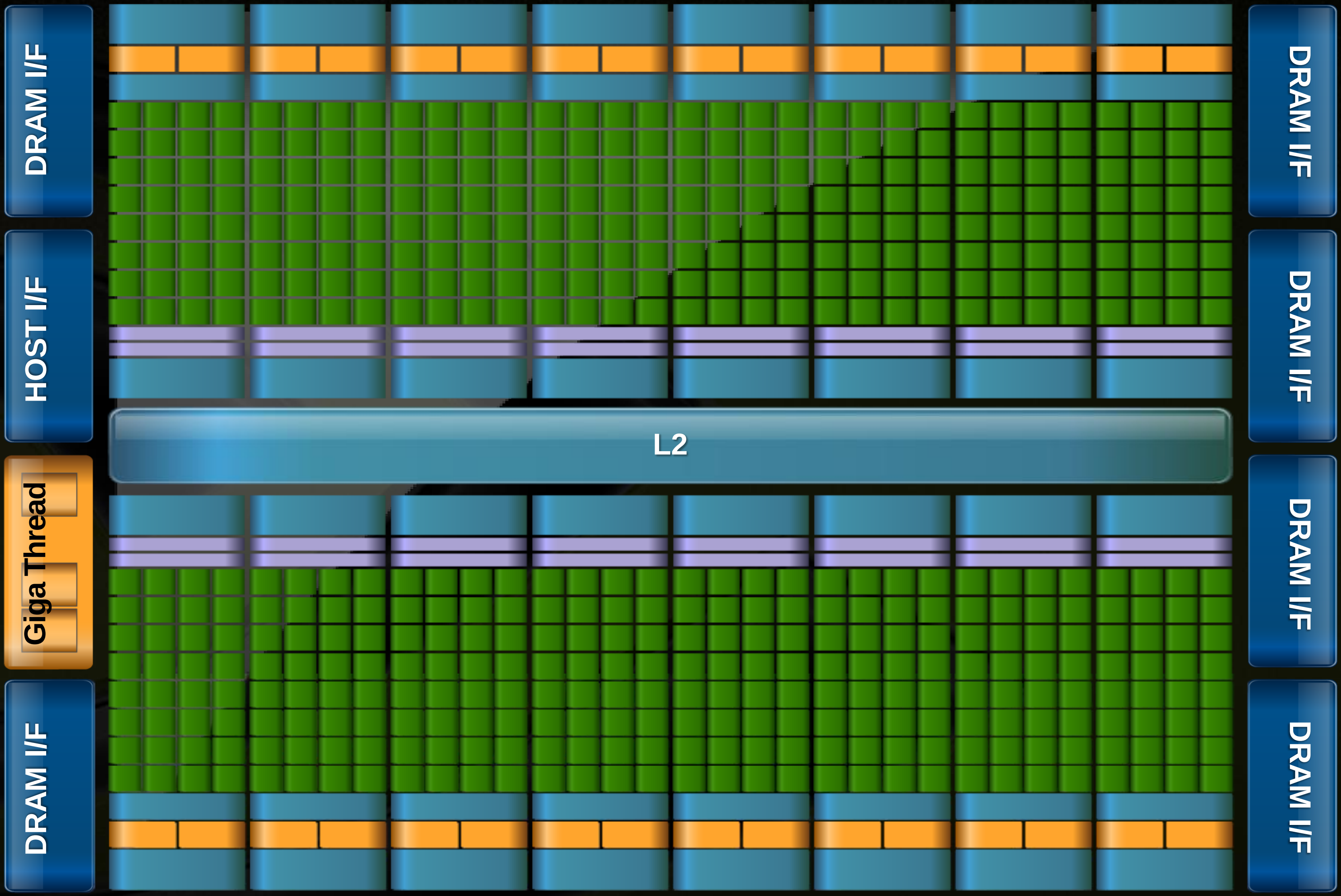
- Sort particles by depth via CUDA Radix Sort
  - dead particles sorted to end of list => no deletion needed
- Attach noise function per particle, animated to erode over particle lifespan

$$\alpha(x, y) = \mathrm{clip}(noise(x, y) - |r| + offset, cutoff).$$

[Video]

GPU Optimizations

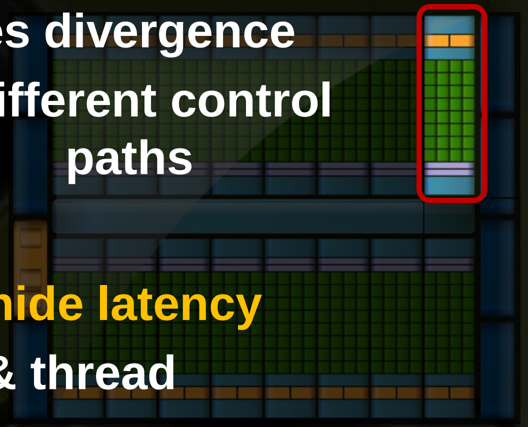
Digression: CUDA Architecture

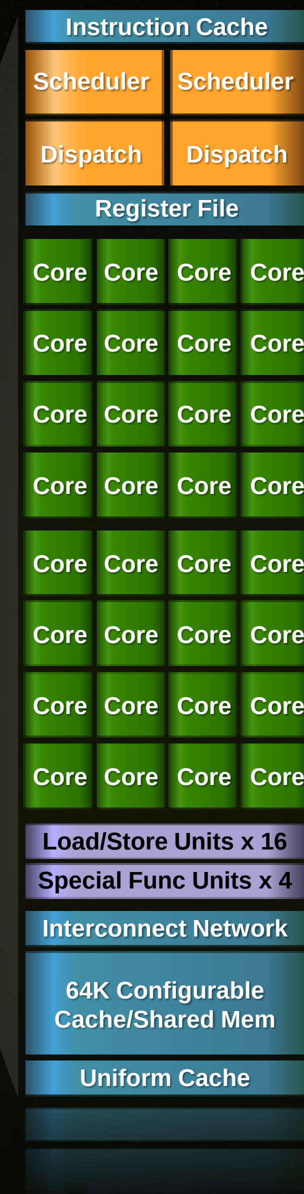


NVIDIA next-gen "Fermi" architecture

CUDA Execution Model

- **SIMT** (Single Instruction Multiple Thread) **execution**
  - **threads run in groups of 32 called warps**
  - **threads in a warp share instruction unit**
  - **1 instruction x 32 threads per clock**
  - **HW automatically handles divergence**
  - **divergence penalty = # different control paths**
- **Hardware multithreading**
  - **HW relies on threads to hide latency**
  - **HW resource allocation & thread scheduling**
  - **any warp not waiting for something can run**
  - **context switching is free**





Optimization: Per-warp particles

- Particle flow-of-control decided per-warp, rather than per-particle

- Particle birth/death decided by PRNG
- Seed set based on (Particle ID) / 32
- All particles in warp follow same flow of control

- 2x performance improvement

- See paper for more like this

Results

Performance

Frames per second
Single GPU (Dual GPU*)

	64 x 16 x 64	128 x 32 x 128	256 x 64 x 256
1 car	26(20)	25(16)	11 (8)
2 cars	26(15)	16(11)	5 (NA)
3 cars	19 (12)	11(8)	NA (NA)

*** Rendering on GPU 0, Simulation on GPU 1**

Relative Performance

Frames per Second (Simulation Only)

	Present Work	[Crane 2007]	[Long 2009]
32 x 32 x 32	182	233	96.3
64 x 64 x 64	86	65	14.3
128 x 128 x 128	22	10	1.6

[Crane 2007] and Present work on GTX285
[Long 2009] on quad-core CPU (reported in their paper)

Other applications

NVIDIA

[see DarkVoid video]

i3D 2010

Thanks!

- DevTech Team
- NVIDIA Research
- PhysX and APEX Teams
- DarkVoid

Questions ?