Shape of Things to Come: Next-Gen Physics Deep Dive

Jean Pierre Bordes
NVIDIA Corporation

NVIDIA

Free PhysX on CUDA

- PhysX by NVIDIA since March 2008
- PhysX on CUDA available: August 2008

GPU PhysX in Games

Game World Entities	Physical Characteristics
Smoke/Flames	Gaseous Fluid
Water/Oil/Mud	Liquid Fluid
Fabric/Clothing Plants/Pillows/Fat	Deformable Solids
Leafs/Slivers	
Sparks Gravel Splinters/Shrapnel Trash/Rocks Furniture	Rigid Solids
Characters Vehicles/Machines	
Buildings	
World Architecture	Static Solid, Infinite Mass

Number of Computational Elements

Opportunity for Parallelization

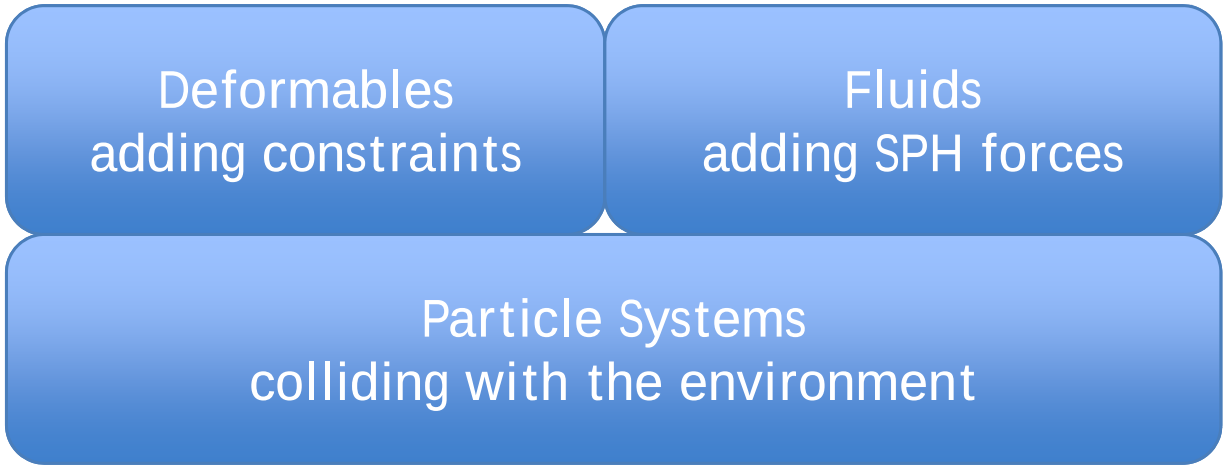
Scale of Visual Motion

PhysX Representation
Particle Fluids
Deformables
Particles
Rigid Body Actors
Jointed RB Actors
Static Actors

Presentation Overview

- Introduction
- Feature presentations
  - Particle systems
  - Fluids
  - Deformables
- Parallel PhysX SDK
- Conclusion
- Demos

Particle Based Features



- Seamless collision with rigid bodies
- Algorithmic reuse and consistency
- Massively parallel execution on CUDA enabled GPUs

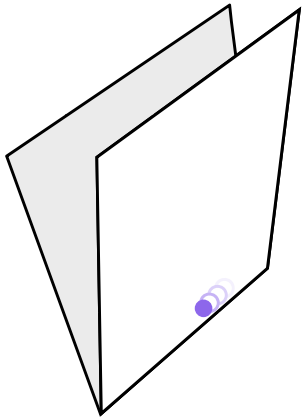
Particle Systems

- Ubiquitious in games
- Increased realism:
  - Universal collision with game environment
- Basis for more advanced features

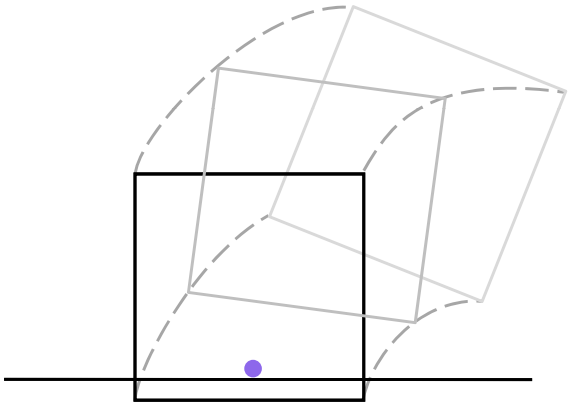
Robust Particle Collision

Features

- Minimize penetration and artificial energy loss
- Handle high velocity collisions
- Configurable collision radius
- Static over dynamic precedence



„Edge“ Case
(avoid leaking and sticking)

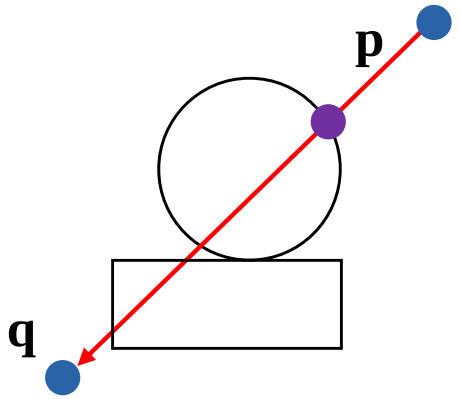


Static Precedence

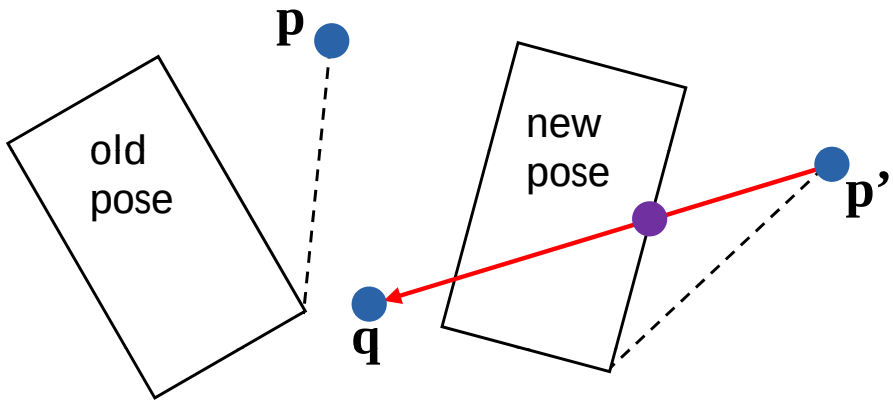
Robust Particle Collision

Mixed Approach

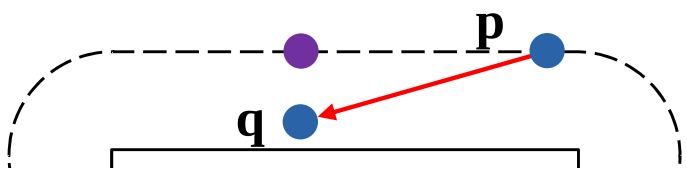
- Continuous CD: motion ray against shapes
- Discrete CD: motion target against shape



Continuous Test
(multiple shapes)



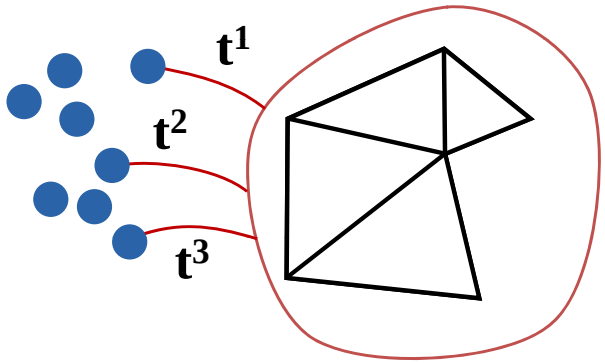
Continuous Test
(moving shape)



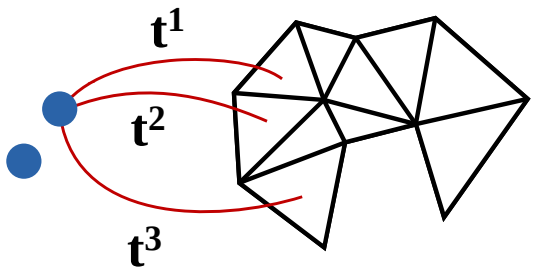
Discrete Test

CUDA Implementation

- GPU Workloads
  (set of particles, set of shapes)



- Particle level parallelism
- Optimization for |shapes| >> |particles|
  → particle-shape pair parallelism for early outs



- Caching of static geometry
- Texture cache and shared memory for shapes

Applications

- Small objects that should collide with the environment
  - Debris (gravel, shrapnel, dirt bits, etc…)
  - Shells
  - Sparks
  - Leaves, Slivers
- Animate Meshes: Oriented particles that rotate based on collisions

GRAW-2

NVIDIA.

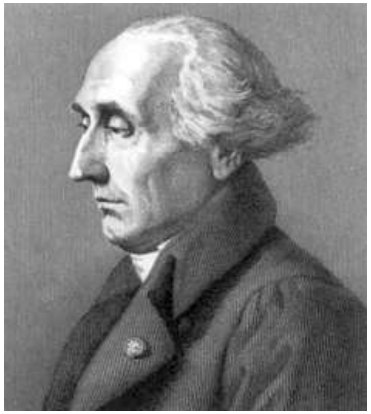
Fluids

- Mathematical descriptions for continuous materials
  - Lagrangian (particles)
  - Eulerian (grids)

- Particle vs. Grid based fluid simulation in games
  - Particles already used in games (easier integration)
  - Seamless collision with lagrangian game objects
  - Mass conservation for free

Leonhard
Euler
1707-1783

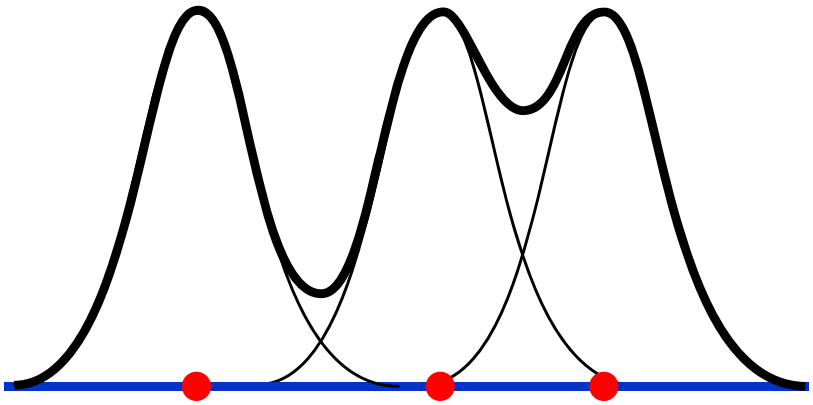
Joseph-Louis
Lagrange
1736-1813



Smoothed Particle Hydrodynamics

- Particle acceleration given by Navier-Stokes eqn.
  - External acceleration (e.g. gravity)
  - Internal acceleration
    - Pressure (pressure field gradient)
    - Viscosity (smoothed velocity field)

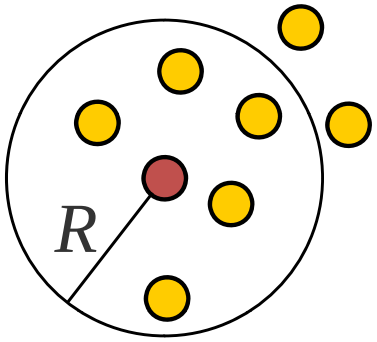
- Particles = sampling points for physical fields
- Kernels to reconstruct smooth fields



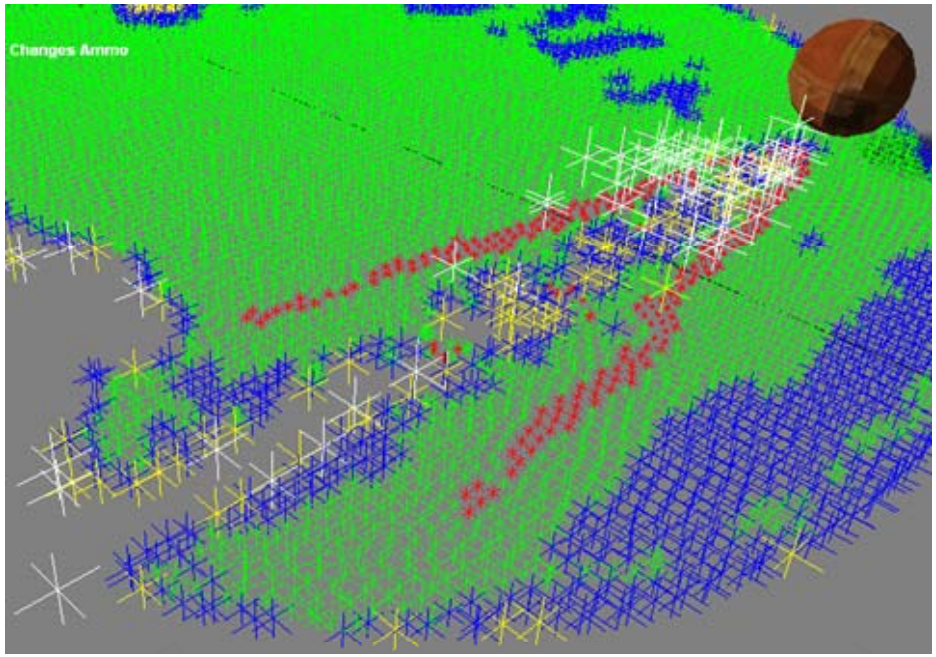
SPH Algorithm

2 Passes over particle neighborhood

- Density
- Force (Pressure, Viscosity)



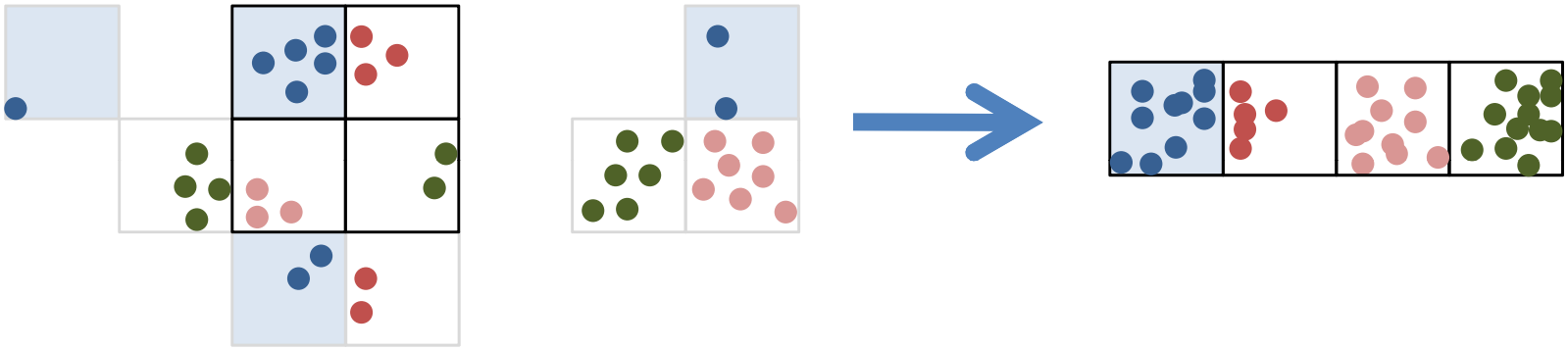
Particle Neighborhood



Densities at Particle Locations

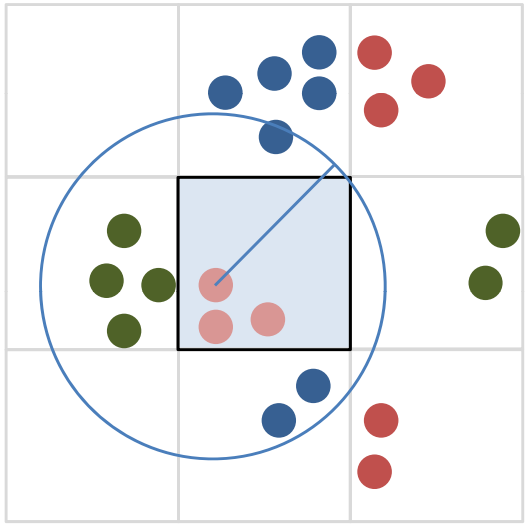
CUDA Implementation

- Neighbor finding optimizations
  - Group particles into cells
  - Map cells to regular cube (e.g. 16x16x16)
  - Radix sort particles according to mapping



CUDA Implementation

- Neighborhood pass
  - Particle level parallelization
  - Texture cache particle access



Applications

- Dynamic, interactive scenarios
  - Bursting liquid containers
  - Flame throwers
  - Mud puddles
  - Splashes
  - …

SPH Demo

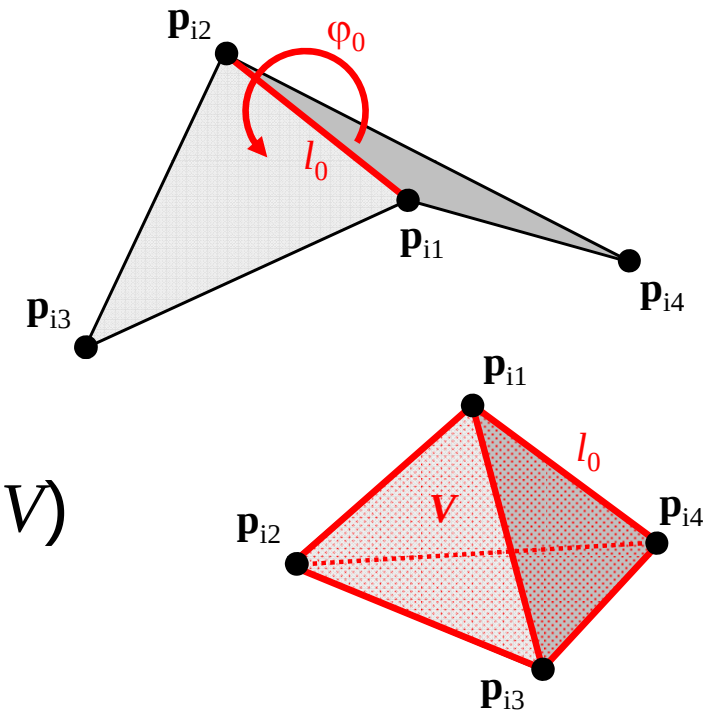
Deformables

- Deformables
  = Cloth, Softbody, …
  = Particles + Constraints + Meshes

- Specific features
  - Tearing
  - Self-collision
  - Pressure
  - … and more

Constraints

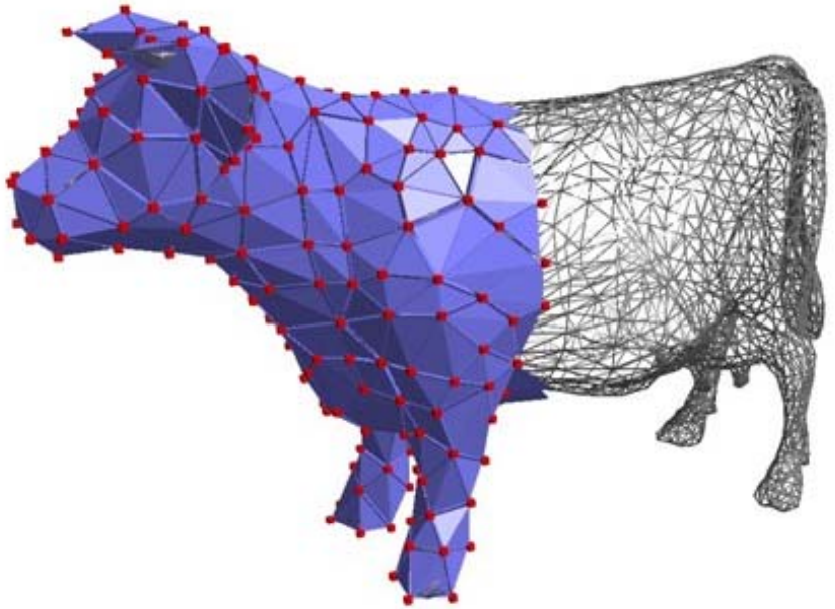
- Internal
  - Stretching (l_0)
  - Bending (φ_0)
  - Volume conservation (V)



- External
  - (Self-)collision
  - Attachments, adherence, dominance
  - Custom constraints

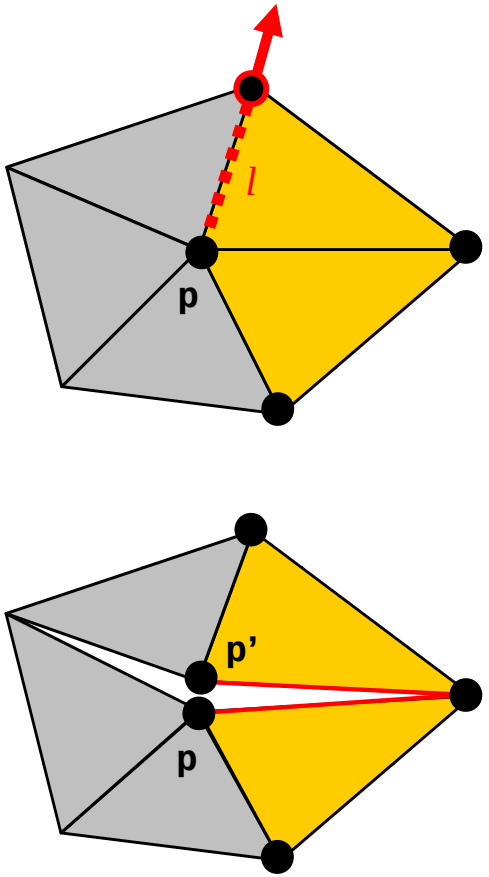
Meshes

- Cloth
  - Triangle mesh
- Softbody
  - Tetrahedral mesh
  - TetraMaker tool
- Simulation vs. graphics mesh



Tearing

- Algorithm
  - Choose vertex *p* adjacent to overstretched edge *l*
  - Duplicate vertex *p* → *p'*
  - Split mesh perpendicular to *l* at vertex *p*

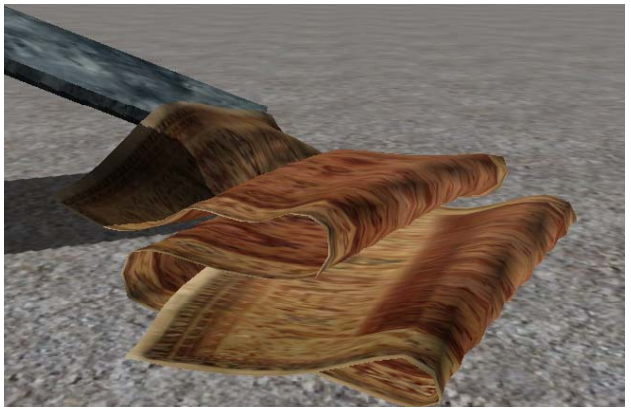


- Updates
  - Add particle (*p'*)
  - Modify constraints (impacted by *p* and *p'*)
  - Split graphical mesh too!

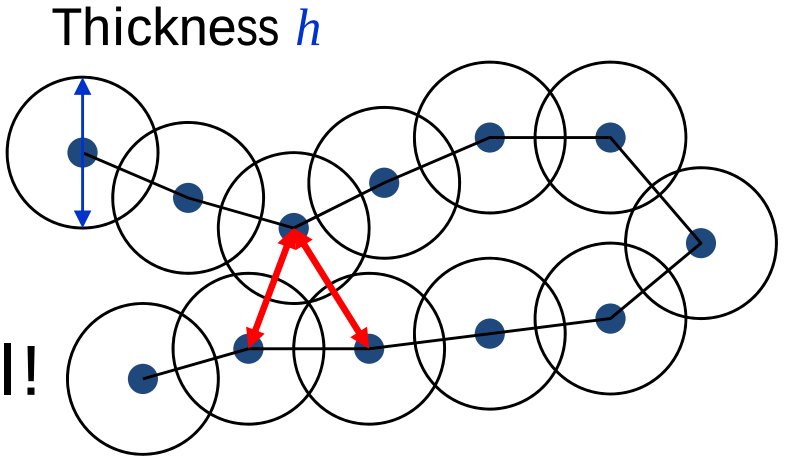
Tearing

Self-collision

- "Perfect" self-collision is tricky & slow
  - Uncontrollable external effects
  - Needs untangling



- Simplified approach
  - Particle repulsion
  - Works surprisingly well … and fast … in parallel!



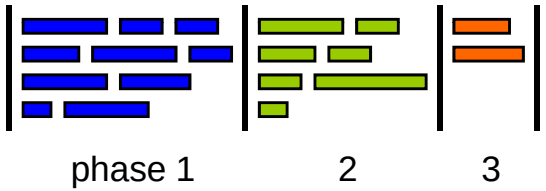
Simulation Algorithm

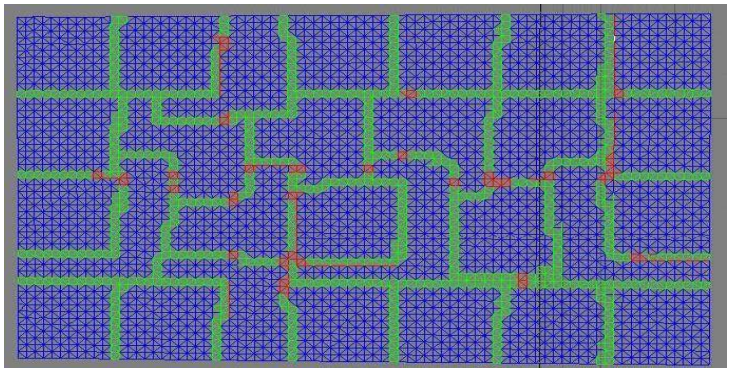
- Position based dynamics

1	Update	positions $\mathbf{p}_1, \ldots, \mathbf{p}_n$ velocities $\mathbf{v}_1, \ldots, \mathbf{v}_n$
2	Prediction	$\mathbf{q}_i = \mathbf{p}_i + \mathbf{v}_i \cdot \Delta t$
3	(Self-)collision	add constraints based on ray $\mathbf{p}_i \rightarrow \mathbf{q}_i$
4	Solver	for all constraints c_i: modify $\mathbf{q}_{i1}, \ldots, \mathbf{q}_{im}$ to satisfy c_i
5	Integration	$\mathbf{v}_i = (\mathbf{q}_i - \mathbf{p}_i) \,/\, \Delta t$ $\mathbf{p}_i = \mathbf{q}_i$
6	Damping	modify $\mathbf{v}_1, \ldots, \mathbf{v}_n$

Parallel Solver

- Work packet $w_i = (c_1, \ldots, c_{ni}, p_1, \ldots, p_{mi}, phase)$
  - Each c_i is contained in exactly one work packet
  - All w_i contain all particles referenced by $c_1, \ldots, c_{ni}$
  - For each *phase*, the partition of particles is disjoint





- Parallel Algorithm

For *phase* = 1 to *#phases*

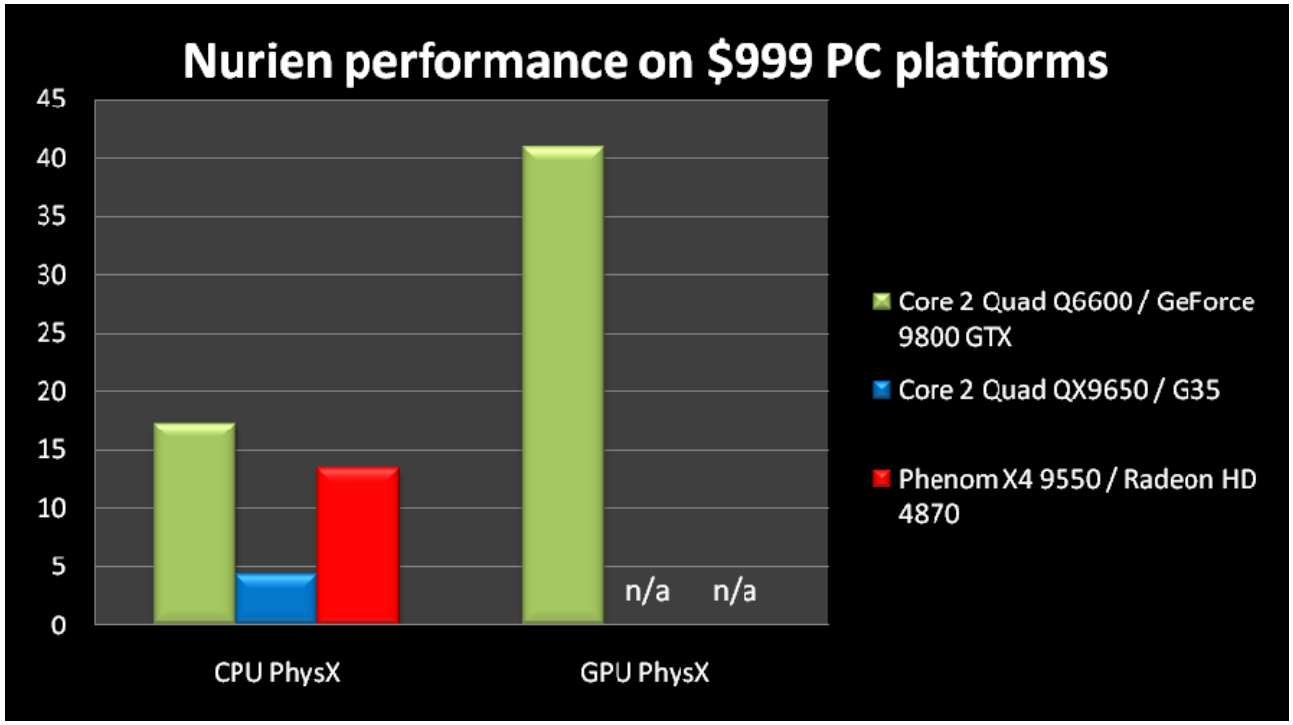
Process work packets of *phase* in parallel

Sync

End for

CUDA

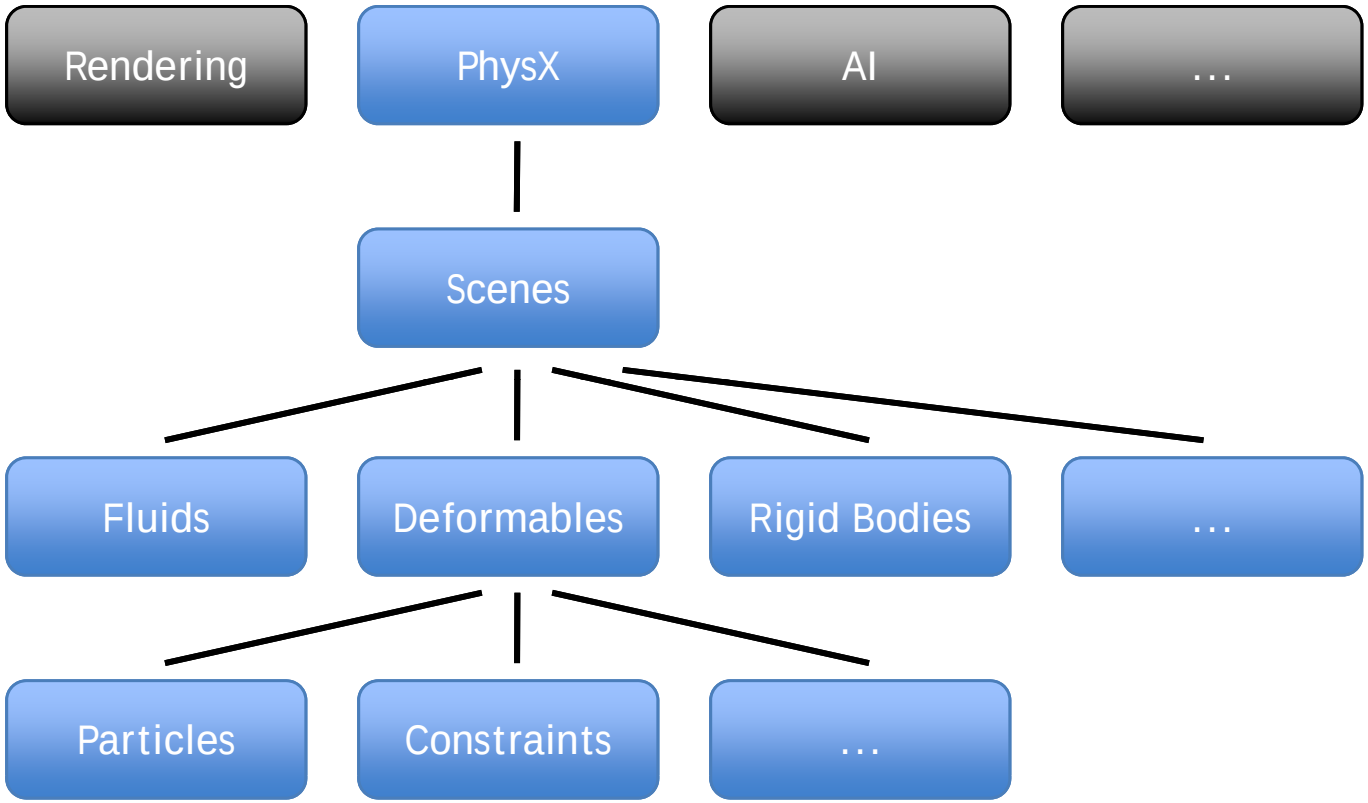
- Implementation
  - Update, (self-)collision, solver kernels
  - Work on batched instance data
  - Heavily use shared memory for work packets



Nurien

Parallel PhysX SDK

- Exploits task- and data-parallelism in different layers and modules



Parallelization Challenges

- Algorithms
  - Robustness vs. massive parallelism
  - Generality vs. specialization
- Scheduling
  - Immediate vs. batched execution
  - Task- and data-dependencies
  - Load balancing
- Data management
  - DMAs, cache hierarchies
  - Asynchronous access, double-buffering

Conclusion

- Particle based features
  - Broad range of applications
  - Highly scalable content
  - Suitable for parallelization
- NVIDIA PhysX SDK
  - Exploits massive parallelism offered by CUDA
  - Delivers solution to challenges and complexity
  - Drives next-gen games and applications
  - Available now!

The Great Kulu Demo

NVIDIA Fluid

The End

Questions?