Game Developers Conference®
March 9-13, 2010
Moscone Center
San Francisco, CA
www.GDConf.com

GDC 10
Learn. Network. Inspire.

Physics Simulations on the GPU

Takahiro Harada

havok™

Senior Software Engineer

takahiro.harada@havok.com

Introduction

- Based on my research at the university of Tokyo
  - Not at havok

Agenda

- Introduction
- Particle-based Simulations
- Rigid bodies
- Solving Constraints
- Using Multiple GPUs

Current GPUs

- Current GPUs are designed for graphics
- Good at
  - Many similar computations
  - **Simple** computations
- Restrictions
  - Not complicated logic (CPU is good at)
  - All the thread taking the same path is ideal
  - The number of computations run in a kernel have to be very large (otherwise, cannot hide memory latency etc)
- Not good for branchy code
  - Narrow phase like 2 box-box, 4 cvx-cvx
- Great for large number of same simple computation
  - 1M particles <- Good at Particles

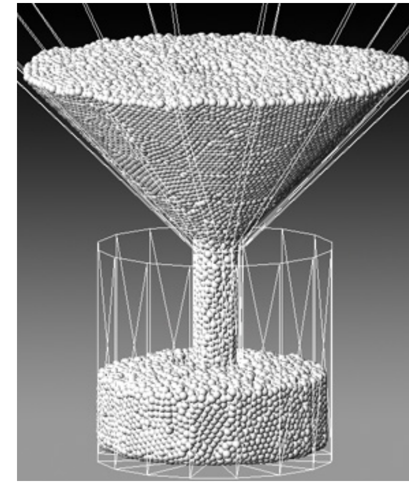
Particle based Simulations

- Granular Materials

  DEM(Distinct Element Method)

$$\frac{d\mathbf{v}_i}{dt} = \frac{1}{m}\sum_{j\in contact}\mathbf{f}^c_{ij} + \mathbf{g}$$

$$\mathbf{f}^n_{ij} = -k\mathbf{\Delta r}^n_{ij} - \eta\mathbf{v}^n_{ij}$$



- Fluids

  SPH(Smoothed Particle Hydrodynamics)

  MPS(Moving Particle Semi-implicit)

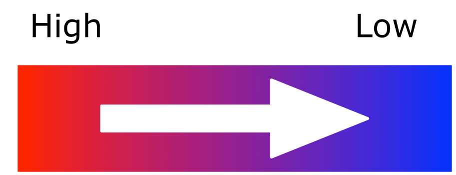
  Governing Equations for Fluid

$$\frac{D\mathbf{U}}{Dt} = -\frac{1}{\rho}\nabla P + \nu\nabla^2\mathbf{U} + \mathbf{g}$$

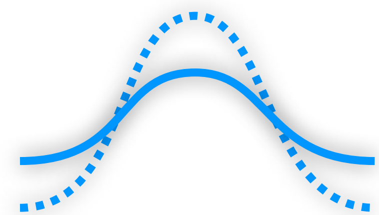
$$\frac{D\rho}{Dt} + \rho\nabla\cdot\mathbf{u} = 0$$

The Equation

$$\frac{D\mathbf{U}}{Dt} = -\frac{1}{\rho}\nabla P + \nu\nabla^2\mathbf{U} + \mathbf{g}$$



- Pressure Gradient
  - Makes fluid incompressible
- Viscosity



  - Reduce the difference of velocities
  - Very small and we already have numerical viscosity, sometimes it can be ignored in graphics
- Gravity
- Lagrangian Derivative

$$\frac{D\phi}{Dt} = \frac{\partial\phi}{\partial t} + \mathbf{U}\cdot\nabla\phi$$

  - Value change on a particle moving on the flow
  - When particles are used for simulation, dont have to do anything special. Just advect particles

The Equation

$$\frac{D\mathbf{U}}{Dt} = -\frac{1}{\rho}\nabla P + \nu\nabla^2\mathbf{U} + \mathbf{g}$$

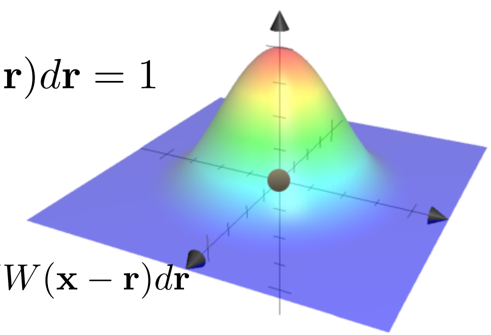
- ∇ and ∇^2 have to be modeled when simulating using particles
  - How to on SPH?
  - How to on MPS?

SPH

» Idea

A value at x is calculated by integration of the function in space(in continuous)

$$\phi(\mathbf{x}) = \int \phi(\mathbf{r}) W(\mathbf{x}-\mathbf{r}) d\mathbf{r} \qquad \int W(\mathbf{x}-\mathbf{r}) d\mathbf{r} = 1$$



Gradient

$$\nabla\phi(\mathbf{x}) = \int \nabla\phi(\mathbf{r}) W(\mathbf{x}-\mathbf{r}) d\mathbf{r}$$

$$\nabla\phi(\mathbf{x}) = \int \nabla(\phi(\mathbf{r}) W(\mathbf{x}-\mathbf{r})) d\mathbf{r} + \int \phi(\mathbf{r}) \nabla W(\mathbf{x}-\mathbf{r}) d\mathbf{r}$$

$$\nabla\phi(\mathbf{x}) = \int \phi(\mathbf{r}) \nabla W(\mathbf{x}-\mathbf{r}) d\mathbf{r}$$

» On Discrete particles

$$\phi(\mathbf{x}) = \sum_j m_j \frac{\phi_j}{\rho_j} W(\mathbf{x}-\mathbf{x}_j)$$

$$\nabla\phi(\mathbf{x}) = \sum_j m_j \frac{\phi_j}{\rho_j} \nabla W(\mathbf{x}-\mathbf{x}_j)$$

Laplacian

$$\nabla^2\phi = \nabla \cdot (\nabla\phi)$$

$$\begin{aligned} \nabla^2\phi_i &= \sum_j m_j \frac{\nabla\phi_j}{\rho_j} \cdot \nabla W(\mathbf{x}_i-\mathbf{x}_j) \\ &= \sum_j m_j \frac{\phi_j-\phi_i}{\rho_j} \frac{\mathbf{x}_j-\mathbf{x}_i}{|\mathbf{x}_j-\mathbf{x}_i|^2} \cdot \nabla W(\mathbf{x}_i-\mathbf{x}_j) \end{aligned}$$

MPS

» Idea

Differential operators are modeled directly

Generalized gradient $$\nabla \phi_{i,j} = \frac{\phi_j - \phi_i}{|\mathbf{r}_{ij}|} \frac{\mathbf{r}_{ij}}{|\mathbf{r}_{ij}|}$$

Weighted average of neighbors

$$\langle \nabla \phi \rangle_i = \frac{d}{n^0} \sum_{j \neq i} \frac{\phi_j - \phi_i}{|\mathbf{r}_{ij}|^2} \mathbf{r}_{ij} w(\mathbf{r}_{ij})$$

$$\langle \nabla^2 \phi \rangle_i = \frac{2d}{n^0 \lambda} \sum_{j \neq i} (\phi_j - \phi_i) w(\mathbf{r}_{ij})$$

Solving Poisson equation on particles(to make incompressible)

$$\frac{\mathbf{u}_i^{n+1} - \mathbf{u}_i^*}{\Delta t} = -\frac{1}{\rho} \nabla P_i^{n+1} \rightarrow \nabla^2 P_i^{n+1} = \rho \frac{\nabla \cdot \mathbf{u}_i^*}{\Delta t}$$

$$\downarrow$$

$$\nabla^2 P_i^{n+1} = -\frac{1}{n_0} \frac{n^* - n_0}{(\Delta t)^2}$$

$$n_i = \sum_{j \neq i} w(\mathbf{r}_{ij})$$

Wrap up

» SPH

A value is a weighted sum of neighboring values

X: Have to choose kernels carefully

X: Cannot calculate correct pressure

X: Can be compressed

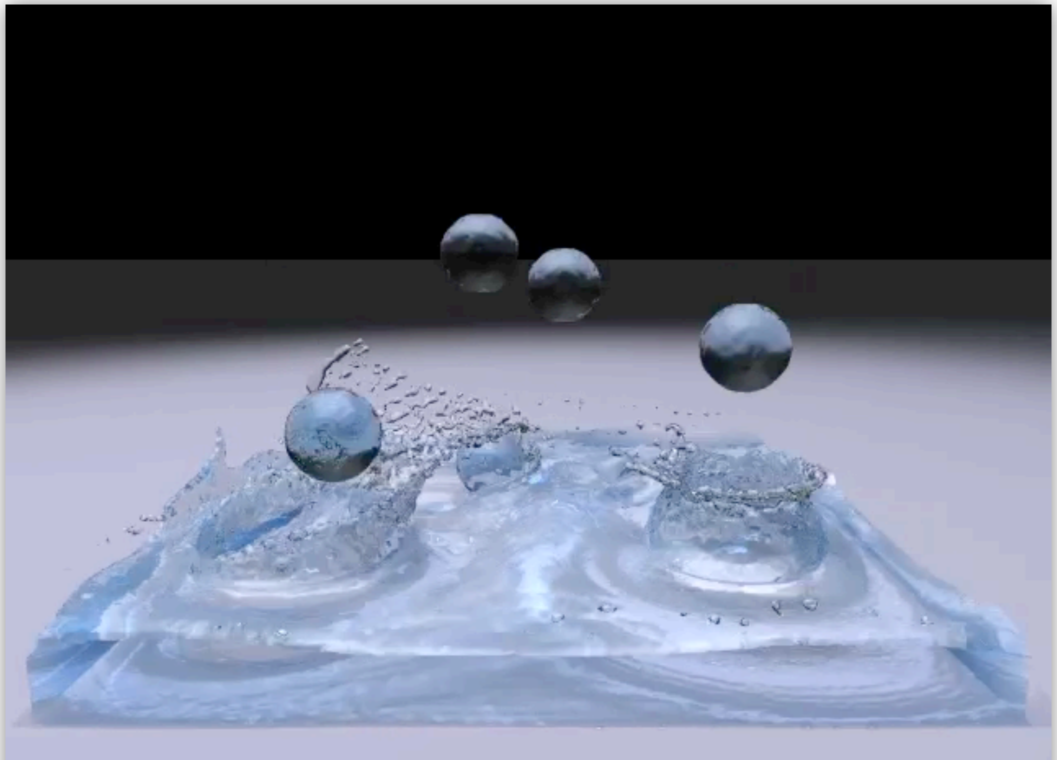
» MPS

O: No need to be bothered from choice of kernels

O: Can calculate correct pressure like grid based

O: Doesnt compress

X: Can have some oscillation of pressure

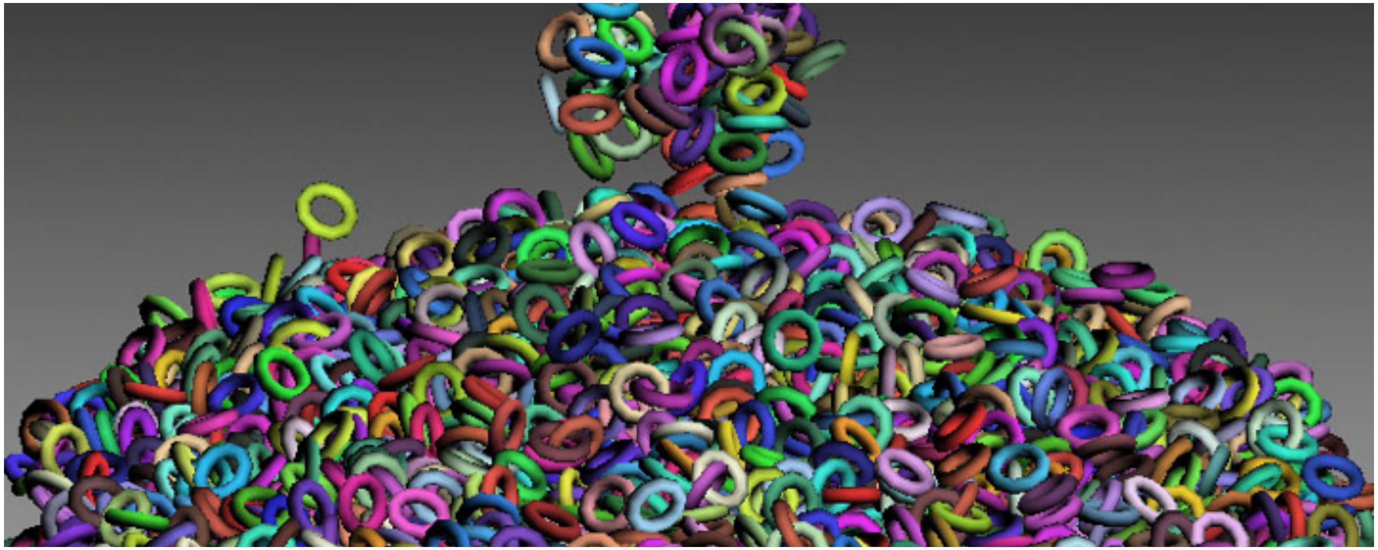


Agenda

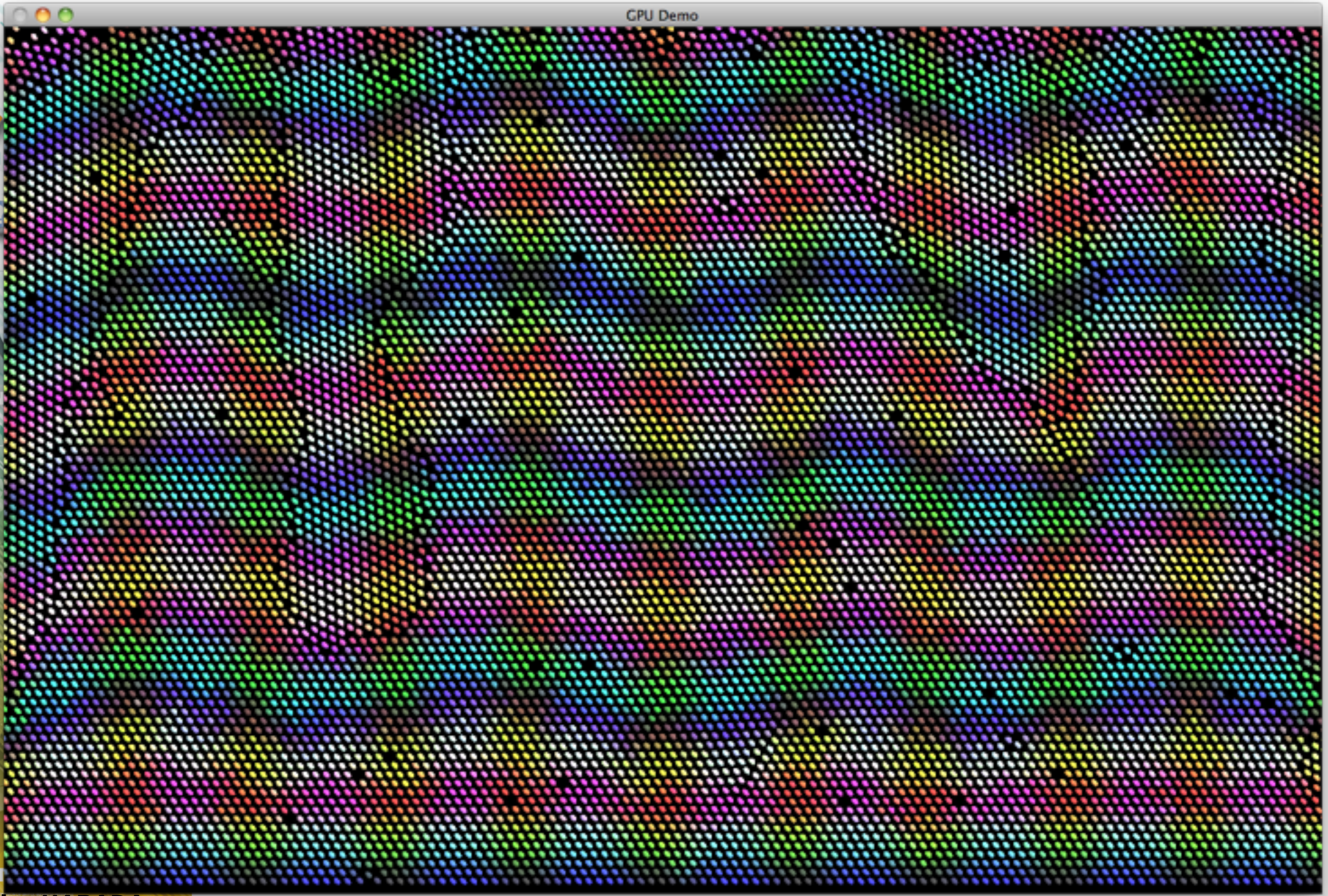
» Introduction
» Particle-based Simulations
» Rigid bodies
» Solving Constraints
» Using Multiple GPUs

Physics Simulation

- Some physics simulation is highly parallel
  - Grid-based fluid simulation
  - Particle-based simulation
- How about rigid bodies?
  - X Accurate simulation
  - O Simplified simulation
    - Takahiro Harada, "Real-time Rigid Body Simulation on GPUs", GPU Gems3



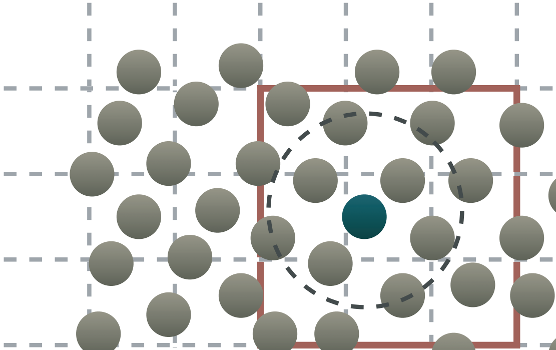
Particle-based Simulation



DEM Simulation

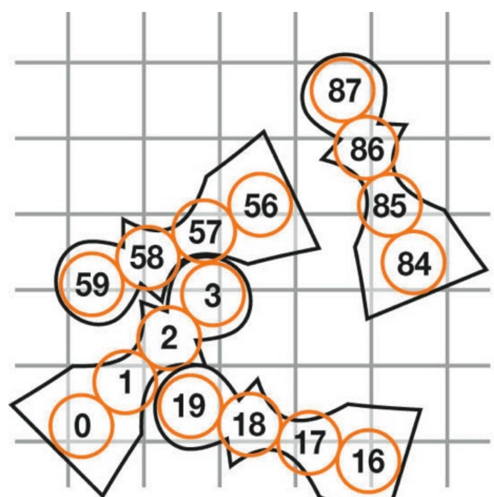
- Overview
  - For each particle
    - Look for neighboring particles
  - For each particle
    - Force on a particle is calculated using values of neighbors
  - For each particle
    - Integrate velocity and position

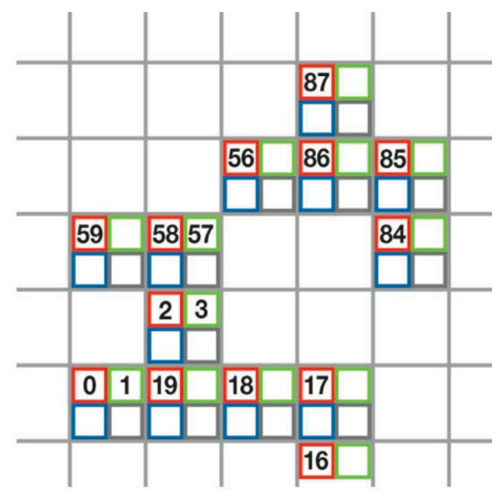
- Neighbor search using Uniform Grid
- Problem is neighbor search
  - Use uniform grid to accomplish this



Grid Construction

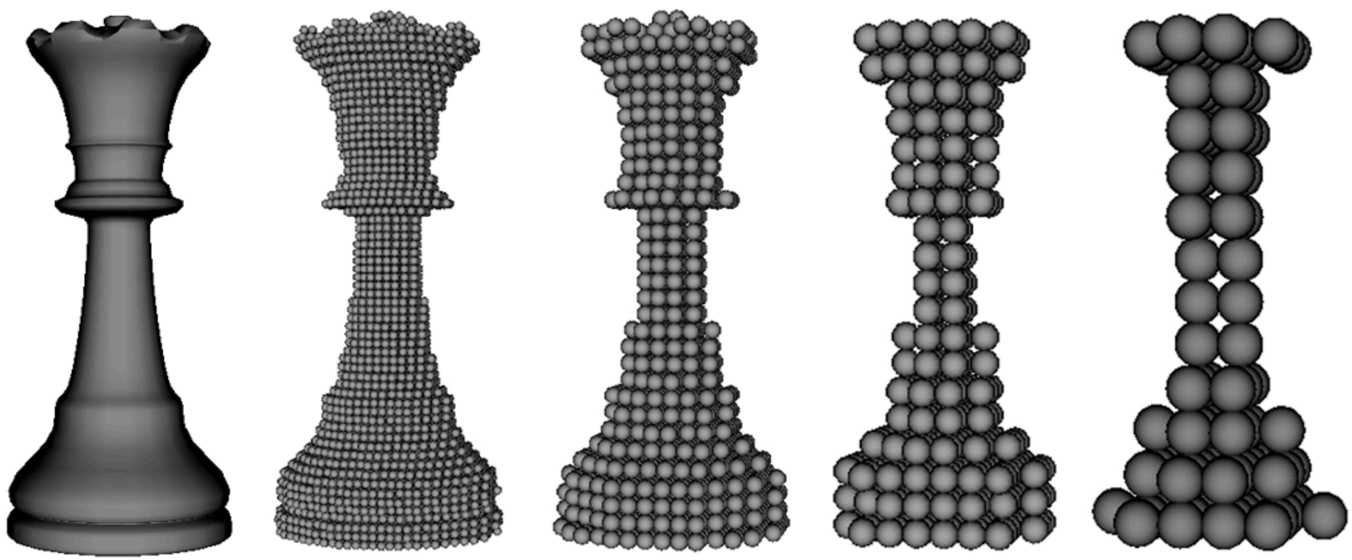
- Storing particle indices to 3D grid
- Each thread read particle position, write the index to the cell location
- But this fails when several particles are in the same cell
  - Divide this into several pass
  - 1 index is written in a pass
  - Repeat n times (max number of particles)
- Can limit the max number of particle in a cell if particles does not penetrate





Rigid Body Simulation using Particles

- Extension to particle based simulation
- Rigid body is represented by particles
    - Not accurate shape
    - Trade off between accuracy and computation
    - Simple, uniform computations -> Good for GPUs
- Use particles to calculate collision

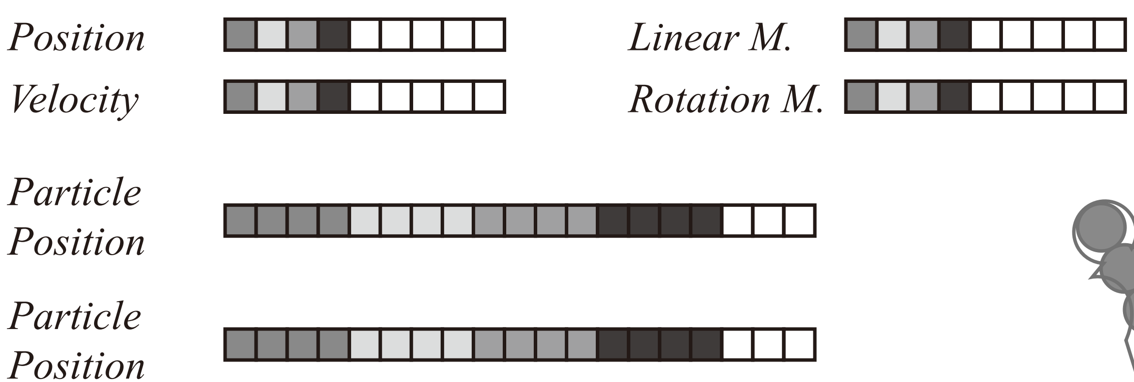


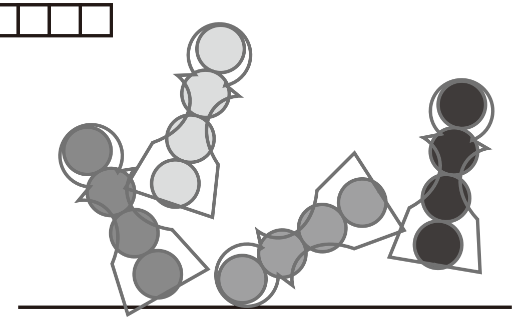
Overview

- Prepare for collision
  - Computation of particle values
    - For each particle: read values of the rigid body and write the particle values
  - Grid construction
- Collision detection and reaction
  - For each particle: read neighbors from the grid, write the calculated force (spring & damper)
- Update
  - Update momenta
    - For each rigid body: sum up the force of particles and update momenta
  - Update position and quaternion
    - For each rigid body: read momenta, update these

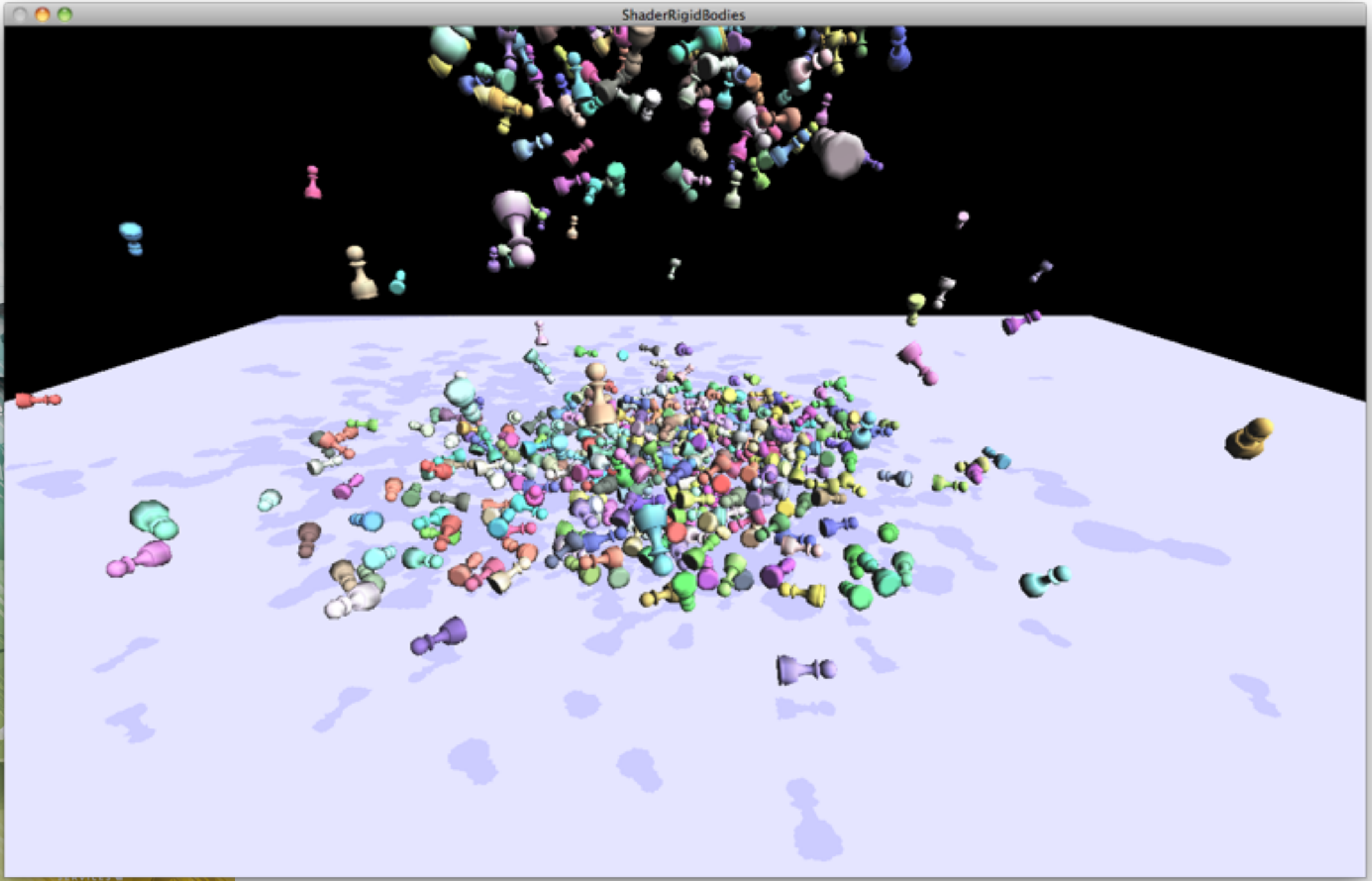
Data Structure

- All physical values are stored in arrays (texture)
- For each rigid body
  - Positions
  - Quaternion
  - Linear momentum
  - Angular momentum
- For each particle
  - Position
  - Velocity
  - Force
- For neighbor search
  - 3D grid



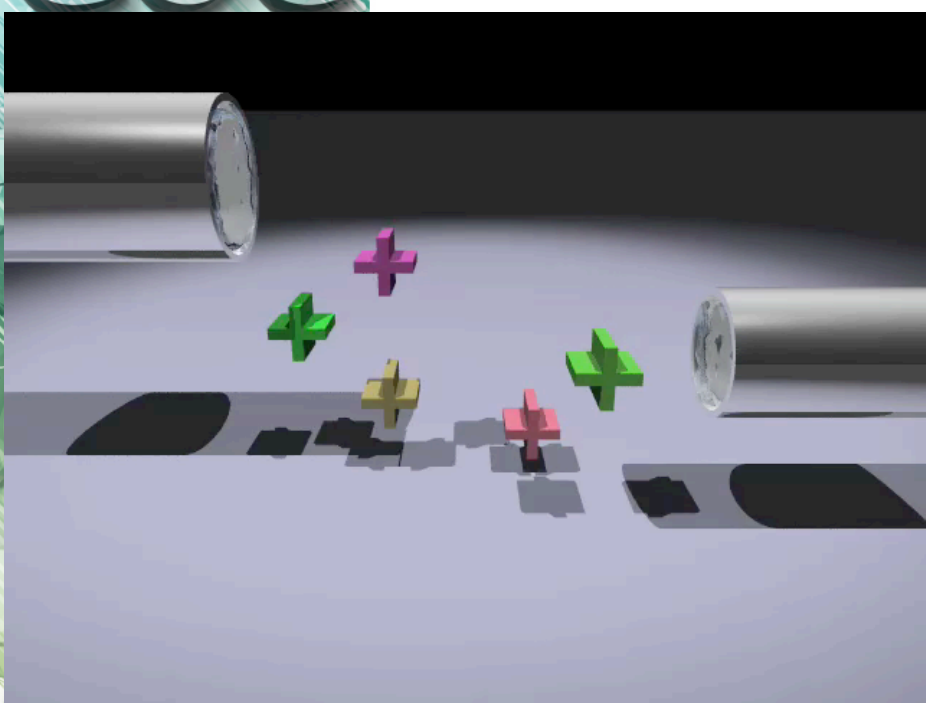


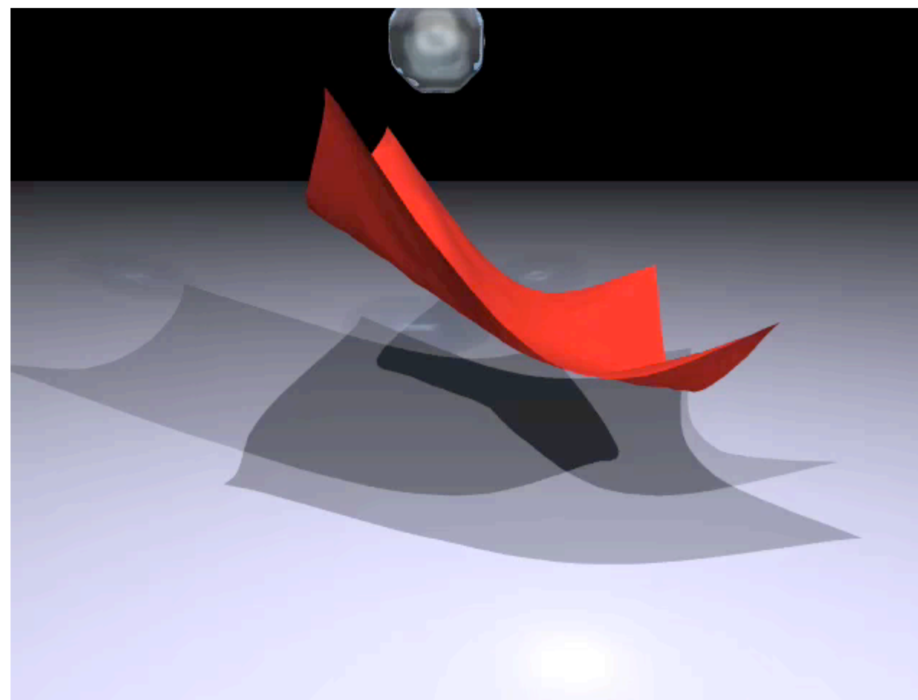
Demo



Extension

- It is easy to couple with fluid simulation and rigid bodies
- If there are more than particles
  - Particles + Mesh(cloth)
- Can solve using several grids
  - A grid for particles
  - A gird for mesh





Agenda

- » Introduction
- » Particle-based Simulations
- » Rigid bodies
- » Solving Constraints
- » Using Multiple GPUs

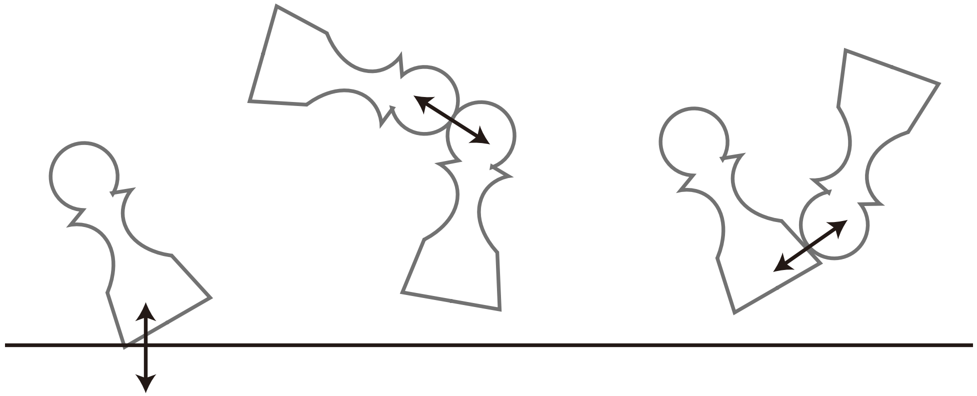
Solving Constraint

- Constraints are solved for velocity

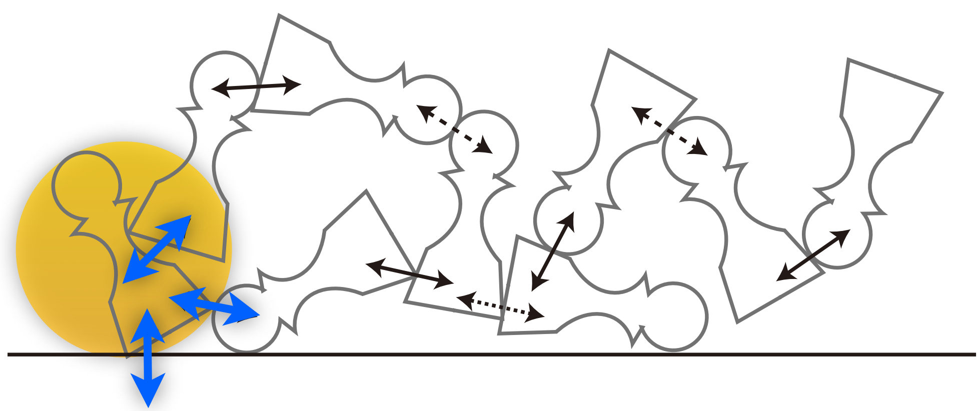
- Penalty based
  - Input: position, output: force
    - No dependency btwn input and output
  - No problem for parallel computation
- Impulse based
  - Input: velocity, output: velocity
    - Dependency btwn input and output
  - Problem when parallelizing
  - How to parallelize on the GPU?

Problem of Parallel Update

» 1:1 collision

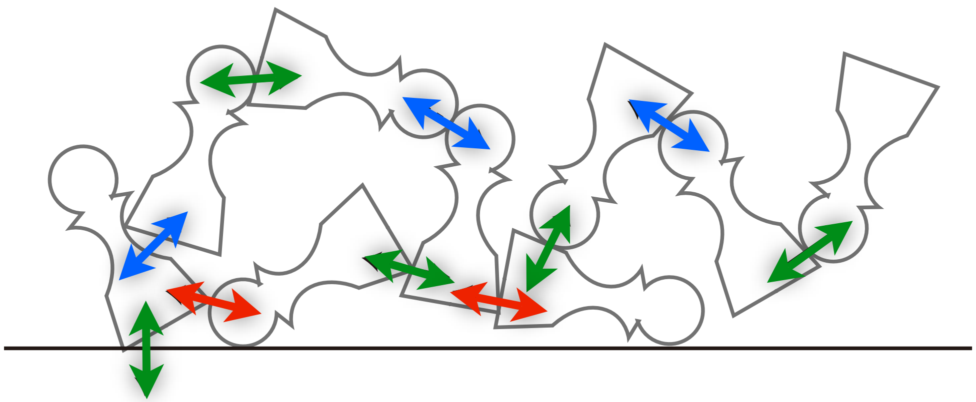


» 1:n collision



Batching

- Not update everything at the same time
- Divide them into several batches
- Update batches in sequential order
  - Update collisions in a batch can be parallel



- But how to divide into batches?? GPU??

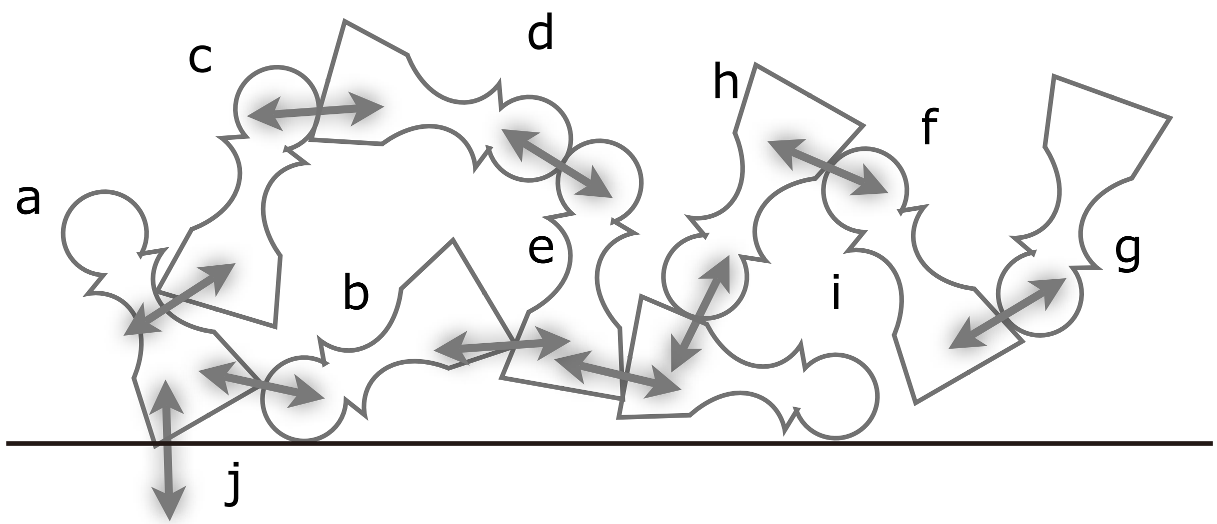
Chen et al., High-Performance Physical Simulation on Next-Generation Architecture with Many Cores, Intel Technology Journal, 11, 04

Dynamic Batching on GPU

- CPU can do this easily
  - Chen et al., High-Performance Physical Simulation on Next-Generation Architecture with Many Cores, Intel Technology Journal, volume 11 issue 04
- To implement on the GPU, the computation has to be parallel
- Do it by partially serialize the computation
  - Synchronization of several threads, which is available on CUDA, OpenCL

Dynamic Batching

» A thread is assigned for a constraint



Thread ID	0	1	2	3	4	5	6	7	8	9
Constraint	a,b	f, g	a, c	f, h	d, e	a, j	e, i	h, i	c, d	b, e

Dynamic Batching

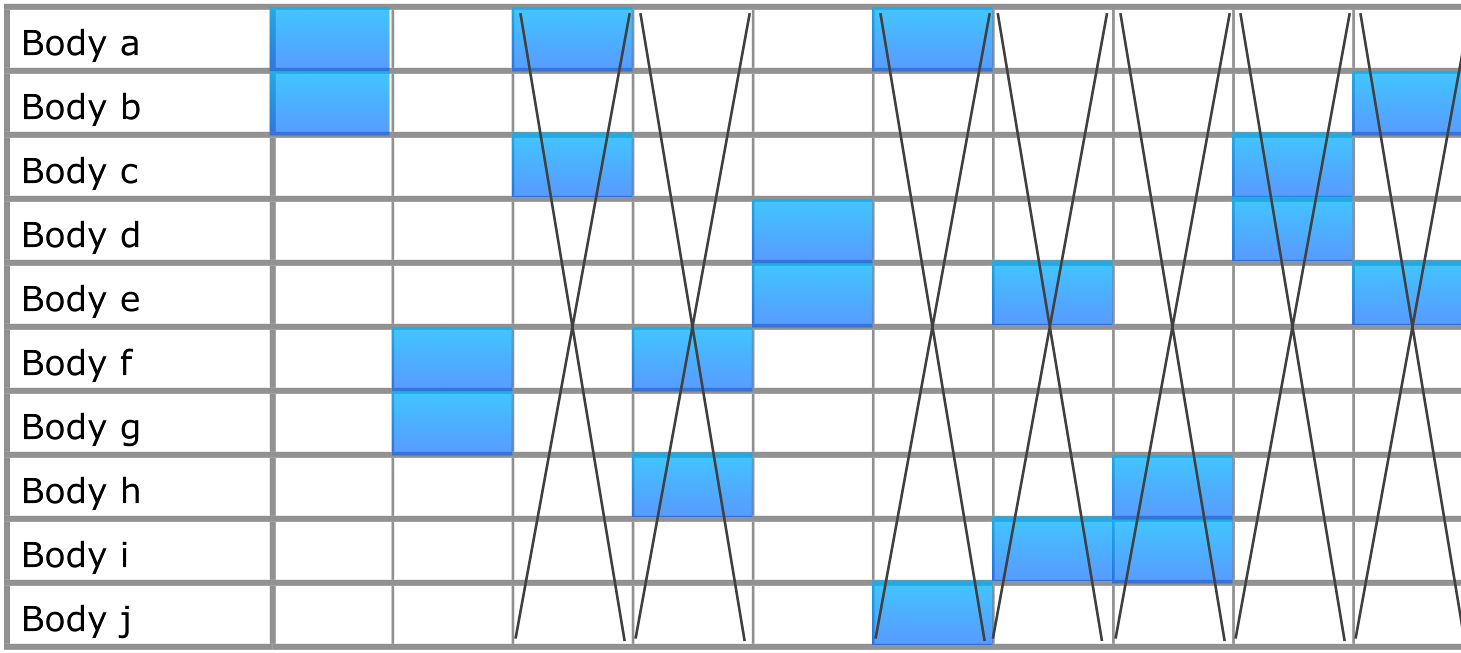
- A thread reads a constraint data
  - Thread0 reads a, b
- And flag a, b, if they are not flagged
- Can serialize operation in a block

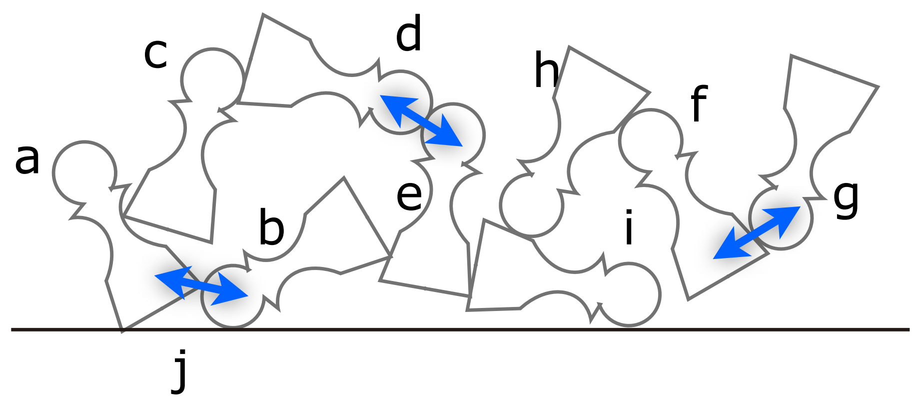
Thread ID	0	1	2	3	4	5	6	7	8	9
Constraint	a,b	f, g	a, c	f, h	d, e	a, j	e, i	h, i	c, d	b, e

Body a										
Body b										
Body c										
Body d										
Body e										
Body f										
Body g										
Body h										
Body i										
Body j										

Dynamic Batching

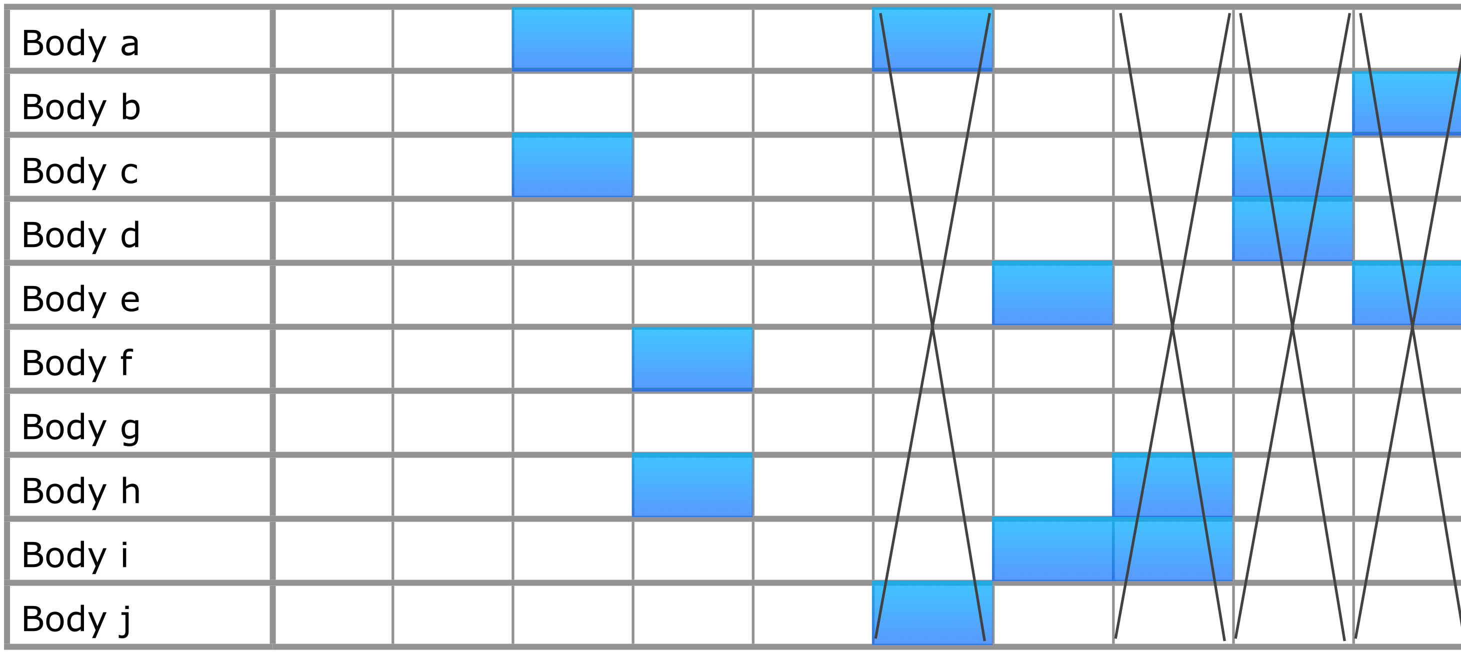
Thread ID	0	1	2	3	4	5	6	7	8	9
Constraint	a,b	f, g	a, c	f, h	d, e	a, j	e, i	h, i	c, d	b, e

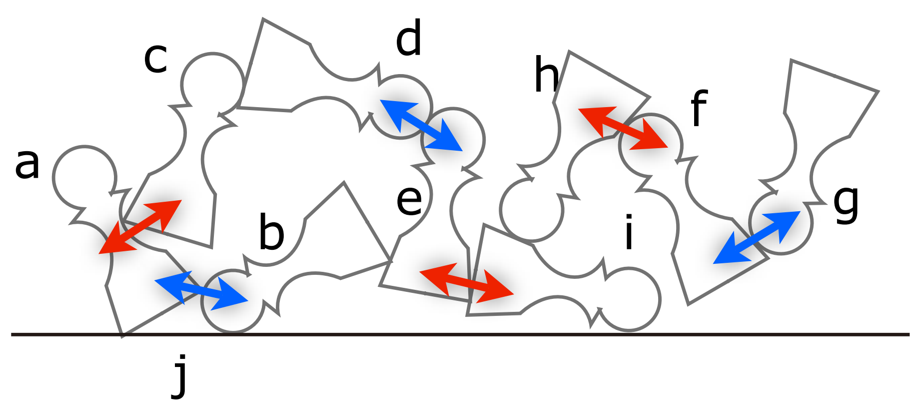




Dynamic Batching

Thread ID	0	1	2	3	4	5	6	7	8	9
Constraint	a,b	f, g	a, c	f, h	d, e	a, j	e, i	h, i	c, d	b, e

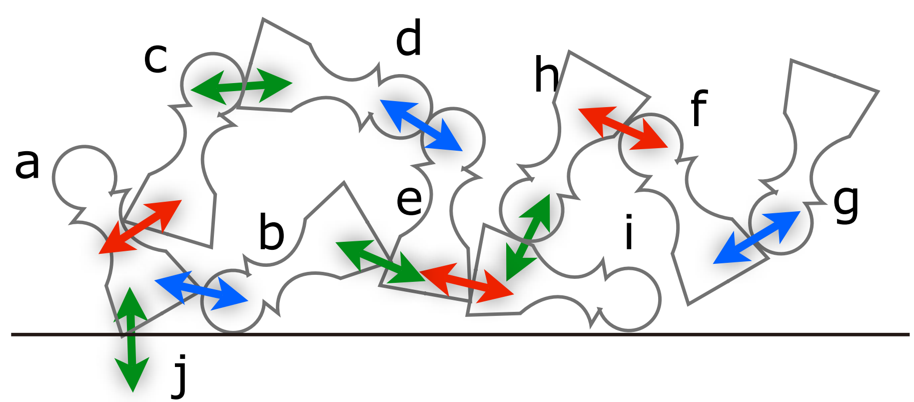




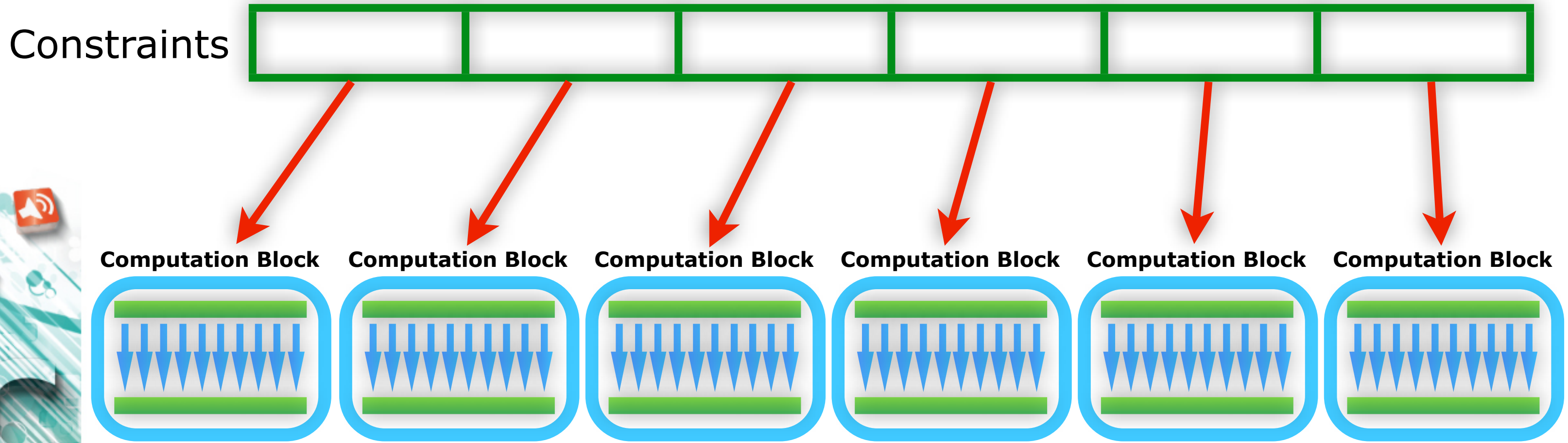
Dynamic Batching

Thread ID	0	1	2	3	4	5	6	7	8	9
Constraint	a,b	f, g	a, c	f, h	d, e	a, j	e, i	h, i	c, d	b, e

Body a										
Body b										
Body c										
Body d										
Body e										
Body f										
Body g										
Body h										
Body i										
Body j										



Parallelization of Batch Creation

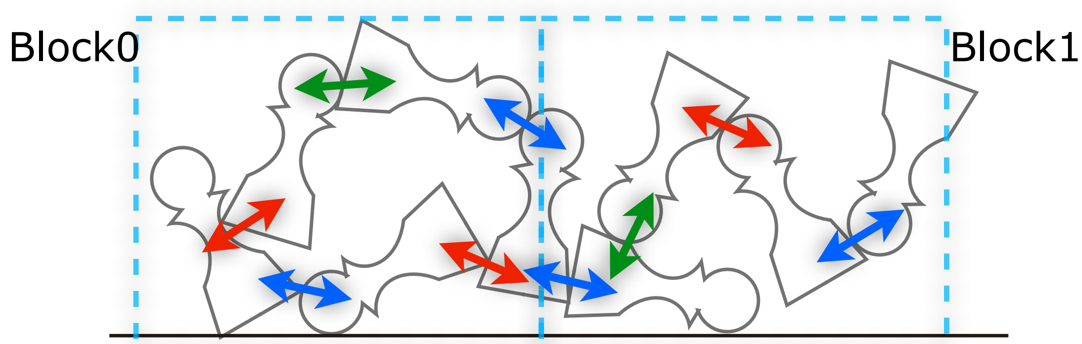


» Partially serial

» This operation creates wrong batch

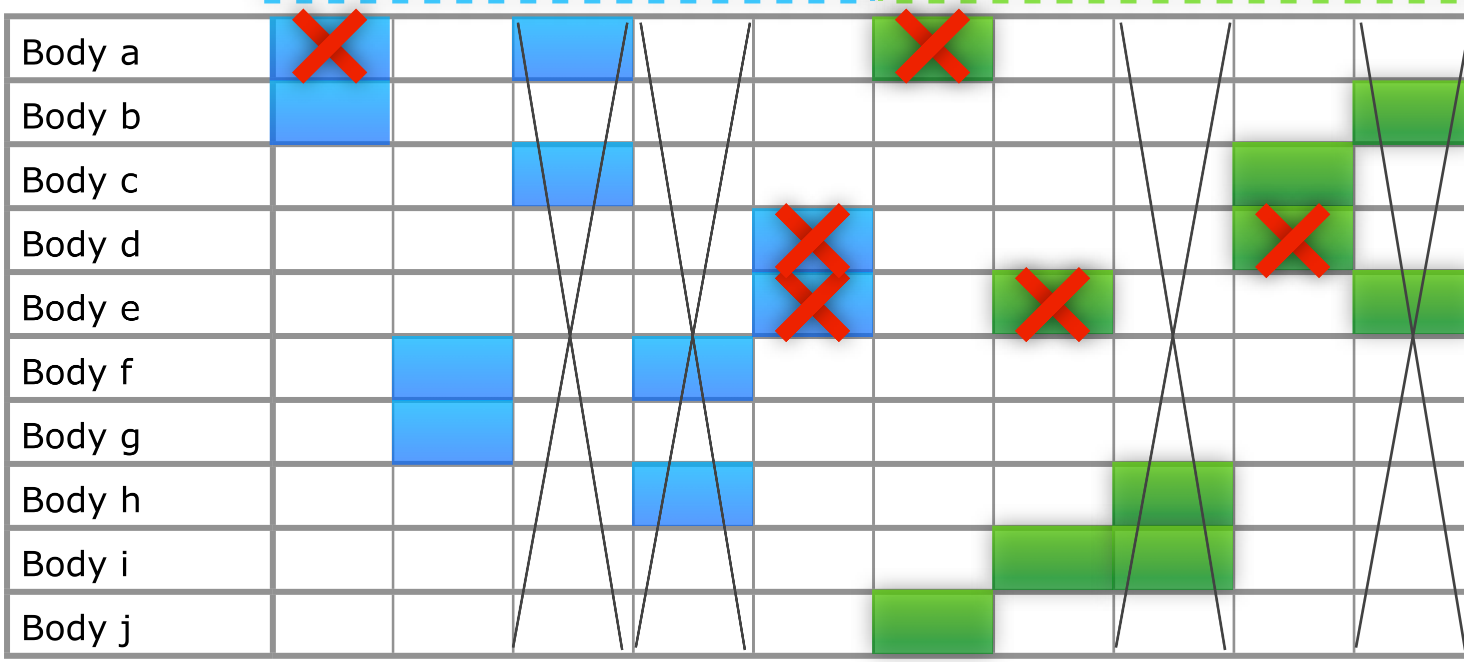
Each block doesn't know what the others are doing

Need another mechanism to solve this situation



Conflict

	Block0					Block1				
Thread ID	0	1	2	3	4	5	6	7	8	9
Constraint	a,b	f, g	a, c	f, h	d, e	a, j	e, i	h, i	c, d	b, e



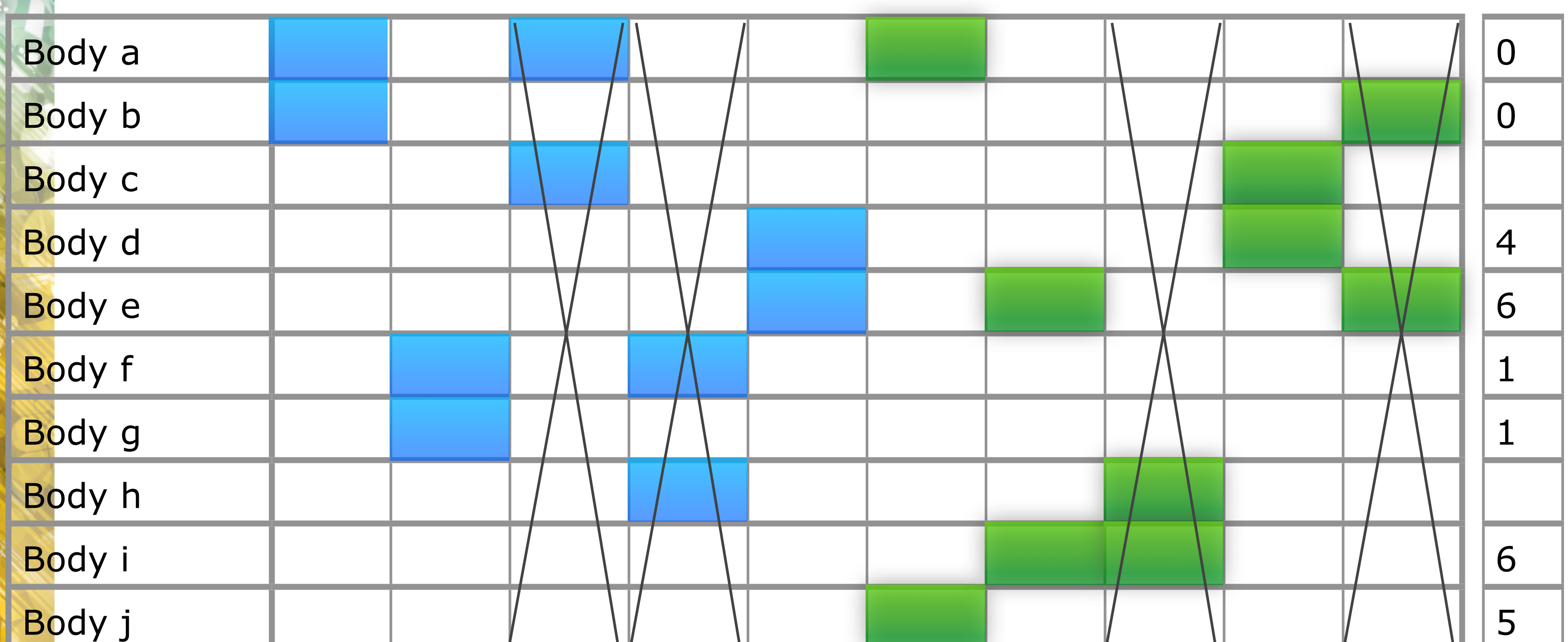
Solving Inconsistency

- Solving inconsistency requires global sync
  - Don’t want to do it often
- Run a kernel to check constraints are not shared after creating batch on each blocks

Solving Inconsistency

- On batch creation, write constraint idx to a buffer without any sync (cheap)
  - The value can be overwritten by other thread if the body is shared among several constraint
- After batch creation, check if the constraint idx matches between two bodies

Thread ID	0	1	2	3	4	5	6	7	8	9
Constraint	a,b	f, g	a, c	f, h	d, e	a, j	e, i	h, i	c, d	b, e



GDC 10
Learn. Network. Inspire.
THINK SERVICES

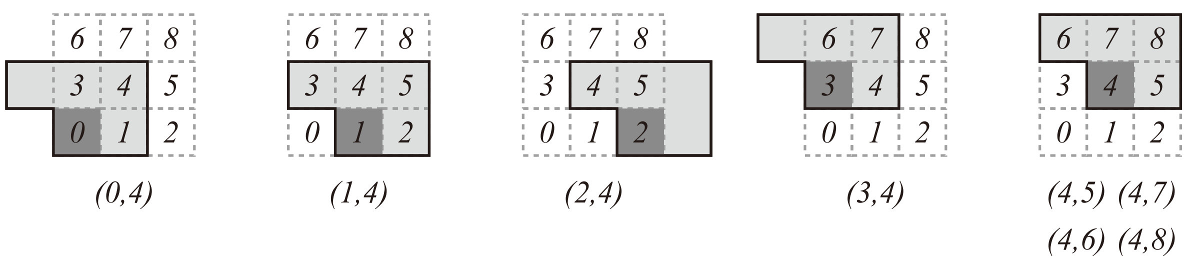
Procedure

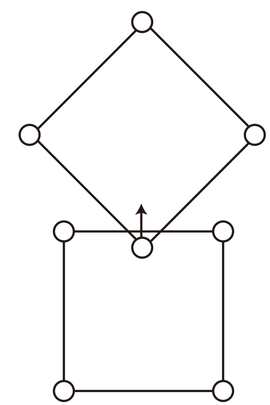
- Batch 0
  - Clear the buffer
  - Write indices sequentially in a warp
  - Check if the write is valid
- Batch 1
  - Clear the buffer
  - Write indices sequentially in a warp
    - Skip a constraint registered already
  - Check if the write is valid
- Batch 2
  - Clear the buffer
  - Write indices sequentially in a warp
    - Skip a constraint registered already
  - Check if the write is valid

Game Developers Conference®
March 9-13, 2010
Moscone Center
San Francisco, CA
www.GDConf.com

Demo Overview

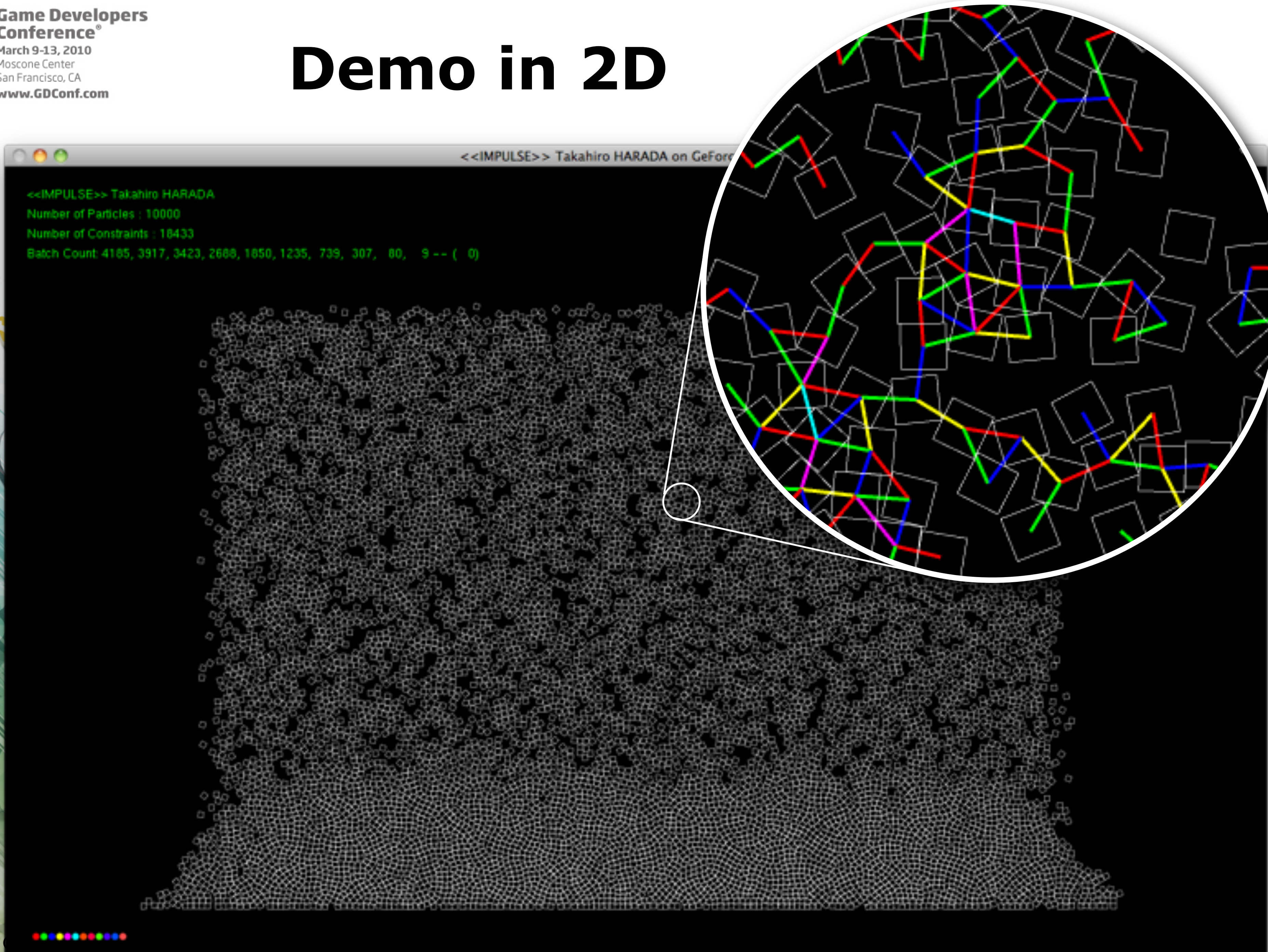
- » Broadphase
  - Uniform grid(using symmetry)
- » Narrowphase
  - Distance function with sample points
- » Constraint Solve
  - Projected Gauss Seidel





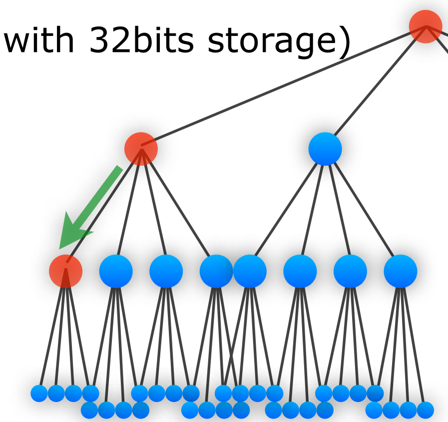
Learn. Network. Inspire.

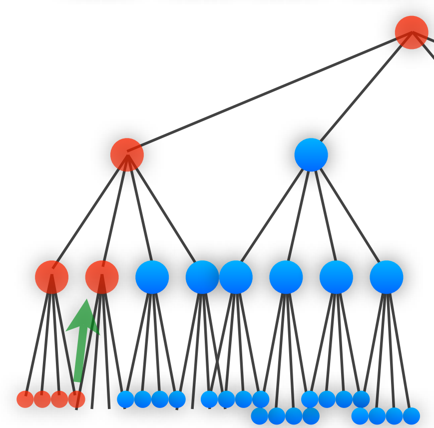
Demo in 2D


<<IMPULSE>> Takahiro HARADA on GeForc
<<IMPULSE>> Takahiro HARADA
Number of Particles : 10000
Number of Constraints : 18433
Batch Count: 4185, 3917, 3423, 2688, 1850, 1235, 739, 307, 80, 9 -- (0)

TreeTraversal using HistoryFlags

- Traversal
  - Using stack is not GPU friendly - Need a lot of local storage (↑Resource, ↓Performance)
  - Adding Escape sequence is not good choice for GPU
- Traversal using History Flags
  - Simple
  - **Small** local resource (2^{32}leaves can be traversed with 32bits storage)
  - No additional computation on build
- Quad Tree
  - 4 bits for each level
  - Flag traversed node
- Procedure
  - Initialize:“0000”
  - Next node:Find first “0”
  - After visit a node “1” (Flag)
  - Descend:Move to next bits
  - Ascend:Move to prev bits
    - No more “0” on this level
    - Clear this level to “0000”
- **Good** Performance compared to stack traversal





Agenda

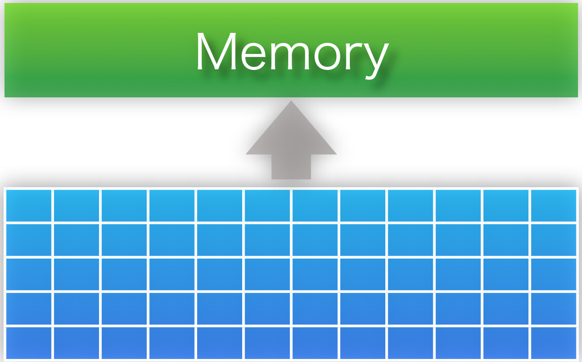
- Introduction
- Particle-based Simulations
- Rigid bodies
- Solving Constraints
- Using Multiple GPUs

Learn. Network. Inspire.

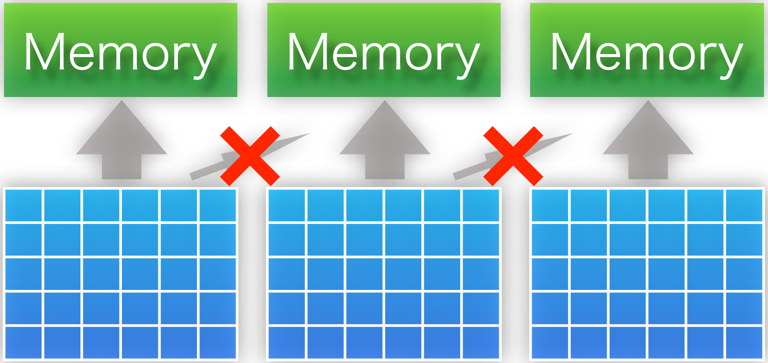
THINK SERVICES

Using Multiple GPUs

- Cannot run applications developed for a GPU
- Need two levels of parallelization
- 1GPU



- Multiple GPUs



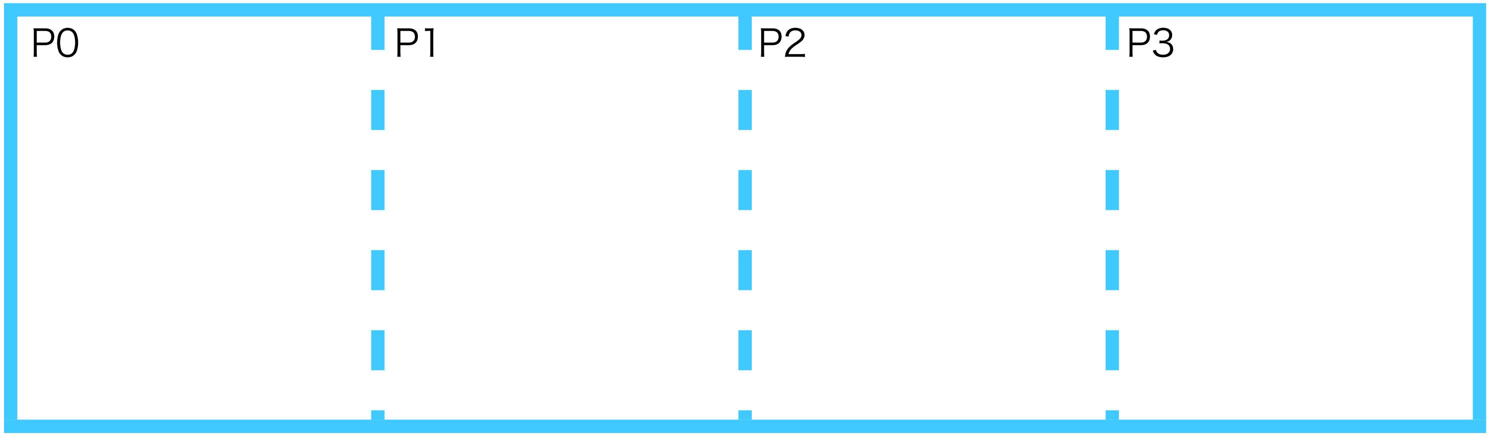
Parallelization using Multiple GPUs

- Grid-based simulation
  - Connectivity of simulation elements is fixed
  - Boundary elements are also fixed
  - Easy to split a simulation by space
  - Relatively easy to parallelize on Multiple GPUs
- Particle-based simulation
  - Connectivity of simulation elements is dynamic
    - Have to search for neighbors each time step
  - Not obvious how to parallelize on Multiple GPUs

Decomposition

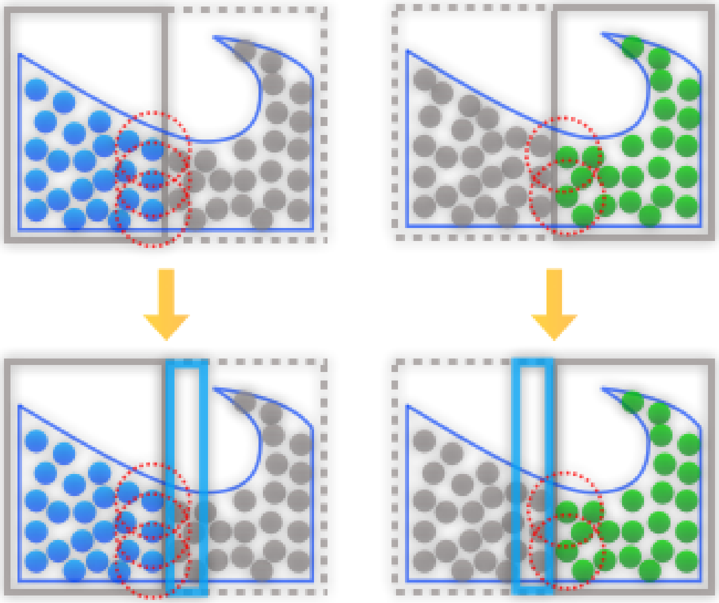
- Space decomposition is employed (instead of index based)
  - Simulation domain is split into sub domains

- Have to assign particles to each processor dynamically



Decomposition of Computation

- Computation of particle values requires values of neighbors
  - Inside of subdomain: all the data is in the memory of its own
  - Boundary of subdomain: some data is in the memory of others
- Have to read data from other GPUs
  - Communicating when required makes the granularity of transfer smaller and inefficient
- Transfer only “Ghost Region” and “Ghost Particles”
  - Ghosts are not updated
  - Just refer the data



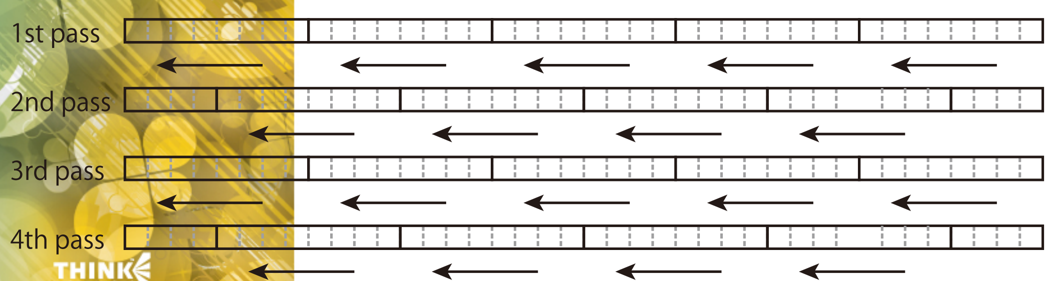
Data Management

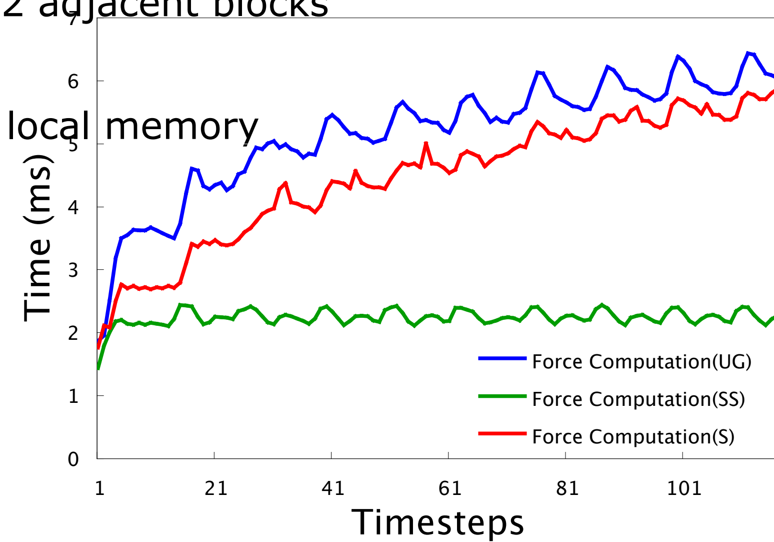
- Don't have to send all the data
- The particles have to be sent are
    1. Particles go out from my subdomain(Escaped particles)
    2. Particles in ghost region(Ghost particles)

- Scanning all the particles to flag them are too expensive for high frequency simulation
- Instead, reused grid constructed for neighbor search to select particles to be sent(Sliced grid is used instead of uniform grid)

Introduction of sorting to improve cache efficiency

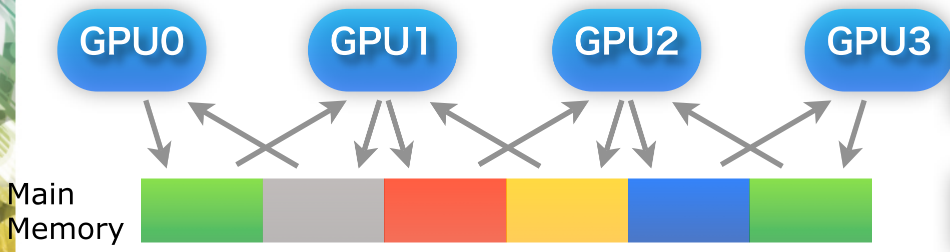
- As simulation proceeds, particles are mixed up
  - Spatial coherency is lost
  - Can sort by spatial order to increase cache hit
- ~~Full sort is an option~~
  - ~~**But** timing Radix sort > neighbor search~~
- Observation
  - Full sort isnt necessary
  - Can exploit frame coherency
- Block Transition Sort
  - Generalization of Odd-even transition sort
    - 2 adjacent elements -> 2 adjacent blocks
  - No random read/write
  - Good for processor with fast local memory

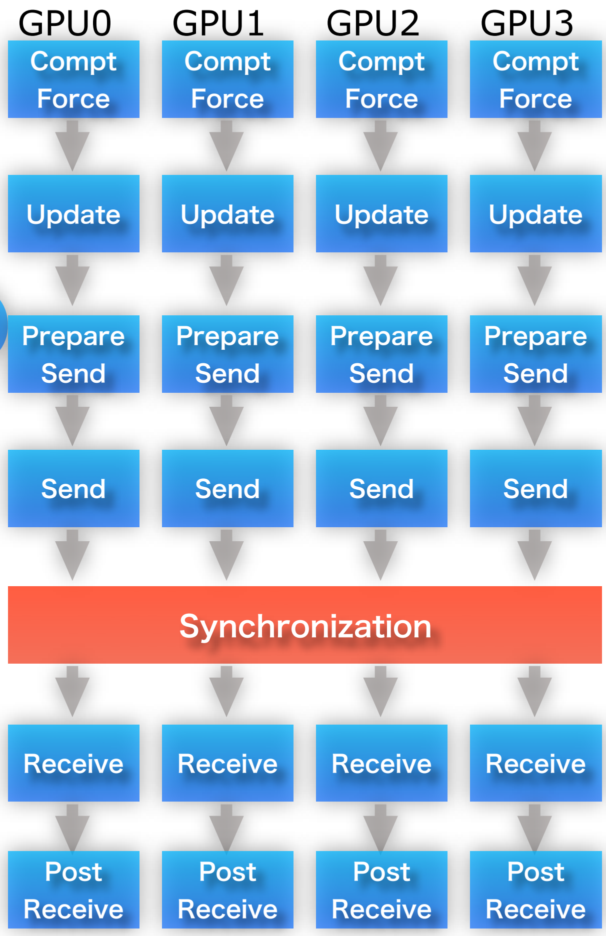




Data Transfer btwn GPUs

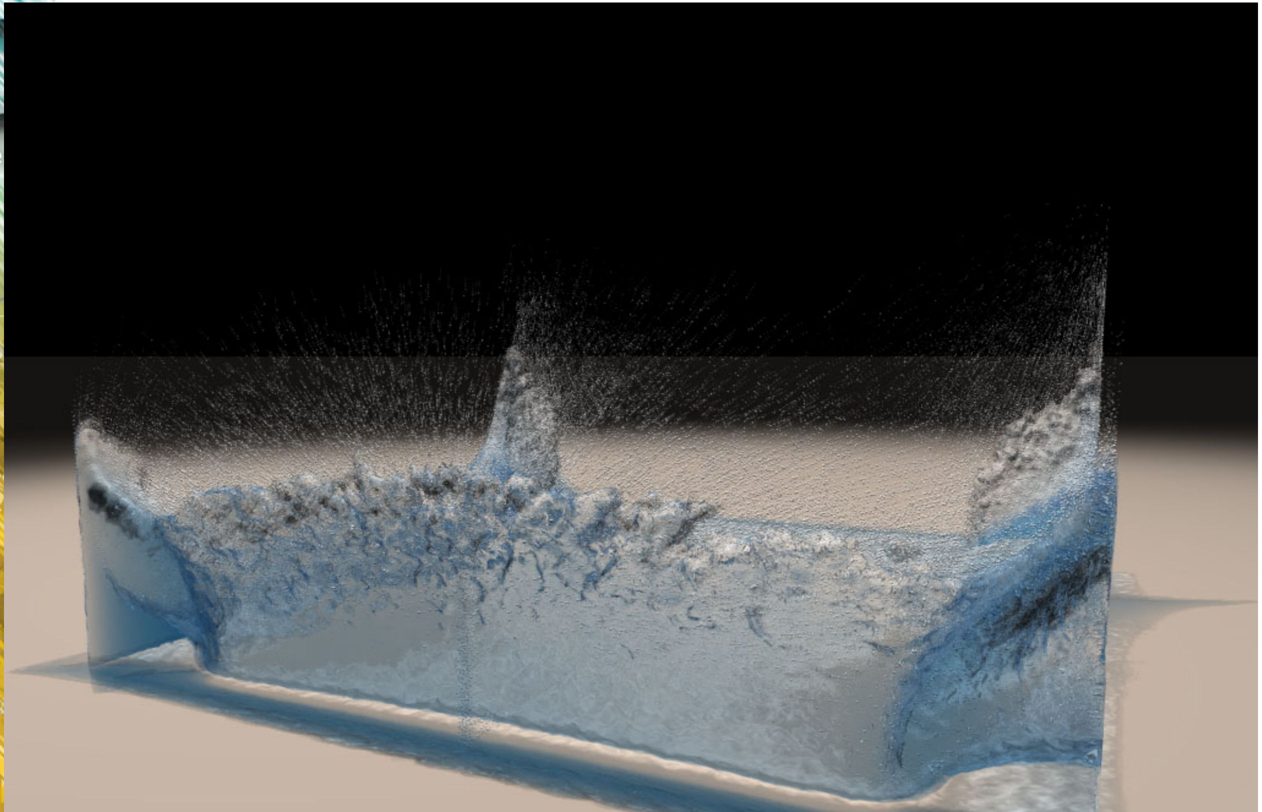
- No way to direct transfer
- Write to main mem from GPU mem, then read from another GPU



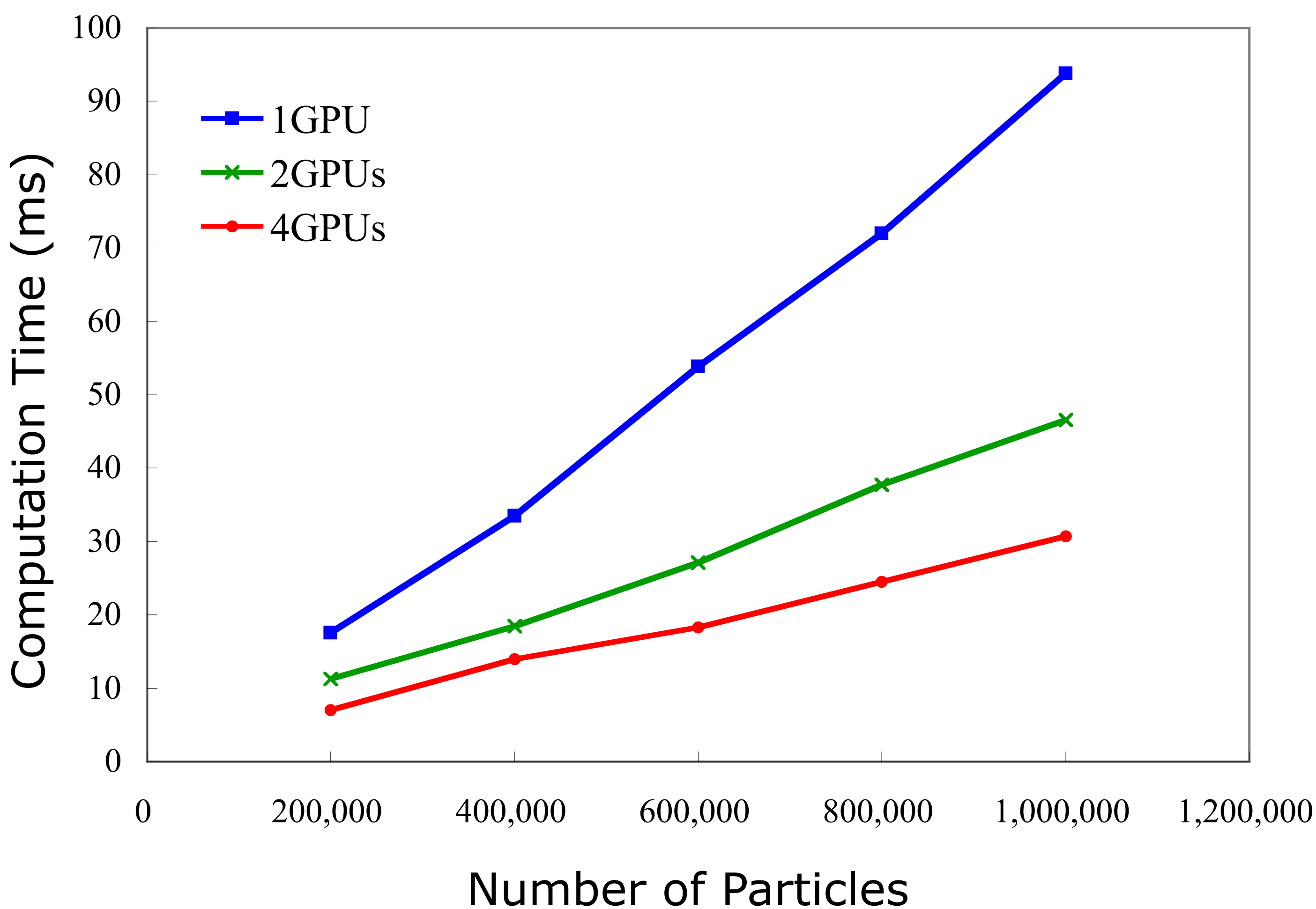


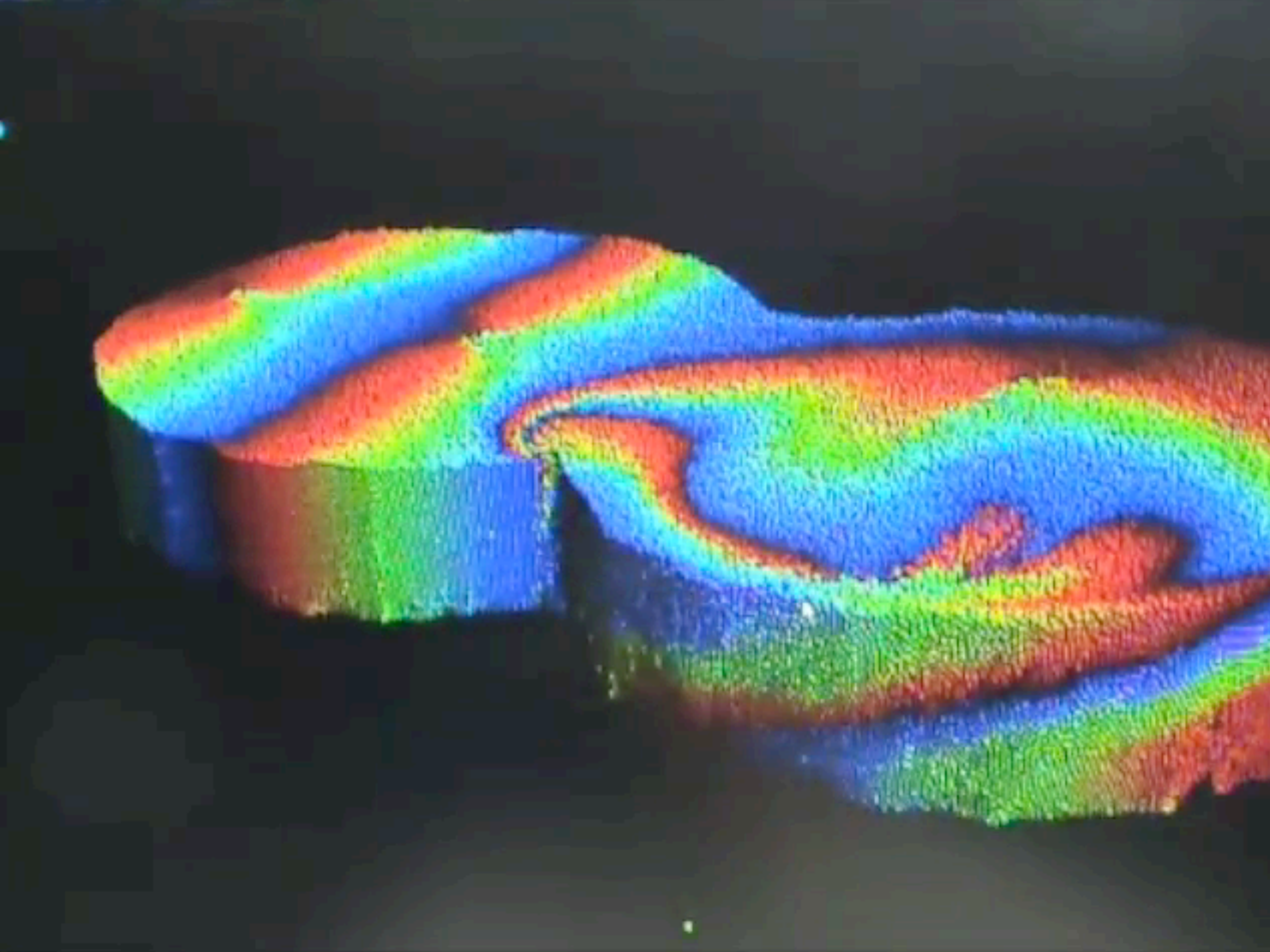
Environment

- 4GPUs(Simulation) + 1GPU(Rendering)
  - S870 + 8800GTS
- 6GPU(Simulation) + 1GPU(Rendering) @GDC2008
  - QuadroPlex x 2 + Tesla D870 + 8800GTS



Results


1GPU
2GPUs
4GPUs
Computation Time (ms)
0
10
20
30
40
50
60
70
80
90
100
0
200,000
400,000
600,000
800,000
1,000,000
1,200,000
Number of Particles



Thanks

» Slides and Demos :
  http://www.iii.u-tokyo.ac.jp/~takahiroharada/