# Character Clothing in PhysX-3

Tae-Yong Kim, NVIDIA Corporation

NVIDIA

# Outline

- Introduction to PhysX-3
- Algorithms for New Clothing Solver
- Character Clothing Pipeline

## What is PhysX?

***PhysX is NVIDIA's game physics technology, scalable across a wide range of platforms from multicore CPUs, GPUs, PS3, Xbox 360 to iPhone and Android.***

# PhysX SDK Timeline

2004
*AGEIA acquires Novodex*

2008
*NVIDIA acquires AGEIA*

**2011:**
***PhysX-3…***

Novodex SDK

PhysX by AGEIA
(from v2.0 through 2.8.1)

PhysX by NVIDIA

2002:
*Novodex SDK ships for PC/Windows*

2005/06:
*PS3 and XBox360 ports appear*

2007:
*AGEIA PhysX PPU available*

2008:
*GPU/CUDA port released*

2010:
*Support for iPhone, iPad and Android*

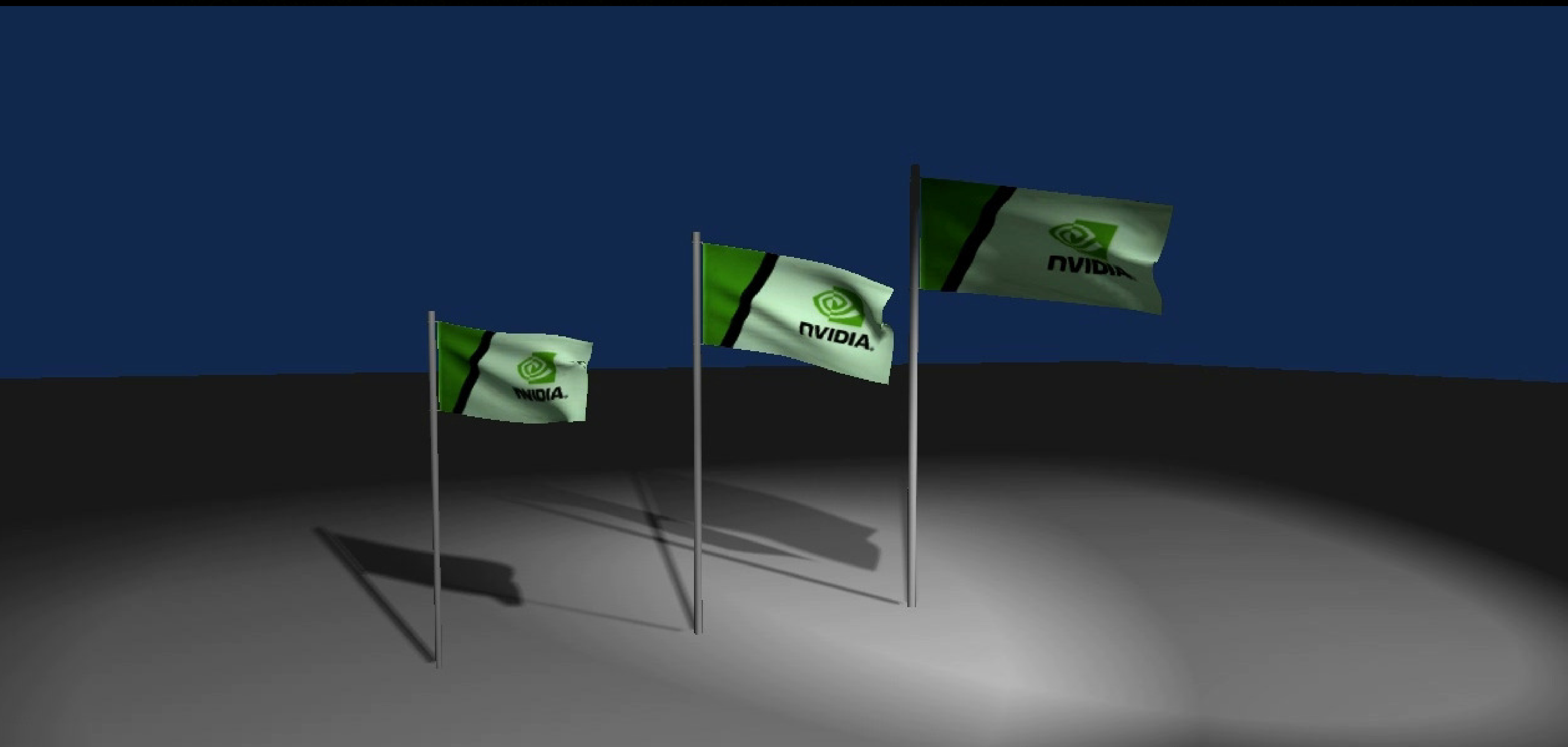
# PhysX-3

***PhysX-3 is the first complete rewrite of PhysX SDK since 2004, targeted for better performance and usage, built upon experience from supporting 200+ game titles.***

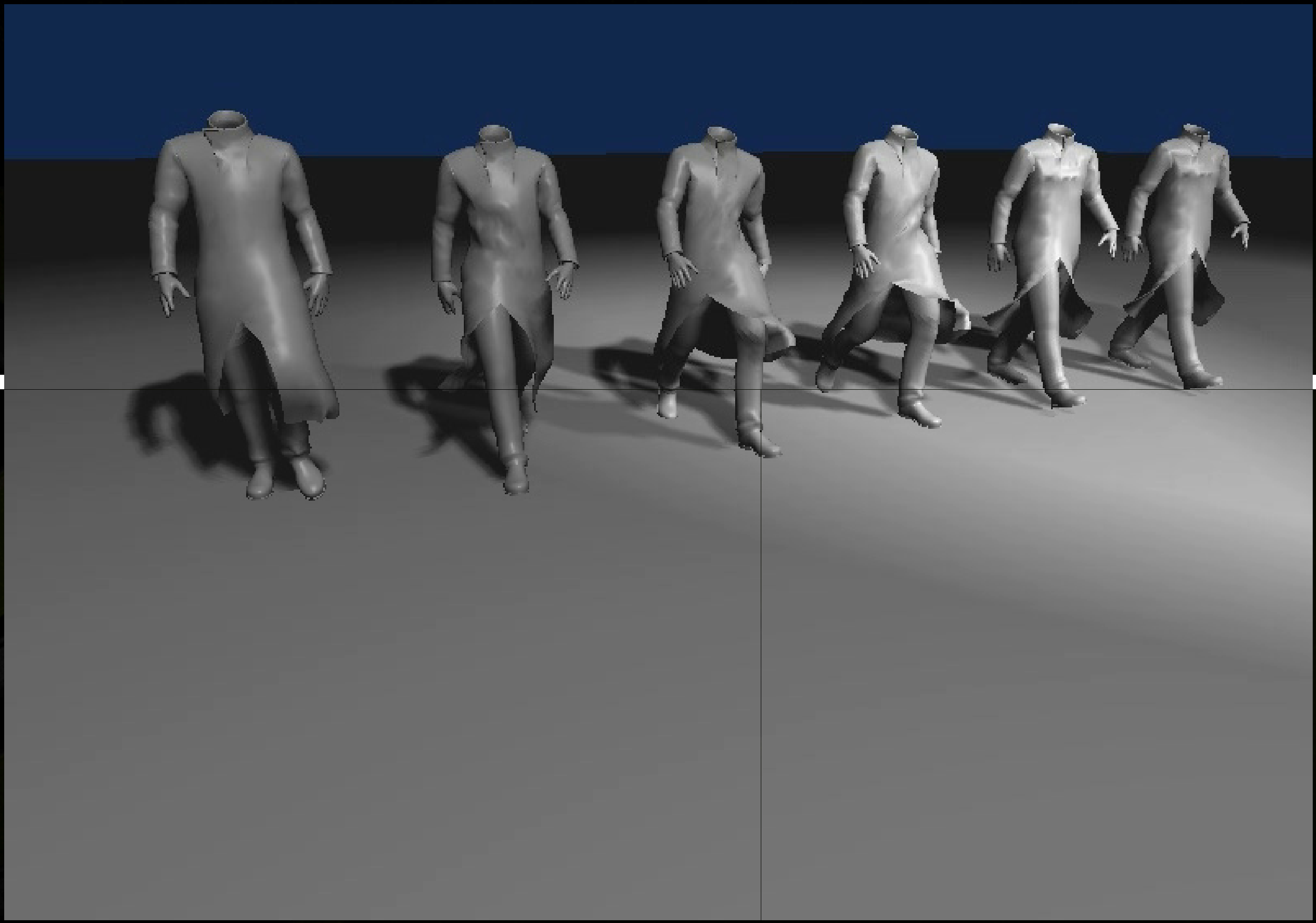
# PhysX-3 Clothing

- Complete rewrite of cloth engine
- New constraint solvers
- New collision algorithms
- Character oriented API and features
- Supports CPU (SSE), GPU (Cuda), PS3, Xbox, Android, iOS, Linux, …

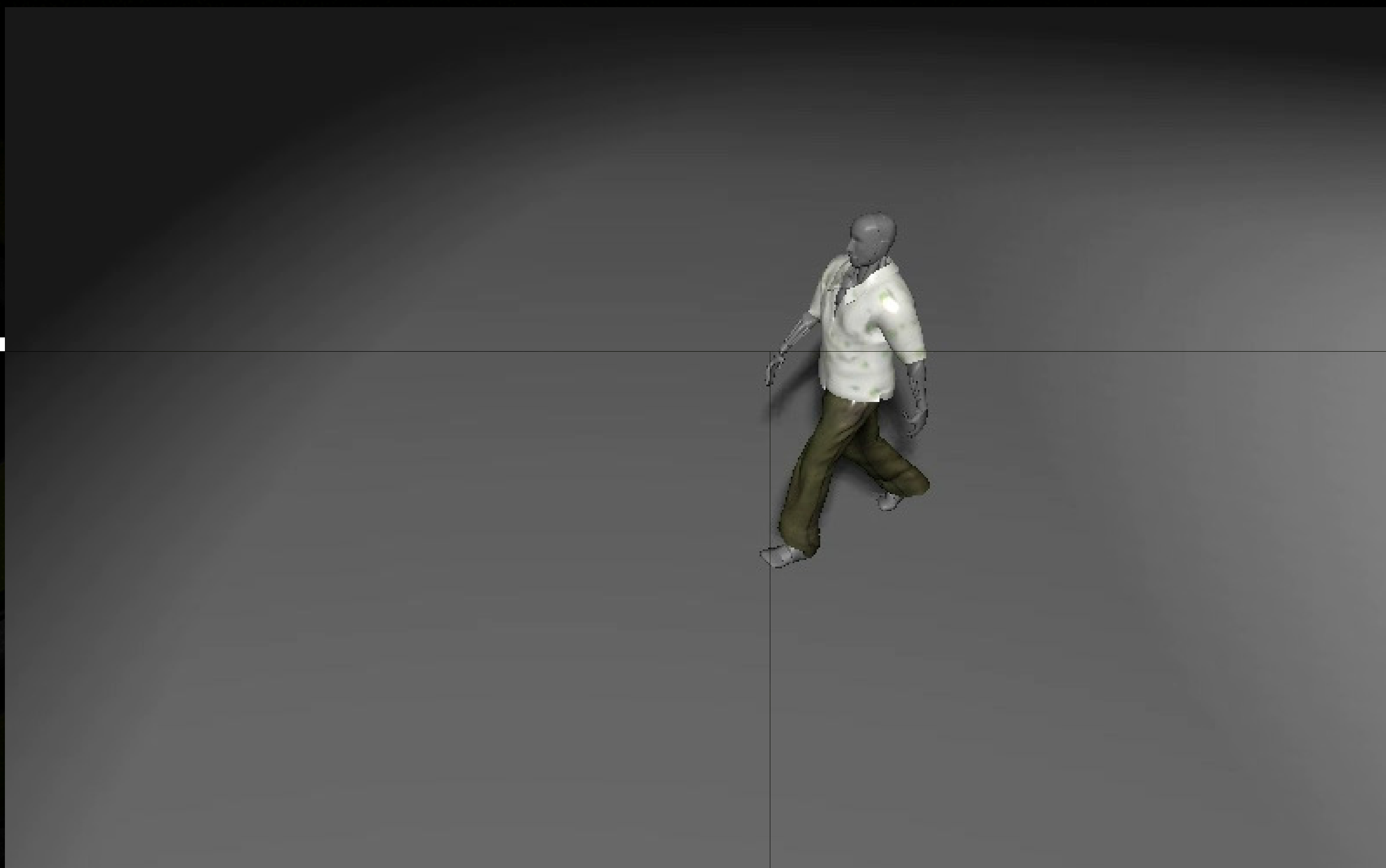
# PhysX-3 Clothing



# PhysX-3 Clothing



# PhysX-3 Clothing



# Solving Cloth Motion

# Position Based Dynamics (PBD)

- **Introduced in 2006 (Müller et al.)**
- **Apply constraints on distance, angle, collision, etc.**
- **Stable and efficient for real-time applications**
  - **Near industry standard in game physics (e.g. Bullet dynamics)**

M. Müller, B. Heidelberger, M. Hennix, J. Ratcliff, **Position Based Dynamics**, *Proceedings of Virtual Reality Interactions and Physical Simulations (VRIPhys) 2006*

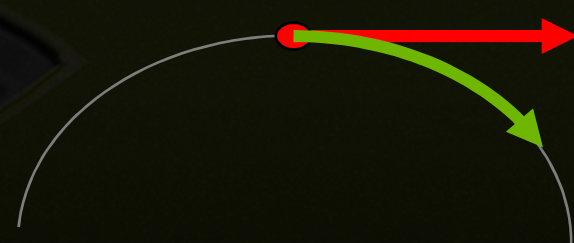
# Explicit Euler Integration

- Explicit Euler with $\Delta t$

$$\mathbf{v}_i^{t+1} = \mathbf{v}_i^t + \Delta t \frac{1}{m_i} \sum_j \mathbf{f}(\mathbf{x}_i^t, \mathbf{v}_i^t, \mathbf{x}_j^t, \mathbf{v}_j^t)$$

$$\mathbf{x}_i^{t+1} = \mathbf{x}_i^t + \Delta t \, \mathbf{v}_i^{t+1}$$

- Assumes velocity and force constant within $\Delta t$



# Position Based Dynamics

Init all $\mathbf{x}_i^0, \mathbf{v}_i^0$

**Loop**

$\mathbf{p}_i = \mathbf{x}_i^t + \Delta t \cdot \mathbf{v}_i^t$ // prediction

$\mathbf{x}_i^{t+1}$ = modify $\mathbf{p}_i$ // position correction (solve constraints)

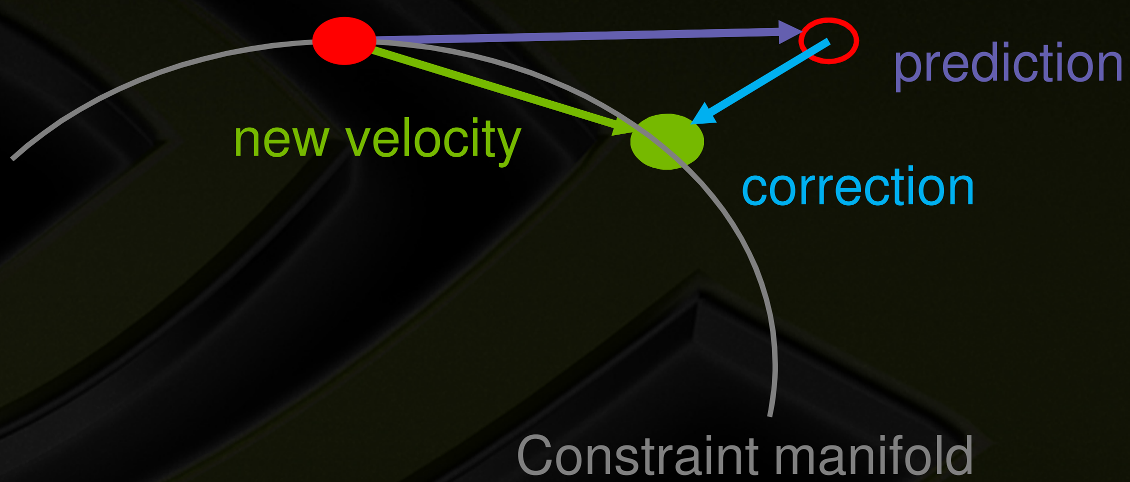
$\mathbf{u}_i = [\mathbf{x}_i^{t+1} - \mathbf{x}_i^t] / \Delta t$ // velocity update

$\mathbf{v}_i^{t+1}$ = modify $\mathbf{u}_i$ // velocity correction (damping, etc.)

**End loop**

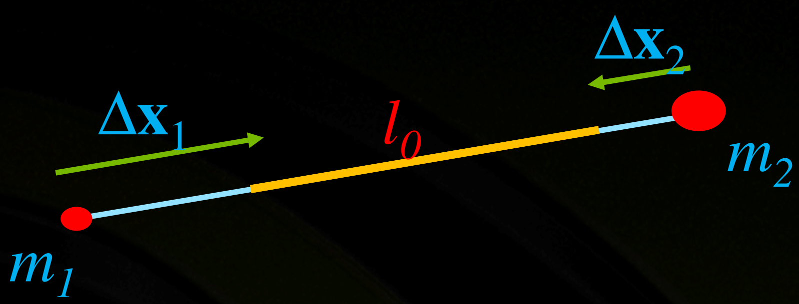
# Position Based Dynamics

- Integrate without constraints first (prediction)
- Then move particles such that constraints are satisfied (correction)
- Compute new velocity from new position and old position




- Dynamics are stable as long as constraint solvers converge
- Solving constraints (position correction) become key issues

# Distance Constraints




$$\Delta \mathbf{x}_1 = -\frac{w_1}{w_1 + w_2}\left(|\mathbf{x}_1 - \mathbf{x}_2| - l_0\right)\frac{\mathbf{x}_1 - \mathbf{x}_2}{|\mathbf{x}_1 - \mathbf{x}_2|}$$

$$\Delta \mathbf{x}_2 = +\frac{w_2}{w_1 + w_2}\left(|\mathbf{x}_1 - \mathbf{x}_2| - l_0\right)\frac{\mathbf{x}_1 - \mathbf{x}_2}{|\mathbf{x}_1 - \mathbf{x}_2|}$$

$$w_i = 1 / m_i$$

- **Move each end point toward or away from each other**
- **Scale by inverse mass *w***
  - **preserve momentum**
  - **zero *w* for constrained particle**
- **We need to solve ALL the constraints**

# Solving Distance Constraints

- Global constraint enforcement
  - Non-linear problem to solve
  - Apply linearization

# Solving Distance Constraints

- **First linearize constraints**

$$C(p) = 0, G = \nabla C$$
$$GM^{-1}G^{T}\lambda = b$$

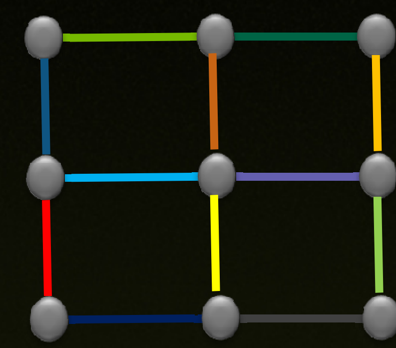
- **We solve λ for each constraint**

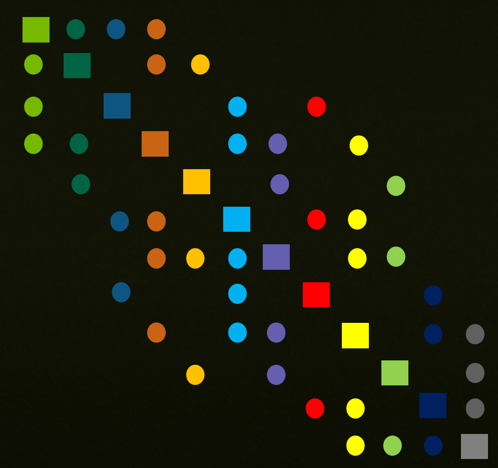
$$A\lambda = b$$

- **Then compute x (displacement)**

$$x = -M^{-1}G^{T}\lambda$$

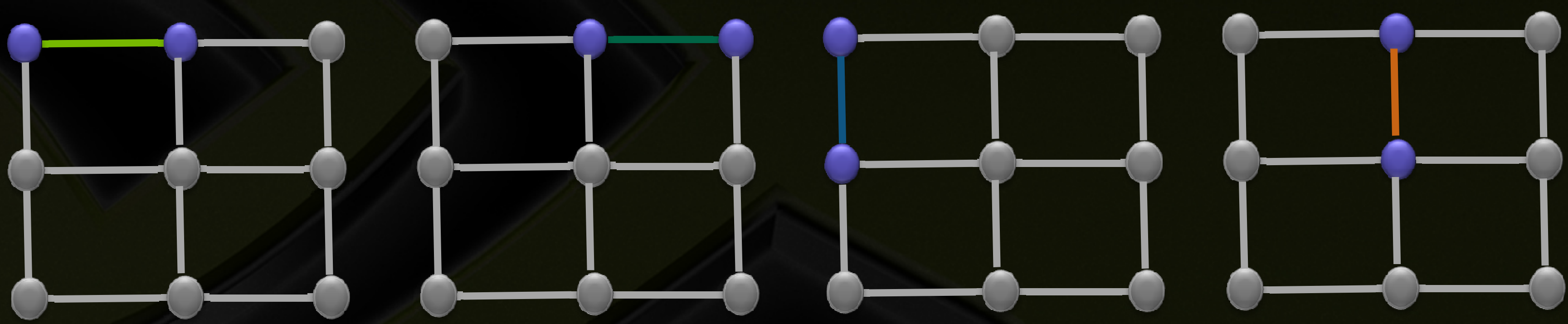
- **Sparse matrix problem**





# Gauss-Seidel Method for Constraints

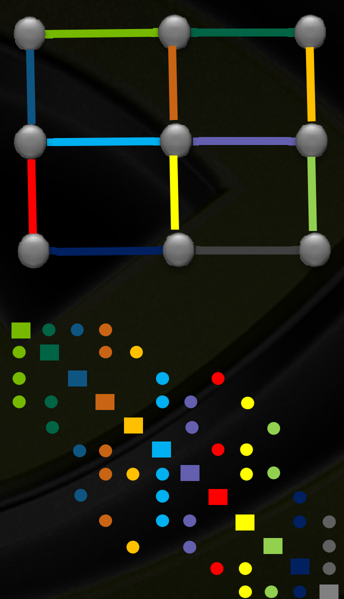
- Iterate through each constraint and update positions immediately



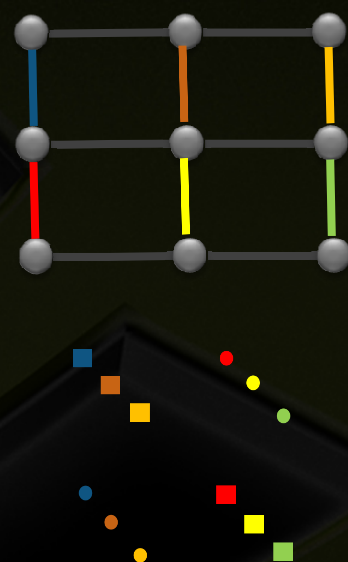
- May not converge very well for large number of constraints

# Gauss Seidel - Can we do better?

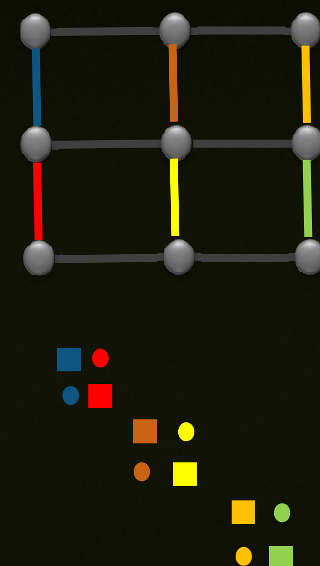
- **Pivoting for better convergence**



All the constraints



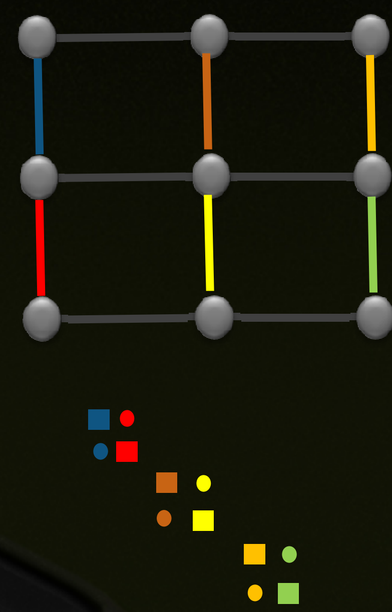
Vertical constraints



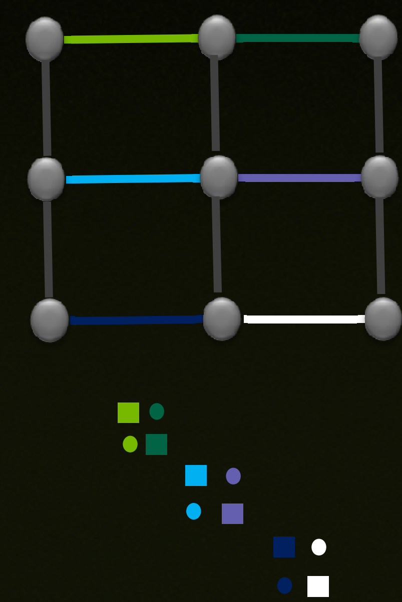
Vertical constraints
after pivoting

# Gauss Seidel - Can we do better?

- Block Gauss-Seidel
  - Assemble neighboring constraints to a block
  - Solve each block independently
  - Iterate over every block (in parallel)



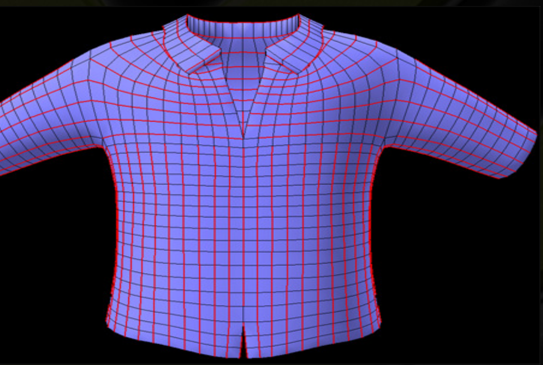
Vertical constraints after pivoting

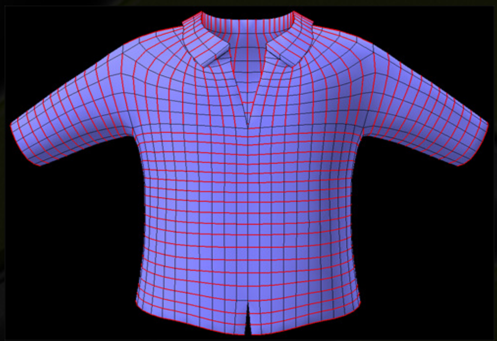


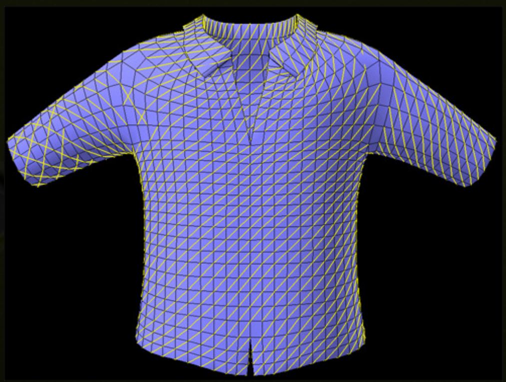
Horizontal constraints after pivoting

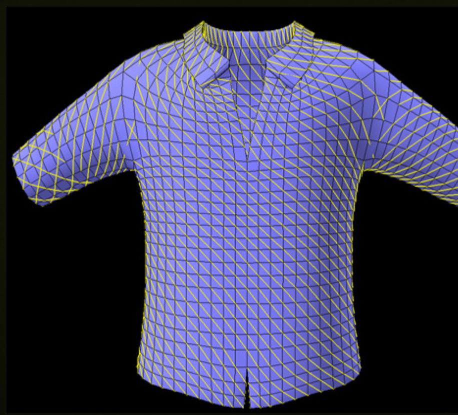
# Fiber and Set

- **Fiber** Independent set of connected constraints
- **Set** Non-overlapping fibers that can solved in parallel
- **Cooker** A special program that generates fibers and sets from mesh


Stretch set 1

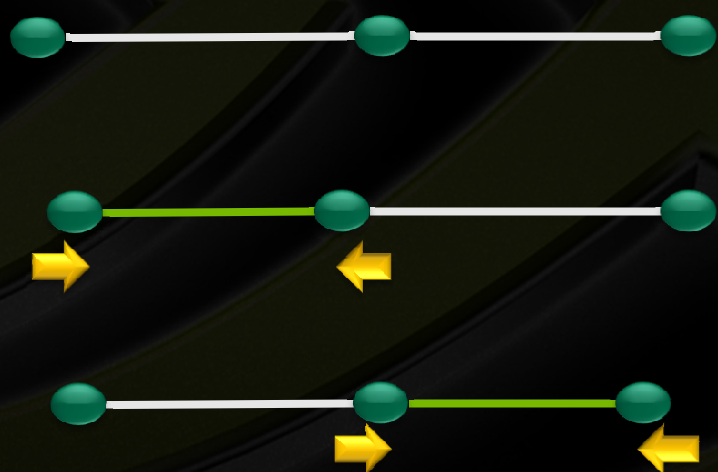

Stretch set 2


Shear set 1


Shear set 2

# Gauss-Seidel Solver for a Fiber Block

- One constraint at a time
- Immediately update results and work on the next

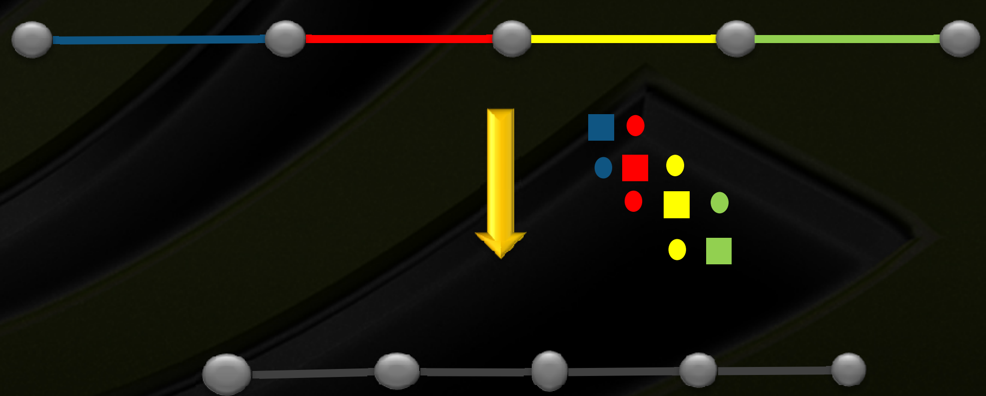


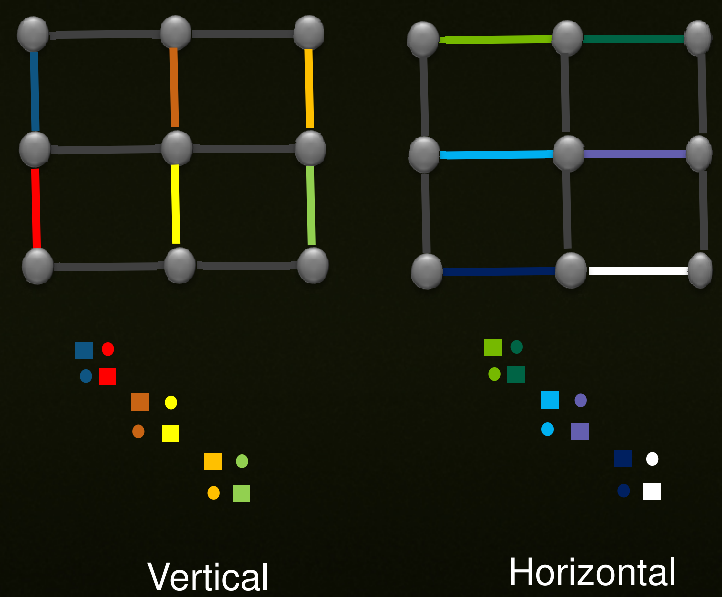
- Slow Convergence for stiff constraints
- Many iterations needed for inextensible fibers

**Idea:** use direct solver instead

# Semi Implicit Solver

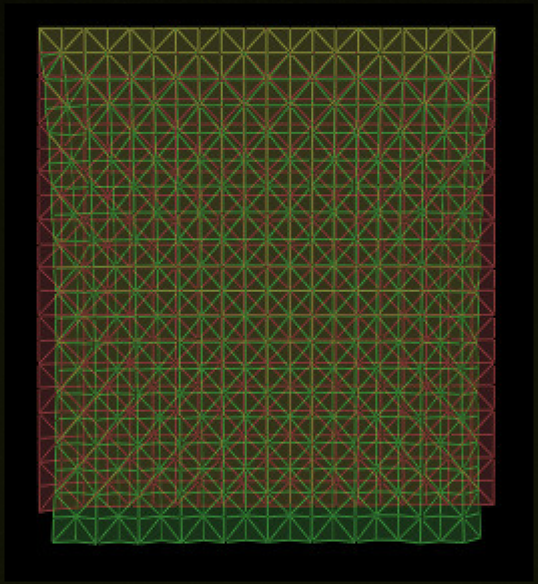
- $LDL^T$ factorization of tri-diagonal system
- For performance, L and D not explicitly constructed
- Can be ill-conditioned, special treatment needed
- A momentum preserving implicit method





# Gauss Seidel vs Semi Implicit Solver

- **Gauss-Seidel**
  - Simple and easy to tweak
  - Per iteration cost is low
  - Convergence is low → stretchy cloth
- **Semi Implicit**
  - 10x + convergence
  - Per iteration cost is higher (2.5x)
  - Some modification (e.g. limit) difficult

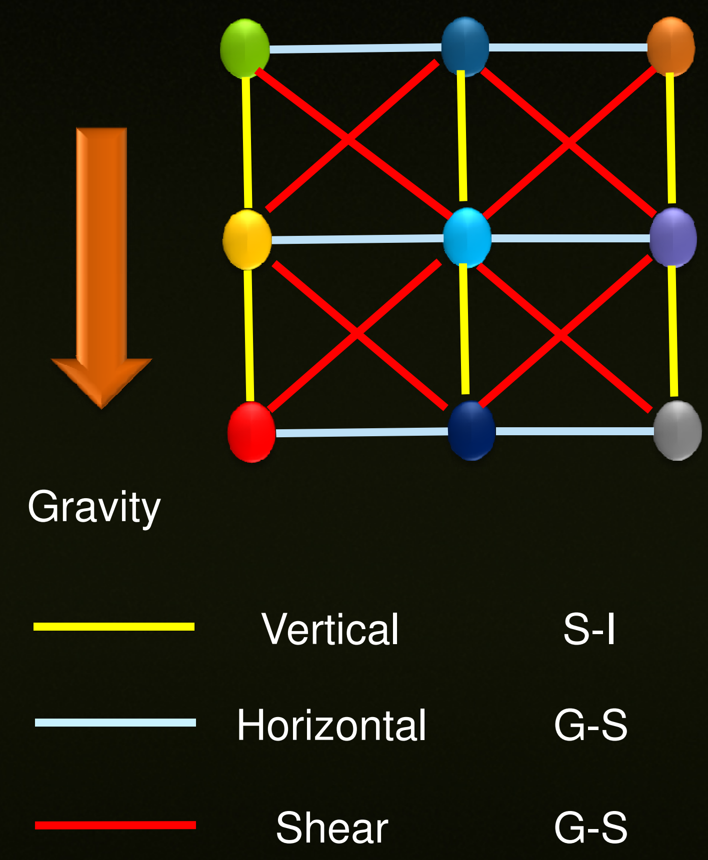


Green – G-S with 15 iters

Red – S-I with 1 iters

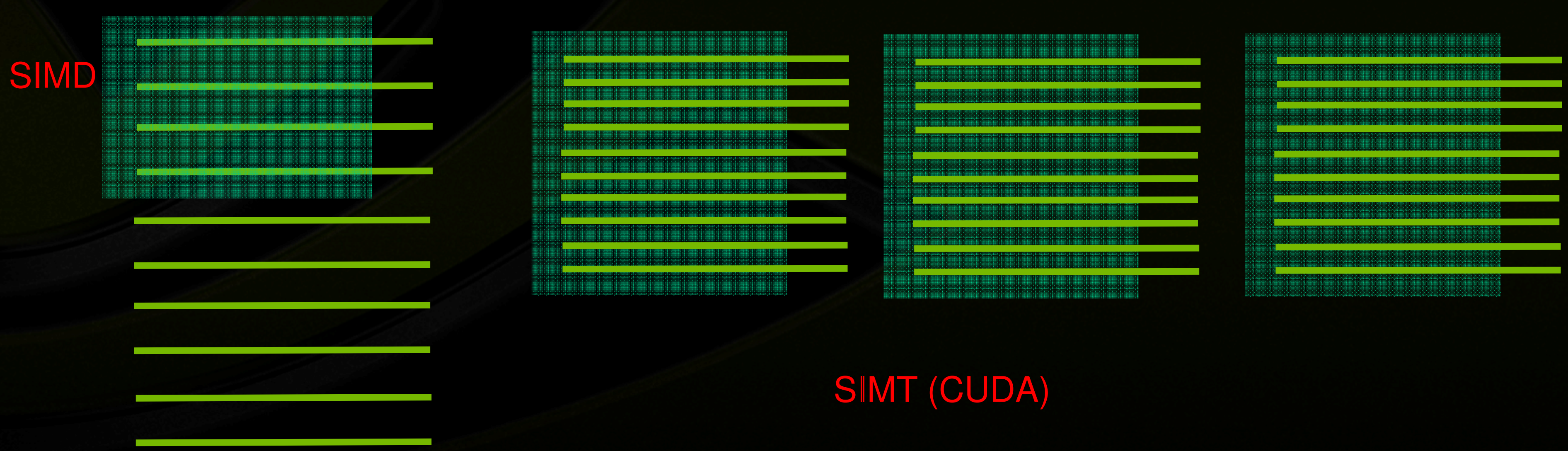
# Optimally Combining Solvers

- **Observation**
  - → stretching happens the most along gravity
- **Semi-Implicit for vertical fibers**
- **Gauss-Seidel for horizontal and shear (diagonal) fibers**



# Parallelism in Clothing Solver

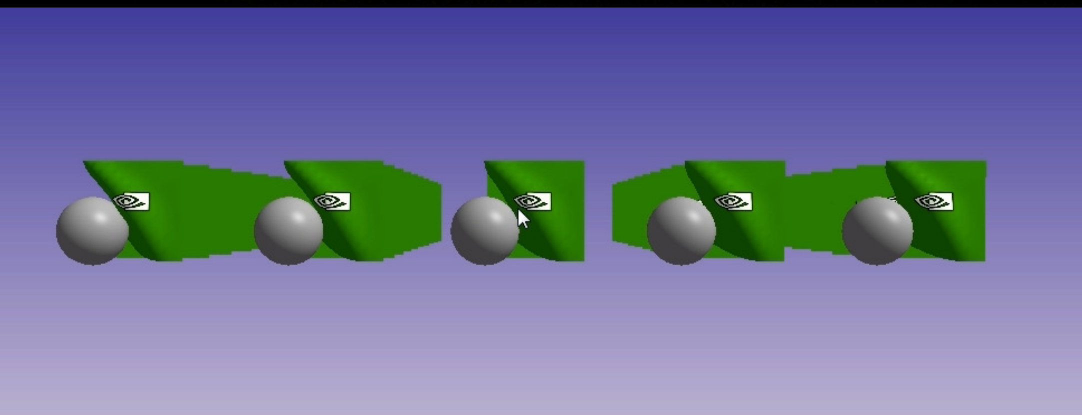
- Fiber level parallelism
  - Each computation unit work on each fiber
  - Good enough for SIMD (only 4 sets are sufficient for SSE2)
  - Typical data not enough to fill entire pipe in CUDA



# CUDA for Clothing Solver

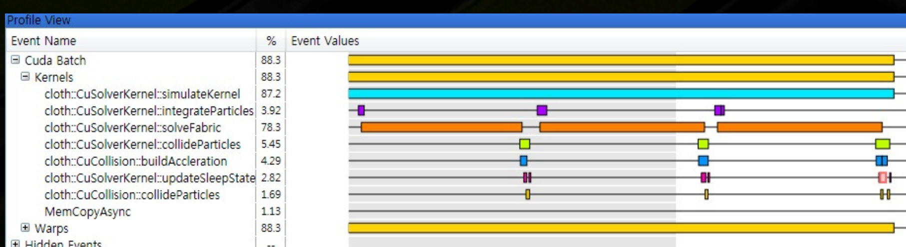
- **Maximum parallelism with multiple cloth**
  - One cloth per SM (up to 16 in Fermi)
  - Each thread works on one fiber
  - Shared memory per SM for optimized performance
- **Resolution limit**
  - ~1.2K particles on Fermi
  - *Fits the bill* for most game content

# Performance Statistics



|  | Time / frame | # of Cloth / ms |
|---|---|---|
| CPU (1Core) | 2.7 ms | 18.5 |
| CPU (4 Core) | 1.0 ms | 50.0 |
| GPU (14 SM) | 0.65 ms | 76.9 |

50 Cloths (each 256 particles)

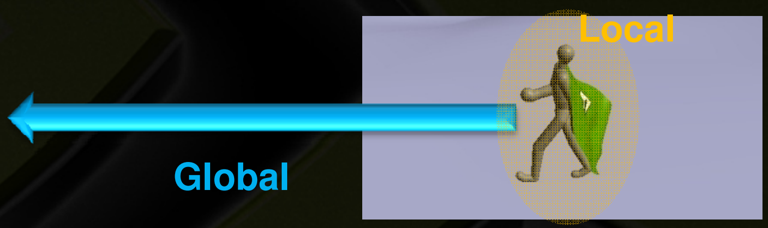



CPU : 2.8 GHz i7 GPU: GTX 470

# Character Clothing Pipeline

# Challenges in Game Character Motion

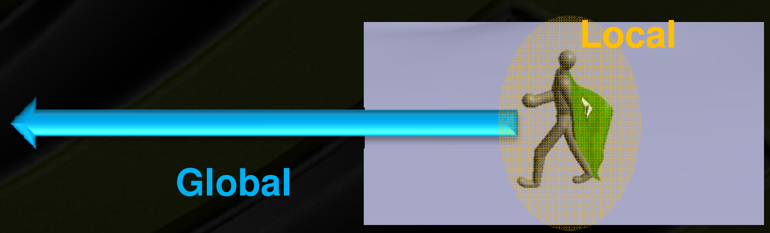
- Floating point precision issue from large offset
- Too much energy from unphysical game characters acceleration




**Idea** – use local space simulation

# Challenges in Game Character Motion

- Floating point precision issue from large offset
  - Local space simulation for cloth and colliders
- Too much energy from unphysical game characters acceleration
  - User control on inertia effects



# Inertia Control for Local Simulation

- Global acceleration is applied to local space cloth particles
- Users can control amount of inertia effect

Full inertia

No Itertia (Local only)

# Collision for Character Clothing

- Main collision shape: tapered capsules
  - Convex hull of two spheres (with potentially different radii)
  - Flexible to model character with few shapes
  - Simple enough for high collision performance

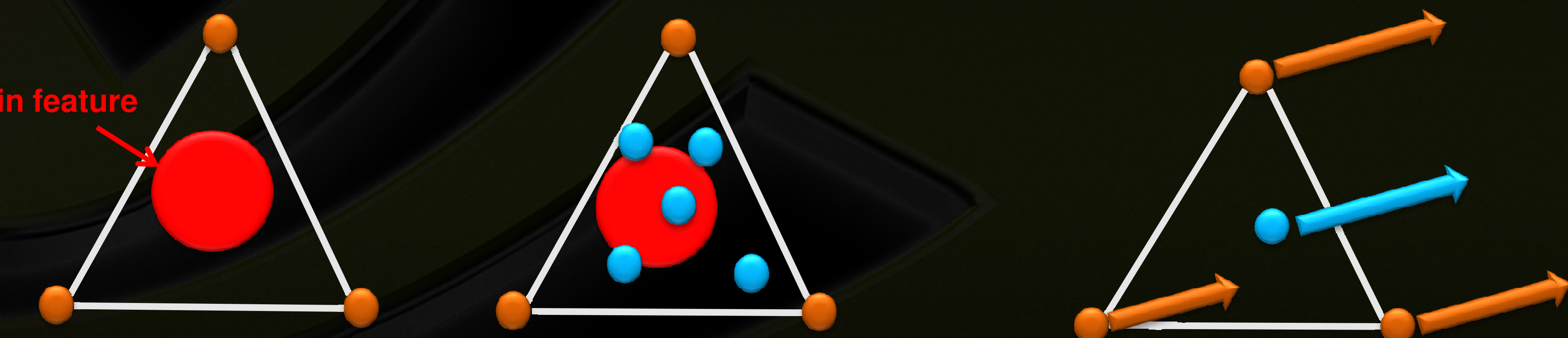
# Discrete vs Continuous Collision

- Discrete Collision
  - Avoid penetration at the frame boundary
  - Fastest, but less robust
- Continuous Collision (CCD)
  - Solves for trajectory of capsule and particle for frame interval
  - Robust against fast motion
  - Requires solution of 6-th order polynomial
    - Approximation with quadratic equation
    - Adds about 2x more computation

# Virtual particles

- Low resolution samples miss thin features
- Increasing simulation resolution too costly
- Triangle based collision too costly

**Idea:** use more collision points without changing simulation resolution

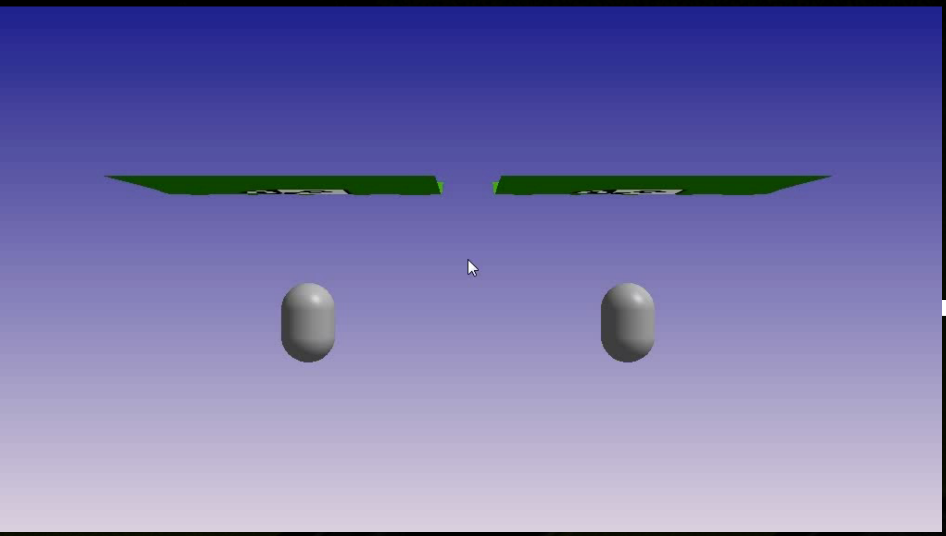


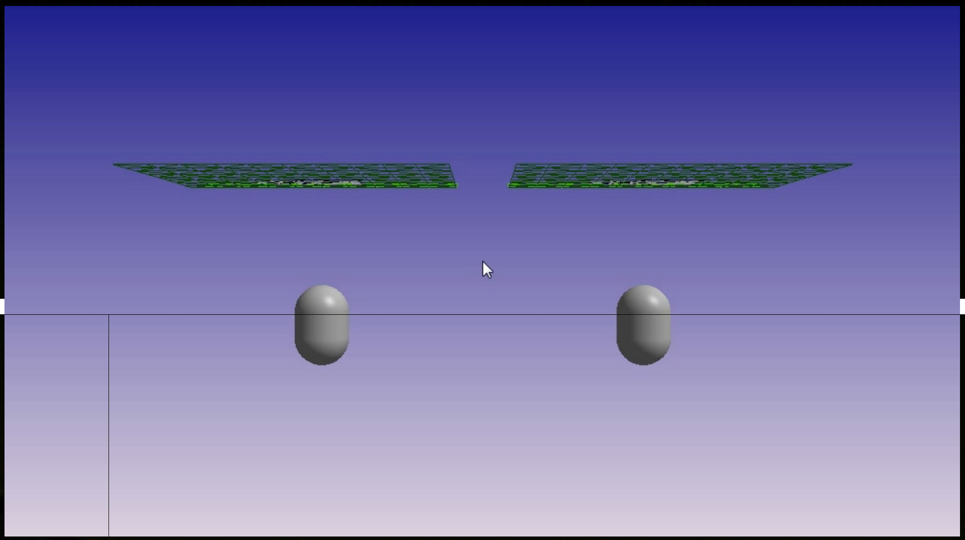

Simulation Particles Only

Simulation Particles + Virtual Particles

Each virtual particle re-distributes impulses to simulation particles

# Virtual particles





Without VP With 4 VP

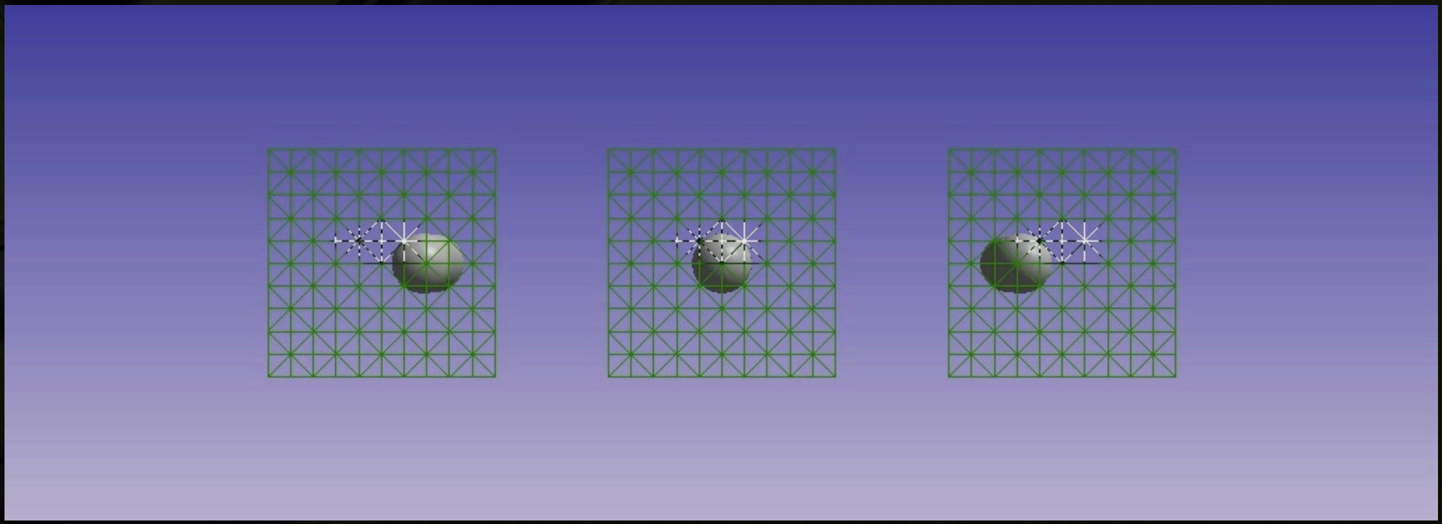
Without VP With 4 VP

# Virtual particles

- Can be placed anywhere on the mesh
- Mimics triangle collision with enough samples
  - Avoids stretching cloth when collision shape slips through
- Only increases collision handling cost, not solver cost

# Mass scaling

- Inextensibility can conflict with collision
  - "Do not stretch" vs "do not penetrate the collider"
- Make colliding particles temporarily heavier
- Re-distributes stretching away from collision area, so less chance of conflict



# Recap

- Solving Cloth Motion
- Character Clothing Pipeline

# Timeline (Schedule)

- PhysX-3.1 (released)
- PhysX-3.2 (beta, release in 2012 Q1)
- PhysX-3.3 and beyond (upcoming)

# How can you access PhysX?

http://developer.nvidia.com/physx-downloads

Either:

1. Register at NVIDIA Developer Zone.
2. Download the PhysX SDK (binary license is free!)

Or:

License a game engine that already uses PhysX.

- Epic's *Unreal Engine 3*
- Trinigy's *Vision 3D*
- Unity Technologies' *Unity 3*
- Emergent's *Gamebryo*

# THANK YOU


Special thanks to:

- Christian Sigg
- Miles Macklin
- And the whole PhysX team