
UMEÅ UNIVERSITET

Interacting particle systems

Neighbor find

Spring-and-damper cloth

Lab assignment

- Spring-and-damper cloth

Particle Simulation Loop

```
Loop {
  emitter/sink
  update attributes
  neighbour find (broad, narrow)
  inter-particle interaction - forces
  external forces
  boundary conditions/collisions
  time-integration
  visualize and render
}
```

Specific order can vary depending on algorithm and application.
Performance bottlenecks depend on model and problem size.
Relevant research area is to combine neighbor find and interaction (solve).
Visualization/interaction can run in its own thread/process.

Interaction forces and potentials

- Interaction force depends on what you want to model.
- For particle-particle interaction often a pair potential, V(r) such that the pair force is,

$$\mathbf{F}(r) = -\nabla V(r)$$

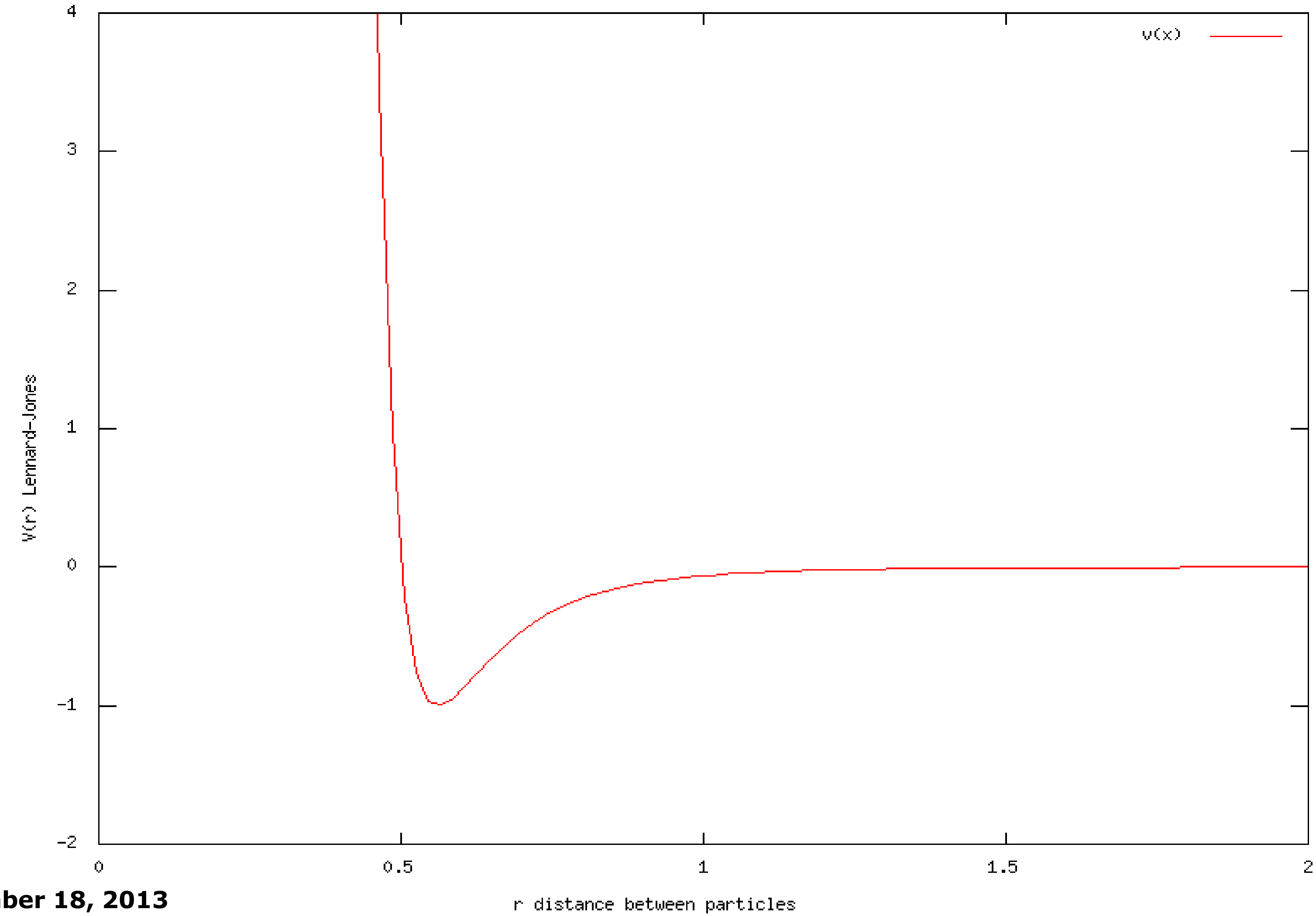
- If spherically symmetric/central force

$$\mathbf{F}(r) = \hat{\mathbf{r}} F(r) = -\hat{\mathbf{r}} \frac{d}{dr} V(r)$$

- In *molecular dynamics*, the Lennard-Jones potential for modeling inter-molecular forces is very common. It is near repulsive near the core, and attractive but short range at longer distances.

$$V^{LJ}(r) = 4\varepsilon \left[\left(\frac{\sigma}{r} \right)^{12} - \left(\frac{\sigma}{r} \right)^{6} \right]$$

Lennard Jones Potential



Interaction forces and potentials

Hard sphere potential, i.e. rigid body sphere. Can also be used to simulate an ideal gas.

$$V^{HS}(r) = \begin{cases} \infty & r \leq \sigma \\ 0 & r > \sigma \end{cases}$$

In modern molecular dynamics the pair-potentials are non-trivial, and may be more complicated than just pair-potentials too. In quantum mechanics, the potential is modeled based on the solution to the Schrödinger equation involving all/many other particles in the system, and requires extensive computations!

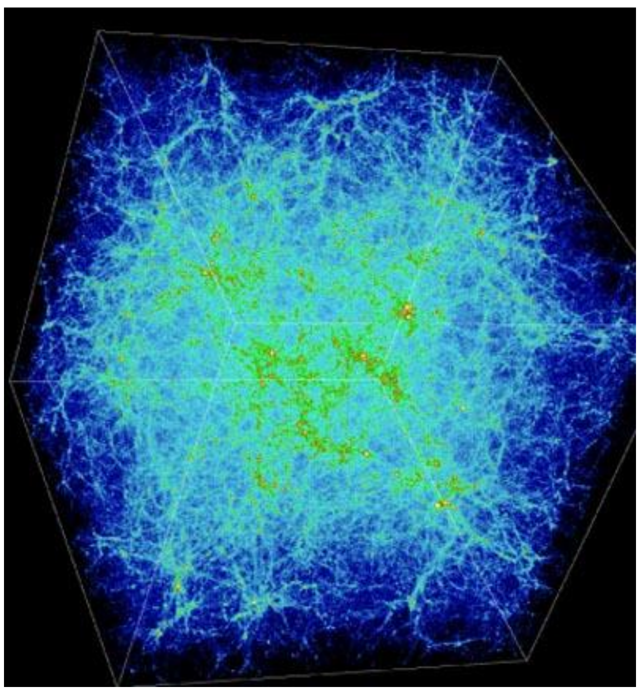
In Smoothed Particle Hydrodynamics (SPH), the interaction potential is weighted by an SPH-density which depends on more particles than just the pair. In this way we can simulate liquids and complex material properties. More about this in a coming lecture.

For e.g. Cloth, one can use a *spring-and-damper interaction*.

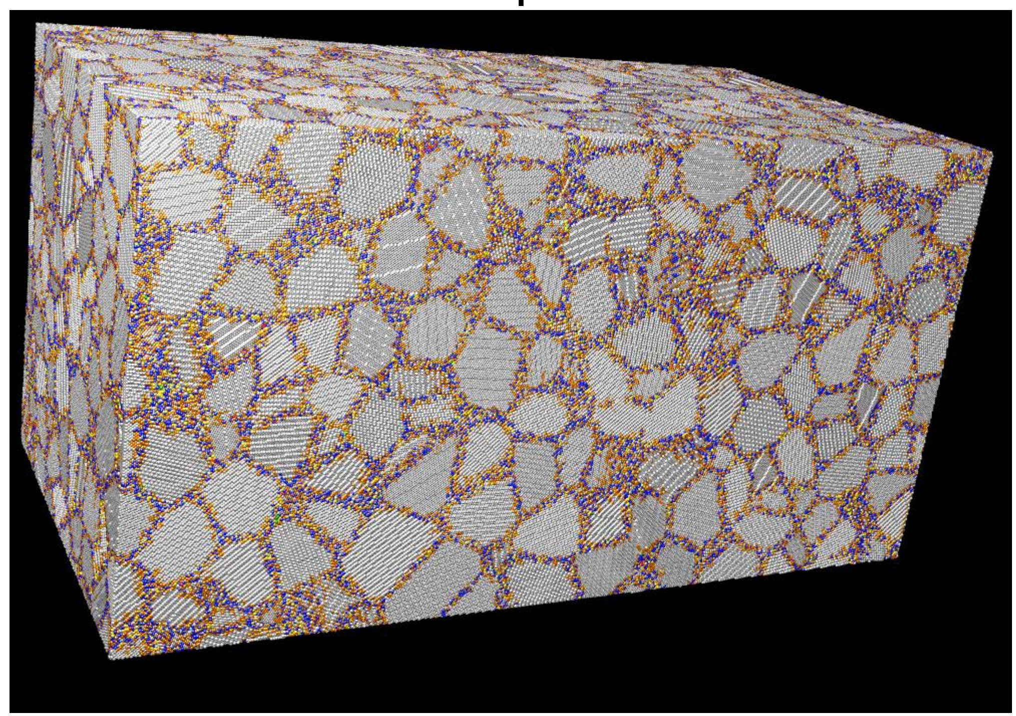
UMEÅ UNIVERSITET

Neighbour find

- Required speed-up if we have inter-particle interaction
- Exhaustive search requires *N(N-1)/2* tests so complexity is $O(N^2)$
- Typically we strive for *O(N)* or *O(NlogN)* complexity
- Overhead typically low. Break even for 100-500 particles.



From Mike Warren – A billion stars (HOT)

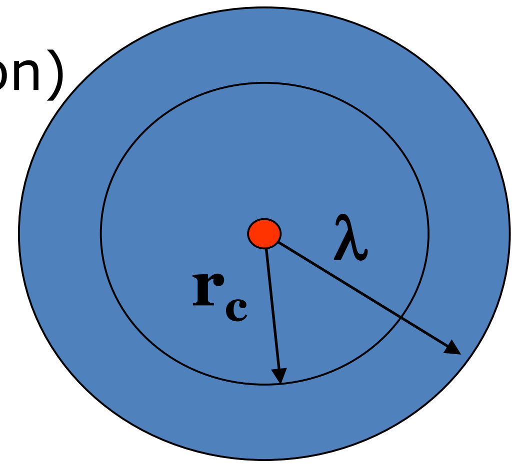


Kadau et.al. 2007, Shockwaves in Iron (SPaSM) http://www.dnatube.com/video/1228/tRNARibosome-Molecular-Dynamics-Simulation

http://www.gauss-centre.eu/gauss-centre/EN/Projects/MaterialsScienceChemistry/Weltrekord_molec_dynamics_sim.html

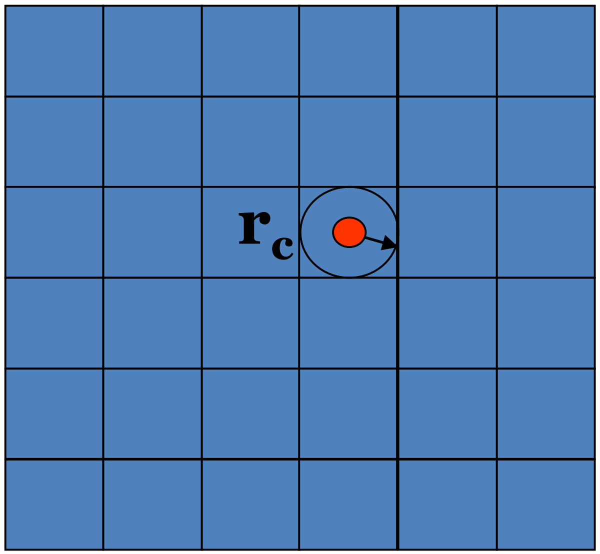
Methods for Neighbour find

- Brute force/exhaustive search $O(N^2)$
  - $N(N-1)/2$ tests. If $r < r_c$ add particle to neighbour list (r_c is the range of the interaction)



- Exhaustive + temporal coherence
  - ”Verlet neighbourhood table”
  - Perform exhaustive search only often enough to track particles from going from exterior $r > \lambda$ into $r < r_c$
  - This can be done incrementally or every Nth time step.
  - Choice of λ depends on dynamics, typically $1.1 – 1.2\ r_c$
  - Very low overhead, but doesn’t scale well at all for large systems. Not ideal in computer graphics that CPU load vary alot between time-steps.

Spatial subdivision – cell linked list



- Spatial subdivision – cell linked list. Quentrec and Brot -75; Hockney and Eastwood 1981
- For a system of size *LxLxL* use an e.g. *MxMxM* sized array of cells where each cell side is $l = L/M < r_c$ such that a particle located in a cell can only interact with particles in its own or in neighbouring cells.
- Cost is reduced to *O(27N)* in 3D (and *O(13N)* if we avoid double counting and use Newton's third law).
- For each particle determine which cell it belongs to, addressed by a cell index.
- For each particle, look for other particles in its cell and for other particles in neighbouring cells, and construct a neighbour list from this.

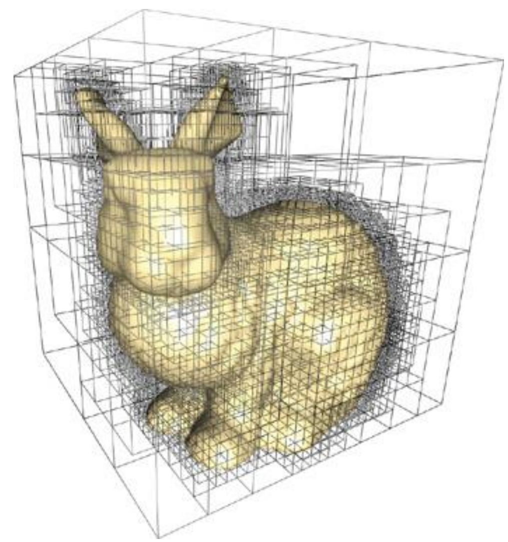
Spatial subdivision – cell linked list construction

```
int M, ncell=M*M*M, npart                    //MxMxM number of cells, npart number of particles
int headofchain[ncell-1]                     //contains largest particle number in cell
int linkedlist[N-1]                          //Number of the next particle in the cell
double cellside, x[npart-1], y[npart-1], z[npart-1]        //Cellside is L/ncell^3
for i = [0,ncell-1]
          headofchain[i] = 0
          for j = [0, npart-1]
                    cellpos = int(x[j]/cellside)
                              + int(y[j]/cellside)*M
                              + int(z[j]/cellside)*M*M
                    linkedlist[i] = headofchain[cellpos]
                    headofchain[cellpos] = j
          endfor
endfor
```

Code is from the classic book: Allen & Tildesley, Computer Simulation of Liquids.
Bare with them - original source in Fortran ☺: See e.g.
http://server.ccl.net/cca/software/SOURCES/FORTRAN/allen-tildesley-book/f.20.shtml

Hierarchical spatial subdivision

- E.g. octree
- Fairly expensive overhead, but gives better handling of inhomogeneous systems (i.e. varying density) - in particular memory-vise.
- Resolution is adaptive.
- Was also covered in the Anders Backmans course
- Search the web for documentation and examples.



From Nvidia, GPU Gems 2

Spatial Hashing and Infinite Grids

- Explicit 3D grids eat your computers memory!

- Map each cell into a hash table of a fixed set of n buckets
- The grid is just conceptual and doesn’t eat up your memory! The hashing is a sort of ”grid compression” where the grid is compressed to a size comparable to the number of objects stored in it.
- It is easy to construct a bad hash function ☺
  Knuth is a good general reference, but better hashing has emerged in recent years.
- For particles in a 3D-space – use a prime number hash.

Spatial Hashing and Infinite Grids

```
//Pseudo Code
//cell position
Struct cell {
   cell (int32 px, int32, py, int32 pz) {x=px, y=py, z=pz}
   int32 x,y,z;
};
#define NUM_BUCKETS 1024

//compute hash bucket index in [0,NUM_BUCKETS-1]
Int32 ComputeHashBucketIndex (Cell cellpos) {
   const int32 h1 = 0x8da6b343;       //Large primes
   const int32 h2 = 0xd8163841;
   const int32 h3 = 0xcb1ab31f;
   int32 n = h1*cellpos.x + h2*cellpos.y + h3*cellpos.z;
   n = n%NUM_BUCKETS;
   if (n<0) n+=NUM_BUCKETS;
   return n
}
```

//Example from Christer Ericson's book "Real-time collision detection".

Spatial Hashing and Infinite Grids

- Hash collisions = mapping of two keys to the same bucket (and they do happen!)
- Problem:
  - There is a risk of mapping several 3D points to the same bucket/hash index.
  - Choose index size to be ”significantly larger than number of object primitives”
  - It is recommended (but not often used?!) to choose size to also be a prime number.
- Not entirely well documented how to best construct hash indexes!
- Obviously we can map e.g. a box onto the hash grid, often an AABB. Common method for broad phase collision detection.
- Be smart about memory – don’t reallocate the whole thing every timestep, amortize cost and allocate in chunks that are reused (only makes a big difference for large particle systems).

Collisions and boundary conditions

Collisions with the surroundings, e.g.

particle-plane

particle-box

Extract intersection point and intersection normal.
Create intersection lists, containing e.g.:

```
entity a, entity b
intersection point
intersection distance    //Can be important to cache to avoid multiple sqrt(r2) computations.
intersection normal
```

a and b are general entities, i.e. a particle with a plane, a particle with a box, etc.

External forces

E.g. Gravity, velocity field (wind), diffusion, dissipation.

Dissipation can be modeled using air friction, fluid friction with forces linearly or quadratically proportional to the velocity.

Note: Large damping is in general not at all numerically stable, so damping for stability can often be a bad strategy!

Velocity reduction is usually stabilizing, but doesn't have much with physics to do, i.e.

$$v = 0.999\, v$$

If you really need it – then your model, algorithm or program is typically broken.... Remember: If the damping is "only" 0.999 per timestep, at 60 Hz the velocity is reduced by 97% after one minute of simulation time!

Time integration

Euler bad as usual

Use Leap-Frog/Verlet (sometimes called symplectic Euler) and variational types of methods

For cloth we might want to use an implicit solver. This is more complicated and requires an equation solver method. It gives improved stability, but exaggerated non-physical damping (i.e. cloth moving in molasses!)

Visualization and rendering

Not covered in depth in this course…

- Points
- Spheres (or some general ”glyphs”)
- Alpha textured quads (perhaps the most common)
- Point sprites in openGL, i.e. Hardware billboards
- Point splatting and splat shaders
- Mesh generation, e.g. Marching cubes methods
- Metaballs, blobs.
- Very common to use environment mapping
- Procedural textures/surfaces desirable but slow and largely unexplored
- Shaders, shaders, shaders…

- Note that some methods use orientation as they represent a surface. Thus, we need a way to compute surface normals. In the SPH method we typically get this from looking at the gradient of the color field (se lab notes and SPH lecture).

- There are also numerous methods for refining the particle surface and for smoothing it, and also for adding more dynamics and visual effects to the surface.

Hardware and parallelism

Particle simulations and particle effects important drive for graphics hardware and GPGPU.

PhysX from Nvidia, important for Nvidia GPGPU.

Havok from Havok/Intel important for Larrabee (somewhat cancelled and delayed...) and Sandy Brigde.

Bullet (in association with AMD) important for AMD Fusion/APU

OpenCL very important for heterogeneous parallelism both for graphics and physics – and many other things.
http://www.khronos.org/opencl/

MD on Bluegene/L

- 320 x 10^9 particles
- Short-range interaction
- SPaSM software – Scalable Parallel Short-range Molecular Dynamics

MOLECULAR DYNAMICS COMES OF AGE: 320 BILLION ATOM SIMULATION ON BlueGene/L

KAI KADAU
Theoretical Division, Los Alamos National Laboratory
MS G756, Los Alamos, New Mexico 87545, USA
kkadau@lanl.gov

TIMOTHY C. GERMANN
Applied Physics Division, Los Alamos National Laboratory
MS F663, Los Alamos, New Mexico 87545, USA
tcg@lanl.gov

PETER S. LOMDAHL
Theoretical Division, Los Alamos National Laboratory
MS B214, Los Alamos, New Mexico 87545, USA
pxl@lanl.gov

Received 1 November 2006
Revised 1 December 2006

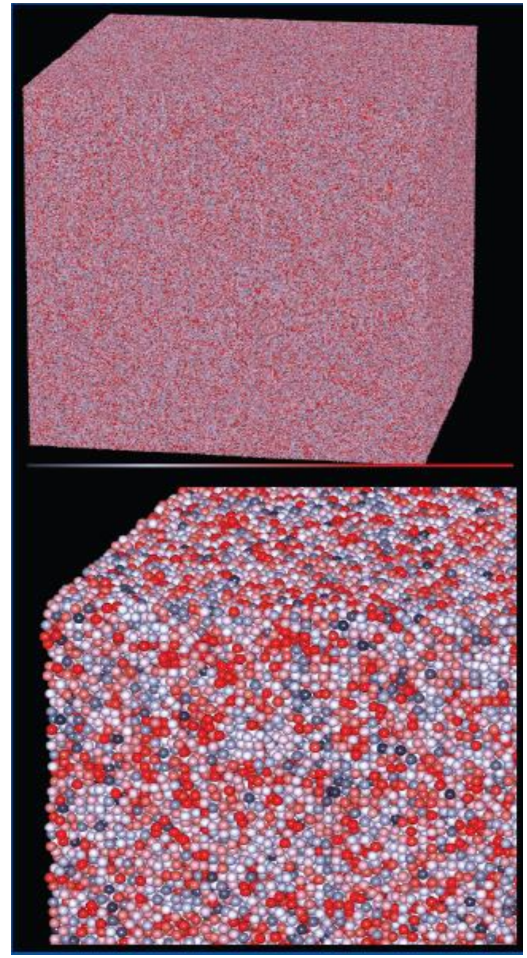
- Memory efficient neighbour lists + domain parallelism
- 131 072 processors
- Lennard-Jones potential
- Atomistic resolution on mesoscopic/macroscopic micron scales
- One (compressed) configuration = 9600 Gb
- 22 226 Gb RAM (32 768 Gb)

Int J Mod. Phys. C Vol 17, No 12 (2006)

MD on Bluegene/L

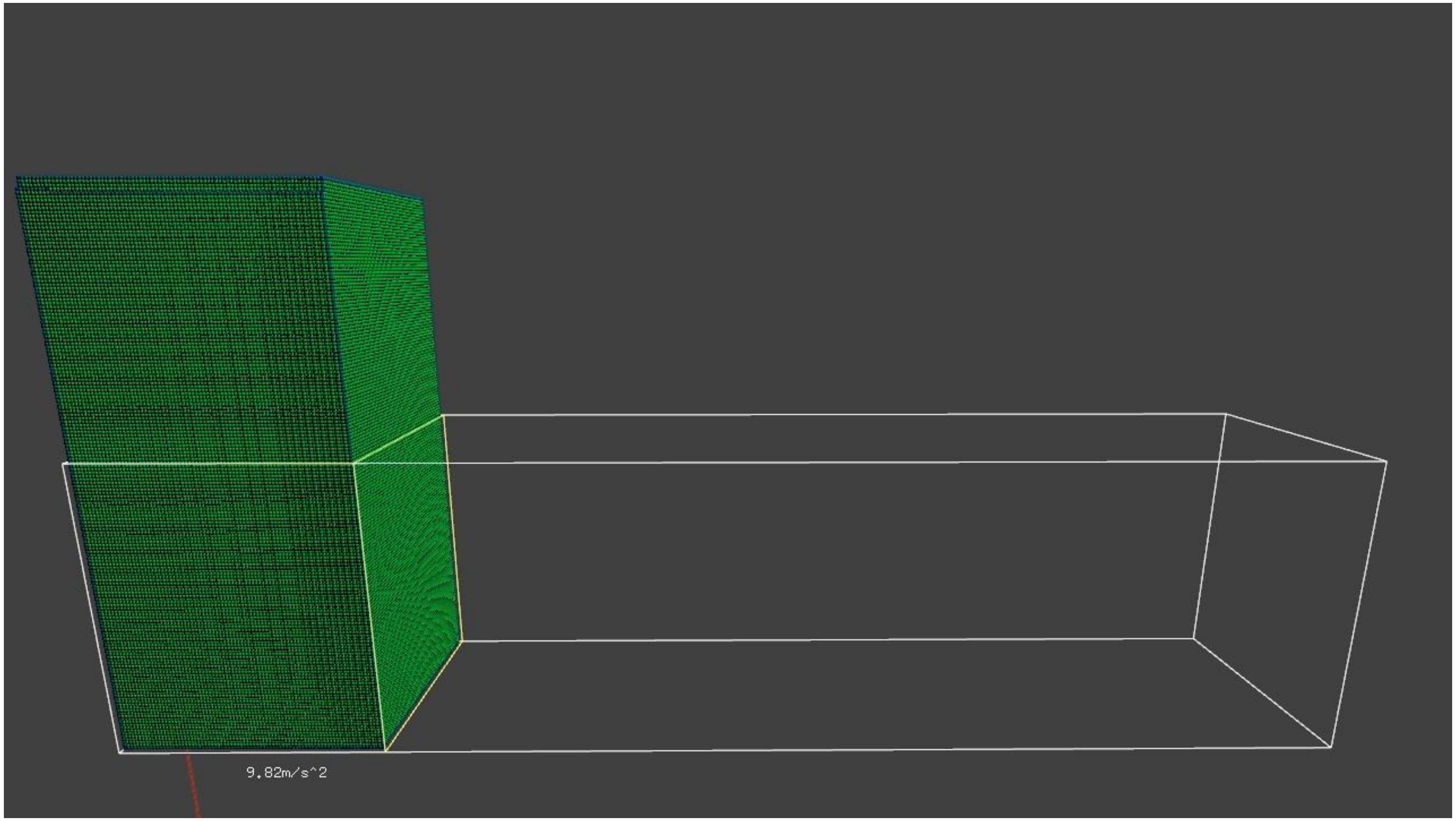
Movies

- 3D Rayleigh-Taylor instability (7.1G)
- Shockwave in Iron (30M)



Kadau et.al. 2005-2007

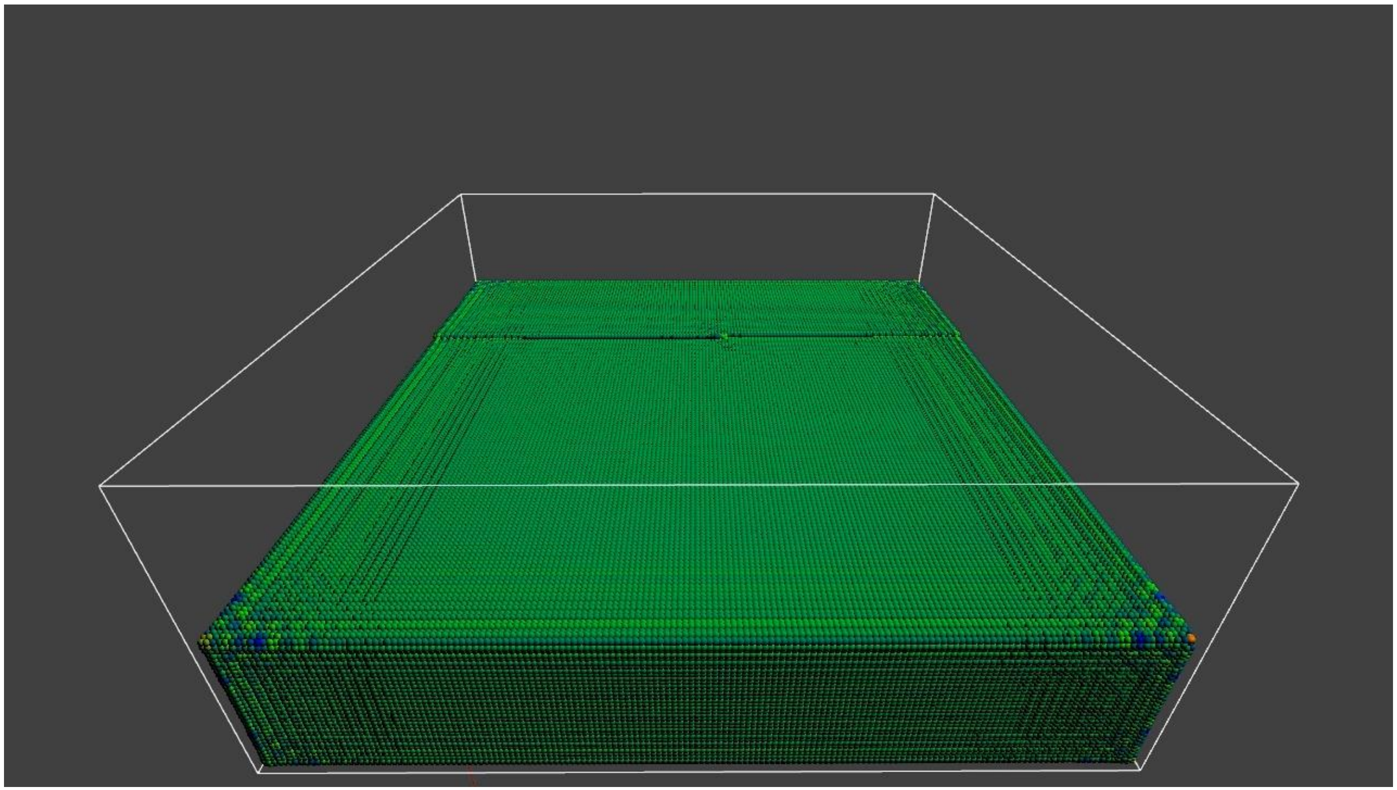
Constraint Fluids on GPU



Martin Nilsson, MSc thesis, 2009

See also: **http://youtu.be/0e3p5iu3zj8**

Constraint Fluids on GPU



Martin Nilsson, MSc thesis, 2009

Constraint Fluids on GPU

With metaballs and ray tracing.
See also: http://youtu.be/kzFam0UKFoA

Cloth

- What is cloth?
- Models and simulation methods
- Implementation guidelines

Coming lecture (Claude)

- Integration methods
- Constraint based methods
- Lab (Wire, Cloth)

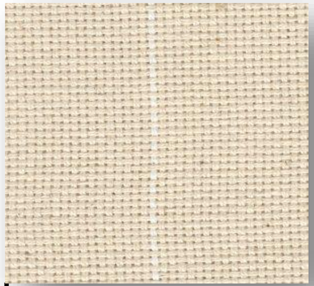
What is cloth?

From the Merriam Webster:

Cloth: a pliable material made usually by weaving, felting, or knitting natural or synthetic fibres and filaments

Pliable: supple enough to bend freely or repeatedly without breaking

Supple : capable of being bent or folded without creases, cracks, or breaks



Crease: a line, mark, or ridge made by or as if by folding a pliable substance

In short, cloth is something made of fibres which can bend easily and without causing permanent distortion. From the definition of weaving, felting, and knitting, cloth is typically very thin in one dimension so it's essentially a two dimensional object.

Easy to bend means that we need a fine grained discretization.

Cloth and 3D graphics

- Cloth is ubiquitous
  - Curtains, lamps shades, table clotl
  - Clothing
  - Flags
  - Bags
- Cloth moves
  - Curtains of flags in the wind
  - Garments
- Cloth improves appearances
  - Drapes, curtains, table cloth
  - Dresses, shirts, scarfs

A good 3D environment should contain cloth to improve realism.

Thus we need both simulation methods and visualization methods for cloth.

Cloth Sim by Syflex (Despereaux)

- http://www.youtube.com/watch?v=_Doo66oSuC0
- http://www.youtube.com/watch?v=NG5C_a6rxrY

Spring-and-damper cloth

- The simplest model of cloth is constructed using springs and dampers.

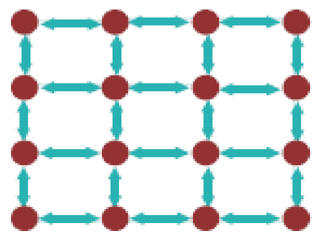
- We will take a closer look at other models later.

Spring-and-damper cloth

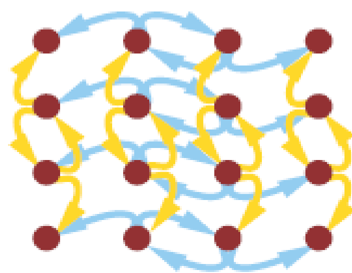
Mass of the fabric to be concentrated at a number of identical grid points.

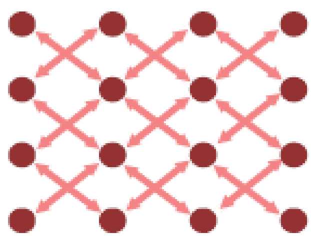
Interactions between these points should reproduce cloth properties:

Stretch forces between nearest neighbors



Bend forces up to next-next nearest neighbors





Shear forces couple next nearest neightbors

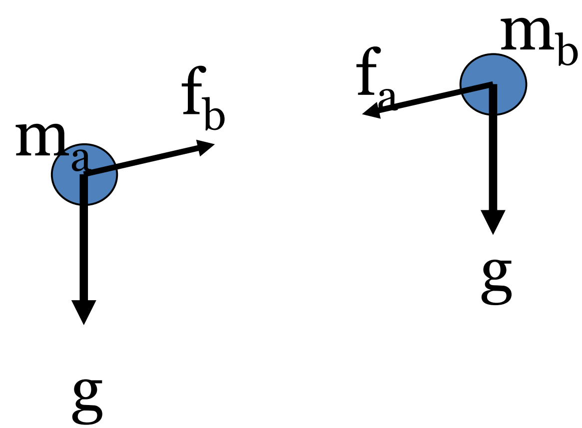
Stretch resistance mostly along warp and weft directions

Bend mostly along warp and weft

Shear prevents distortion of the cloth

Hooke's law for damped spring

$$\boldsymbol{f}_a = -\boldsymbol{f}_b = -\left[k_s(|\boldsymbol{r}_{ab}| - R_0) + k_d \frac{\boldsymbol{v}_{ab} \cdot \boldsymbol{r}_{ab}}{|\boldsymbol{r}_{ab}|}\right] \frac{\boldsymbol{r}_{ab}}{|\boldsymbol{r}_{ab}|}$$



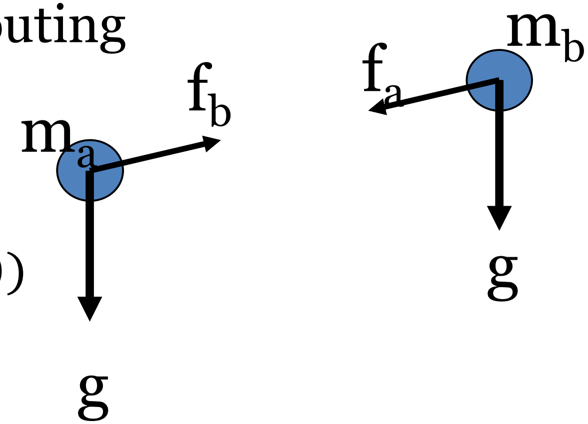
Set up springs for stretching, bending (often weak) and shearing.
Lots of parameters to deal with…

R_o = rest length, k_d = damping coefficient, k_s = spring constant

Stretch resistance in one direction typically stiff – hard to handle numerically, and requires small timesteps since the integrator misses its target, and energy is created by the penalizing potential.

Implicit solver for cloth

Implicit solver uses *future* positions and velocities for computing the current force.



$$r(t+h) = r(t) + h\dot{r}(t+h)$$
$$\dot{r}(t+h) = \dot{r}(t) + M^{-1} f(r(t+h), \dot{r}(t+h))$$

Assume the force is linear

$$f(r(t+h), \dot{r}(t+h)) = f(r(t), \dot{r}(t)) + h\, \nabla f(r(t), \dot{r}(t)) + \cdots$$

From this (and more, see 8.7 in Erleben) we can solve for velocities and positions that are consistent with the stepping above.
This method is stable but highly dissipative for large steps and/or stiff forces.

Implicit vs (semi) explicit

Implicit method for cloth solves a matrix problem for velocities consistent with the integration formula.
Thus, it propagates information through the entire system in one timestep!

If using an explicit method – how many timesteps does it take to propagate information from one side to the other in an NxN simulation of cloth?

What sets the physical propagation velocity of information?

What happens if information propagates faster than the maximum rate set by the timestep? How can this problem be handled?

References and reading

- Read 5.2 – 5.7 and 22.4 in Erleben to review properties of the damped harmonic oscillator (spring-damper system)!
- Erleben chapter 8 - particles.
- Spring-and-damper cloth 8.4.1
- Improved methods covered later: 8.7.1-2
- Collision detection with hashing is discussed in 16.2
- Chapter 23 (the section about implicit integration)

- Lab assignment notes – spring-damper-cloth