Kenneth Bodin
Umeå University
2005-04-22

Lab projects 2.1, 2.2, 2.3
Simulation of interacting particles

The basic task of lab 2.1 is to set up the basic modular structure for a particle system, and to use that in the application projects on smoke (bonus on 2.1), cloth (2.2) and sph (2.3). Various extra features and improvements can give you bonus credits.

Below, a general description of a particle system simulator is given. It is recommended that you look at the specific tasks (smoke, cloth, sph) when designing and implementing this in the code.

Lab 2.1 - General particle system

Develop code to simulate a particle system. Design the code (or library) so that you can have several particle systems with different properties in the same scene. For further information also see the lecture notes. The data structure describing a particle should typically hold the following data:

```
index or address
mass
inverse mass

position (possibly a history of these, depending on integrator)
velocity (-“-)

accumulated force

...
(other attributes)
```

Keep in mind that the design of the particle data structure can be highly important for cache optimization when particles are interacting with each other. However, we will not worry about such optimization here, and instead keep things simple.

The Simulation loop

The simulation loop should roughly have following structure:

```
Setup initial conditions

while (running)

      Create particles at emitter
      (Remove particles at sinks or when they are too old)


      Do inter-particle collision detection and construct a
      neighbour list.

      Loop over neighbour lists and compute interaction
      forces. Accumulate the forces. Use Newton's third law.

      Accumulate external forces from e.g. gravity.

      Accumulate dissipative forces, e.g. drag and
      viscous drag.

      Handle external boundary conditions by reflecting the
      impulses and updating the velocities (i.e.
      intersection find for contact set plus collision
      impulses.


      Take a timestep and integrate using e.g. Verlet/Leap Frog
      Reset the accumulated force and administrate
      data for velocity and position properly for next
      timestep.

      Render the particles and use whatever
      attributes necessary for rendering (e.g.
      mass, age, colour, normal, …).

End
```

It is recommended that you design your code such that the above components of the simulation loop can be used quite freely and easily interchanged, i.e. you shouldn't hard-code what is done during a time step. You could e.g. push the components to a queue and pop them out one by one in the simulation loop. If designed like this, it is also easy to define the simulation loop in e.g. an xml file (which is not necessary for the lab of course).

Keep it simple and don't exaggerate the general purpose design of it. Pay respect to the deadlines for lab projects! ☺

Simulation loop explained

Create particles at emitter

For this lab we only require that particles are created at circular sources (while in general one would like to be able to load general geometric objects, i.e. from a 3D modelling program). Particles should be created at a constant rate (per frame or per second) on random positions on this circle, with an initial velocity that varies randomly from an average initial velocity, and with a direction that varies randomly from the normal of the circle.
Particles may also have other attributes that vary, e.g. variable lifetime, mass and visual attributes.

Age and lifetime

If a particle has a finite lifetime, remove the particle from the simulation if its age exceeds its lifetime. What does this mean to physical conservation laws for mass, momentum and energy?
(We will not consider particle sinks in this lab)

Inter particle collision detection and neighbour lists

There is a handful of different algorithms for doing this, and their efficiency depend on e.g. homogeneity of the particle system (e.g. variations in density and range of interaction potential), overall range of interaction potential, memory efficiency (e.g. cache reuse and size), size of particle system. For basic credits you can implement a regular real grid based method. For a bonus credit you can implement spatial hashing or an octree. In any case you will get a list of interaction pairs formulated as:

```
Interation_pairs[1..N] is a list that returns addresses/indices
to the two particles that define the pair:

(index of first particle, index of second particle)
```

Of course, for non-interacting particles, this task can be left out entirely! For e.g. cloth or some 3D object with permanent connectivity the list is computed at startup and used throughout the simulation.
Note: Make sure you don't double count, e.g. you only need [i, j] and not [j,i] since the [j,i] interaction force can be computed using Newton's third law (i.e. $F_{ij} = - F_{ji}$)

Particle interaction forces

Loop through the list of interaction pairs. Compute the inter-particle force and accumulate the force to the total force on each particle. Don't forget to use Newton's third law (action/reaction), so that both particles in a pair gets their forces updated (and don't double count!)
Obviously, this task can be left out entirely if particles aren't interacting.

External forces

The most common external force is gravity. It acts uniformly on the particles adding a force *(0,0, -mg)* to the accumulated force on each particle.

Dissipative forces

Dissipative forces dissipate energy from the particles and is typically proportional to their velocities, or relative velocities (as in SPH for fluids). Implement viscous friction where the damping force is proportional to the particle velocity, $\boldsymbol{F} = -k\,u\,\hat{\boldsymbol{u}}$ (k is a constant), and air friction where damping is proportional to the velocity squared, $\boldsymbol{F} = -c\,u^2\,\hat{\boldsymbol{u}}$ (c is a constant). Accumulate this to the total force on each particle.

Collision detection and collision response with the environment

Implementation of an efficient general purpose collision detection library is a formidable task, even if it is just for a simple particle system. Here we will limit ourselves to handle a few simple geometries and not worry too much about efficiency or optimization. In order to do the basic SPH modelling for "The Scene", you must be able to handle box interiors (i.e. a box container). For bonus credits you can also implement box exteriors and planes.
Use the Newton impulse collision law to handle collisions with the environment. This is of course an over simplified method for e.g. fluids, but it gives plausible results in many cases. Use can also use Newton-Coulomb friction in impulses.

Time integration

Numerically integrate the system for new velocities and positions. As explained in the lectures, Verlet integration is often a good choice. Update the global time parameter. Reset the accumulated forces. Manage data for current/past position and velocity, properly in order to take another time step later.

Visualization and rendering

Use e.g. textured quads or sprites (OpenGL, hardware supported) for visualization and rendering. Experiment with colors, textures, orientation/normals and alpha channel to achieve acceptable visual models.

Credit Points in lab 2.1

Basic credit (mandatory):

Implementation and documentation of a basic particle system, 1 credit point.

Bonus credit (choice):

Use your particle system to implement a visual effect (or several ☺) in "The Scene", e.g. dust, smoke or some sort of explosion. Implementation and documentation gives 1 bonus credit.

Kenneth Bodin
Umeå University
2005-04-22

Lab 2.2 - Simulation of Cloth

Model and algorithm description

Follow Claude's lecture notes and implement a simulation of cloth using a spring-and-damper model with nearest neighbour connectivity on a 2D grid.

This can all be implemented using the particle system of lab 2.1:

- Set up the initial conditions with ~100 particles on a rectangular 2D grid
- Pre-compute the connectivity list (interaction pairs)
- Accumulate spring interaction forces by looping over interaction pairs. The vector spanning from particle i to particle j is just $\boldsymbol{r}_{ij} = \boldsymbol{r}_i - \boldsymbol{r}_j$ and the spring force is along this vector and proportional to it's magnitude minus the resting distance. Don't forget to use Newton's third law.
- Also accumulate the dissipative forces on each particle, i.e. imagine a damper *along the direction of the spring*, and compute the friction force that is proportional to the velocity along this direction. This velocity is simply the relative velocity of the particles. Don't forget to use Newton's third law.
- Accumulate forces on attachment points, where the piece of cloth is attached to some other body (i.e. the "bone" in "The Scene"). Use a spring-damper coupling here too, and simply fixate some of the cloth particles on the other object. Don't forget to use Newton's third law to accumulate to the torque and force on the body that the particles are attached to.
- Accumulate air friction, proportional to the velocity squared of each particle.
- Time integrate using an explicit integrator, e.g. Verlet/Leap Frog
- Visualize and render, by placing a texture on the resulting 2D mesh.

Basic credits (mandatory):

The task described above gives 2 credit points.
Document the method, stability and performance issues, and present all parameters used.

Bonus credits (choice):

Implement additional spring connections in the system to get additional material behaviour. 1 bonus credit.
Implement additional collision handling with the environment, so that you e.g. can lay out a piece of cloth so that it can relax on a box, or interact with boxes when your hero ("bone") swings through the box piles. 1 bonus credit.
Implement self collisions hindering particles from going through eachother (your piece of cloth will have quite weirdo behaviour without this feature..). 1 bonus credit.
Max 3 bonus credits will be awarded.

Lab 2.3 – SPH simulation

In this lab you will model a container with a fluid inside – a bubbling witch pot. This container will also be integrated in "The Scene". It is non-trivial to get the simulation fully stable, in particular with the simplified boundary conditions we use, and the coupling to the moving container – but some extra turbulence is quite consistent with a witch brew as we all know ☺

Model and algorithm description

Use the basic particle system described in lab 2.1 with the following components:

Emitter

Just create all particles at start up time and place them uniformly inside a box. Roughly ~ 500-2000 particles should work depending on computer, optimization and what other computations you do during the time-step. Around 10000 particles can be simulated in real-time on a Pentium-4 at 3.5 GHz if optimization is done well.

Age and lifetime

Just leave this out and let particles live for the entire simulation.

Finding neighbours

Without this, your simulation will be terribly slow beyond 100 particles or so. For basic credit points use a regular lattice inside the box and a cell index/linked list method as described in the lecture notes, to find neighbours. The search radius and size of each cell is determined by the range of the interaction, i.e. the smoothing distance of the SPH kernel function, *h*.

SPH Interaction forces

Loop through the interaction list, i.e. pairs that are close enough to be inside the smoothing volume, and compute the SPH density parameter and store this value for each particle.

Thereafter use the equation of state to compute the pressure value at each particle.

Finally, loop again over the interaction list and compute and accumulate the SPH force terms, i.e. the pressure force and the viscosity force. These are computed by multiplying the accelerations given in the lecture notes by the particle mass. There was an erroneous density term in earlier versions of the lecture notes so the m_i/ρ_i that appeared in the viscosity and pressure terms, should just be m_i. See the updated lecture notes!

External forces

Accumulate the gravitational force on each particle. In principle you can also apply air friction or viscous friction on each particle, but it doesn't make a lot of sense since viscosity should be internal (you have already computed it in the interaction loop, in the material coordinate system of the fluid), and particles inside the fluid, should not experience e.g. air resistance. However, damping can sometimes be used to stabilize a real-time simulation and compensate for e.g. large time steps.

Collisions with the environment and boundaries

Your particles are placed inside a box, and you need to run a particle-box interior test on each particle. If the particle is outside the box, you project it back by reflecting it's velocity in the normal direction. Make sure you don't reflect separating contacts…
You are free to experiment with relaxation of resting contacts and with Coloumb friction, but it isn't necessary for the lab.
Accumulate impulses also on the container, i.e. every particle impulse should update the linear and angular momentum of the container.

Visualize and render

Use a alpha textured quad or a point sprite to render your particles. See e.g. the osgpointsprite example:
http://openscenegraph.org/viewcvs/*checkout*/examples/osgpointsprite/osgpointsprite.cpp

Parameters

Mass $m = 0.0001$
Smoothing length, h = *0.0055*
Rest density $\rho_0 = 2000$
Time step = *0.003*
Number of particles = *2000* (or 1000, or 10000...)
Surface tension = *0.1*
Viscosity = *0.02*
Pressure constant, $c^2 = 0.5$ (i.e. $P = c^2 (\rho - \rho_0)$)
Side length of container box = *0.05* (i.e. *5x5x5* cm^3 filled with ~ *0.2 l* of fluid)

We recommend using the poly6 kernel for simplicity. It is far from optimal for stability, but it works..
Double check the derivatives for typos (and let us know..)!
The definitions of the gradient and the Laplacian ("nabla2") in spherical coordinates are given here:
http://mathworld.wolfram.com/SphericalCoordinates.html

Kernel function poly6:

W(r) = 315/(64*pi*pow(h,9)) * pow((h*h-r*r),3)
grad(W(r)) = 945/(32*pi*pow(h,9)) * pow((h*h-r*r),2) * r_vector
Nabla2(W(r)) = 945/(32*pi*pow(h,9)) * (h*h-r*r)(7*r*r - 3*h*h)

Basic credit points (mandatory):

Basic implementation will give 2 bonus credits.
If you run into serious time problems, focus on getting the SPH simulation running in a fixed stand-alone container that isn't interacting with the rest of the scene.

Bonus credits (choice):

Implement a hashed grid or octree instead of a plain cell linked list. 1 credit
Implement colour field and surface tension. 1 credit
Implement rendering that uses the normals computed from the colour field to orient a textured quad (or a sprite). 1 credit
Implement more general boundary conditions, i.e. collisions with other bodies and a ground plane so that you can pour out the fluid. 1 credit.
More suggestions might be added.
Max 3 bonus credits will be awarded.