Visual interactive simulation 7.5p 5DV058 2011
Kenneth Bodin, Umeå University

# Lab Projects

## Overview

In the lab projects you will build some of the basic components of a general physics engine. We start by constructing a simple particle system, where particles only interact with a simple environment. Next we extend this to particles that interact through springs and dampers, so that we get a simple model for materials, e.g. cloth. These particles have fixed connections to their neighbours, but in general we need to actually find the neighbours to interact with – so this is what we do next, using spatial decomposition and spatial hashing. Thereafter we introduce constraints as an efficient alternative to springs and dampers for modelling a rope or cloth.
Next we simulate contacts and collisions and we also introduce rotational degrees of freedom so that we can simulate rigid body spheres, using an iterative contact solver. Finally, we go back to a particle system and use this to simulate fluids utilizing the smoothed particle hydrodynamics method.

NEW: The SPH lab is not mandatory. If you wish, you can instead work deeper into the other lab projects, and collect more bonus points there.

Various extra features and improvements can give you bonus credits. See lab instructions and the course web for more information on the credit and grading system. ***Always document very clearly the credit points you aspire for*!**

As a grand finale you build a 3D scene where all the components are integrated and used in a creative way. The scene is presented in a demo session at the end of the course and in a written report. You may show a live simulation or a captured video at the demo session. The lab report must be accompanied with a captured video.

# Lab 1 - General particle system for special-FX

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. 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, e.g. lifetime, age, colour, etc.)
```

We will not worry so much about optimization here, but it should be stressed that if particle-particle interaction is important (e.g. for the cloth and sph labs) the memory access to particle data should be fairly direct and not contain too many levels of C++ indirect referencing etc.

## The Simulation loop

The simulation loop should *roughly* have the structure below. Of course you are free to design things differently, motivated by how the equations should be solved and how you want your code to be modularized etc.

```
Setup initial conditions

while (running)

      Create particles at emitter
      (Remove particles at sinks or when they expire in time)


      Do inter-particle collision detection and construct a
      neighbour list – or use a fixed interaction list (cloth).

      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.

      Find contact sets with external boundaries, e.g. a plane.
      Handle external boundary conditions by reflecting the
      the velocities.

      Take a timestep and integrate using e.g. Verlet/Leap Frog
      If there still are overlaps in the contact set with
      external boundaries, you could project the positions of
      the particles to the constraint manifold, e.g. to the
      surface of the plane.

      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 so 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 of course is not required for the lab!).

## Simulation loop explained

### Create particles at emitter

For this lab we only require that particles are created at a rectangular or circular source (while in general one would like to be able to load general geometric objects, for example, from a 3D modelling program). Particles should be created at a constant rate (per frame or per second) on random positions, with an initial velocity that varies randomly from an average initial velocity, and with a direction that varies randomly from the normal of the creation area. It should be possible to translate and rotate the emitter, so that you later can attach it to a rigid body and let it follow its trajectory.

### Age and lifetime

If a particle has a finite lifetime, remove the particle from the simulation if its age exceeds its lifetime. What is the consequence of this for physical conservation laws of mass, momentum and energy?

### 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) and size of particle system. You will get a list of interaction pairs formulated as:

```
Interaction_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)
```

Alternatively, you can also create such a list by constructing an interaction list for each particle of the system. Later when you loop over particles, you may then also do a secondary loop over the neighbours of each particle.

Of course, for non-interacting particles, this task can be left out entirely! For e.g. cloth or some elastic 3D object with permanent connectivity, the list is computed at startup and used throughout the simulation.
Note: Make sure you don't double count. 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. $F_{ij} = - F_{ji}$). However, keep in mind that in a parallel implementation it may actually be more efficient to do this double counting for symmetry reasons!

**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, $\mathbf{F} = - k\, u\, \hat{\mathbf{u}}$ (*k* is a constant), and air friction where damping is proportional to the velocity squared, $\mathbf{F} = - c\, u^2\, \hat{\mathbf{u}}$ (*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. Here we will limit ourselves to handle a few simple geometries and not worry too much about efficiency or optimization. In order to create particle special effects in the first lab you should at least be able to handle a particle-plane contact. For large impact velocities, use the Newton impulse collision law (that is, velocity reflection with a restitution coefficienc) to handle impacts. For lower velocities you can use a spring-damper model to hinder penetration.

**Time integration**

Numerically integrate the system for new velocities and positions. As explained in the lectures, Leapfrog 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, sprites or shaders for visualization and rendering. Massively large numbers of spheres can be expensive to render unless optimized correctly. Experiment with colors, textures, orientation/normals and alpha channel to achieve acceptable visual models.

## Credit Points in lab 1

### Basic credit (mandatory):

Implementation and documentation of a basic particle system, used to create one special effect, e.g. dust, smoke, fire works, an explosion, or something of your choice. Handle collision detection with a plane, through reflection of velocities and a penalty force. Visualize using e.g. an alpha textured quad. (1 credit point)

### Bonus credits

Implement intersection tests with additional geometries, e.g. a box or a sphere. (0.5 credit point).

Implement a shader of your choice, or other method for more advanced rendering of a special effect. Try to use output from the simulation into the shader (for example, velocity, acceleration). (0.5 credit point)

Add inter-particle interaction with spring-damper interaction forces, so that a spring pushes the particles apart when their spherical shells overlap. An efficient simulation will require a broad phase collision detection algorithm, implemented later in the course. (0.5 credit point)

Implement a GPU optimized algorithm (0.5 credit point).

**Up to 1.5 bonus points can be awarded**. Clearly state what bonus credits you aspire for!

# Lab 2 - Simulation of Cloth

## Model and algorithm description

Follow the cloth 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 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 $\mathbf{r}_{ij} = \mathbf{r}_i - \mathbf{r}_j$ 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 along the spring. 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. 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 (this is needed later for rigid bodies).
- Accumulate air friction forces, proportional to the velocity squared of each particle, or implement an improved model for the cloth surface.
- Time integrate using e.g. Verlet/Leap Frog
- Visualize and render, by placing a texture on the resulting 2D mesh.
- Choose parameters for the spring constant and the damping, following the discussions in the book. Note that gravity sets the length and time scale of stability and that it is harder to simulate a small piece of cloth (e.g. 0.1x0.1 $m^2$) than a large piece of cloth (e.g. 3x3 $m^2$).
- When integrating the simulation in a 3D scene, the piece of cloth should either be attached to a dynamic object or be given initial conditions and/or external forces (e.g. wind/gravity) that make the simulation look interesting. Think of things made out of cloth e.g. .....

**Basic credits in lab 1 (mandatory):**

The task described above gives 1 credit point.
Document the method, stability and performance issues, and present all parameters used. In particular, test stability for large spring constants, i.e. stiff cloth.

**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 interact with boxes in your scene. (1 bonus credit)

Implement self collisions hindering particles or cloth filament from going through each other (your piece of cloth will have quite weirdo behaviour without this feature – just keep in mind that solving this problem in the general is very challenging). (1 bonus credit)

Implement an improved air resistance model using the geometry rather than just the particles, to model things like a sail or a parachute (very cool!). (1 bonus credit)

You can also suggest your own bonus credit tasks!

**Up to 2.5 bonus credits can be awarded.**

## Lab 3 – Constraints

Here, we will use SPOOK to distance constrain particles, as described in the lecture notes.

- Mandatory (1p):
  - Simulate a rope, consisting of particles linked together with a distance constraint.
  - Compute the Jacobian for this constraint, and model a system of N bodies with pairwise attachments using SPOOK, and use Gauss-Seidel iterations for computing the constraint forces.
  - Fix the rope at the upper end
  - Visualize the rope as it swings in the gravitational field.
  - Change the mass of the bottom particle so that it becomes much heavier than the other particles (e.g. 1: 10000). Study and comment convergence and stability aspects of this (e.g. number of required iterations)
  - Study how the direction of the iterations affect the convergence, i.e. do you get different convergence if you start from the top compared to starting from the bottom?

- Bonus credits
  - Study convergence by plotting the change in the constraint force. For example, plot $\sigma = 1/N \sum_i^N |\Delta\lambda_i|^2$ as a function of the iteration count, where $\Delta\lambda_i$ is the difference between consecutive approximations in $\lambda_i$ through the solve. How does $\sigma$ approach zero with iteration number? How does a large mass ratio (as discussed above) affect this convergence? How does the size of the timestep affect the solve error and the convergence?(3p)
  - Implement cloth using this method. Compare stiffness vs. size of maximum allowed timestep with your spring-and damper cloth. (2p)
  - Use a Conjugate Gradient solver instead of Gauss-Seidel. (2p).
  - Implement the Conjugate Gradient Solver on GPU. (2p)
  - Your own suggestions.

**Up to 5p bonus can be awarded.**

## Lab 4 – Rigid Body Simulation

In this programming task you will develop a simulator that can handle a pile of rigid-body spheres. You will implement broad phase collision detection for sphere-sphere interaction and intersection finds for sphere-sphere and sphere-plane interaction.

**The following is mandatory (1 credit points):**

1. Set up the framework you need to represent multiple rigid body spheres and a plane/halfspace.
2. Drop spheres so that they fall in a pile on the plane (or implement something more spectacular!). N.B. to test your simulator, first start with a single sphere on a plane. Since this problem consists of only one contact, you get a single equation to solve, and thus the Gauss-Seidel iterator is exact, and you can check your simulation results with analytical results. Next, add another sphere on top of the first sphere, and so on to sanity check and debug your code. Also use graphics to debug, by visualizing contact points, contact normals and forces.
3. Implement broad phase-collision detection for spheres using e.g. spatial hashing for a uniform 3D grid. For basic credit you only need to make this work for spheres that all have the same size.
4. Implement intersection finds for sphere-sphere and sphere-plane interaction.
5. Use the SPOOK method described in the lectures to resolve the collisions and contacts and then integrate the system.
6. 3D-visualize the system.

Comment on stability and performance!

**Bonus credits**

Visualize and analyze convergence aspects and errors of the iterative solver. For example, check if collisions really are resolved in the first iterative collision loop, and if not, how big are the errors (e.g. mean square deviation, or max error). Check also after the contact loop. Are there no non-separating contacts left? What is the overall convergence rate, i.e. how fast do the errors drop with number of iterations (plot as in the previous lab)? Measure the kinetic energy of a system with dissipation (that should go to rest with time), and plot how it changes with time as a function of e.g. timestep and number of iterations. What is the reason that the kinetic energy does not go identically to zero? (3p)

Generalize your simulation framework, e.g. mutual interaction with cloth (1p)

Implement broad phase collision detection for spheres of arbitrary size (well, within some reasonable bounds) (1p).

Introduce oriented boxes into your simulation. This means you have to find at least box-box and box-plane contact sets, and if you wish to integrate with spheres you also need to compute box-sphere sets. (2p)

**Max 6 bonus credits** will be awarded. Additional tasks for bonus credits might be added. You can also suggest your own. Talk to a tutor or lecturer first.

# Lab 5 – SPH simulation

In this lab you will model a container with a fluid inside – a bubbling witch pot.

## Model and algorithm description

Use the basic particle system developed in lab projects 1 and 2 with an emitter that creates ~ 500-2000 SPH particles. Extend the particle data structure so that it also can hold SPH variables, such as density and pressure etc. Find neighbours using the spatial hashing algorithm used in e.g. lab 3.

### 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 (i.e. the density pressure relation discussed in the lecture notes) 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.

Don't forget to include the self contribution from the particle in the SPH density summations!

### External forces

Accumulate the gravitational force on each particle and possibly also user interaction forces. Interaction with e.g. rigid bodies can also be added as external force, but this is not necessary for the basic credit.

### Collisions with the environment and boundaries

Your particles are placed inside a box or a sphere (your choice), and you need to run an interior test. If the particle is outside the box with a colliding velocity, apply a Newto-Coulmb impulse to deal witht his impact and bounce it back. Add a damped spring force at the contact point to avoid having the SPH particles drifting outside the container. Forces and impulses should also be applied to the container, i.e. every particle impulse should update the linear and angular momentum of the container.

### Visualize and render

Use an alpha textured quad or a point sprite to render your particles. If you choose to do the bonus assignment and compute the colour field to track the surface and surface normals, you can implement more advanced rendering models.

### Parameter values

We consider a container filled with 10 litre of water. SI-units are used for the parameters below. Parameters are rough recommendations – feel free to experiment!

**Number of SPH-particles:** $N = 1000$

**Mass:** $m = 0.01kg$

**Density:** $\rho_0 = 1000\,\frac{kg}{m^3}\;(water)$

**Interaction radius (smoothing length):** h is chosen such that 15-20 particles are interacting on average

**Dynamical viscosity:** $\mu = 0.001\frac{kg}{ms}$

( **Speed of sound:** $c_s = 1500\,m/s\;\;(water)\;\;\Rightarrow\;Time\,step: dt = 0.0001s$ )

**Speed of sound:** $c_s = 1 - 10\,m/s\;\;(water)\;\;\Rightarrow\;Time\,step: dt = 0.01 - 0.03s$

**Surface tension:** $\sigma_s = 0.073\frac{N}{m}\;\;(water)$

We recommend using the poly6 kernel below for simplicity. It not optimal for stability, but it works. 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 (NEW: not mandatory):

Basic implementation will give 1 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. When debugging, try to reduce the problem into independent components, i.e. check that the pressure force works reasonably, the viscosity force, the boundary projections, free fall in a gravity field etc.

## Bonus credits (choice):

Implement colour field and surface reconstruction. 1 credit
Implement colour field and fire propagation. 1 credit
Implement colour field and surface tension. 1 credit
Use povray and the scripts provided to render a movie of the fluid. 1 credit.
Implement rendering that uses the normals computed from the colour field to orient a textured quad (or figure out your own rendering method). 1 credit
Implement more general boundary conditions, i.e. collision impulses with other bodies and a ground plane so that you can pour out the fluid. 1 credit.
More suggestions might be added.
**Max 2 bonus credits** will be awarded.

As explained the SPH lab is not mandatory and instead you can collect more lab scores from bonus points on the other lab projects. It is certainly useful to learn about hands-on fluid simulations, so go for it if you are interested!