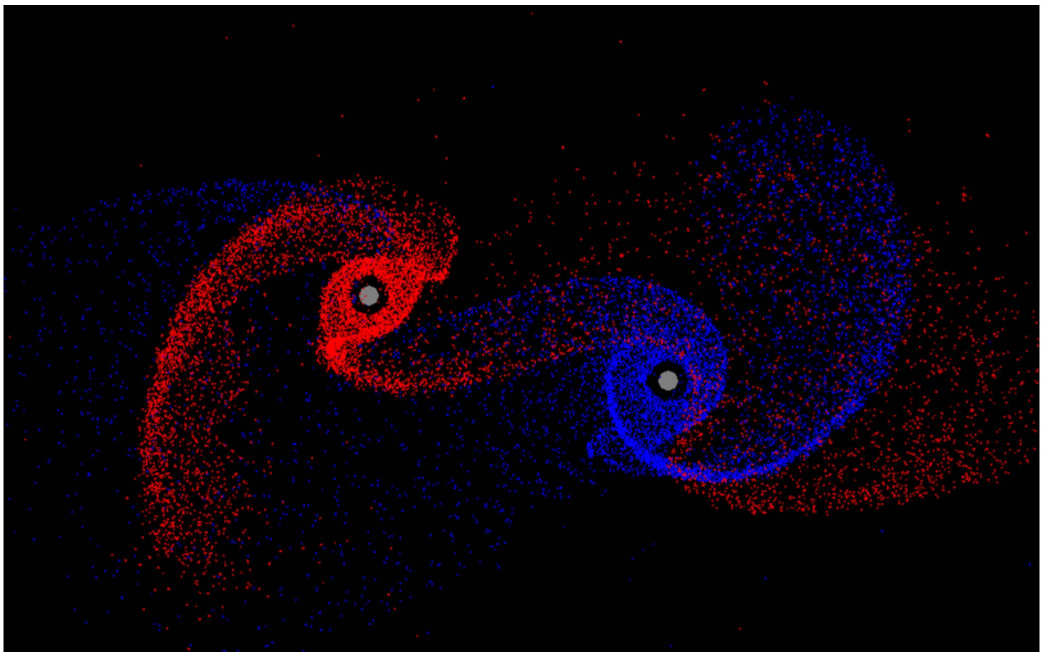
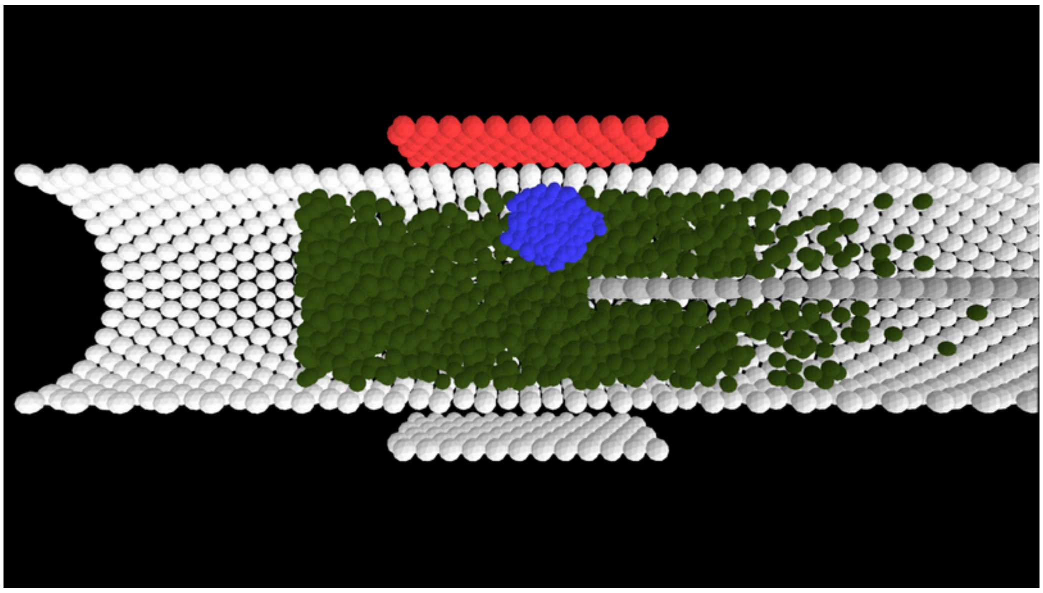
N-body simulations using the GPU





Adrian Lenkeit

Intel ISEF 2015

Contents

1. Abstract

The goal of my project is to develop software that allows the simulation of many-body systems. Numerical modelling can save a lot of time and money both in basic and applied research. I decided to exploit the parallel computing capabilities of the graphics adapter in order to be able to model many-body systems at low cost and I used the programming language C++. My first goal was to model collisions of galaxies and eventually I was able to simulate such collisions in two and three dimensional space. My results are consistent with astrophysical observations. Furthermore, I modelled solids, liquids and gases using the Lennard-Jones potential. I optimized these and my astrophysical simulations throughout my project. My simulations based on the Lennard-Jones potential reached their limit when I tried to model water molecules as I did not include quantum mechanical effects. However, I obtained interesting results in the area of nanofluidics, which clearly demonstrated differences between our macroscopic world and the nanoscale. Finally, I developed components for controlling liquids on the nanoscale, which could be used for the realization of novel devices.

2. Advantages of simulations

The development of functional and safe products requires both a lot of money and time. Fortunately, the advancements in computational power have made it possible to simulate the safety and functionality of a product and thus to reduce the number of tests required with a real prototype. In the field of fundamental research, the steady increase in computational power has made it possible to use computer simulations to test existing knowledge and even gain new knowledge required for future experiments.
One of the widely used methods for simulating physical systems is the N-body simulation. It has the advantage of being a very accurate model of the real physical world and thus creates accurate results. Unfortunately, this comes at the cost of using up a lot of computational power.

3. The GPU

An N-body solver has to calculate the interaction of each of the N particles with every other particle. The number of interactions I as a function of N can be described by $I(N)=\frac{N*(N-1)}{2}$.
For example, 500 particles result in ~125.000 interactions, which all need to be calculated. In many cases though, 500 particles are not enough. Often, thousands of particles are required, which quickly results in millions or even billions of interactions.
What makes it possible to calculate such huge numbers of interactions is, that each interaction can be calculated independently from every other interaction. In theory, you should be able to split up the workload of calculating the interactions of each particle onto several processors and thus to speed up the simulation. This has been successfully done before, but only with the help of huge computer clusters, which only a few people have access to.
This made me wonder whether it would be possible to speed up an N-body simulation avoding the need for big computer clusters, to model complex N-body systems on a normal desktop computer. The solution I had in mind to make this possible is to use the graphics processing unit (GPU) rather than the central processing unit (CPU) to calculate the interactions.
At first glance the GPU is a very powerful, but specialized piece of hardware. Over the years it has

been developed to render more and more complex 3 dimensional scenes. Each scene can consist of up to several billion vertices, creating triangles and more complex meshes, which a texture can then be applied to. Many shader effects can then also be applied to portions of the scene, like a mirror reflecting light or glass distorting objects behind it. To be able to render such complex scenes, graphics cards are built based on a different architecture than a CPU, in fact like a tiny supercomputer. Many modern graphic cards contain a minimum of one thousand CPUs per card which can calculate complex shader effects in a parallelized manner and thus speed up the rendering of a scene. Through recent technologies like Compute Unified Device Architecture (CUDA) [1], it has become possible to reprogram the graphics card to not calculate complex shaders, but nearly any kind of formula.
Because the graphics card is a basic requirement for any computer, it is available to every computer user. Reprogramming the graphics card through CUDA into an N-body solver should allow me and other people to simulate complex N-body systems at home.

4. Galaxy collisions

I wrote a program to set up the graphics card, upload and download data to and from it and to display the results of my N-body simulations. The program itself is relatively complex and thus only critical key components will be discussed here, in particular the kernel code running on the graphics card itself.
My first goal was to simulate the collision of two galaxies with each other. The results of such collisions are well known from astronomical observations, which allows me to compare and verify the results of my simulations. This simulation requires a formula that describes the forces which arise from the interaction of two bodies of a given mass. The force behind this is gravity which can be approximated by Newton’s Law of Gravitation.

$$\vec{F}_{(ij)}=G\frac{m_i * m_j}{|\vec{r}_{ij}|^2}\hat{r}_{ij}$$

where indices *i* and *j* represent the indexes of the two interacting bodies, *F* the resulting force, *G* the gravitational constant, *m* the mass of one body and r_{ij} the position of body *j* subtracted from the position of body *i*. Any force that acts on a mass results in an acceleration of the mass, described by $\vec{a}=\vec{F}_{ij}/m_i$, with *a* being the acceleration. Using Newton's second law to simplify the Law of Gravitation results in a shortened version of the formula, which directly returns the acceleration and requires less computation, thus speeding up the simulation later on.

$$\vec{a}_{ij}=G\frac{m_j}{|\vec{r}_{ij}|^2}\hat{r}_{ij}$$

To calculate the total acceleration one mass experiences, the partial accelerations described by the above formula must be summed over every other mass in the system

$$\vec{a}_i=\sum_{j=1}^{N} G\frac{m_j}{|\vec{r}_{ij}|^2}\hat{r}_{ij} \quad , (i\neq j)$$

After the total acceleration of every mass has been calculated, the velocity and position need to be updated accordingly. The resulting equations are differential equations which can only be solved using a numerical integrator. Many numerical integrators exist, with some having a high accuracy and others being fast. For my simulations of galaxy collisions, the Leap-Frog-Integrator [2] can be used. In comparison to other integrators, it has a slightly lower accuracy, but it is faster. The drop in accuracy is negligible when using 32 bit floating point values, but the increase in speed can result in overall faster simulation times. The Leap-Frog-Integration for velocity is $v_{t+\vec{1/2\mathrm{dt}}} = v_{t-\vec{1/2\mathrm{dt}}} + \vec{a}_t dt$, with *dt* being the time each of the simulation, *t* the time of the current step and *v* the velocity. The position is integrated in nearly the same manner with $\vec{x}_t = x_{\vec{t-dt}} + v_{t-\vec{1/2\mathrm{dt}}} dt$ and *x* being the position of the particle. These three basic equations are sufficient to program a simple kernel for the graphics card which is able to simulate an N-body system.

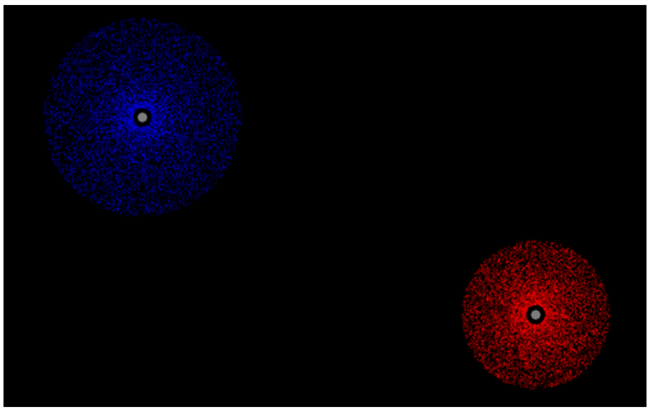


Fig. 4.1: Two disk galaxies moving toward each other

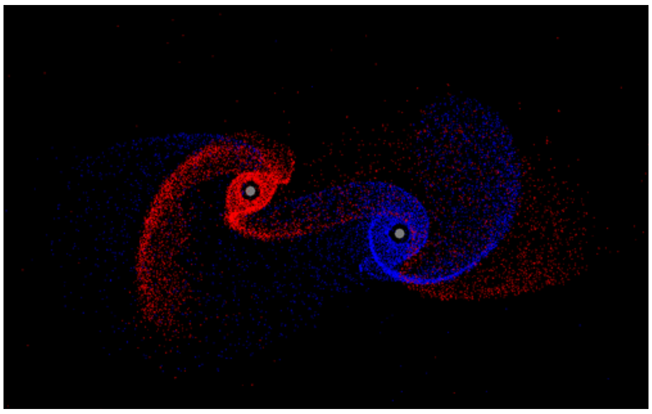


Fig. 4.2: Two disk galaxies transforming into spiral galaxies after a collision

I tested my N-body solver by simulating a collision of two disk galaxies in two dimensions. The result of such a collision is described by the Hubble Sequence [3], which predicts that, depending on the size of the galaxies and the collision vector, the two galaxies should merge and form one spiral galaxy or both galaxies survive the collision and also become spiral galaxies. In my simulation, I consider a near-hit of two galaxies of similar mass and size, resulting in the two galaxies not merging with each other but still strongly interacting through gravity. The result of the simulation at each time step is displayed live through OpenGL. In the simulation itself, when the two disk galaxies come closer together, they attract each other's star disk and central black hole (Fig. 4.1). When the galaxies are close enough, so that the black hole of one galaxy is within the star disk of the other, the shape of the galaxy changes rapidly. Some stars are flung away into space, while other stars change their orbit. Over a short period of time, this results in both galaxies loosing their initial disk form. The density of the stars is highest near the black hole, where a very tiny, but dense and bright disk has formed. Both galaxies also develop a long arm spiraling out of the center into space and another, shorter arm pointed at the other galaxy's black hole (Fig. 4.2). Both former disk galaxies transformed into spiral galaxies, as predicted by the Hubble Sequence and shown by my simulation. In Fig. 4.3, a collision of two real disk galaxies can be seen that formed a very similar structure in comparison to the results of my simulation.

Fig. 4.3: A potograph of a galactic collision, resulting in two spiral galaxies being created [I1]

My initial simulation returned accurate results but was limited to two dimensional space and thus being restricted to galactic collisions where both galaxies lay on one plane. Therefore, I extended my program to also allow the simulation of three dimensional collisions, allowing every possible configuration of galaxies. Using my modified program, I created a new simulation again representing the collision of two disk galaxies, but with one galaxy being rotated by 90°, so that it is perpendicular to the other galaxy. The galaxies collide in a near-miss scenario. After the collision both galaxies develop two spiral arms. The arms of the rotated galaxy lengthen over time and as a result they become very thin. This galaxy also did not develop a very dense disk near its center like in the previous simulation (Fig. 4.6). Instead, the other galaxy develops the thin, but very dense disk near its black hole. It also develops two spiral arms considerably shorter and thicker than the spiral arms of the rotated galaxy (Fig. 4.7). The galaxy also develops a warp that can be seen when viewed sideways (Fig. 4.8), which is also the case for real galaxies that developed in a similar manner (Fig. 4.9) [4].

5. Solids

After my simulations of galactic collisions, I wanted to simulate a very different field of physics that can be described by N-body simulations. I choose to simulate the formation of solids from gaseous and liquid phases and the transition into these phases. The force I based my simulations on to model the interactions of atoms with each other is the Lennard-Jones-Potential [5], given by $\vec{a}_{ij}=-24\epsilon\{2(\frac{\sigma^{12}}{|\vec{r}_{ij}|^{13}})-(\frac{\sigma^{6}}{|\vec{r}_{ij}|^{7}})\}\hat{r}_{ij}$, where ε is a constant for the strength of the potential and σ a constant for the range. When plotted (Fig. 5.1), the use of the Lennard-Jones-Potential for the simulation of atoms becomes clear. It approximates the behavior of two uncharged atoms that attract each other slightly when they are a small distance apart, and repulse each other with a very strong force when the distance between them gets very small. Therefore, it can be used to simulate the formation of solids, liquids and gases.

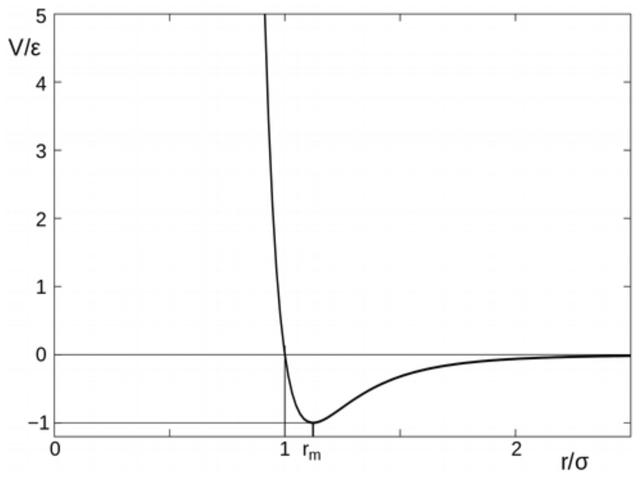


Fig. 5.1: A plot of the Lennard-Jones-Potential relating the potential enegry to the distance

As a first simulation I put 1000 atoms in a box in two dimensions. Not only the Lennard-Jones-Potential was acting between the atoms, but also a uniform downward force resembling the gravity on earth's surface. What I expected to

happen was, that the atoms clump together to form a solid slowly falling down to the bottom of the box. Instead, I noticed that the atoms got more and more kinetic energy until they were unable to form a solid. After some analysis to find out why this happens, I concluded that the repulsive force between two colliding atoms compared to the attractive force is so high, that the whole system starts to heat up over time due to tiny errors in the numerical integrator that become important because of this big force difference. To solve this, the time step could be reduced, which increases the overall accuracy of the simulation, but this would only slow down the unwanted heating effect. Another numerical integrator could also be used, but after some tests with other integrators like the Verlet-Integrator [6], the heating effect stayed or even reversed into a cooling effect. Because this severely violates the conservation of energy, I needed another method to make the energy stay constant without slowing the simulation down too much. The solution I came up with was to enable the manual and automatic cooling and heating of the whole system, or in other words, to enable the removal or addition of kinetic energy to all atoms at run time. Because the total energy of the system is known at the start of the simulation, unphysical changes in the total energy can be adjusted for with changing the kinetic energy of each atom accordingly, so that the overall energy of the whole system stays constant. This also enables manual changes of the total energy at run time, so that the system can be cooled or heated up to transition between the different phases of matter.
After changing my program to conserve energy, I repeated the simulation. After some time, the atoms positioned themselves into a hexagonal grid, which is the lowest energy state for an atom in a Lennard-Jones solid (Fig. 5.2). Because of the uniform downward force, the solid also dropped to the bottom of the box. After adding enough energy to the system, the atoms at the surface of the solid broke out of the hexagonal grid and were fast enough to create a gas (Fig. 5.3). Over time, the whole solid transitioned into a gaseous phase.

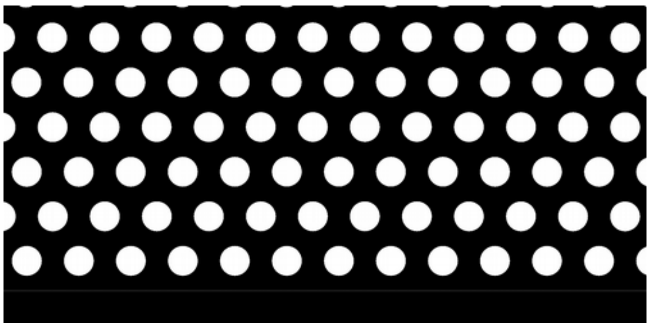


Fig. 5.2: Atoms arranged in a hexagonal grid forming a solid

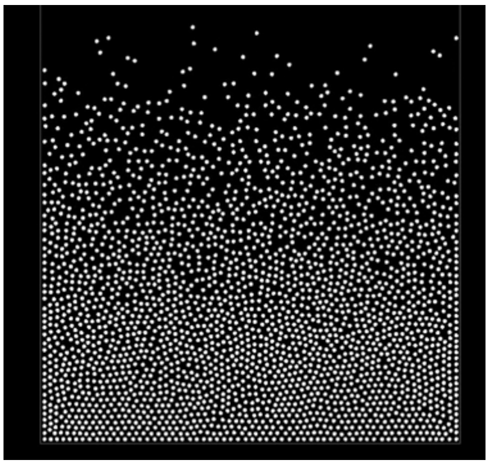


Fig. 5.3: A solid being heated, creating gas at the surface

6. Optimizations

The previous simulations all worked well, but were limited by the number of particles I could put into the simulation. With my unoptimized program, simulations with more than 3000 particles created a noticeable delay per step. To simulate more complex physical systems, I would have to wait a long time before the simulation would return a proper result. Therefore, I needed to optimize my program to be faster, while still returning accurate results. One optimization that works for every type of N-body system that comes from the fact that I use the graphics card relates to memory access [7]. When one process tries to gather data about the position of other particles in the N-body system, it needs to access the global memory of the graphics card. Global memory access in itself is very slow and usually takes between 500-1000 clock cycles in between which nothing can be calculated. To reduce the access times to global memory, the memory controller of the graphics card tries to group the access of memory regions together. This only works though, when there is no gap between the requested memory regions. In my case, when I try to read the position of a particle, the position data is composed as a three dimensional vector with the three elements *x, y* and *z*. Because the vector cannot be read from memory at once, every thread first requests the *x* component of the vector. The *y* and *z* component then create a gap that significantly slows down the access to the memory. To speed the access up, the memory can be reorganized to represent three continuous lists that only store one vector component each, but have no gap when accessed. This speeds up my simulations by a maximum of 40% and an average of 30%, depending on the size of the simulation. Another optimization that only works for the Lennard-Jones-Potential is to cut off the potential after a set distance. The idea behind this comes from the fact, that the Lennard-Jones-Potential quickly looses in strength with distance until it becomes insignificant. Calculating interactions further than this radius has no significant impact on the simulation's outcome, so they can be ignored and a lot of time can be saved.

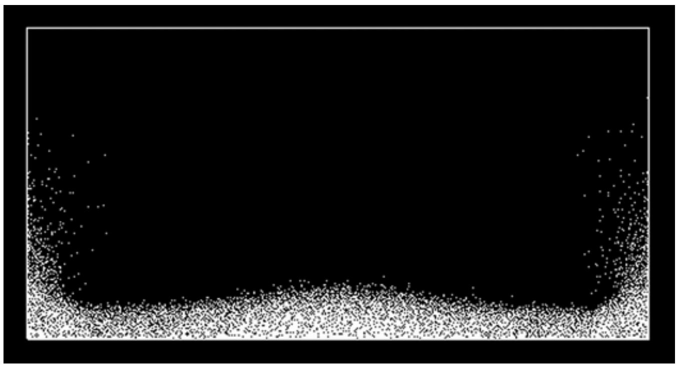


Fig. 6.1: A fluid simulated using 10000 particles

A way to implement this cutoff radius very effectively is to restrict the area the simulation can take place in and to then overlay a grid onto that area. Each cell of the grid gets an edge length as big as the cutoff radius and gets filled with particles. Then, only the interactions between particles within neighboring cells are calculated, which speeds up the simulation substantially. I was able to simulate a fluid using the Lennard-Jones-Potential with 10000 particles in real time (Fig. 6.1).

7. Nanofluidics

Using my optimized program, I wanted to test whether I could use my simulations on the graphics card for research in the field of nanofluidics [8] in which the the flow of fluids in the scale of up to 100 nanometers is studied to enable nanoscale Lab-on-a-Chip devices.
In the macroscopic world, the flow of fluids through a pipe is described by Bernoulli and Venturi, but it proves to be problematic to apply the same laws to pipes with diameters of only a few nanometers. In a first simulation I wanted to test this using the Lennard-Jones-Potential to simulate a liquid flowing through a pipe with a diameter of only a few nanometers which has a narrowing build into it. The first problem that arised was, that the attractive forces between the liquid and the inner wall of the pipe makes the liquid stick to the wall and therefore it resists being pushed trough the narrowing. Applying high pressure and therefore high energy to the liquid causes parts of it to heat up and transition into a gaseous phase. This shows that the laws of the macroscopic world cannot be transferred directly into the microscopic world.
One of the problems I had in my previous simulation was how to pump the liquid through the pipe without mechanical parts and it sticking to the inner wall of the pipe. An idea I had to solve this problem was to exploit electrostatic forces to control the flow of the liquid. A droplet of liquid that is electrically charged by ions could be pumped through a pipe by electrostatic forces if a section of the pipe behind the liquid is also charged with the same polarity. This will cause the ions in the liquid and thus the liquid to be repelled by the charged section of the pipe. Furthermore, repulsive

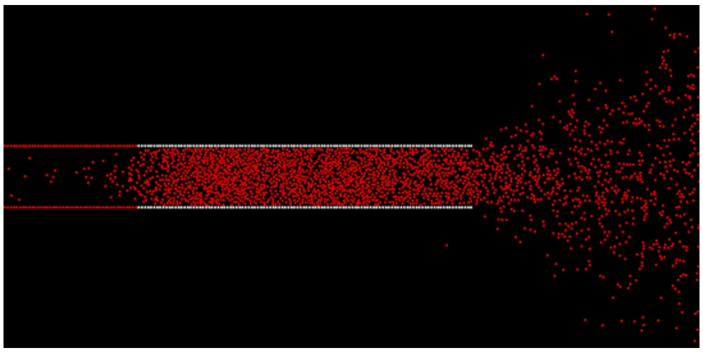


Fig. 7.1: A pump for electrically charged liquids

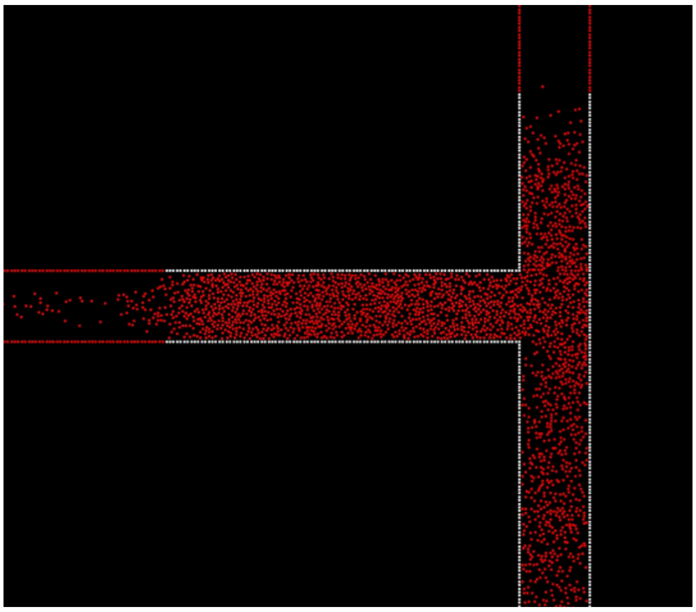


Fig. 7.2: A T-junction pipe with transistors and a pump attached to its arms. Only the upper transistor is active, therefore the liquid flows down

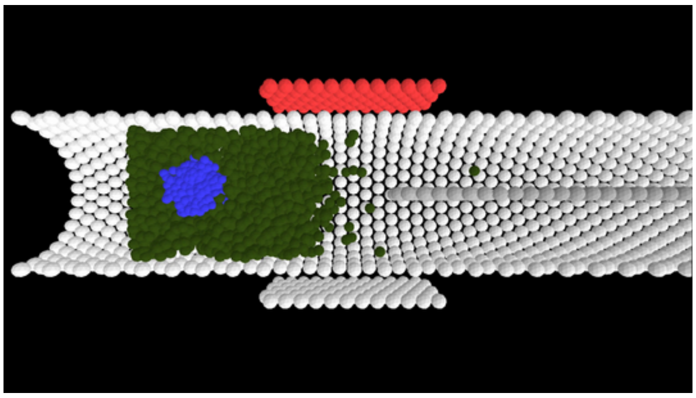


Fig. 7.3: A junction using an electrode to attract a water droplet and a carrier liquid to transport the droplet

forces between the ions in the liquid may prevent the liquid from forming a solid that can stick to the walls of the pipe. To simulate this, Coulomb's Law is required to model the forces that arise from the different charges. It is the same as Newton's Law of Gravitation, except that the masses are replaced with charges and the gravitational constant becomes Coulomb's constant.

In the simulation the part of the pipe behind the liquid gets charged, which causes the liquid to get pumped through and out of the pipe (Fig. 7.1). The concept of using charges to control liquids can even be extended to include an element that behaves like a transistor for liquids, enabling or disabling liquids the flow of liquids by using a long segment of the pipe in front of the liquid, which can either have no charge and let the liquid flow through it or be charged to repel the liquid. Attaching two of these liquid transistors to the arms of a T-junction allows one to build a switch that allows choosing which path a liquid takes (Fig. 7.2).

One downside of using ions to make the liquid react to the charge-based components build into the pipe is that these ions limit what can the transported along with the liquid: any chemicals that react with the ions are not suitable. To solve this, I combined my method of using static charges to control the liquid flow with ideas from the field of microfluidics. There, to pump a liquid through a pipe, the pipe is filled with a pressurized carrier liquid. Then, water droplets are put into the carrier liquid as container for chemicals, biological samples, etc. As the water molecule has a permanent dipole, it has the property of reacting to static charges on its own, it is attracted by them. Using my simulations, I wanted to simulate the T-junction from my previous simulation, but using the non-polar pressurized carrier liquid as a pump to move a water droplet along the nanopipe and two electrodes placed directly at the beginning of the junction to control which path the polar water droplet takes (Fig. 7.3). Applying a charge to one of the electrodes then attracts the water droplet and redirects it onto the corresponding path (Fig. 7.4).

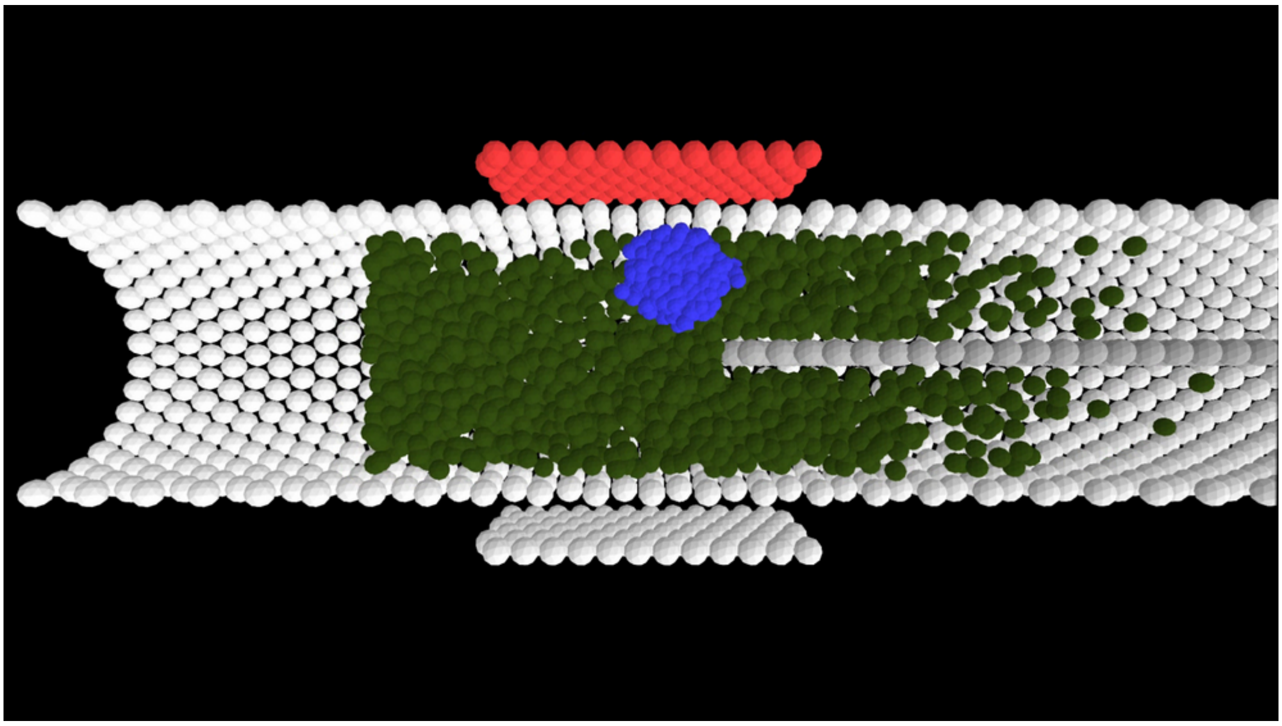


Fig. 7.4: Because the upper electrode is active, it attracts the water droplet, which is therefore redirected to the upper path

8. Software

The program I wrote to simulate all my previous simulations requires knowledge of C++ and CUDA to use and to create new initial setups for new simulations. People who are not familiar with CUDA or C++, but want to use my program would first need to learn C++ or CUDA. To still allow the fast creation of N-body systems based on my research, I set myself the goal to attach a user interface to my base program, which allows the fast creation of N-body simulations using the GPU even without being familiar to technologies like CUDA.
I am still developing the software, which currently enables the user to create a two dimensional n-body simulation.

9. Summary

I was able to show that complex N-body systems can be simulated in large scale on a graphics card using recent technologies like CUDA. My simulations provide correct results and are still very fast, thus my method can be used for research. The software I am currently developing would enable other people to create N-body simulations on their desktop computer.

10. References

[1] - http://www.nvidia.de/object/cuda-parallel-computing-de.html

[2] - http://einstein.drexel.edu/courses/Comp_Phys/Integrators/leapfrog/

[3] - http://www.spacetelescope.org/images/heic9902o/

[4] - http://hubblesite.org/newscenter/archive/releases/2001/23/

[5]- http://www.chemgapedia.de/vsengine/glossary/de/
lennard_00045jones_00045potenzial.glos.html

[6] - http://www.physics.udel.edu/~bnikolic/teaching/phys660/numerical_ode/node5.html

[7] - http://on-demand.gputechconf.com/gtc-express/2011/
presentations/NVIDIA_GPU_Computing_Webinars_CUDA_Memory_Optimization.pdf

[8] - https://www.london-nano.com/research-and-facilities/themes/
techniques/microfluidics-nanofluidics

[I1] - http://www.nasa.gov/images/content/223974main_wildgalaxies1_20080424_HI.jpg