Report to the Third Research Turn

1800012969 Haoxiang Yang
Advised by Baoquan Chen

Abstract

In this third research turn, I was engaged in physical simulation direction in Professor Chen Baoquan's team. During these five weeks I mainly completed the simulation task of the course Dartmouth COSC 89.18/189.01, tried to use C++ to simulate springs, particle fluid, rigid body and Quadrotors--a type of flying machines, and participated in the talks and meetings weekly.

Description of the experience and what's learned and achieved

1. Experiment: Abstract

In this rotation, I mainly completed the assignment of the course: Dartmouth COSC 89/18/189.01, Computational Methods for Physical Systems (Fall 2019). I used six course notes and four assignments. They taught me the simulation of springs, particle fluid, rigid body and Quadrotor. In this course, the students were given some base codes (including initialization and visualization units). They were asked to completed some essential parts using the formulas they have learned from the course. After coding, they could check the correctness of their codes by running the simulation interactively, and can see whether the results are stable and logical.

In the following part, the codes are completed by myself (with some seniors' advice). All my codes are available on Github now at: https://github.com/yanghaoxiang7/Yang-Haoxiang-Third-Research-Turn.git.

2. Mass-Spring Model

Springs are common in our everyday life: Ball-pens, keyboards, bicycles, cars, clocks all have springs inside. In the simulation of a mass-spring system, we design some points (with mass, see the points in Figure. 1.1, 1.2 and 1.3) and connected them with springs (shown as lines). We release the spring from a certain height and watch it fall, rebound, and rebound again towards another side.

We mainly use these tools to accomplish the simulation: spring force equation, damping force equation, Newton's second law, Explicit Euler Method and Implicit Euler Method.

Spring force: the spring tension. Proportional to the spring growth.

Damping force: the impediment of movement. Proportional to the relative velocity of two ends of the spring.

Explicit Euler Method: Use the position and the velocity in time t to calculate the position and the velocity in time t+1.

Explicit Euler Method: Express the position and the velocity in time t+1 using itself, solve the equation to get the position and the velocity in time t+1 (using Taylor series).

Results of Explicit Euler are shown below in Figure 1.1, 1.2 and 1.3: (corresponding to Pattern 1, 2, 3)

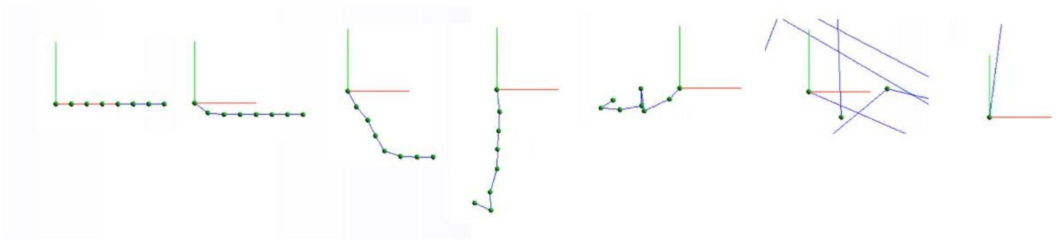
Figure 1.1: Mass-Spring Model, Explicit Euler, not optimized.

Pattern 1: A spring composed of 8 points. Failed to converge.

From Left to right are pictures taken at some time point in the simulation.

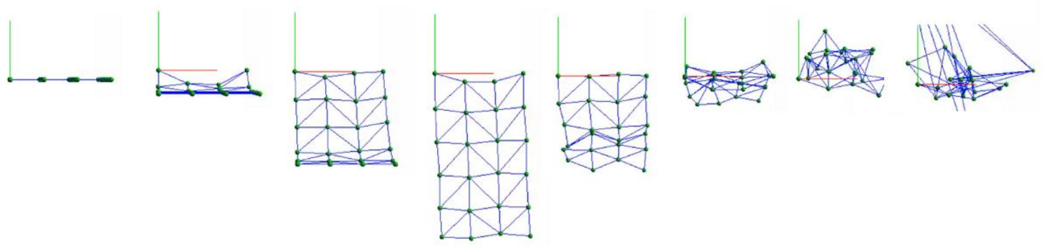
Figure 1.2: Mass-Spring Model, Explicit Euler, not optimized.

Pattern 2: A 2D grid. Failed to converge.

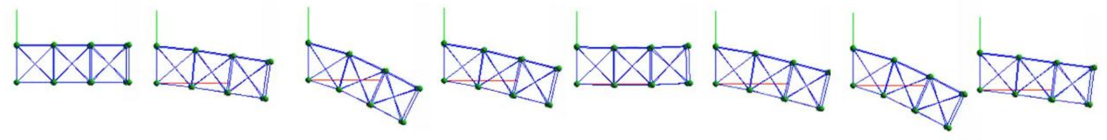
Figure 1.3: Mass-Spring Model, Explicit Euler, not optimized.

Pattern 3: A 3D grid. Converged.

Results of Implicit Euler are shown below in Figure 2.1, 2.2 and 2.3: (corresponding to Pattern 1, 2, 3)

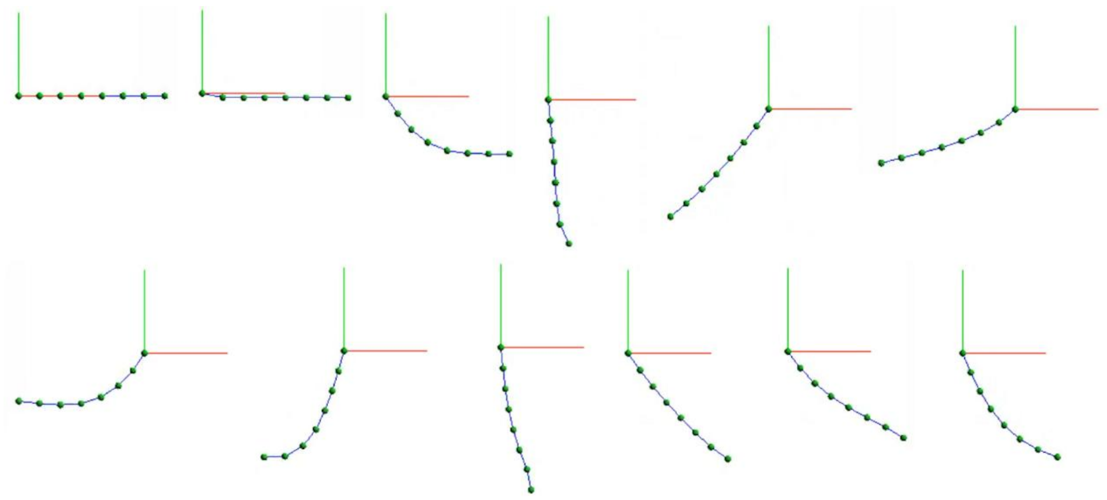
Figure 2.1: Mass-Spring Model, Implicit Euler.

Pattern 1: A spring composed of 8 points. Converged.

From up to down, from left to right are the pictures at different time points.

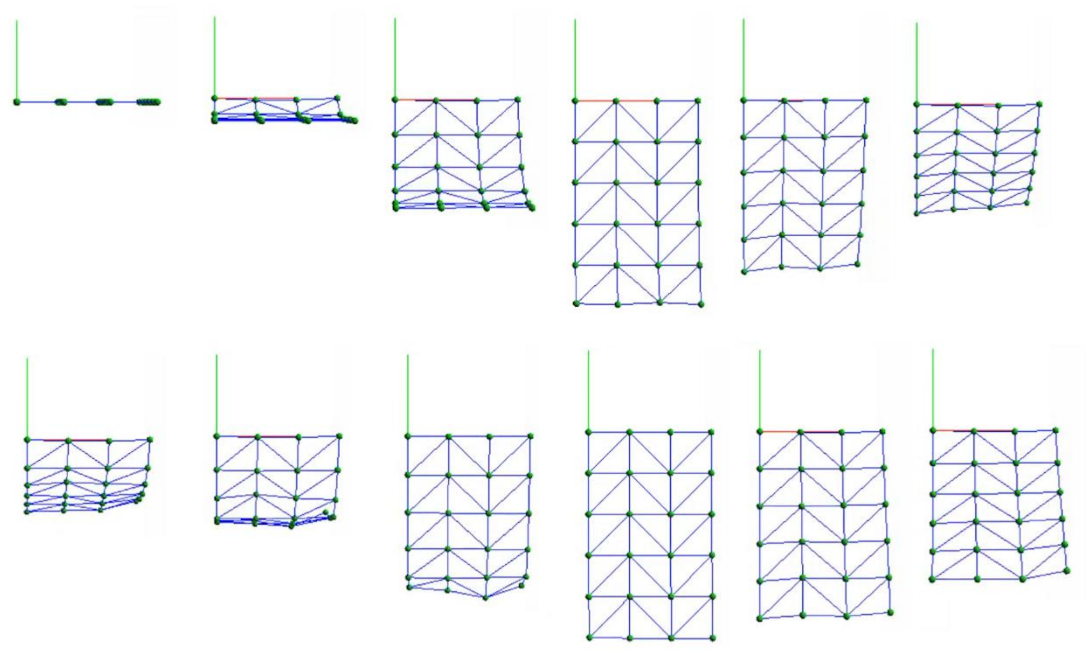

Figure 2.2: Mass-Spring Model, Implicit Euler.

Pattern 2: A 2D grid. Converged.

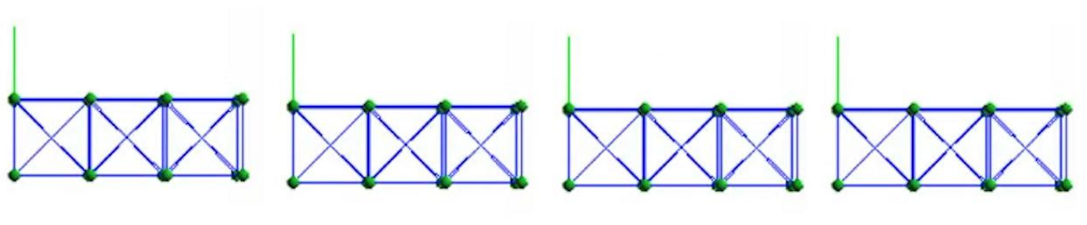

Figure 2.3: Mass-Spring Model, Implicit Euler.

Pattern 3: A 3D grid. Converged. The springs stay unmoved.

The Explicit Euler Method is much less stable than its implicit counterpart, as its approximation is kinds of 'inaccurate'. However, I found that we can improve the stability of the results significantly by rearranging the order of the calculation of position and velocity (the idea is similar to Implicit Euler Method) and lowering the elasticity a little. But the results are still not good. We can find some small strange movements. Also, in the third pattern, as the elasticity is reduced (k = 5e2 to k = 1e2), the spring system goes down. This is only a small trial, but I would suggest that we may dig into that.

Results of Explicit Euler Method (edited) are shown below in Figure 3.1, 3.2 and 3.3: (corresponding to Pattern 1, 2, 3)

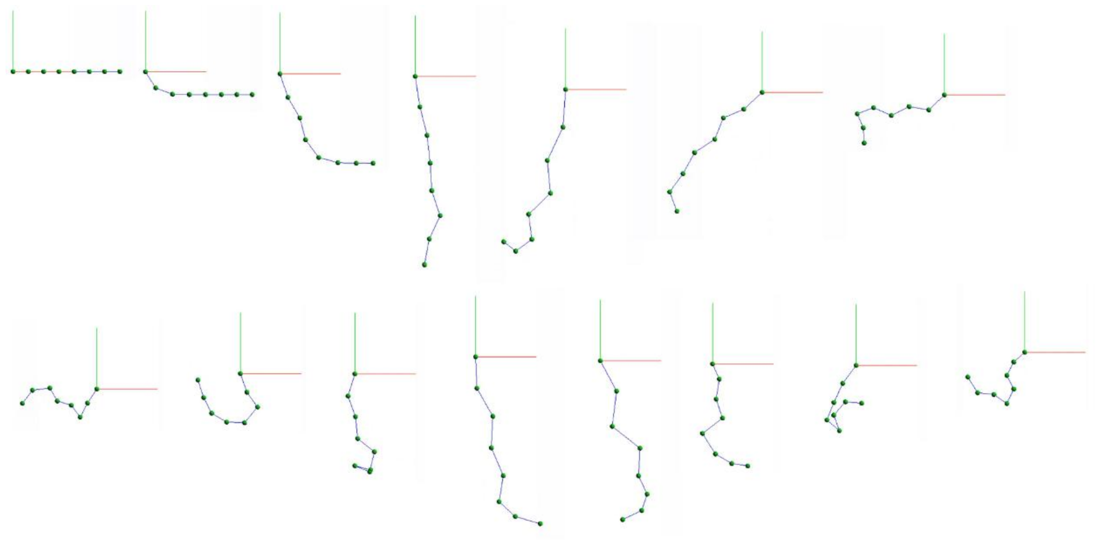

Figure 3.1: Mass-Spring Model, Explicit Euler, edited.

Pattern 1: A spring composed of 8 points. Converge.

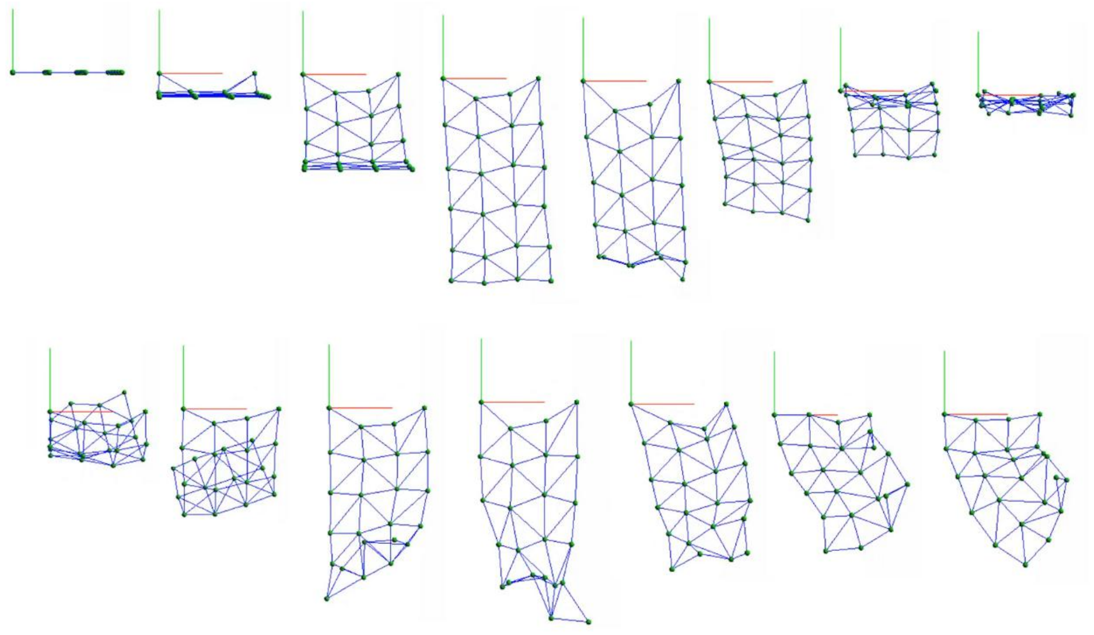

Figure 3.2: Mass-Spring Model, Explicit Euler, edited.

Pattern 2: A 2D grid. Converged.

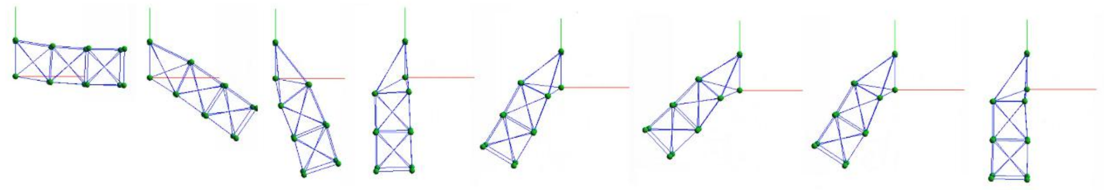

Figure 3.3: Mass-Spring Model, Explicit Euler, edited.

Pattern 3: A 3D grid. Converged.

Note that the results are rather strange: the gravity pulls all the spring system down.

3. Particle Physics: Fluids

We cannot simulate fluids directly nowadays, as the number of particles in even a drop of water is too large to calculate accurately. Yet we can simplify the process by simulating less particles as long as the results are acceptable. Here I have completed a simulation of particle fluids, in which several colorful particles fall down to a bowl, press each other, and rebound to the sky. You can add new particles to the bowl interactively.

The tools we use are: collision detection and response, neighbor search, fluid force calculation and kernel functions.

Collision detection and response: once the two particles are too close (as the distance between them is shorter than the sum of their radius), we add a force similar to that in the mass-spring model to detach the two particles from each other.

Neighbor search: We use a data structure, hash table, to acquire the neighbors of a particle as quickly as possible. The table key is the cell coordinate while the table item is the list of indices. The size of the grid cell is set to be greater than the kernel radius, so that a one-ring search can get all the neighbors.

Fluid force calculation: We use the momentum equation for the particle fluid and approximate the density, pressure, smoothed pressure force, smooth viscosity force and the body force using the following kernel functions.

Kernel functions: We use Spiky kernel to approximate every term in momentum equation except that in calculating viscosity force we use viscosity kernel instead.

Results of Particle Fluids are shown below in Figure 4.1 and 4.2: (corresponding to Pattern 1 and Pattern 2)

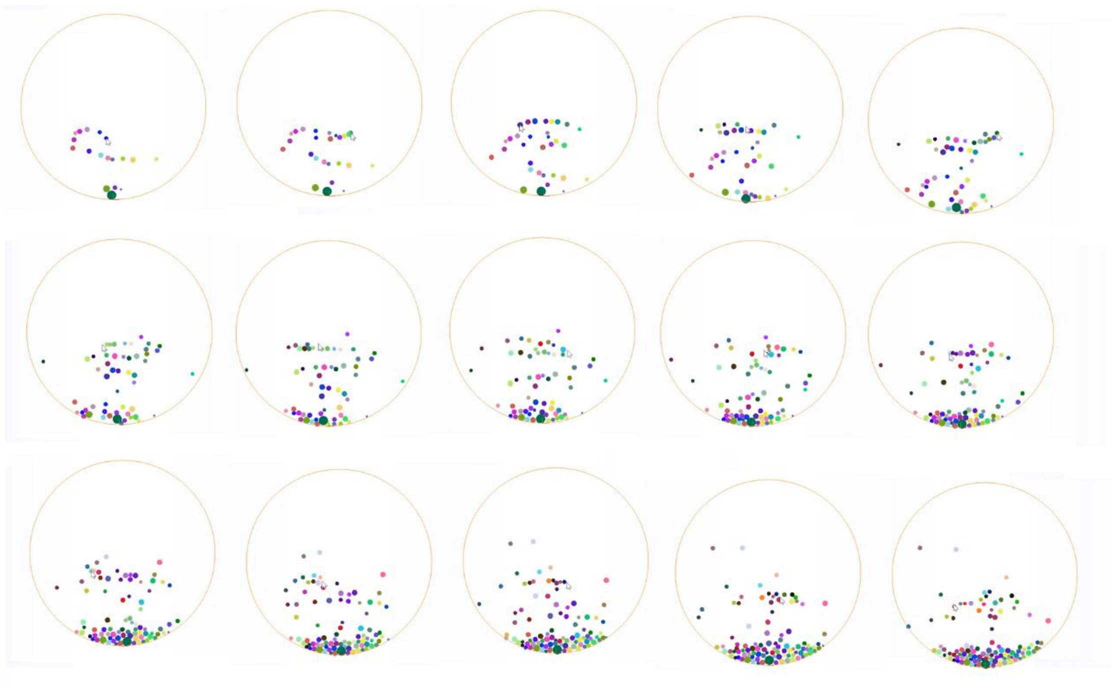

Figure 4.1: Particle Physics and Particle Fluids.

Pattern 1: colorful marbles. From up to down, from left to right are pictures at different time.

We can add new particles by a simple click, and watch the colorful particles collide.

The particles added are at random size.

We can add new points outside the bowl and we will see it come back to the bowl quickly.

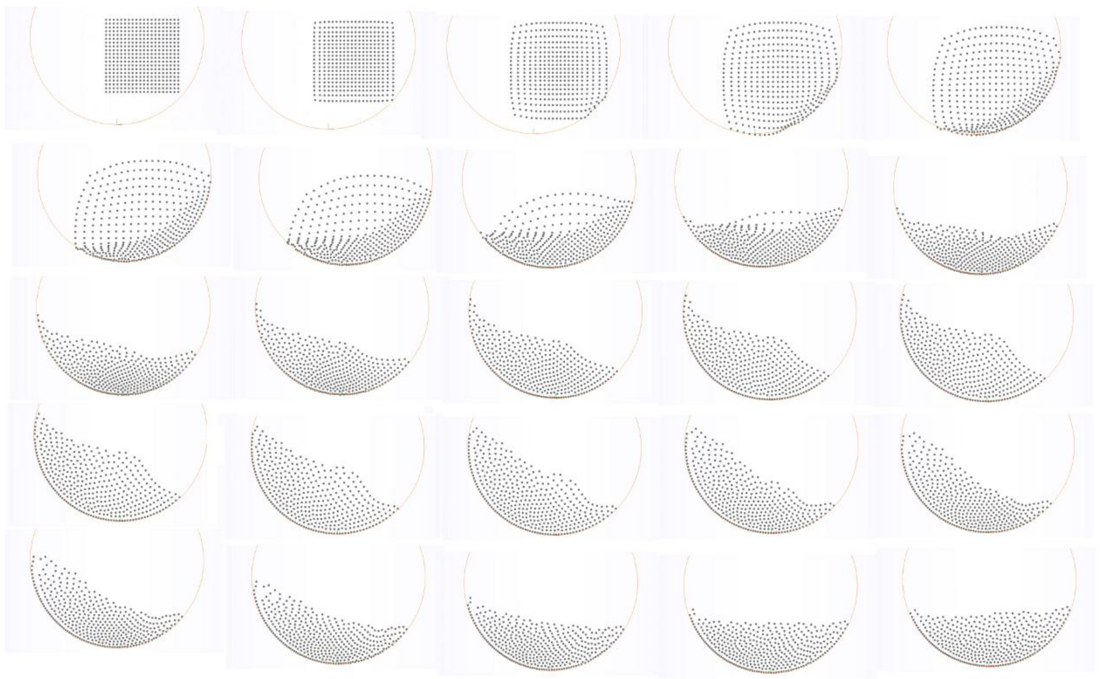

Figure 4.2: Particle Physics and Particle Fluids.

Pattern 2: grey particles. We can see more particles in this simulation.

We can also add new grey particles interactively.

To make the result more vivid I altered the number of particles, changed their radius, and gain the pictures below (Figure 4.3):

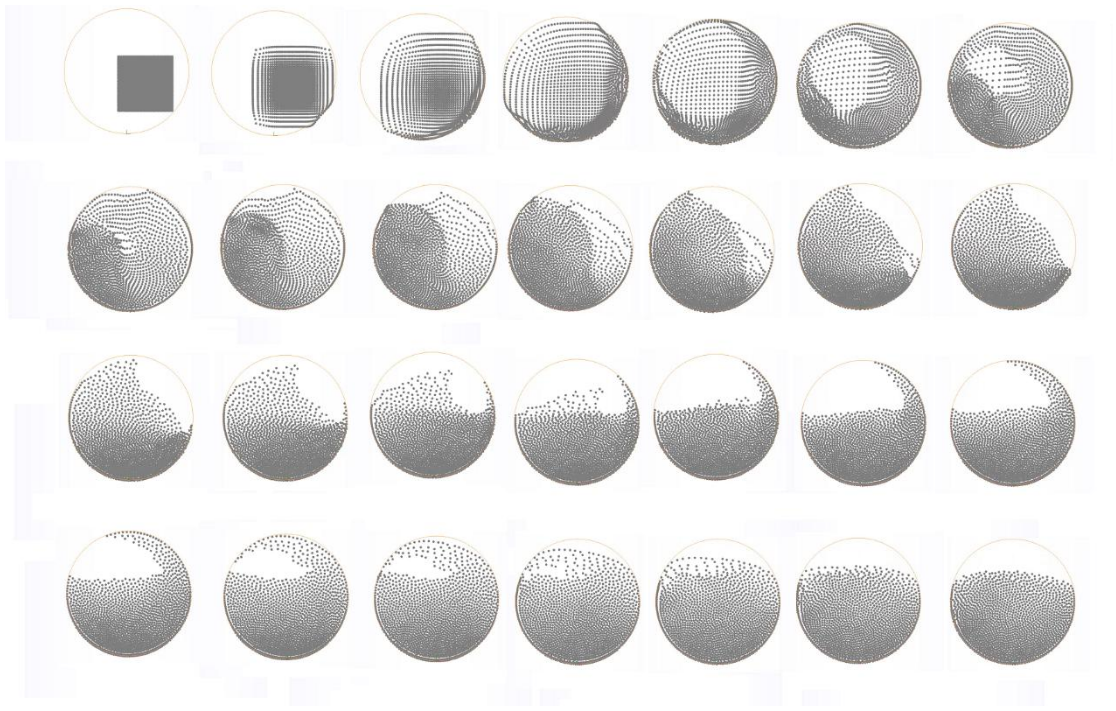

Figure 4.3: Particle Physics and Particle Fluids.

Pattern 2: grey particles.

The number of particles each row and each column are doubled.

The size of the particles is largened to perform a denser result.

4. Grid-based Fluid

In order to simulate fluid, not only can we approximate it by reducing the number of particles we use, but we can also divide the space into grids, and replace all the particles in one grid by the four grid points. I have completed a simulation of Grid-based Fluid, which can be seen in Figure 5.1 and 5.2. As you can see in this graph, we can print the grid points (as points) and their velocity (as longer or shorter lines), and we can add additional particles with small curl to the simulation. When we do that, we will see the particles flowing towards right hand side, just like a leaf floating down the stream.

The tools here is inviscid and incompressible Navier-Stokes equation (already stated in the note of the last part), Euler equation (which is later recomposed to be a Poisson equation), Advection, Projection and Vorticity confinement.

Advection: use semi-Lagrangian scheme, which utilizes the idea of tracing back a position and the technique of interpolation.

Projection: Solving the Poisson equation with Gauss-Seidel smoothing operator, which requires gradient, divergence, curl and Laplacian of a grid. As stated above, we use the information of the four grid points to approximate this.

Vorticity confinement: used to preserve the vorticity we add extra force along the rotation.

Particle Fluids are shown below in Figure 5.1 and 5.2:

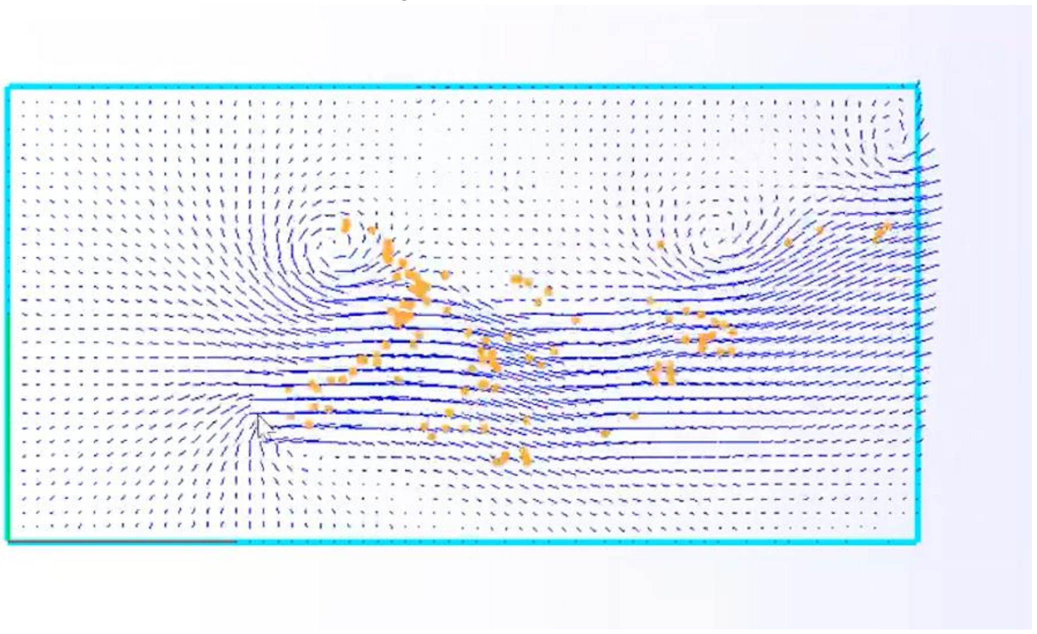

Figure 5.1: Grid-based Fluid Simulation.

Click and you will see a new curl and some yellow points.

The points will flow downstream (toward right-hand side).

The blue lines demonstrate the moving direction of the fluid.

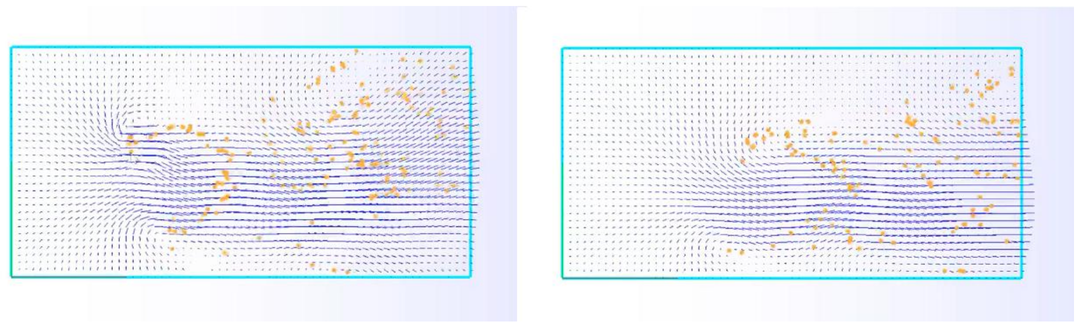

Figure 5.2: Grid-based Fluid Simulation without Vorticity confinement.

The picture on the left shows the stream just after a click.

The right shows the stream in one second.

In this second graph, as Vorticity confinement is off, the curl is weakened very fast due to the noise.

We can see from Figure 5.2 that the Vorticity confinement part is not required. Yet with Vorticity confinement it is easier for us to keep the vortexes.

5. Quadrotors: Rigid-body simulation and control

Quadrotors is a flying machine with four 'wings' to keep its balance. Drones (UAV) are extremely popular nowadays with its convenience and reasonable price. They are widely used in police, city management, agriculture, geography, climate prediction, video shooting and other fields. Many of these flying machines utilize rotors (especially four rotors) to acquire thrust and keep balance, as the four rotors design have the benefit of better control. In the following graph (Figure 6.1 and 6.2) you can see the quadrotor flying from the origin point, travel along each of the four destinations and go back to the origin with some background noise. This indicates that with this rigid-body simulation, we can reproduce the flying pattern of this type of flying machine.

We only need basic rigid body simulation techniques to simulate a quadrotor. However, we will need more coding since the design of a quadrotor is a little more complex than those models above. We will need four actuators, an altitude controller, an attitude controller and a horizontal controller. Finally, Euler time integration similar to those in the sections above can help to update the information.

Rigid body simulation: Here I have learned 2D and 3D rigid body simulation. The crucial thing is to update the position, velocity, orientation and angular velocity of a particle in the next time frame using the information in this time frame. In 3D problem we need to replace the vector with the corresponding matrix.

Actuators: four actuators provided the flying machine with force to keep its balance. The direction of the rotation of rotors are designed to ensure that the rotor can remain unmoved in default mode.

Altitude controller: It enables the quadrotor to go up by adding to the force of the four engines and enhance the force of the four rotors.

Attitude controller: It allows the quadrotor to move towards some x points or y points.

Horizontal controller: It helps to stabilize the quadrotor when it is given a horizontal noise (possibly from wind). Without the horizontal controller the quadrotor may fail to remain stationary given a horizontal push.

Results of Rigid Body and Quadrotors is shown below:

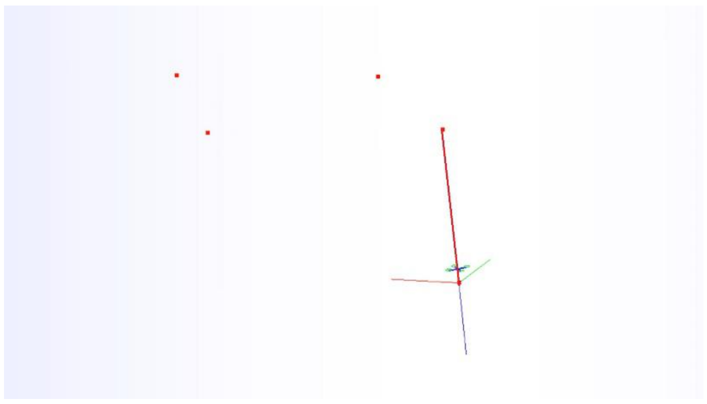

Figure 6.1: Rigid Body and Quadrotors. The Quadrotor is with blue skeletons and four green rotors. Red points show four destinations. Red lines indicate the direction.

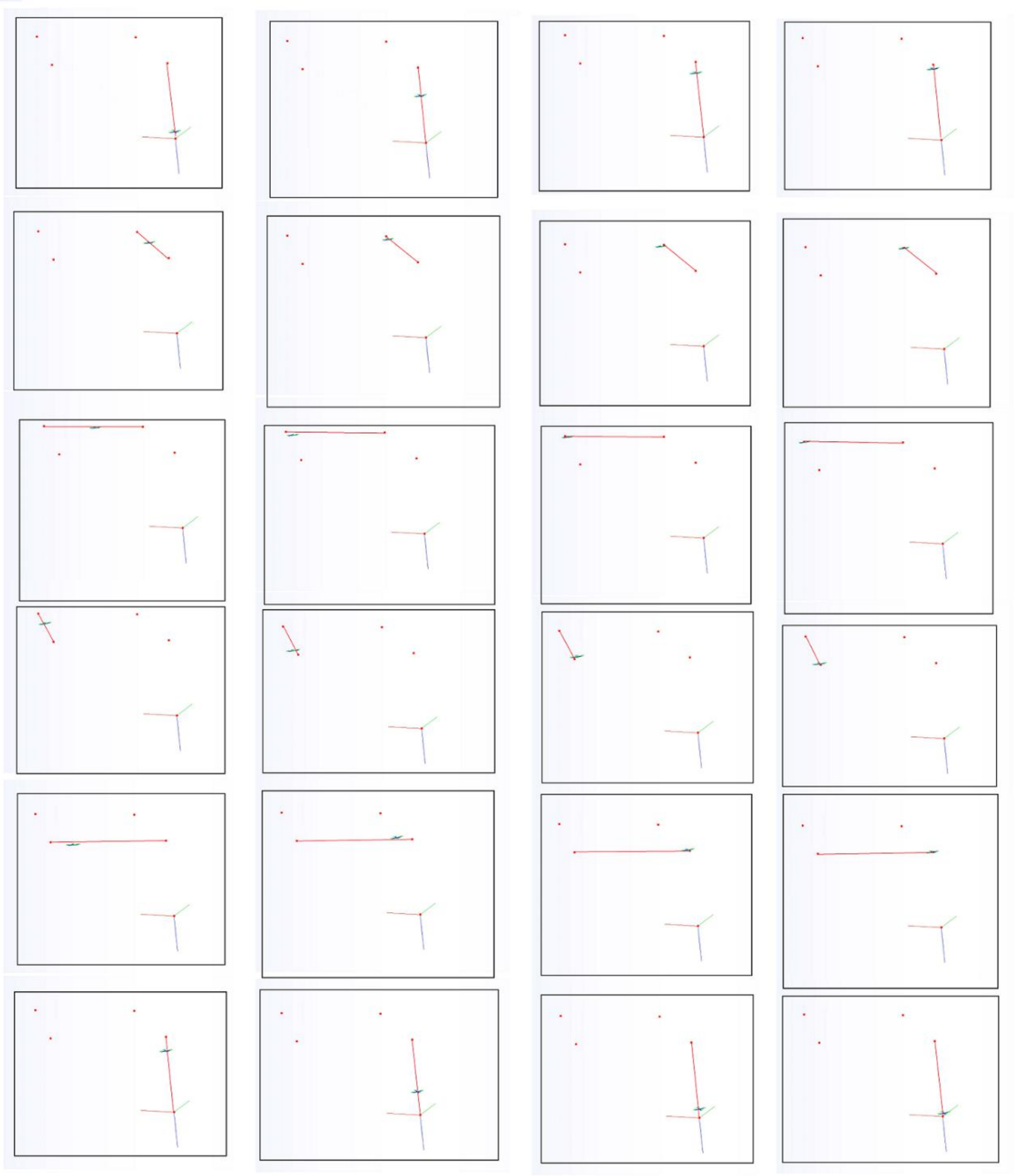

Figure 6.2: The Quadrotor fly toward four red destinations one by one, and finally fly back to the origin.

Reflection on the experience

The reflection here is less about values than those in my first and second rotation report, but more about detailed experience, feelings and small ideas and thoughts.

1. Last Summer in Jing Yuan

Last Summer in Jing Yuan 5th courtyard, I met Dong Siyan by chance when I self-studied there. At that time, he was talking about hash table over phone, auditioning a graduate student. I was attracted and felt curious about which field they are in and what Professor Chen Baoquan is in. (Anyway, the head of Center on Frontiers of Computing Studies in Peking University.)

We talked for nearly an hour that day. He asked me what I was interested in, and I told him that I wanted to simulate a cell, or a smaller-scaled structure, utilizing the power of computers. He indicates that Professor Chen's team may provide me with a chance. He said that he was doing 3D modeling, and mention ray tracing, DLSS, as far as I can remember. And he asked me a question:

How will I simulate the light and produce a picture we see in games?

The answer is that the lights are approximately computed, not from the origin, but from the players' eyes. Dong also told me that they were designing algorithms that can rebuild the skeleton of Jing Yuan 5th courtyard. From my point of view at that time, the algorithm of rebuilding a molecule, a cell or a house can be similar in their idea. Therefore, from that time I had an idea that I would step into this team one day and experience it by myself. Dong Siyan invited me to do a project last winter. Yet it was a pity that I didn't got enough time to do that. Well, too little time is always a pity for me.

2. The team and teammates

When I entered Professor Chen's team, Jiang Hongda contacted me over phone for a brief introduction, and I invited another student in the same rotation together with us. Jiang introduced us the basic concepts in the Computer Graphics and the various directions here in the group. Following is my brief summarization about them. After knowing about these directions, I began to realize that what I wanted to do (to simulate a cell) is more related to physical simulation. (It seems that Raytrace by Dong Siyan cannot fully solve my problem. The instinct of materials and light differ a lot. Yet the technique may still be useful, since the idea of the simulation and approximation is similar, and we cannot ignore light in simulating real world anyway.) I then contact Senior Ni Xingyu in physical simulation direction, and contacted with Professor Chen Baoquan for more advice. My seniors also advised me that I should choose my direction carefully, or I would waste the time and the chance.

Note: Assuming that only Professor Chen and members in the team would read this report, I am here stating something inside this group. You are highly recommended to skip this section if you are not a member of Professor Chen's group or is not interested in it (as it is a little hard to understand).

Computer Vision: It is about the detection of objects like cats, dogs and so on. In this group we take more interest in differentiate objects in three-dimensional space, including 3D point cloud.

Physical simulation: It is about the simulation of smoke, snow, fire, fluids, and rigid body. Senior Ni Xingyu is now in this field, interested in magnetic fluid. Sometimes rendering is also taken into consideration.

Animation and rendering: Professor Caffer is now researching is this field. The problem in this field is to transfer one person's movement of skeleton to another person without 3D reconstruction (only using 2D images as inputs and passing them through some certain neural networks).

3D reconstruction and robot collaboration: 3D reconstruction has a long history, with a series of perfect algorithms to learn. The research in this field requires enough resources and lots of time. This is the field that Dong Siyan is in. He is now interested in camera relocation--to relocate the position of the camera given a 3D reconstruction and a photo shoot by the camera. Robot collaboration is about to use several robots to rebuild 3D modeling of a place.

Geometry: Wu Rundi is in this field now. It's about 3D generation of objects. He is interested in finding an implicit expression of geometric objects using neural networks like GAN. With this technique he is able to reproduce the splicing process of a chair.

Movies: This is what Jiang Honda is doing. One problem in this field is to extract the movie recording techniques used by human photographers and video makers. Another problem is to acquire the location of a camera by synthesizing the recorded video. This can be useful in the broadcast of sports events.

There are also other fields like visualization in this team, but will not be stated in detail in this report.

3. The field of simulation

In one of our past rotation reports, I clearly remember that one student was asked this question: What is the difficulty in the field? At that time, I was stocked to some extent. I quickly realized I didn't know about the difficulty of this field. How is physical simulation difficult? How can the algorithms nowadays perform? This is actually the core of this rotation for me. I want to know the advances of physical simulation today, so that I can assess the accessibility of simulating a cell.

Therefore, on one of our meetings I asked these questions. Since I am not good at memorizing things, I chose to take notes. The difficulty in this field, physical simulation, is composed of three parts: speed, modeling and controllability.

First, the speed of simulation is much slower than the speed of rendering, not to say the living changes of a cell. In fact, video games (as far as I know) requires at least 30 or 60 fps to be fluent (using PLA-Based Methods), but the simulation is rough; we can simulate with higher accuracy (at least middle, meaning the accuracy that can be published in a paper), but we can only produce 4 to 5 fps with a simulation as long as one hour.

Second, the phenomena that can be simulated is rather limited, and many objects in our everyday life still cannot be simulated. When solids and fluids come together, problems also occur.

Third, even if we have many formulas in mathematics and physics, we cannot apply them directly to our model, or else we may fail to get a convergent result. The controllability of the tools is rather important, as can be seen in my experiment, that even small edition may lead to collapse.

I also questioned if scientists have used GPU to speed up their codes. The answer is no. My seniors said that many algorithms in this field cannot run in parallel. In some cases, the fetch of data is so slow that it offset the benefits of GPU speeding up. It might be a pity for me since in another rotation team there's a project about pushing Neuron on CPU now to GPU. I told them that they should merge multi-scale simulation. But it seemed that the simulation of particles cannot be moved to GPU nowadays, at least not directly.

But it's also a chance: nobody has done that successfully before. And from my view, paralleling is the only way to go. (GPU is only a small component of parallel computing.) Who knows?

4. Pity

It's a great pity that I have no time to further complete a brand-new project. It's all due to me myself: I started the rotation too late, and failed to arrange my time well towards the final exams. I know that some students in this direction have done more than me. Also, although I don't know why, the number of students who chose this direction is so large. Maybe it is due to the popularity of physical simulation--according to one senior, the best topic in Graphics nowadays is still simulation (But I didn't know about that before). Anyway, I have known about something I wanted to know. The rotation turn is worthy, and that's enough.

The codes provided by the course only include a basic simulation. As a result, I cannot get a gif or a list of images directly from the codes. In order to visualize the results better, I recorded the whole video (using OBS Studio), cropped it and transferred it into jpeg (using Adobe Premiere Pro), and assembled the images (using Microsoft Paint). Vivid results are essential in papers, and it will be better if I could have done some coding to visualize the results automatically.

The adjustment of parameters is similar. I tried to adjust some parameters in the codes as much as possible. But I did not know why these parameters are set to those certain values. And I also didn't know whether better parameters exist. Another problem is that in simulation there isn't a standard for 'better'. We consider the results as reasonable when they seem similar to what we see in the real world. But it's hard to give a criterion except that we can do some kind of detection in the real world, translate them into data, and compare them with the simulation results. As far as I can imagine, we can use whether the model converges as a criterion. If that is valid, then I am interested in the boundary condition since it may bring us better understanding of the failure of convergence.

Here is another small idea. It is the prediction of noise in the Quadrotor model. As I can see in the results, the Quadrotor move towards the destination point quickly at the beginning, but fails to adjust to the noise quickly when it approaches the point. This is also true when we operate a flying machine in the real world. Can we make prediction of the noise, so that the adjustment can be speeded up? In real world we humans may not keep the balance of the machine only because we cannot respond very quickly. Yet computers can do that. If we regard the flying machine as a player, the noise as the opponent, this problem can be modelled into an online learning problem. Maybe we can use related methods or perhaps neural networks to solve it.

It's also a pity that I cannot touch any 'real things' in the team now. I heard that the equipment for 3D modelling is high-end, and I am always kinds of interested in watching machines work. But this is not the problem for the third rotation turn. It is for all the semester. The Coronavirus simply brings all things away. Hopefully human beings will one day recover from today, and come back to the normal life in the foreseeable future.