HW1 REPORT

110511010 楊育陞

1 Introduction

In this homework, we implement particle systems with different integrators and parameters. Using these particle systems, we simulate the motion of cloth and collision with a sphere. Furthermore, by comparing the results of different integrators and parameters, we analyze the effect of them on the simulation. Finally, I implement a bonus feature to constrain some particles of the cloth.

2 Fundamentals

2.1 Structure of Cloth in Particle System

The structure of cloth is a 2D grid of particles, with each particle connected to its neighbors by springs. The springs are divided into three types:

1. **Structural Springs**: connect the particle to its neighbors in the same row and column.
2. **Shear Springs**: connect the particle to its diagonal neighbors.
3. **Bend Springs**: connect the particle to its neighbors in the same row and column with a distance of 2.

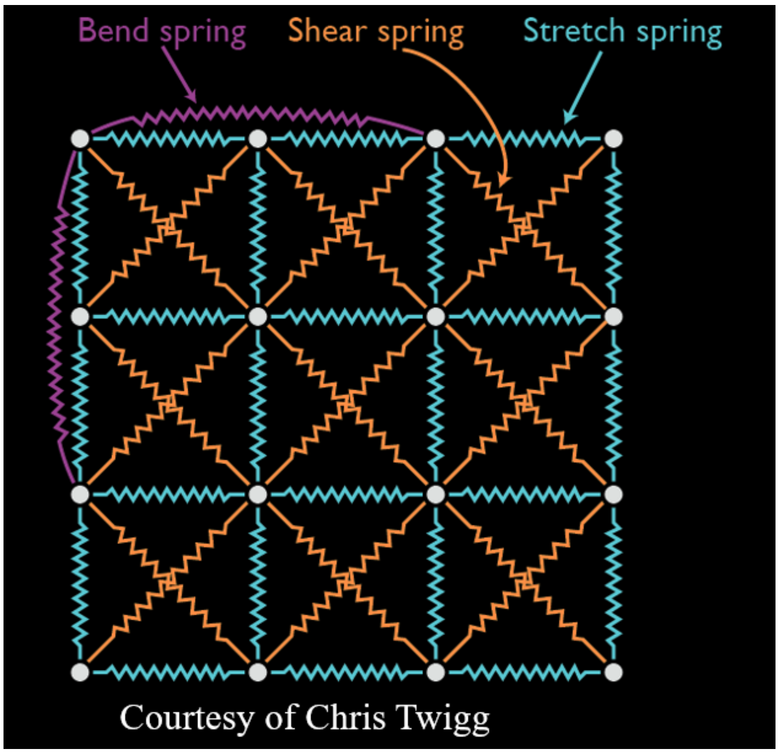


Figure 1: structure of cloth in particle system

2.2 Structure of Cloth in Particle System

There are two types of internal forces in the cloth:

1. **Spring Force**: the force exerted by the springs connecting the particles. In this homework, this is calculated by following formula, where k_{spring} is the parameter 'springCoef':

$$F_{spring} = -k_{spring} \cdot (|x_a - x_b| - l_0) \cdot \frac{x_a - x_b}{|x_a - x_b|}$$

2. **Damping Force**: the force that resists the motion of the particles. In this homework, this is calculated by the following formula, where $k_{damping}$ is the parameter 'damperCoef':

$$F_{damping} = -k_{damping} \cdot \left(\frac{(v_a - v_b) \cdot (x_a - x_b)}{|x_a - x_b|} \right) \frac{x_a - x_b}{|x_a - x_b|}$$

2.3 Collision

In this homework, the collision is detected by the distance between the particle and the sphere, also the velocity of the particle. If collision happends, the particle will be applied with a collision force, which is calculated by the following formula:

$$v_a = \frac{m_a u_a + m_b u_b - m_b C_R (u_a - u_b)}{m_a + m_b}$$

We only consider the normal direction of the collision, also we assume the sphere is fixed.

2.4 Integrators

1. **Explicit Euler**: The simplest integrator, which is easy to implement but not stable for large time steps.

2. **Implicit Euler**: a more stable integrator than explicit Euler, but requires simulating a linear system in each time step.

3. **Midpoint Method**: similar to Euler's method, but uses the average of the current velocity and the velocity at the midpoint of the time step.

4. **Runge-Kutta 4th Order**: the most accurate integrator among the four, but also the most computationally expensive.

$$\begin{aligned}
k_1 &= h \cdot f(t, y) \\
k_2 &= h \cdot f\left(t + \frac{h}{2}, y + \frac{k_1}{2}\right) \\
k_3 &= h \cdot f\left(t + \frac{h}{2}, y + \frac{k_2}{2}\right) \\
k_4 &= h \cdot f(t + h, y + k_3) \\
y(t + h) &= y(t) + \frac{k_1 + 2k_2 + 2k_3 + k_4}{6}
\end{aligned}$$

3 Implementation

3.1 Construct the connections of springs

To construct the connections of each spring, I find the index of the particles that the spring connects to, and calculate the length of the spring, then store them in the '_springs' vector with type of spring. I handle edge cases by only iterating the particles that are not on the edge of the cloth.

3.2 Compute the internal forces

To compute the internal forces, I iterate all the springs and calculate the spring force and damping force by the formulas mentioned above. Then I calculate the acceleration caused by the forces and accumulate them to the particles.

3.3 Handle the collision

To handle the collision, I iterate all the particles and calculate the distance between the particle and the sphere. If the distance is less than the radius of the sphere by a correction value, the particle is in collision. Then it will first be moved out of the sphere, and then be applied with the collision force normal to the sphere. I use 0.01 as the correction value.

3.4 Implement the integrators

1. **Explicit Euler**: The position and velocity of the particles are updated by current integration of velocity and acceleration.

```
for (int i = 0; i < particles.size(); i++) {
    particles[i]->velocity() += particles[i]->acceleration() * deltaTime;
    particles[i]->position() += particles[i]->velocity() * deltaTime;
}
```

2. **Implicit Euler**: The position and velocity of the particles are updated by the next integration of velocity and acceleration. I back up the original position and velocity first, then update the position and velocity th get the next state. Finally, I restore the original position and velocity and update with the next state.

```
std::vector<Eigen::Matrix4Xf> oriPositions, oriVelocitys;
for (int i = 0; i < particles.size(); i++) {
    // Back up the original state
    oriPositions.push_back(particles[i]->position());
    oriVelocitys.push_back(particles[i]->velocity());
    // To next state
    particles[i]->position() += particles[i]->velocity() * deltaTime;
    particles[i]->velocity() += particles[i]->acceleration() * deltaTime;
}
simulateOneStep(); // Get the next state
// Restore the original state and update with the next state
for (int i = 0; i < particles.size(); i++) {
    particles[i]->position() = oriPositions[i] + deltaTime * particles[i]->velocity();
    particles[i]->velocity() = oriVelocitys[i] + deltaTime * particles[i]->acceleration();
}
```

3. **Midpoint Method**: The position and velocity of the particles are updated by the average of the current state and the next state. This is similar to the implicit Euler, but the next state is calculated by the average of the current state and the next state.

```
std::vector<Eigen::Matrix4Xf> oriPositions, oriVelocitys;
for (int i = 0; i < particles.size(); i++) {
    // Back up the original state
    oriPositions.push_back(particles[i]->position());
    oriVelocitys.push_back(particles[i]->velocity());
    // To the midpoint
    particles[i]->position() += particles[i]->velocity() * deltaTime * 0.5f;
    particles[i]->velocity() += particles[i]->acceleration() * deltaTime * 0.5f;
}
simulateOneStep(); // Get the next state at the midpoint
// Restore the original state and update with the next state
for (int i = 0; i < particles.size(); i++) {
    particles[i]->position() = oriPositions[i] + deltaTime * particles[i]->velocity();
    particles[i]->velocity() = oriVelocitys[i] + deltaTime * particles[i]->acceleration();
}
```

4. **Runge-Kutta 4th Order**: The position and velocity of the particles are updated by the weighted average of the four states. The four states are calculated by the weighted average of the current state and the next state.

```
std::vector<Eigen::Matrix4Xf> oriPositions, oriVelocitys, k1, k2, k3, k4, l1, l2, l3, l4;
for (int i = 0; i < particles.size(); i++) {
  oriPositions.push_back(particles[i]->position());
  oriVelocitys.push_back(particles[i]->velocity());
  k1.push_back(particles[i]->velocity() * deltaTime);
  l1.push_back(particles[i]->acceleration() * deltaTime);
  particles[i]->position() = oriPositions[i] + k1[i] * 0.5f;
  particles[i]->velocity() = oriVelocitys[i] + l1[i] * 0.5f;
}
simulateOneStep();
for (int i = 0; i < particles.size(); i++) {
  k2.push_back(particles[i]->velocity() * deltaTime);
  l2.push_back(particles[i]->acceleration() * deltaTime);
  particles[i]->position() = oriPositions[i] + k2[i] * 0.5f;
  particles[i]->velocity() = oriVelocitys[i] + l2[i] * 0.5f;
}
simulateOneStep();
for (int i = 0; i < particles.size(); i++) {
  k3.push_back(particles[i]->velocity() * deltaTime);
  l3.push_back(particles[i]->acceleration() * deltaTime);
  particles[i]->position() = oriPositions[i] + k3[i];
  particles[i]->velocity() = oriVelocitys[i] + l3[i];
}
simulateOneStep();
for (int i = 0; i < particles.size(); i++) {
  k4.push_back(particles[i]->velocity() * deltaTime);
  l4.push_back(particles[i]->acceleration() * deltaTime);
  particles[i]->position() = oriPositions[i] + (k1[i] + 2 * k2[i] + 2 * k3[i] + k4[i]) / 6.0f;
  particles[i]->velocity() = oriVelocitys[i] + (l1[i] + 2 * l2[i] + 2 * l3[i] + l4[i]) / 6.0f;
}
```

4 Result and Discussion

4.1 The difference between integrators

Through experiments among the four integrators, the time steps at which the simulation becomes unstable and the fps on my laptop are as follows:

Table 1: Results of the integrators

Integrator	Unstable Time Step	FPS
Explicit Euler	0.001	144
Implicit Euler	0.0006	144
Midpoint Method	0.0011	144
Runge-Kutta 4th	0.0015	77

The results show that the Runge-Kutta 4th Order integrator is the most accurate among the four, but also the most computationally expensive. The Explicit Euler integrator is the simplest but not stable for large time steps. The Implicit Euler integrator is unstable at relatively small time steps in my experiment, which is different from the theory. Observing that the Midpoint Method integrator looks more stable than Explicit Euler Method, I think the Implicit Euler integrator may have some problems when getting the next state.

4.2 Effect of parameters

1. **springCoef**: The springCoef parameter controls the stiffness of the springs.

   When the springCoef is small, the cloth is more flexible and the simulation is more stable.

   When the springCoef is large, the cloth is more rigid and the simulation is less stable.

2. **damperCoef**: The damperCoef parameter controls the damping of the springs.

   When the damperCoef is small, the cloth oscillates more and the simulation is less stable.

   When the damperCoef is large, the cloth oscillates less and the simulation is more stable.

5 Bonus

5.1 Introduction

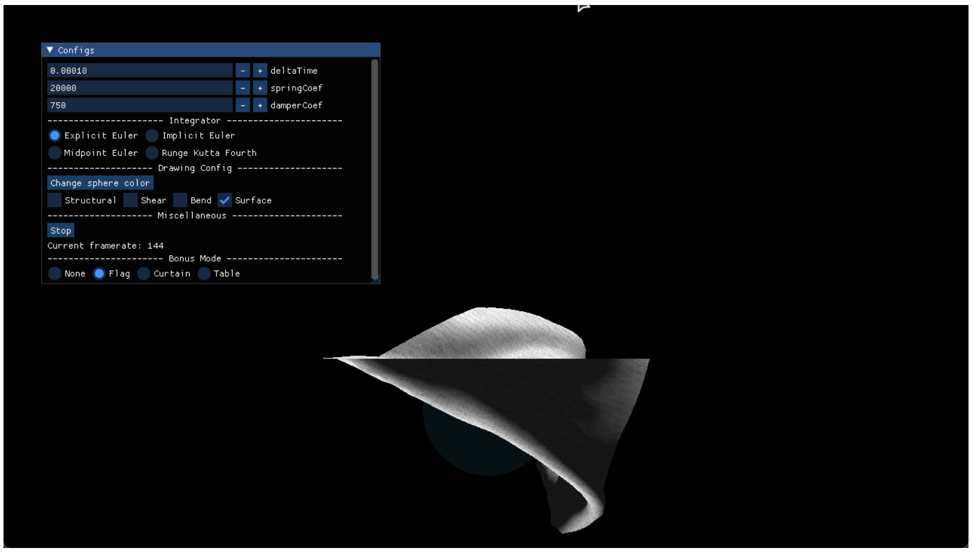
For the bonus feature, I implement constraints on some particles of the cloth. There are three bonus features:

1. **Flag**: constrain the front edge of the cloth to the initial position.
2. **Curtain**: constrain the front edge of the cloth to **x-axis**.
3. **Table**: constrain particles inside a circle of cloth to **x-z plane**.

5.2 Implementation

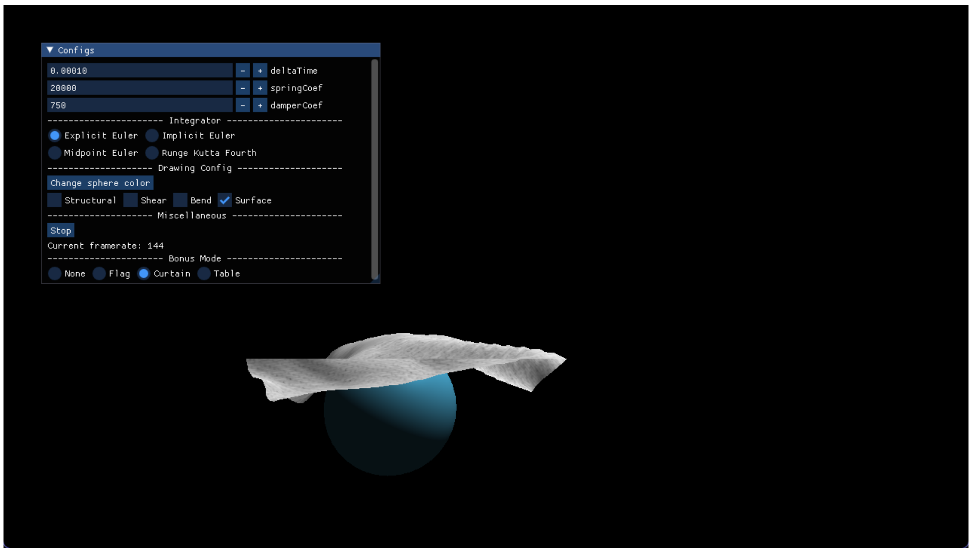
To implement the bonus feature, I add a new element ‘constraint‘ to the particle class. The constraint is a 4x1 vector, corresponding to the axises. When the constraint is all one, the particle is free. When some elements of the constraint are zero, the particle is constrained to the corresponding axis. I also modify the function ‘computeExternalForces‘ and ‘computeSpringForces‘ to handle the constraints. In these functions, the acceleration and velocity of the particles are set to zero if the corresponding element of the constraint is zero. To demonstrate the bonus feature, I add a new switch on the gui panel to select the bonus feature.

5.3 Demo



```
▼ Configs
0.00010   - + deltaTime
20000     - + springCoef
750       - + damperCoef
--------------------- Integrator ---------------------
[✓ Explicit Euler] Implicit Euler
Midpoint Euler     Runge Kutta Fourth
------------------- Drawing Config -------------------
Change sphere color
Structural  Shear  Bend  [✓ Surface]
------------------- Miscellaneous --------------------
Stop
Current framerate: 144
--------------------- Bonus Mode ---------------------
None  [✓ Flag]  Curtain  Table
```

Figure 2: Flag



```
▼ Configs
0.00010   - + deltaTime
20000     - + springCoef
750       - + damperCoef
--------------------- Integrator ---------------------
[✓ Explicit Euler] Implicit Euler
Midpoint Euler     Runge Kutta Fourth
------------------- Drawing Config -------------------
Change sphere color
Structural  Shear  Bend  [✓ Surface]
------------------- Miscellaneous --------------------
Stop
Current framerate: 144
--------------------- Bonus Mode ---------------------
None  Flag  [✓ Curtain]  Table
```

Figure 3: Curtain

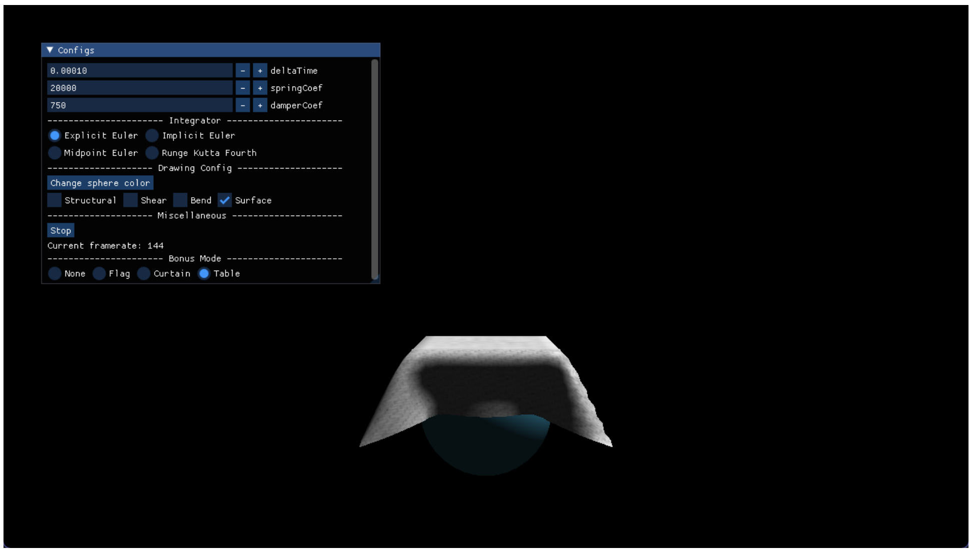

```
▼ Configs
0.00010  - + deltaTime
20000    - + springCoef
750      - + damperCoef
---------------------- Integrator ----------------------
[✓ Explicit Euler]  Implicit Euler
Midpoint Euler  Runge Kutta Fourth
-------------------- Drawing Config --------------------
Change sphere color
Structural  Shear  Bend  [✓ Surface]
-------------------- Miscellaneous ---------------------
Stop
Current framerate: 144
---------------------- Bonus Mode ----------------------
None  Flag  Curtain  [✓ Table]
```

Figure 4: Table: not moving

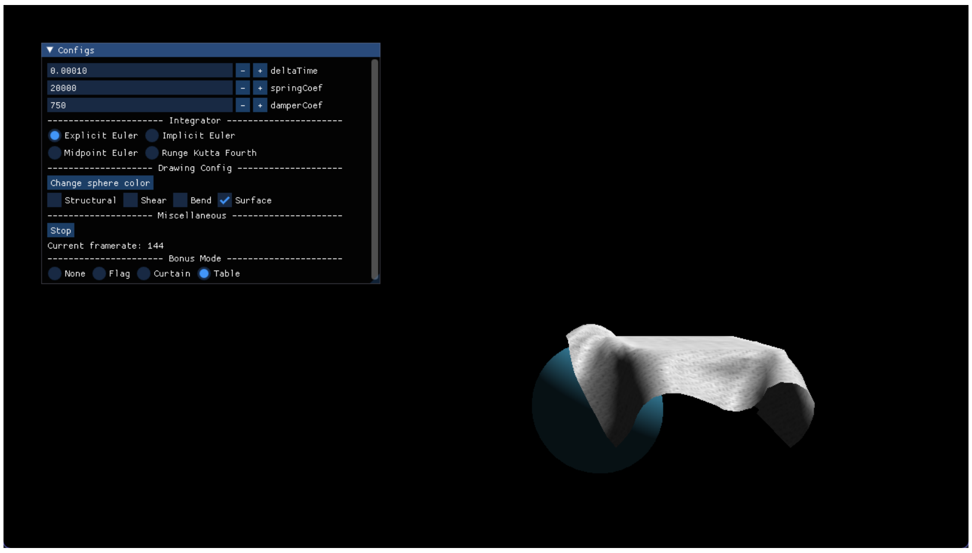

```
▼ Configs
0.00010  - + deltaTime
20000    - + springCoef
750      - + damperCoef
---------------------- Integrator ----------------------
[✓ Explicit Euler]  Implicit Euler
Midpoint Euler  Runge Kutta Fourth
-------------------- Drawing Config --------------------
Change sphere color
Structural  Shear  Bend  [✓ Surface]
-------------------- Miscellaneous ---------------------
Stop
Current framerate: 144
---------------------- Bonus Mode ----------------------
None  Flag  Curtain  [✓ Table]
```

Figure 5: Table: moving

6 Conclusion

In this homework, I implement particle systems with different integrators and parameters. By simulating the motion of cloth and collision with a sphere, I analyze the effect of integrators and parameters on the simulation. Finally, I implement a bonus feature to constrain some particles of the cloth. Through this homework, I have a better understanding of the particle system and the effect of integrators and parameters on the simulation. Though in the real work, the simulation may be more complex and need not do the simulation from scratch, the basic concepts and methods are still useful. Also, I have lots of fun doing this homework.