Assignment 5:
Animation with Cloth Simulation

NAME: HIDDEN
STUDENT NUMBER: HIDDEN
EMAIL: HIDDEN@SHANGHAITECH.EDU.CN

1 INTRODUCTION

In this project, a physically-based animation of cloth simulation is implemented. The following tasks are done.

- Force computation with Hooke's law
- Structural, shear, and bending springs
- Wind simulation
- Obstacle collision handling
- Drag a mesh point to move the cloth with mouse in real-time

2 IMPLEMENTATION DETAILS

To simulate cloth movement, we divide a cloth into a number of masses, each have its own weight. The masses are connected together by virtual 'springs', and can have a force on each other.

2.1 Transformation between world and local coordinate

To transform a vector from local coordinate to world coordinate, we need to multiply position by the model matrix.

To transform a vector from world coordinate to local coordinate, we need to compute the inverse of the model matrix, and multiply position by it.

2.2 Force computation using Hooke's Law

We can compute the force between masses by Hooke's law, $F = kx$.

The force is given by

$$F = kx = k \cdot (L_0 - \|p - q\|) \cdot \frac{p - q}{\|p - q\|}$$

where k is a fixed number, p and q are two masses, L_0 is the zero-force length (distance less than this length will not cause force), and $\frac{p-q}{\|p-q\|}$ is the normalized direction of the force.

2.3 Structural, Shear and bending springs

Now we know how to compute forces between two masses, but we haven't implemented how masses are connected and how they can have effect on each other. We need to implement the springs.

There are three types of springs, namely **structural**, **shear** and **bend**.

- **Structural springs** connect two masses that are adjacent to each other.
  For example, mass at position $[i, j]$ is connected to mass at position $[i + 1, j]$, $[i - 1, j]$, $[i, j + 1]$, $[i, j - 1]$.
- **Shear springs** connect two masses that are diagonal to each other.
  For example, mass at position $[i, j]$ is connected to mass at position $[i + 1, j + 1]$, $[i - 1, j + 1]$, $[i + 1, j - 1]$, $[i - 1, j - 1]$.
- **Bending springs** connect two masses that are two steps away from each other.
  For example, mass at position $[i, j]$ is connected to mass at position $[i + 2, j]$, $[i - 2, j]$, $[i, j + 2]$, $[i, j - 2]$.

An intuitive description of the springs is shown in the following figure.

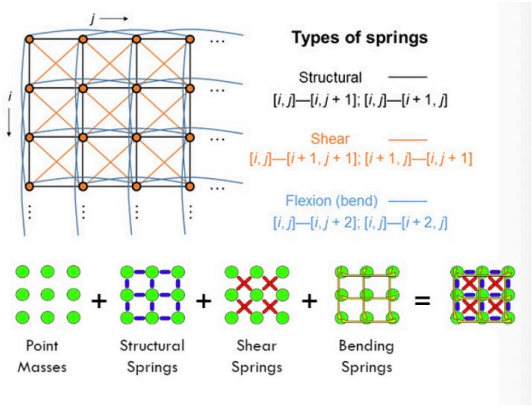

Fig. 1. Structural, shear and bending springs

2.4 Wind simulation

To simulate wind, we need to apply an additional external force to the masses. Randomness is added to the wind force to make it more realistic. The cloth will be blown continously by the wind in the direction of the wind force, instead of converging to a fixed surface position.

The pseudocode is given below.

Listing 1. Wind simulation

```
void computeAcceleration(){
  // randomly generate wind force
  Vec3 wind = {0, 0, random_number / 50};

  // compute acceleration for each mass
  for (int i = 0; i < massCount; i++) {
    if (mass is pinned) continue;
    // compute force
    // ...
```

```
    // compute acceleration
    acceleration[i] = force[i] / weight;
    // add wind force
    acceleration[i] += wind / mass[i];
  }
}
```

2.5 Sphere (obstacle) collision handling

To simulate the collision between the cloth and a sphere, we need to compute the distance between the sphere and the cloth. If the distance is less than the radius of the sphere, we need to remove the acceleration and velocity of the masses along the direction of the normal of the sphere. Here the normal of the sphere is given by the vector from the center of the sphere to the mass.

The pseudocode is given below.

Listing 2. Sphere collision handling

```
vec3 distance = mass_pos - sphere_center;
if (length(distance) <= sphere_radius) {
  vec3 normal = normalize(distance);
  // update position
  mass_pos = sphere_center + normal *
      sphere_radius;
  // add 'velocity along normal'
  // to remove velocity along sphere center
  world_velocities[i] += glm::dot(-
      world_velocities[i], normal) * normal;
}
```

2.6 Drag a mesh point to move the cloth with mouse in real-time

To handle mouse drag event, we need to do the following steps.

(1) When a mouse left click is detected, we need to find out which mass the mouse is pointing at.
We accomplish this by getting the mouse position, and transform it from the screen coordinate to the world space using `glm::unProject` function. The function takes in 4 arguments: the mouse position, the camera LookAt matrix, the camera projection matrix, and the viewport. The viewport is a 4-element array, which is the position of the lower-left corner of the viewport, and the width and height of the viewport. The function returns the world space position of the mouse.

```
// transform mouse position to world
    space
vec3 world_mouse_position =
glm::unProject(mouse_pos, LookAtMat(),
    PerspectiveMat(), Vec4{0, 0,
    screen_width, screen_height});
```

Then we generate a ray from the camera origin to the world position we have just calculated. For each mass center, we assume it is a small sphere and do ray-sphere intersection. If the ray intersects with the sphere, we have found the mass we are looking for. We traverse each mass and find the closest one to the ray. We then set its acceleration to zero and fix its position. The pseudocode is given below.

```
// ray origin and ray direction
vec3 ray_o = camera_pos;
vec3 ray_d = glm::normalize(
    world_mouse_position - camera_pos);
```

(2) When a mouse left click is maintained, it means the user is dragging the mass. We move the mass to the mouse position, and set its acceleration to zero. We also need to fix the position of the mass, so that it won't be moved by the cloth simulation.
To accomplish this, similar to the previous step, we need to transform the mouse position from the screen coordinate to the world space. Then, to find out the new world position of the mass, we need to do ray-plane intersection. We assume the plane is parallel to the camera plane, and the normal of the plane is the camera direction. The plane passes through the mass. We then get the new world position of the mass. The pseudocode is given below.

```
// Plane origin and normal
vec3 normal = camera->Forward();
vec3 x0 = mass_pos[drag_index];
// Do ray-plane intersection and get
// solution 't' of the equation
float t = (dot(normal,x0) - dot(normal,
    camera_pos)) / dot(normal, ray_dir);
vec3 new_pos = ray_o + t * ray_d;
```

(3) When a mouse click is released, we simply remove the mass from the fixed mass list, so that it can be moved by the cloth simulation.

3 RESULTS

3.1 Obstacle collision detection

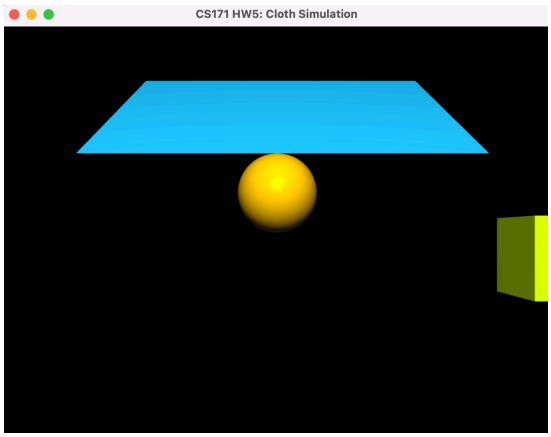

Fig. 2. Initial state

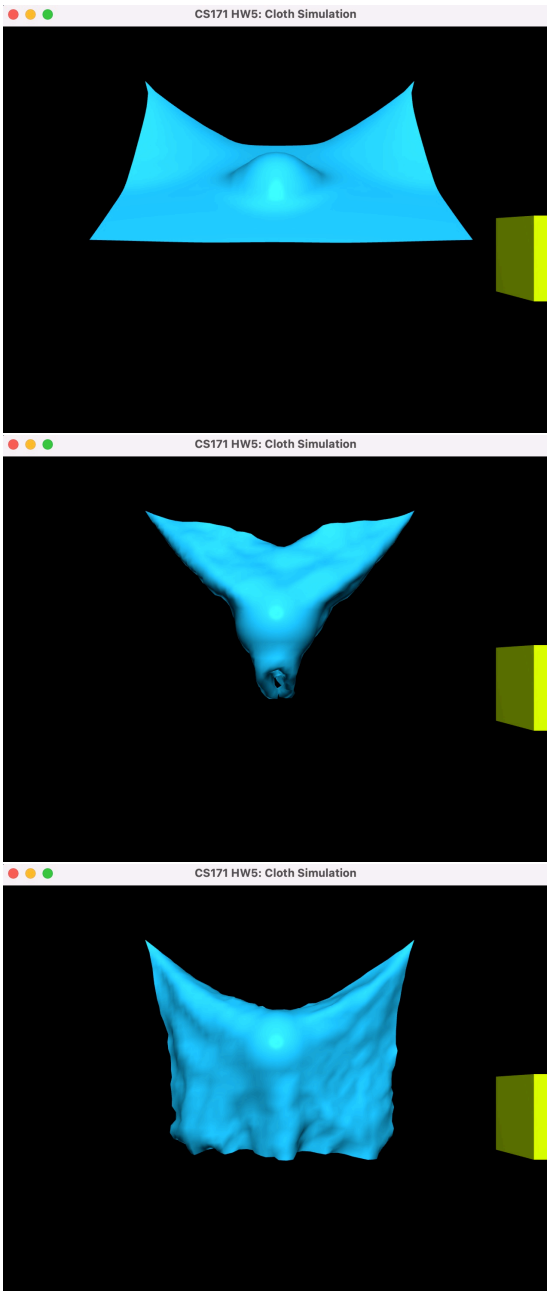

Fig. 3. Falling on a sphere

3.2 Mouse interaction

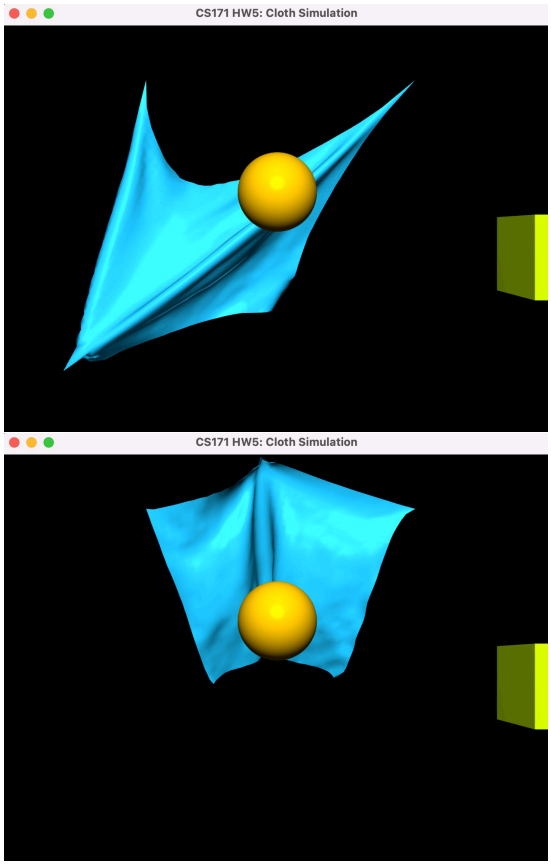

Fig. 4. Drag a mesh point to move the cloth with mouse in real-time

The results will also be shown in real-time to TAs.