Assignment 5:
Animation with Cloth Simulation

NAME: ENTROPY-FIGHTER
STUDENT NUMBER: 2020533
EMAIL: XXXX@SHANGHAITECH.EDU.CN

1 INTRODUCTION

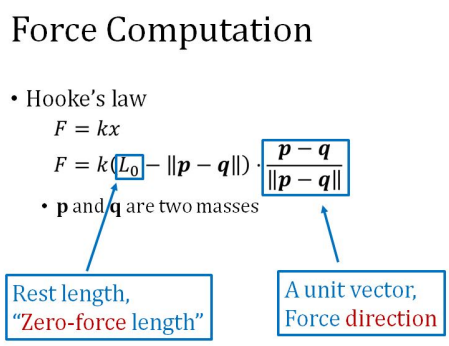
In this homework, I do the following things.

- Force computation with Hooke's law (30 pts)
- Structural, shear, and bending springs (40 pts)
- Fix the location of two mesh points to stop the cloth falling down (10 pts)
- Real-time and stable animation (10 pts)
- (bonus)Apply external forces to the cloth to simulate the behavior of wind (5 pts)
- (bonus)Add a sphere or cube obstacle to simulate a piece of cloth falling on a sphere or a cube with collision handling (10 pts)
- (bonus)Drag a mesh point to move the cloth with mouse in real-time (15 pts)

2 IMPLEMENTATION DETAILS

2.1 Force computation with Hooke's law

This section is implemented in the ComputeHookeForce function in the "cloth.cpp". I write this function based on the Hooke's law. The Hooke's law is clearly shown in the tutorial's PPT below.

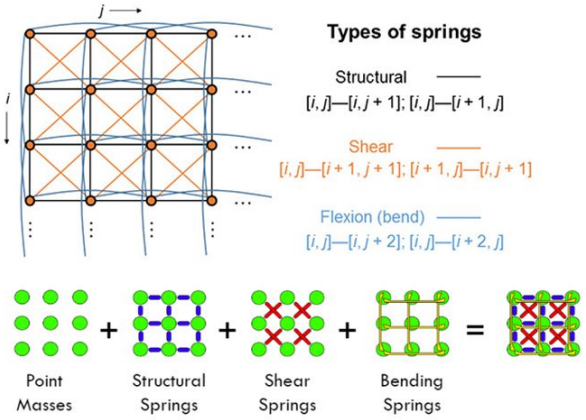


Note that before we use the hooke's law to compute hooke force, we should consider some special cases and set the force to 0, 0, 0 The code about the special cases is listed below.

```
if(iw_this < 0 || iw_this >= mass_dim.x || iw_that
     >= mass_dim.x || ih_this < 0 || ih_this >=
    mass_dim.y || ih_that >= mass_dim.y){
   return {0, 0, 0};
}
```

2.2 Structural, shear, and bending springs

This section is implemented in the ComputeSpringForce function in the "cloth.cpp". Firstly, let us look at the graphical examples to understand what are the structural, shear, and bending springs.



THerefore, when we compute the spring force, we should consider all 3 situations. The code is shown below.

```
Force += ComputeHookeForce(iw, ih, iw - 1, ih,
    dx_local * scale[0]);
Force += ComputeHookeForce(iw, ih, iw, ih - 1,
    dx_local * scale[1]);
Force += ComputeHookeForce(iw, ih, iw + 1, ih,
    dx_local * scale[0]);
Force += ComputeHookeForce(iw, ih, iw, ih + 1,
    dx_local * scale[1]);

Force += ComputeHookeForce(iw, ih, iw - 2, ih, 2 *
     dx_local * scale[0]);
Force += ComputeHookeForce(iw, ih, iw, ih - 2, 2 *
     dx_local * scale[1]);
Force += ComputeHookeForce(iw, ih, iw + 2, ih, 2 *
     dx_local * scale[0]);
Force += ComputeHookeForce(iw, ih, iw, ih + 2, 2 *
     dx_local * scale[1]);

Float hypotenuse = std::sqrt(scale[0] * scale[0] +
     scale[1] * scale[1]);
```

```
Force += ComputeHookeForce(iw, ih, iw - 1, ih - 1,
      dx_local * hypotenuse);
Force += ComputeHookeForce(iw, ih, iw - 1, ih + 1,
      dx_local * hypotenuse);
Force += ComputeHookeForce(iw, ih, iw + 1, ih - 1,
      dx_local * hypotenuse);
Force += ComputeHookeForce(iw, ih, iw + 1, ih + 1,
      dx_local * hypotenuse);
```

2.3 Fix the location of two mesh points to stop the cloth falling down

This part is implemented in the ComputeAccelerations, ComputeVelocities and ComputePositions function in the "cloth.cpp". As for the fixed points, we consider them as special cases, we should set their accelerations, velocities and positions properly. The pesudo code is shown below.

```
if(is_fixed_masses[i]) {
    world_accelerations[i] = Vec3(0.0f, 0.0f, 0.0f)
        ;
    world_velocities[i] = Vec3(0.0f, 0.0f, 0.0f);
    continue;
}
```

2.4 Real-time and stable animation (10 pts)

This section is implemented in the ComputeAccelerations, ComputeVelocities and ComputePositions function in the "cloth.cpp". As for accelerations, the formula is a = F/m - g. The code is shown below.

```
world_accelerations[i] = (ComputeSpringForce(iw,
    ih) - damping_ratio * world_velocities[i]) /
    mass_weight - gravity;
```

As for velocities, the formula is v = v0 + at. The code is shown below.

```
world_velocities[i] += world_accelerations[i] *
    fixed_delta_time;
```

AS for positions, the formula is s = s0 + vt

```
local_or_world_positions[i] += fixed_delta_time *
    world_velocities[i];
```

2.5 (bonus)Apply external forces to the cloth to simulate the behavior of wind

This section is implemented in the fluid function in "cloth.cpp". This function is to add another force to simulate the wind. The core code for constructing such a force is shown below.

```
return Float(0.02) * glm::dot((Vec3(0, 0, 3) - v),
      vertices[i].normal) * vertices[Get1DIndex(iw,
      ih)].normal;
```

Also, when we caculate the accelerations, we should add this new force.

2.6 (bonus)Add a sphere or cube obstacle to simulate a piece of cloth falling on a sphere or a cube with collision handling

In this section, I add a sphere to simulate a piece of cloth falling on a sphere with collision detection. The collision detection section is implemented in the s_intersect function in "cloth.cpp". The key idea is to compare r and the distance. The core code is shown below.

```
if(glm::length(pos - c) >= r)
    return false;
else
    return true;
```

Note that when the cloth falls on the sphere, we should accordingly modify the acceleration, velocity and the position. The pesudo code for such cases is shown below.

```
if(s_intersect(local_or_world_positions[i]) ||
     is_fixed_masses[i]) {
    world_accelerations[i] = Vec3(0.0f, 0.0f, 0.0f)
        ;
    world_velocities[i] = Vec3(0.0f, 0.0f, 0.0f);
    local_or_world_positions[i] = c + r * glm::
        normalize(local_or_world_positions[i] - c)
         + Float(0.04) * glm::normalize(
        local_or_world_positions[i] - c);
    continue;
}
```

The code for position seems a bit long. If we don't add the last term, the position will be on the sphere, which would look like weird.

2.7 (bonus)Drag a mesh point to move the cloth with mouse in real-time

This section is implemented in the "main.cpp". In this section, I add 2 features. If we drag the cloth by left mouse, the whole clothes would move. Otherwise, if we drag the cloth by right mouse, the effect is similar to dragging the cloth by finger. As for the first feature, my key idea is similar to my second homework's bonus(drag the control points). I use two important and useful functions, which are glReadPixels() function and glm::unProject() function. By using this 2 functions, I can get my cursor's world position on the object. When I move my mouse, I modify the all mass's world position to achive the moving effect.

As for the second feature, I use another way. Like homework3, I generate a ray from the camera. The generated ray will intersect the near plane. Then, I use some tricks to find the closest mass. How to change the position of the mass? I caculate the moving distance by similar triangle, then I change the position accordingly.

3 RESULTS

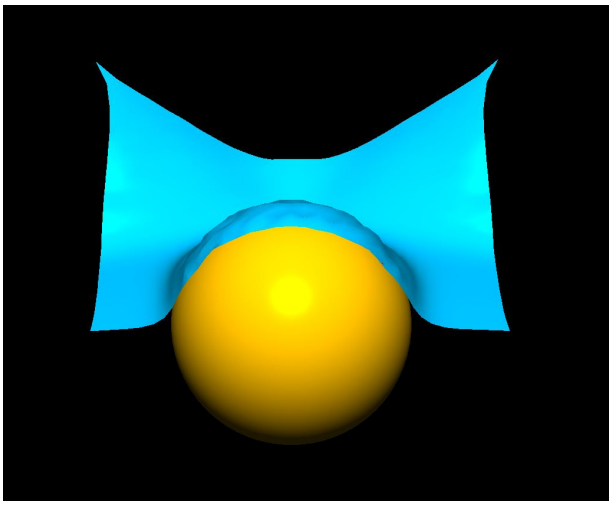


Fig. 1. fall on the sphere

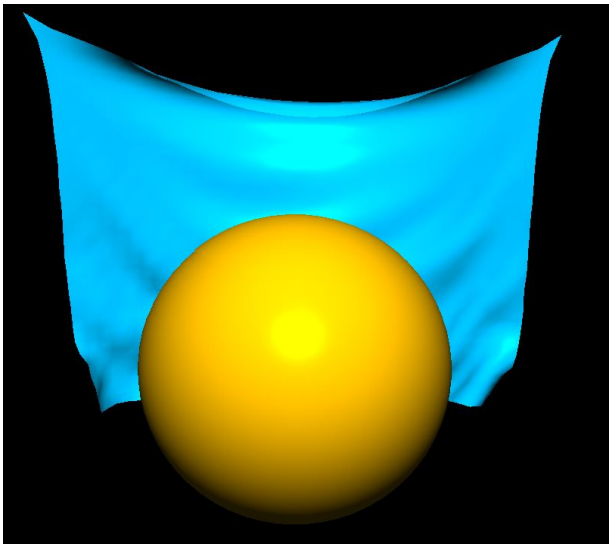


Fig. 2. cloth with wind

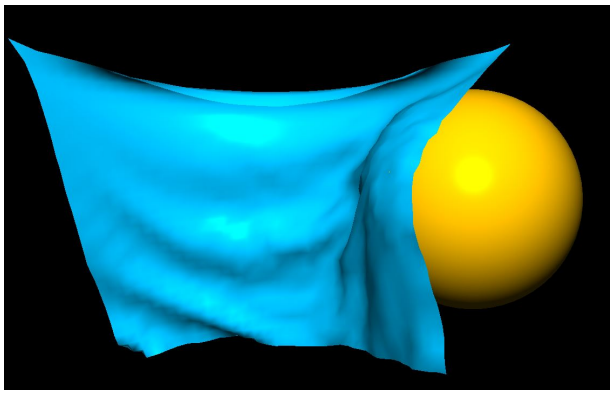


Fig. 3. drag with left mouse(move the whole cloth)

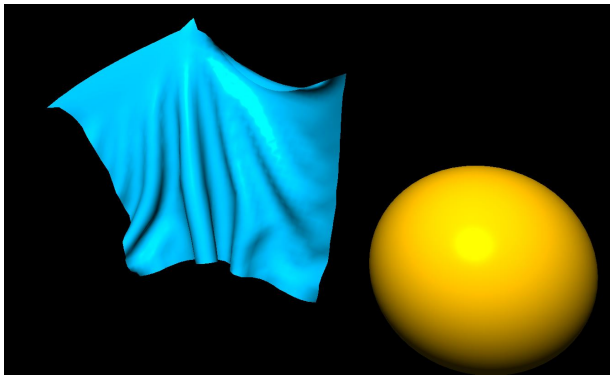


Fig. 4. drag with right mouse(move one point)