Assignment 6: Mass-Spring Simulation

学号：17343124

姓名：伍斌

Assignment 6: Mass-Spring Simulation
- Task 1
  - 代码
  - 实现效果
  - 简述做法
- Task 2
  - 代码
  - 实现效果
  - 简述做法
- Task3
  - 代码
  - 实现效果
- Task4

Task 1

用前面的公式（4）即显式积分法来实现质点弹簧系统的动画仿真。

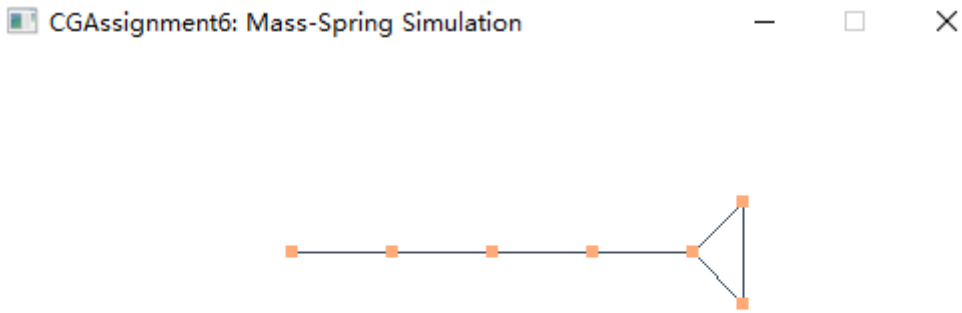
代码

```
void Simulator::simulate()
{
    static constexpr float dt = 0.0001f;
    static constexpr glm::vec2 gravity = glm::vec2(0.0f, -9.8f);
    std::vector<glm::vec2> m_v1;
    // TODO: for each particle i, calculate the force f, and then update the
m_v[i]
    for (unsigned int i = 0; i < m_numParticles; ++i)
    {
        // Gravity force
        glm::vec2 force = gravity * m_particleMass;
        // You should use m_restLength[i][j], m_stiffness, m_x, dt, and
m_particleMass herein
        for (unsigned int j = 0; j < m_numParticles; ++j)
        {
            if (m_restLength[i][j] != 0) {
                glm::vec2 Fij = m_x[i] - m_x[j];
                float dis = sqrt(pow(m_x[j].x - m_x[i].x, 2) + pow(m_x[j].y -
m_x[i].y, 2));
                force += -m_stiffness * (dis - m_restLength[i][j]) *
glm::normalize(Fij);
            }
        }
```

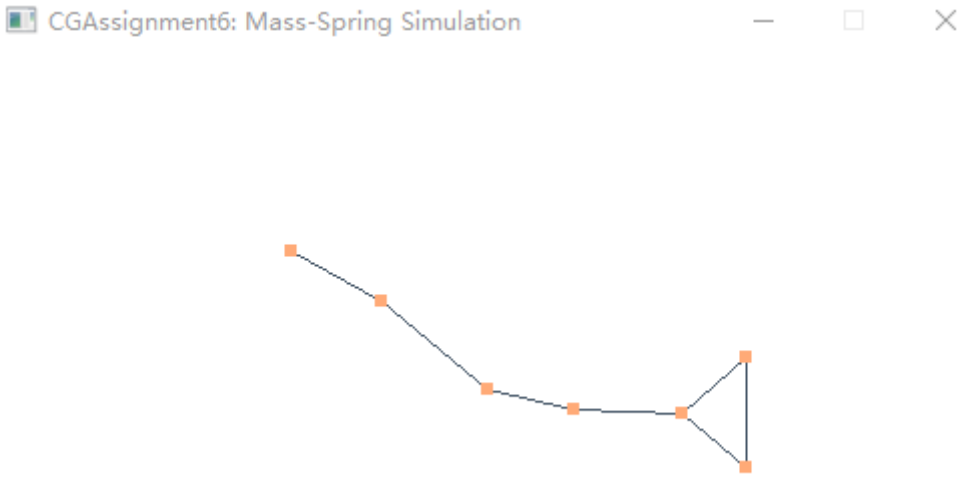
```
        //Update the m_v[i]
        m_v1.push_back(m_v[i]);
        m_v[i] += dt * force / m_particleMass;
    }

    // Collide with the ground
    // Note: no need to modify it
    for (unsigned int i = 0; i < m_numParticles; ++i)
    {
        if (m_x[i].y < 0.0f)
        {
            m_x[i].y = 0.0f;
            m_v[i].y = 0.0f;
        }
    }

    // Todo: update the position m_x[i] using m_v[i]
    // Note: you should use the time step dt
    for (unsigned int i = 1; i < m_numParticles; ++i)
    {
        //Task1：显式法
        m_x[i] += m_v1[i] * dt;
    }
}
```

实现效果

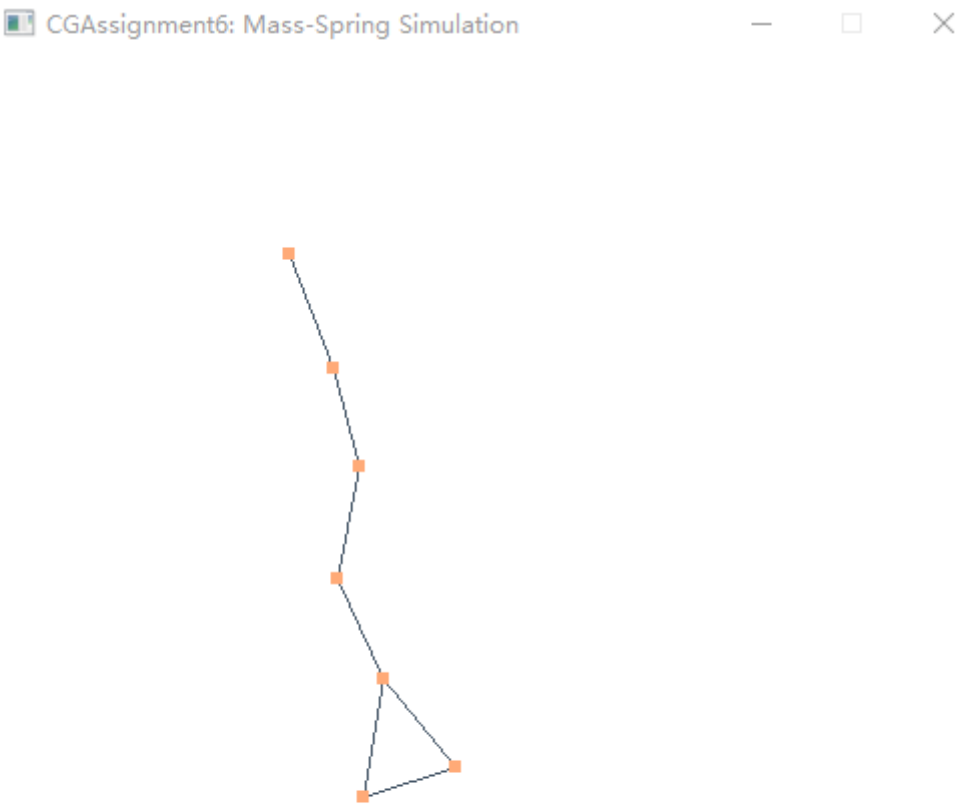
初始状态



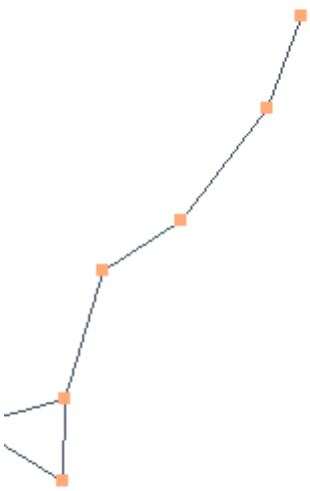
运动1



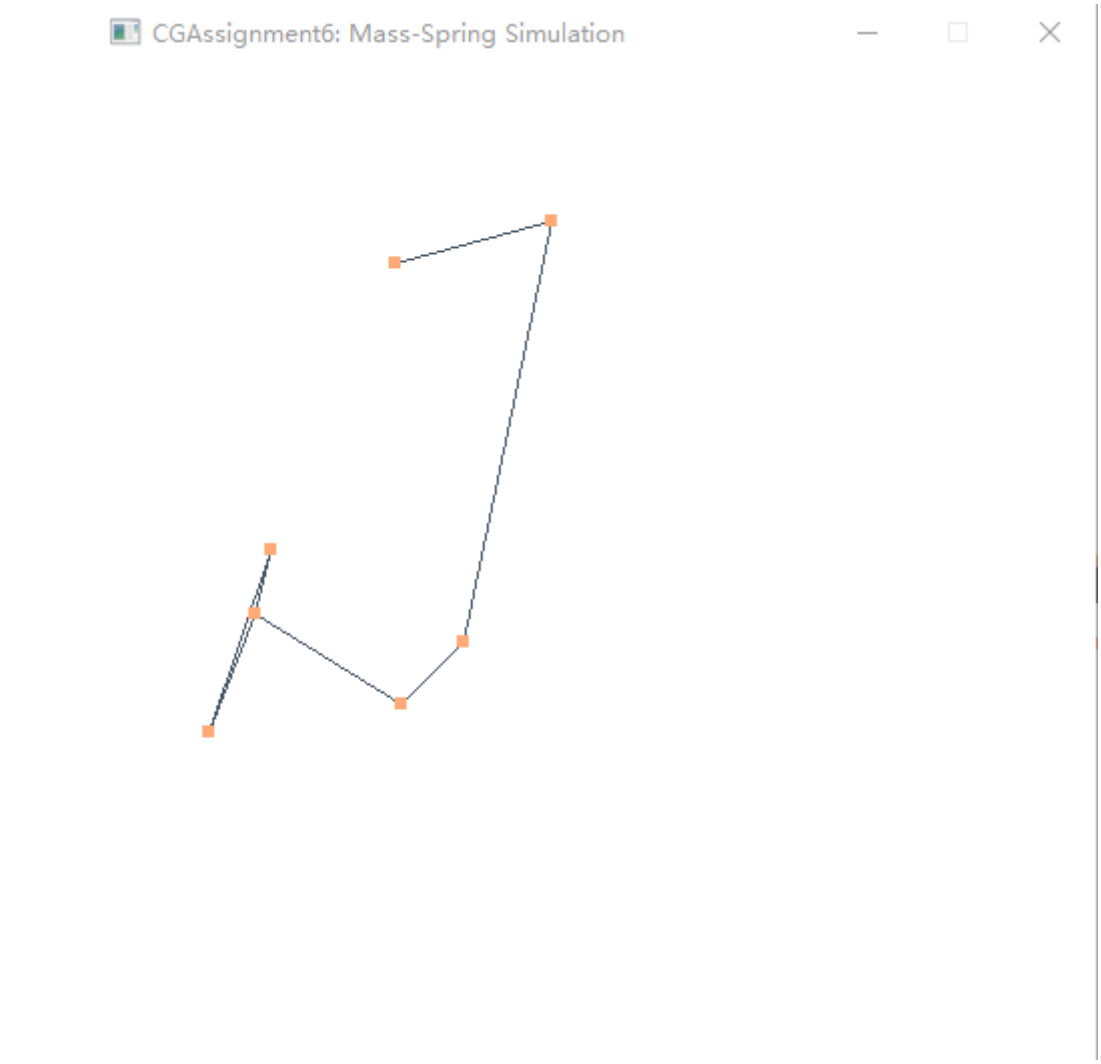
运动2



运动3



最后状态



可以看出晃动幅度越来越明显，动荡的比较厉害，时间越来越长后出现了爆炸的现象。

简述做法

先由胡克定律（1）（2）求出每个质点的合力，然后根据合力计算速度值m_v1[i]，最后计算质点的位置（4）。

Task 2

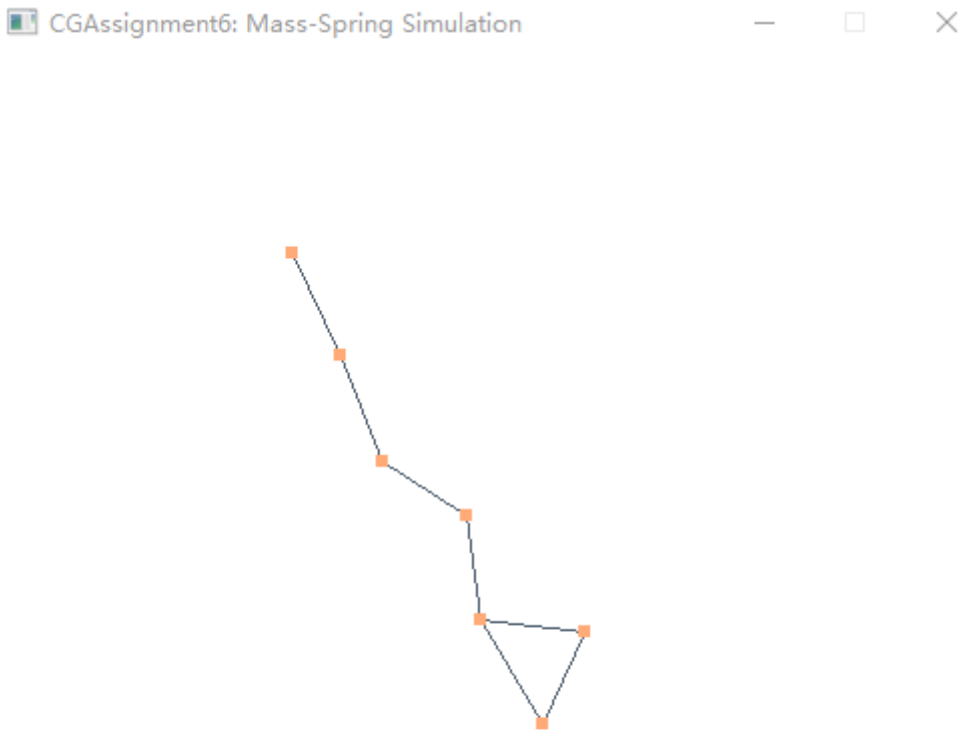
用前面的公式（5）即半隐式积分法来实现质点弹簧系统的动画仿真。

代码

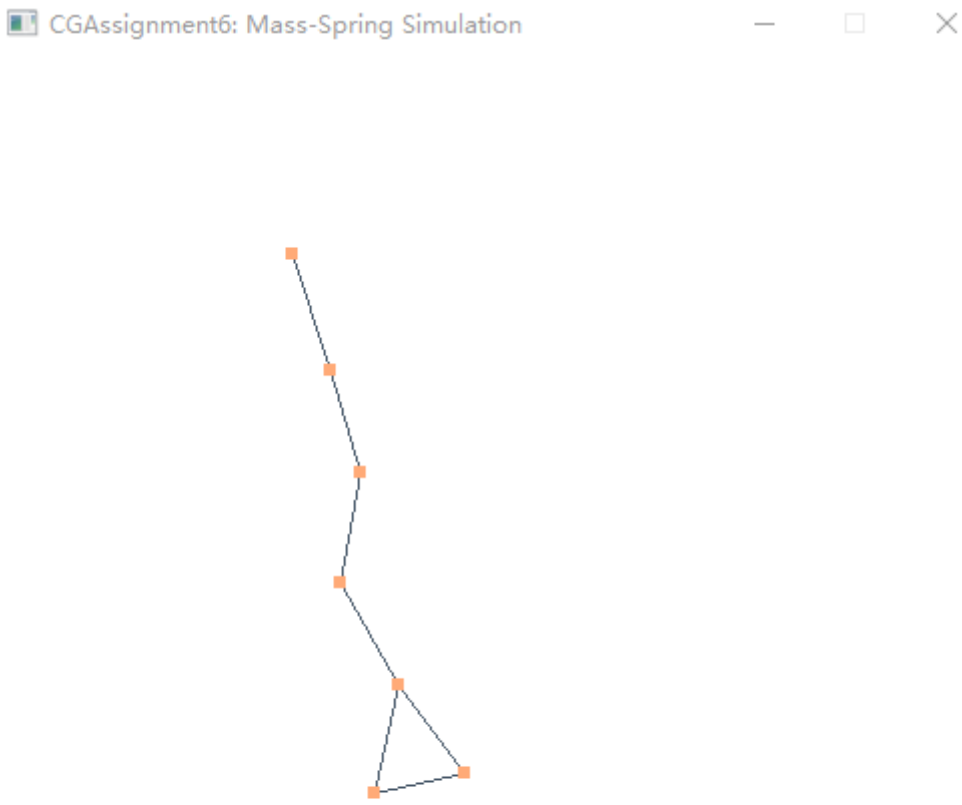
```
    // Todo: update the position m_x[i] using m_v[i]
    // Note: you should use the time step dt
    for (unsigned int i = 1; i < m_numParticles; ++i)
    {
        //Task1：显式法
        //m_x[i] += m_v1[i] * dt;

        //Task2：半隐式法
        m_x[i] += m_v[i] * dt;

    }
```

实现效果

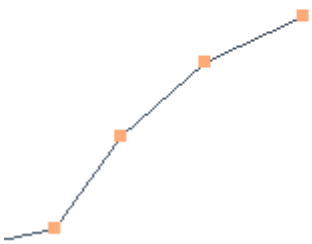
运动1



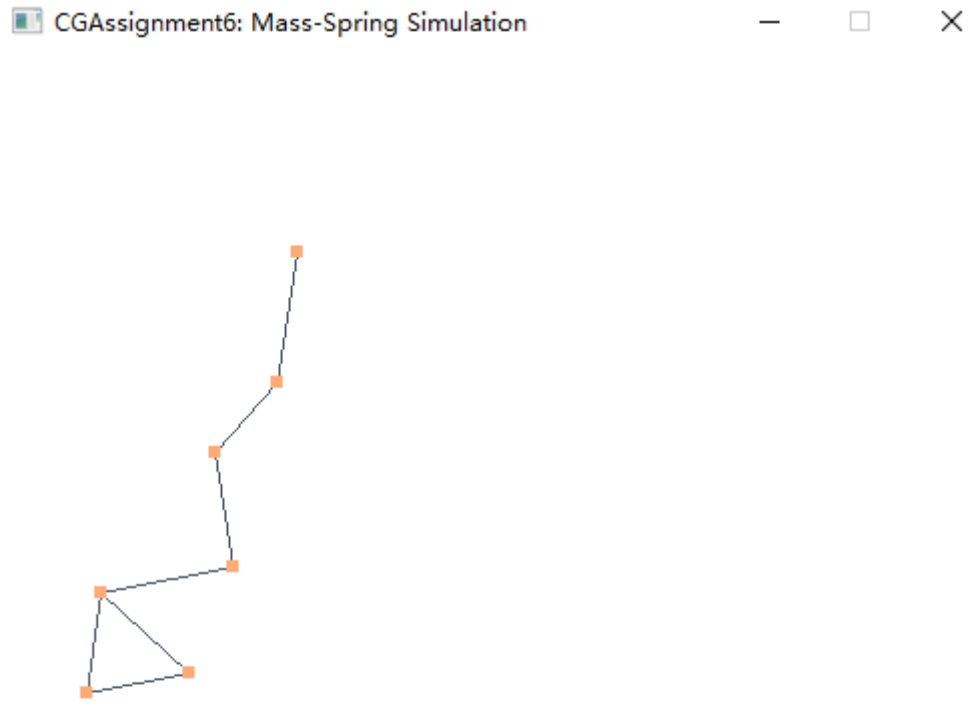
运动2



运动3



最后状态



可以看出晃动幅度相对于Task1有明显减小，但由于没有能量损耗所以动荡并不会停止。

简述做法

和Task2相似，只是计算出的不是前一步的速度值 m_v1[i] ，而是后一步的 m_v[i] 。

Task3

对质点的速度 按照给定的阻尼系数进行衰减，以实现更贴近物理真实的弹簧效果。

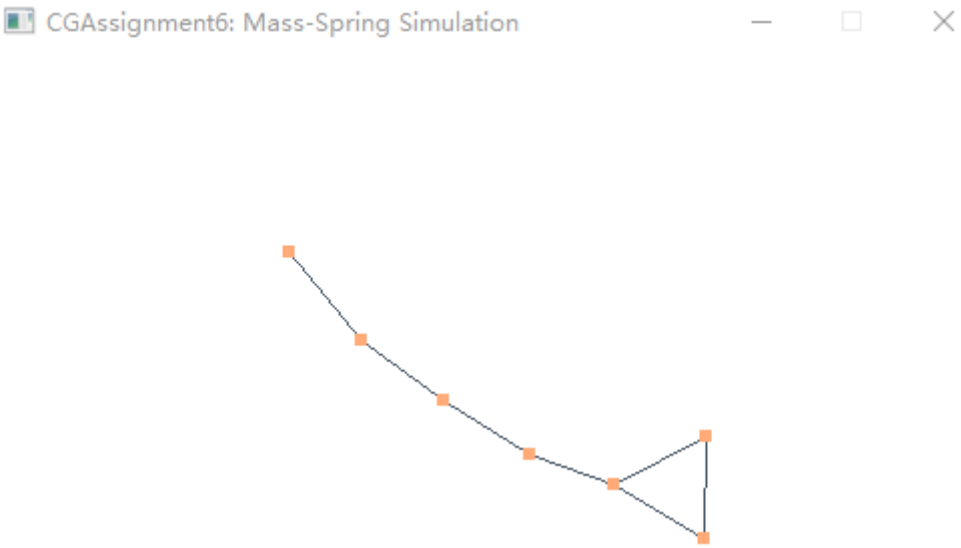
代码

```
    // Todo: update the position m_x[i] using m_v[i]
    // Note: you should use the time step dt
    for (unsigned int i = 1; i < m_numParticles; ++i)
    {
        //Task1: 显式法
        //m_x[i] += m_v1[i] * dt;

        //Task2: 半隐式法
        //m_x[i] += m_v[i] * dt;

        //Task3: 添加阻尼系数
```

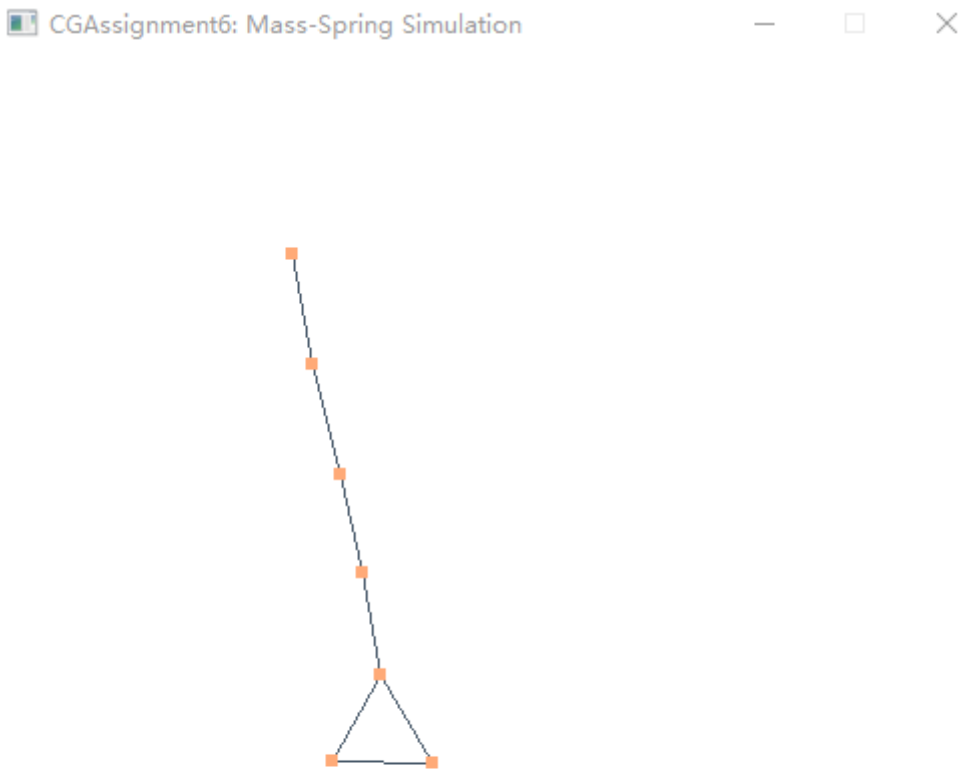
```
        m_v[i] *= exp(-dt * m_damping);
        m_x[i] += m_v[i] * dt;
    }
```

实现效果

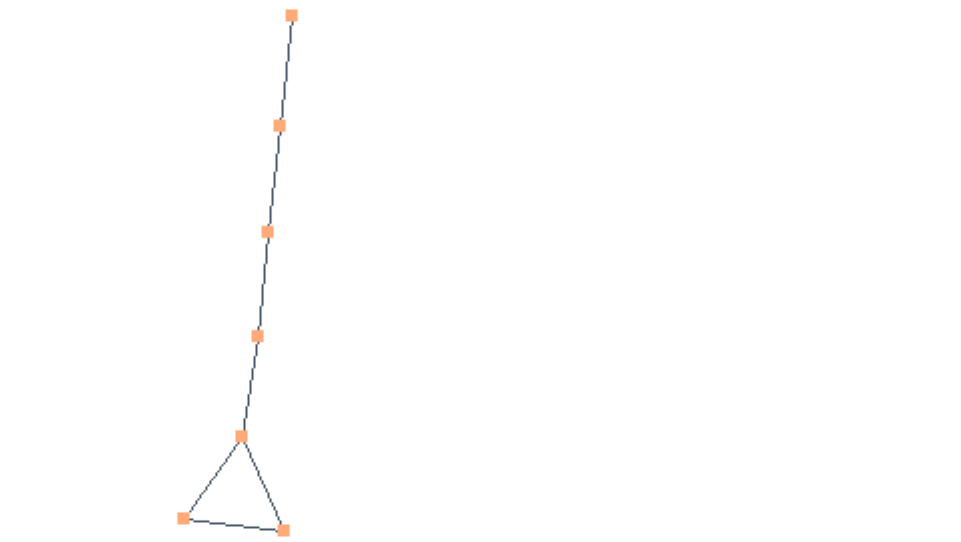
运动1



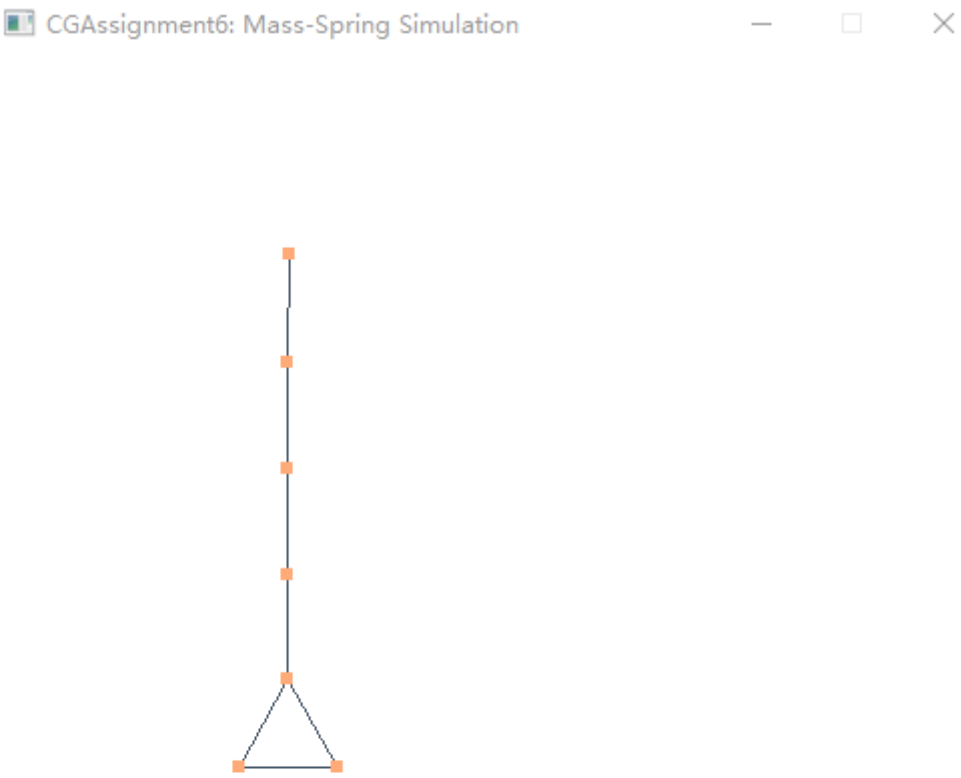
运动2



运动3



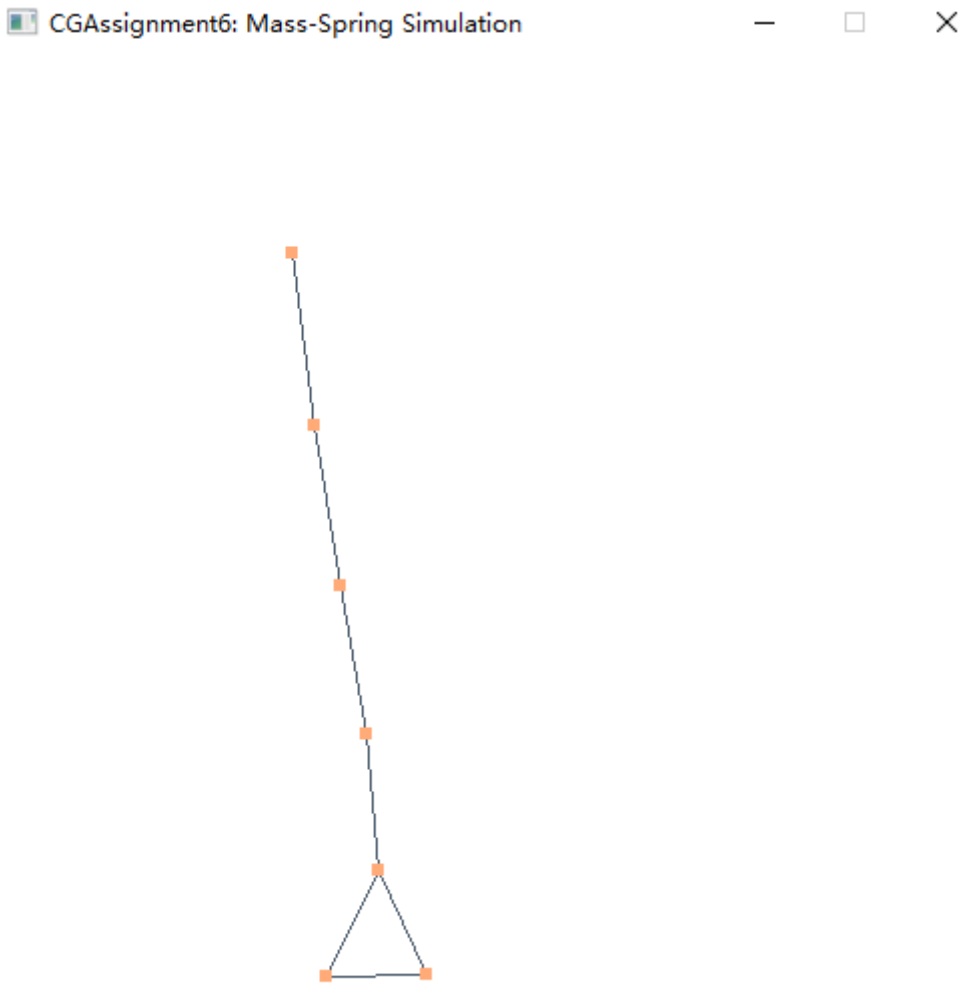
最终状态



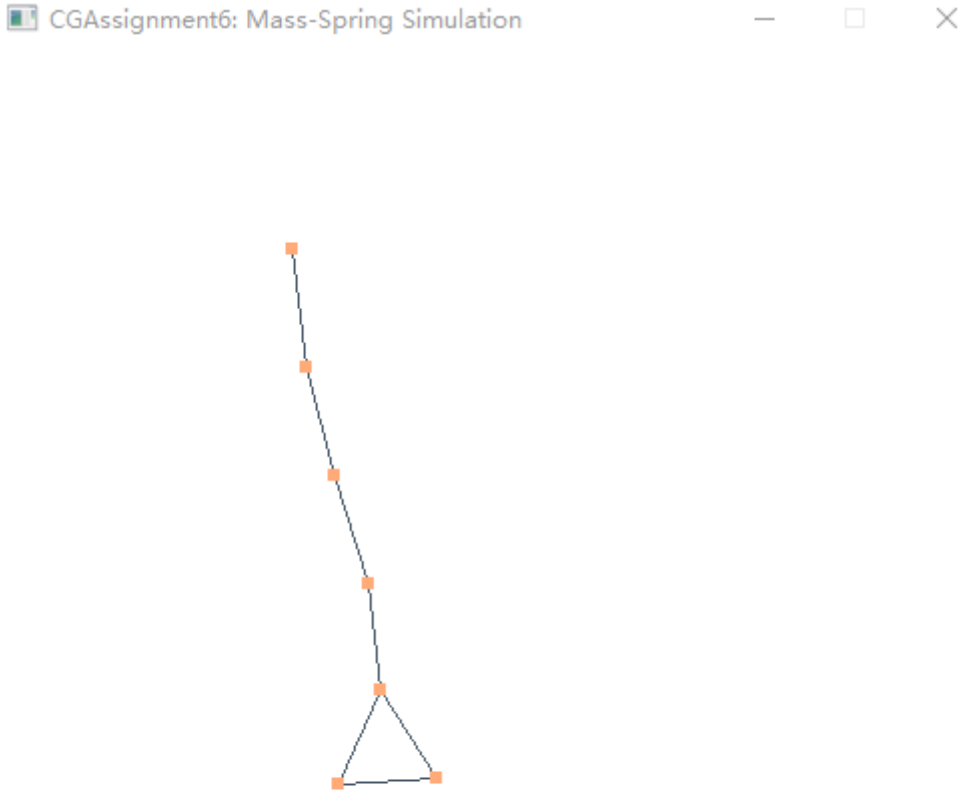
Task4

尝试修改的 Simulator 的 m_stiffness 系数运行动画仿真，看看效果有什么不同，仔细体会公式（1）中的弹簧刚度系数 k 的物理意义。

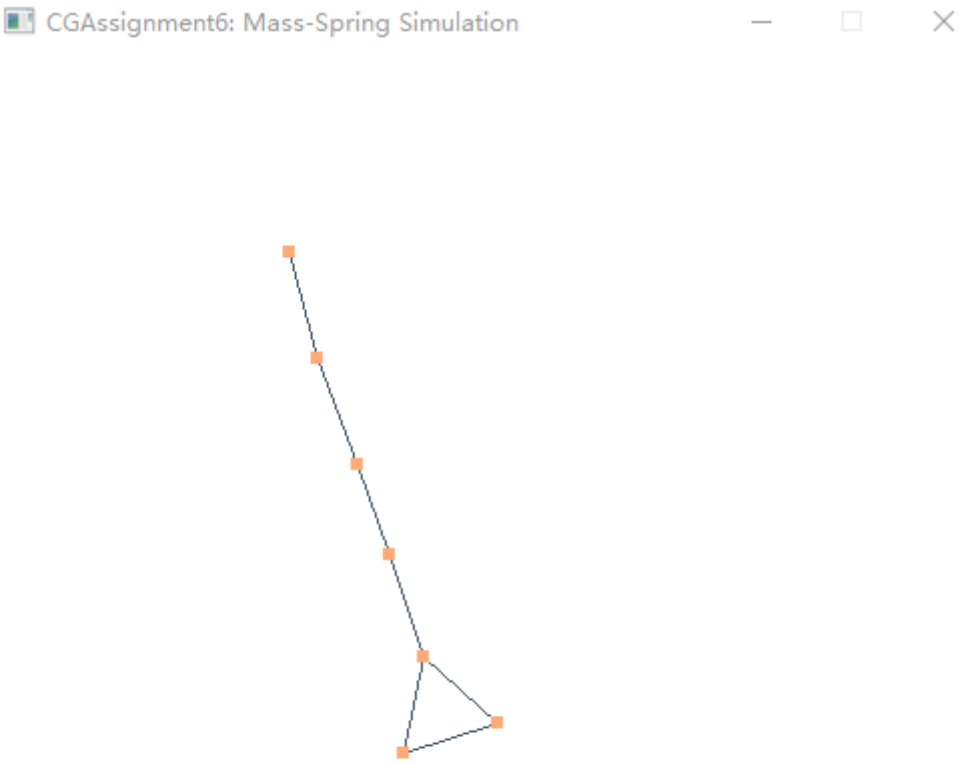
m_stiffness = 1000时：



m_stiffness = 5000时：



m_stiffness = 10000时:



Simulator 的 m_stiffness 系数，即公式中的弹簧刚度系数 k ，它描述的是单位形变量时所产生弹力的大小。 k值大，说明形变单位长度需要的力大，或者说弹簧“韧”。 弹簧的刚度系数，也叫劲度系数或者倔强系数。

k越大，弹簧越硬，实验中下落时拉长效果越小；k越大，弹簧越软，实验中下落时拉长效果越大。