电子科技大学信息与软件工程学院

实 验 指 导 书

（实验）课程名称 图形与动画 II

电子科技大学教务处制表

目　　录

实验四 弹性网络

一、实验目的

1. 掌握弹性网络的基本原理和实现方法；
2. 掌握基本弹性网络实现简单的一维弹性物体的方法；
3. 掌握基本弹性网络实现二维的弹性物体。

二、实验原理

（一） 摆动的绳子

本节我们以模拟一根可以自由摆动的绳子为例（如图 1 所示）介绍如何利用弹簧-质量-阻尼系统模拟一个弹性对象。在该例子中，绳子的一段固定，另一端可自由摆动。绳子所受外力仅为重力。

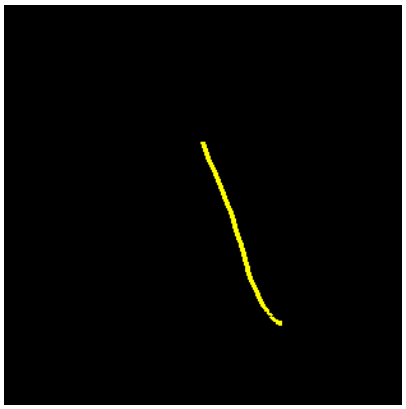

图 1 简单绳子动画

1 绳子建模

在开始模拟之前，首先利用弹簧-质量-阻尼系统为绳子建模，建模方法如图 2 所示。假设绳子由 n 个质点构成，每两个连续的质点之间加入带阻尼的虚拟弹簧，共计 n-1 个弹簧。

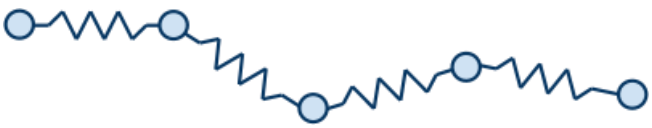

图 2 绳子建模

质点个数 n 决定模拟的精度。质点数越多自然精度越高，绳子运动显得更自然。请大家

注意，在计算机动画中，模拟的运动并不一定和自然界完全吻合。我们使用的运动模型，必须和我们需要模拟的目的有关，目的决定了模拟的精确度。计算机动画的目标不同于科学研究和工程应用，需要准确模拟物体运动，而是在于强调模拟效果的可信度。因此，建模方案中 n 的取值只要能满足模拟效果就可以。

接下来根据建好模型更新质点运动状态，主要步骤如下：

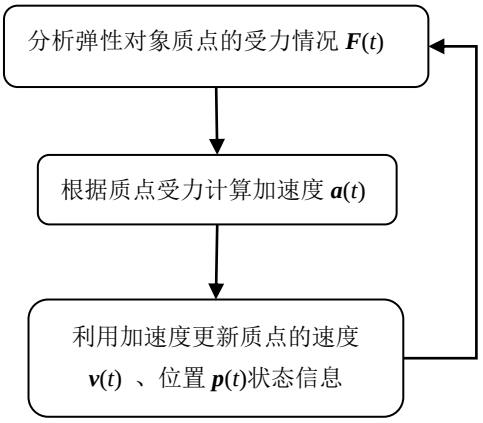

图 3 质点运动状态更新

更新质点运动状态采用欧拉方法作数值积分求解其在任意时间 t 的状态 ：

$$\begin{aligned} \boldsymbol{v}(t+\Delta t) &= \boldsymbol{v}(t)+\frac{\boldsymbol{F}(t)}{m}\Delta t \\ \boldsymbol{p}(t+\Delta t) &= \boldsymbol{p}(t)+\boldsymbol{v}(t)\Delta t \end{aligned} \qquad (1)$$

从上述公式可见，要更新质点状态关键在于对质点进行受力分析。

2 质点受力分析

摆动绳子内部质点所受力有弹力、阻尼力以及重力。重力计算非常简单，下面我们重点推导弹力、阻尼力的计算公式。

1) 一维空间质点受力分析

首先分析质点在一维空间中受力情况，即质点只沿着水平方向运动，质点所受弹力、阻尼力同样位于水平方向。假设两个质点通过一个弹簧-阻尼系统连接，如图 所示。质点质量分别为 m_1 和 m_2。随着质点 1 和质点 2 运动，弹簧-阻尼系统产生弹力和阻尼力，分别作用在两个质点上。两个质点所受合力大小相等方向相反。假设质点 1 所受合力为 f_{sd}，质点 2 所受合力为-f_{sd}。

图 4 一维空间下弹力和阻尼力计算

质点 1 所受弹力计算公式为 f_s：

$$f_s = -k_s\left(d - x\right) \tag{2}$$

质点 1 所受阻尼力计算公式为 f_d：

$$f_d = -k_d\left(v_1 - v_2\right) \tag{3}$$

质点 1 所受合力为：

$$f_{sd} = -k_s\left(d - x\right) - k_d\left(v_1 - v_2\right) \tag{4}$$

- k_s：弹性系数；
- d：弹簧位于稳定状态时，相邻质点间距离；
- x：相邻质点间距离；
- k_d：阻尼系数。

2) **三维空间质点受力分析**

在三维空间中，质点可能沿着任意方向运动，质点所受弹力、阻尼力位于空间任意方向。因此，公式（4）不再适用于质点位于三维空间合力计算，我们需要重新推导。我们注意到两个质点所受弹力和阻尼力一定位于这两个质点连线方向，因此我们可以利用前面结果推导出在三维空间中质点所受弹力和阻尼力计算公式，其计算过程分为如下三大步：

1) 将质点速度值从三维空间投影至一维空间（质点连线方向）；
2) 在一维空间计算质点弹力、阻力；
3) 将计算得到弹力、阻力从一维空间转到三维空间。

2.1 弹力

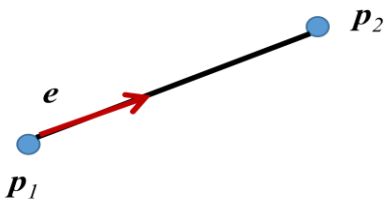

图 5 弹力方向

假设质点 m_1 和 m_2 位置向量分别为 $\boldsymbol{p}_1$，$\boldsymbol{p}_2$ ，定义弹簧向量 $\boldsymbol{s}$ 为:

$$\boldsymbol{s} = \boldsymbol{p}_1 - \boldsymbol{p}_2 \tag{5}$$

该弹簧向量长度为$\|\boldsymbol{s}\|$，方向为：

$$\boldsymbol{e} = \frac{\boldsymbol{s}}{\|\boldsymbol{s}\|} \tag{6}$$

质点 1 位于弹簧方向 $\boldsymbol{e}$ 所受弹力大小为：

$$f_s = -k_s\left(d - \|\boldsymbol{s}\|\right) \tag{7}$$

2.2. 阻尼力

质点 m_1 和 m_2 速度向量分别为 $\boldsymbol{v}_1$ 和 $\boldsymbol{v}_2$。阻尼力大小和质点运动速度相关并且阻尼力大小只和沿着弹簧方向的速度相关。因此，要求得阻尼力首先需要求得速度向量在弹簧方向上的分量。

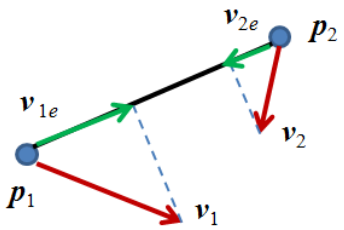

图 6 速度分解

$\boldsymbol{v}_1$ 和 $\boldsymbol{v}_2$ 在弹簧向量方向 $\boldsymbol{e}$ 上速率为：

$$\begin{aligned} v_{1e} &= \boldsymbol{e} \cdot \boldsymbol{v}_1 \\ v_{2e} &= \boldsymbol{e} \cdot \boldsymbol{v}_2 \end{aligned} \tag{8}$$

阻尼力大小为：

$$f_d = -k_d(v_{1e} - v_{2e}) \tag{9}$$

其中，k_d 为阻尼系数。

2.3.合力

质点 m_1 受力大小为：

$$f_{sd} = -k_s(d - \|\boldsymbol{s}\|) - k_d(\boldsymbol{v}_{1e} - \boldsymbol{v}_{2e}) \tag{10}$$

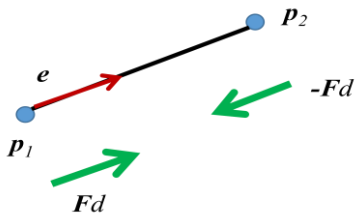

图 7 质点受力

质点 m_1 受弹力及阻尼力 $\boldsymbol{F}_{sd}$ 为：

$$\boldsymbol{F}_{sd} = f_{sd}\boldsymbol{e} \tag{11}$$

质点 m_2 受弹力及阻尼力为：

$$-\boldsymbol{F}_{sd} = -f_{sd}\boldsymbol{e} \tag{12}$$

注意， $\boldsymbol{F}_{sd}$ 为矢量。

3 代码实现

下面给出实现上述例子的关键代码：

1) 构建质点类

质点类 Mass 中记录的质点信息包括：质量、位置、速度和受力。除此以外，该类中包含实现质点运动状态更新的成员函数。

```
class Mass
{
    public:
        float m;                                    // Mass value
        Vector3D pos;                               // Position in space
        Vector3D vel;                               // Velocity
        Vector3D force;                             // Force applied on this mass at an instance

    Mass(float m)                                   // Constructor
    {
      this->m = m;
    }

    void applyForce(Vector3D force)
    {
        this->force += force;                       // The external force is added to the force of the mass
    }

     void init()                                    //void init() method sets the force values to zero
```

```
    {
        force.x = 0;
        force.y = 0;
        force.z = 0;
    }

    void simulate(float dt)
    {
        vel += (force / m) * dt;                    // Change in velocity is added to the velocity.
        // The change is proportinal with the acceleration (force / m) and change in time
        pos += vel * dt;                            // Change in position is added to the position.
        // Change in position is velocity times the change in time
    }
};
```

2) 构建弹簧类

接下来在 Mass 类基础上构建一个弹簧类 Spring。Spring 类中记录的信息有：链接弹簧的两个质点、弹簧静止长度、弹性系数和阻尼系数。 除此以外，该类中包含一个成员函数 solve()用于计算弹簧链接的两个质点受力。

```
class Spring
{
    public:
        Mass* mass1;                                        //m1 at one tip of the spring
        Mass* mass2;                                        //m2 at the other tip of the spring

        float springConstant;                               // constant of stiffness of the spring
        float springLength;                                 //rest length of spring
        float frictionConstant;                             //constant for inner friction of the spring

    Spring(Mass* mass1, Mass* mass2, float springConstant, float springLength, float frictionConstant)
    {                                                                                   //Constructor
        this->springConstant = springConstant;                          //set the springConstant
        this->springLength = springLength;                              //set the springLength
        this->frictionConstant = frictionConstant;                      //set the frictionConstant
        this->mass1 = mass1;                                                            //set mass1
        this->mass2 = mass2;                                                            //set mass2
    }

    void solve()                                    //solve() method: the method where forces can be applied
    {
        Vector3D springVector = mass1->pos - mass2->pos;            //vector between the two masses

        float r = springVector.length();                            //distance between the two masses
```

```
            Vector3D    e, force;                                   //force initially has a zero value
            if (r != 0)                                         //to avoid a division by zero check if r is zero
            {
                  e = springVector / r;
                  force += e * (r - springLength) * (-springConstant);       // spring force is added to the force
                  force += - e * (mass1->vel * e - mass2->vel * e) * frictionConstant;
                                                                        // friction force is added to the force
               }
             mass1->applyForce(force);                                          //force is applied to
mass1
            mass2->applyForce(-force);                              //opposite of force is applied to mass2
      }
};
```

3) 构建模拟类

完成弹簧类的构建后，需要按照图 3 所示的方法更新质点运动。定义类 RopeSimulation 实现质点运动更新。其中成员函数 solve 更新所有弹簧链接质点所受的来自弹簧的合力以及重力。该类定义主要部分如下：

```
class RopeSimulation : public Simulation    //An object to simulate a rope interacting
{
      public:
            Spring** springs;          //Springs binding the masses (there shall be [numOfMasses - 1] of them)
           Vector3D gravitation;              //gravitational acceleration (gravity will be applied to all masses)
            Vector3D ropeConnectionPos;              //A point in space that is used to set the position of the
                                                              //first mass in the system (mass with index 0)
            Vector3D ropeConnectionVel;                 //a variable to move the ropeConnectionPos

      RopeSimulation(
            int numOfMasses,                                          //1. the number of masses
            float m,                                                   //2. weight of each mass
            float springConstant,                                    //3. how stiff the springs are
            float springLength,                          //4. the length that a spring does not exert any force
            float springFrictionConstant,                          //5. inner friction constant of spring
            Vector3D gravitation                                     //6. gravitational acceleration
) : Simulation(numOfMasses, m)

            springs = new Spring*[numOfMasses - 1];         //create [numOfMasses - 1] pointers for springs
                                                  //([numOfMasses - 1] springs are necessary for numOfMasses)

            for (a = 0; a < numOfMasses - 1; ++a)                  //to create each spring, start a loop
            {
           //Create the spring with index "a" by the mass with index "a" and another mass with index "a + 1"
```

```
            springs[a] = new Spring(masses[a], masses[a + 1], springConstant, springLength,
                                                                    springFrictionConstant);
            }
        }

    void solve()                            //solve() is overriden because we have forces to be applied
    {
            int a;
            for (a = 0; a < numOfMasses - 1; ++a)       //apply force of all springs
            {
                springs[a]->solve();                    //Spring with index "a" should apply its force
            }

            for (a = 0; a < numOfMasses; ++a)//Start a loop to apply forces which are common for all masses
            {
                masses[a]->applyForce(gravitation * masses[a]->m);   //The gravitational force
            }
     }
```

RopeSimulation 类是 Simulation 类的派生类。Simulation 类中成员函数 operate 更新所有质点的运动状态。Simulation 类部分实现如下：

```
class Simulation
{
     public:
          int numOfMasses;                                  // number of masses in this ontainer
          Mass** masses;                          // masses are held by pointer to pointer. (Here Mass**
                                                            represents a 1 dimensional array)
     virtual void init()                                    // call the init() method of every
mass
     {
          for (int a = 0; a < numOfMasses; ++a)                         // init() every mass
              masses[a]->init();
     }

     virtual void solve()          // no implementation because no forces are wanted in this basic container
     {
              // in advanced containers, this method will be overrided and some forces will act on masses
     }

     virtual void simulate(float dt)                    // Iterate the masses by the change in time
     {
          for (int a = 0; a < numOfMasses; ++a)         // Iterate the mass and obtain new position
              masses[a]->simulate(dt);                                  //and new velocity
     }
```

```
    virtual void operate(float dt)                    // Complete procedure of simulation
    {
        init();                                       // Step 1: reset forces to zero
        solve();                                      // Step 2: apply forces
        simulate(dt);                                 // Step 3: iterate the masses by the change in time
    }
};
```

三、实验内容

1. 在一维弹性物体基础上设计实现二维弹性物体模拟，并加入和环境的交互；
2. （选作）在二维弹性物体基础上设计实现三维弹性物体模拟，并加入和环境的交互。

四、实验步骤

1. 明确项目需求；
2. 编写代码；
3. 编译代码；
4. 测试程序；
5. 根据测试结果对程序进行调试改进。