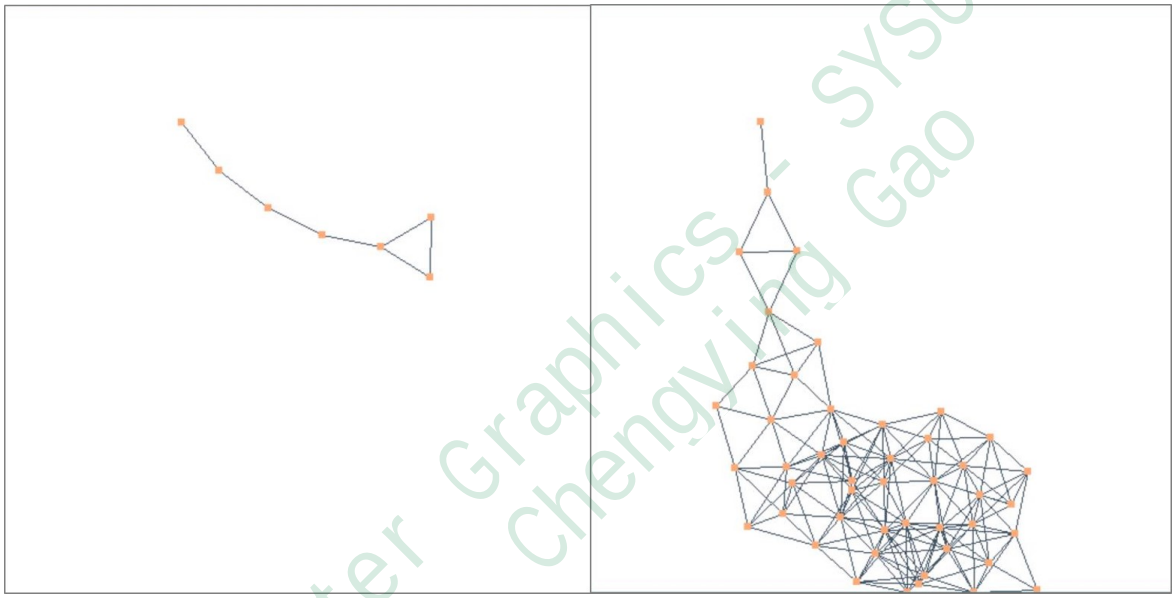
Assignment 6: Mass-Spring Simulation

Computer Graphics Teaching Stuff, Sun Yat-Sen University

Due Date: **1月9号晚上12点之前**

Submission: `sysu_cg_2022@163.com`

> 渲染、建模和动画是计算机图形学领域的三大主题。所谓动画也就使一幅图像"活"起来的过程。使用动画可以清楚的表现出一个事件的过程，或是展现一个活灵活现的画面。动画是一门通过在连续多格的胶片上拍摄一系列单个画面，从而产生动态视觉的技术和艺术。本次的作业是让同学动手实践一个简单的质点弹簧模型仿真动画，对基于物理的仿真动画做一个初步的了解。



本次作业你们将实现的效果实例图

1、作业概述

本次作业的主题为质点弹簧模型的动画仿真，要求你们上机动手实现相关的仿真动画算法。本次提供的作业框架已经帮你们搭好了绝大部分的工作，你只需实现核心的质点弹簧模型算法即可。下面对质点弹簧模型做一个简述。

质点弹簧系统是一种拉格朗日视角的物理模拟，把物体看成由一个个质点构成，质点之间通过弹簧相连，从而产生一定程度的弹性形变。质点弹簧系统适用于模拟布料软体、刚体等物理材质。对于物体中的任意两个不同的质点x_i和x_j，从j作用到i的作用力可由以下的胡克定律（Hooke's law）给出：

$$\mathbf{f_{ij}} = -k(\sqrt{\mathbf{x_i} - \mathbf{x_j}} - l_{ij})(\widehat{\mathbf{x_i} - \mathbf{x_j}}) \tag{1}$$

其中k是弹簧刚度系数（spring stiffness），l_{ij}是质点之间的弹簧静止长度，$\widehat{\mathbf{x_i} - \mathbf{x_j}}$是从质点$i$指向质点$j$的单位方向向量（即$\widehat{\mathbf{x_i} - \mathbf{x_j}} = \frac{\mathbf{x}_i - \mathbf{x}_j}{||\mathbf{x}_i - \mathbf{x}_j||}$）。对于质点$\mathbf{x}_i$，它受到的作用力就是与之相连的所有弹性力的合力：

$$\mathbf{f_i} = \Sigma_j^{j \neq i} \mathbf{f_{ij}} \tag{2}$$

求出了合力，再根据牛顿第二定律可求出质点的加速度和位置的变化梯度：

$$\frac{\partial \mathbf{v_i}}{\partial t} = \frac{1}{m_i}\mathbf{f_i} \tag{3}$$

$$\frac{\partial \mathbf{x_i}}{\partial t} = \mathbf{v_i}$$

上面就是质点弹簧系统的核心公式，比较简单，基本都是高中物理的内容。求出了质点的加速度和速度之后，最简单的方法就是采用下面的显式积分公式来更新每个质点的速度$\mathbf{v}$和位置$\mathbf{x}$，从而生成符合弹簧特性的动画：

$$\begin{aligned} \mathbf{v_{t+1}} &= \mathbf{v_t} + \Delta t \frac{\mathbf{f_t}}{m} \\ \mathbf{x_{t+1}} &= \mathbf{x_t} + \Delta t \mathbf{v_t} \end{aligned} \tag{4}$$

亦或者用更为稳定一点的半隐式欧拉积分法（依旧是显式积分）：

$$\begin{aligned} \mathbf{v_{t+1}} &= \mathbf{v_t} + \Delta t \frac{\mathbf{f_t}}{m} \\ \mathbf{x_{t+1}} &= \mathbf{x_t} + \Delta t \mathbf{v_{t+1}} \end{aligned} \tag{5}$$

公式(4)和(5)的区别仅在于更新$\mathbf{x_{t+1}}$时是用$\mathbf{v_t}$还是$\mathbf{v_{t+1}}$，其中Δt是时间步长。本次作业会让你使用这两个公式均做尝试。综上所述，一个简单的质点弹簧模型求解步骤为：

- 计算每个质点受到的弹簧合力，使用公式(1)和公式(2)；
- 根据公式(4)（或(5)）更新质点的速度矢量$\mathbf{v_{t+1}} = \mathbf{v_t} + \Delta t \frac{\mathbf{f_t}}{m}$；
- 碰撞检测和处理，作业框架里面就是简单地对窗口边界做一个判断和处理；
- 根据公式(4)（或(5)）更新质点的位置矢量$\mathbf{x_{t+1}} = \mathbf{x_t} + \Delta t \mathbf{v_t}$。

2、代码框架

关于本次作业的框架代码部署和构建，请仔细阅读`readme.pdf`文档，基本上与`CGAssignment5`没有太大的差别。本次作业依赖的第三方库为：

- **SDL2**：窗口界面库，主要用于创建窗口给并显示渲染的图片结果，本作业不需要你对这个库深入了解
- **GLM**：基础数学库，用于做一些简单的矢量和数学运算

这些第三方库同学们无需太过关注，我们的框架代码已经构建好了相应的功能模块。目录`CGAssignment6/src`存放本次作业框架的所有代码：

- **main.cpp**：程序入口代码，负责执行主要的渲染循环逻辑，无需修改；
- **WindowsApp(.h/.cpp)**：窗口类，负责创建窗口、显示结果、处理鼠标交互事件，无需修改；
- **Utils(.h/.cpp)**：算法核心代码，负责实现质点弹簧的模拟器，此文件绝大部分代码你无需修改。

在本作业框架的`Utils.h`文件中，`Simulator`类用`m_x`、`m_v`和`m_restLength`分别存储质点的位置矢量$\mathbf{x}$、速度矢量$\mathbf{v}$和弹簧静止长度。`m_restLength[i][j]==0`表示质点`i`和质点`j`之间没有相连，反之质点`i`和质点`j`相连，且弹簧静止长度为`m_restLength[i][j]`：

```
    std::vector<glm::vec2> m_x;
    std::vector<glm::vec2> m_v;
    std::vector<std::vector<float>> m_restLength;
```

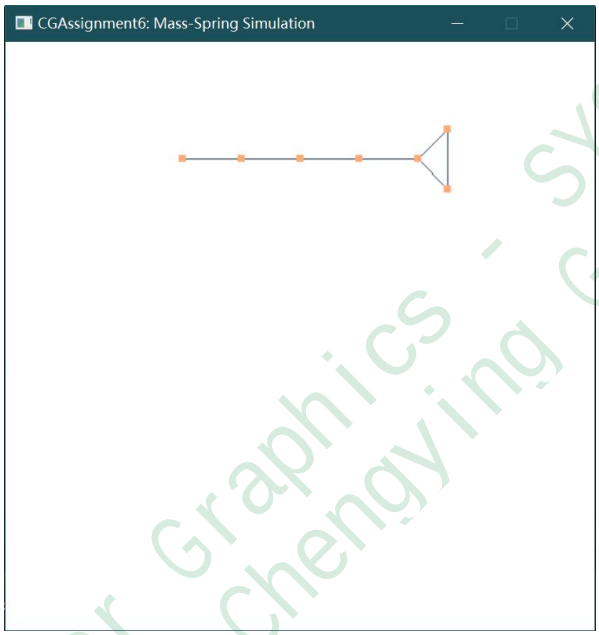
我们用`m_numParticles`存储当前的质点数量，而`m_maxNumParticles`是允许的最大质点数量：

```
    unsigned int m_numParticles;
    const unsigned int m_maxNumParticles = 256;
```

以下的`m_damping`、`m_stiffness`和`m_paerticleMass`分别表示弹簧的阻尼、弹簧刚度系数k和质点的质量，而`m_connectRadius`用于限制质点之间相连的最大长度（小于这个距离就会相连），这个参数你可以不用关注：

```
    float m_damping = 10.0f;
    float m_stiffness = 10000;
    const float m_particleMass = 1.0f;
    const float m_connectRadius = 0.15f;
```

成功编译运行本次提供的作业框架，你应该会得到如下的运行结果：



这里对键盘和鼠标操作做一个说明：

- 鼠标在屏幕上点击会生成一个质点，该质点会与周围相邻的质点自动相连；
- 按下键盘`G`键，就会开始调用`Simulator`的`simulate`函数，执行动画模拟程序，再按暂停。

当然一开始你还没有实现好动画算法，所以按下`G`键也不会有动画效果。

3、作业描述

请你按照下列顺序完成本次做的作业任务。

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。

你需要填充代码的地方在`Utils.cpp`文件的`simulate()`函数，如下所示。其中`dt`是时间步长，`gravity`是重力加速度。你需要先计算每个质点的合力（用前面的公式(1)和(2)），然后用合力更新速度值，最后更新质点的位置矢量。实现好之后，尝试然后按下`g`键查看动画效果（或者点击鼠标生成更多的质点），贴出你的效果，并简述你是怎么做的。

```
void Simulator::simulate()
{
    static constexpr float dt = 0.0001f;
    static constexpr glm::vec2 gravity = glm::vec2(0.0f, -9.8f);
```

```
    // 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, and dt herein
        for (unsigned int j = 0; j < m_numParticles; ++j)
        {

        }
        //Update the m_v[i]

    }

    // 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)
    {

    }
}
```

注意：此时尚未用阻尼对速度进行衰减处理，因此实现的动画效果会一直动荡的比较厉害，这是正常现象。

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。

填充代码的地方与Task1一致，这里不再赘述。简述你是怎么做的。

Task3、对质点的速度$\mathbf{v}$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。

实现了前两个Task之后，弹簧的效果看起来有种不正常的跳动和动荡。现实中的弹簧不会永远跳动，因为动能会因摩擦而减小。为了模拟这种效果，我们用以下的衰减公式对速度进行衰减处理：

$$\mathbf{v} = \mathbf{v} \cdot e^{-\Delta t \cdot \mu}$$

其中μ是阻尼系数，在`Simulator`类中存储为`m_damping`。上述的公式使得速度以指数的方式衰减。请你在`simulate`函数的最后一步（更新质点位置之前），用上述的公式更新质点的速度，使得质点的速度以一定的程度进行衰减。

```
void Simulator::simulate()
{
    ......

    // 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)
    {

    }
}
```

Task4、尝试修改`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。

注意事项:

- 将作业报告、源代码一起压缩打包，文件命名格式为：学号+姓名+HW6，例如19214044+张三+HW6.zip。
- **提交的文档请提交编译生成的pdf文件，请勿提交markdown、docx以及图片资源等源文件！**
- **提交代码只需提交源代码文件即可，请勿提交教程文件、作业描述文件、工程文件、中间文件和二进制文件（即删掉build目录下的所有文件！）。**
- 禁止作业文档抄袭，我们鼓励同学之间相互讨论，但最后每个人应该独立完成。