计算机图形学

弹簧质点系统

Assignment 7

姓名：陈家豪

班级：2018 级计科 1 班

学号：18308013

目录

1 作业任务与要求

(1) Task 1：完善 rope.cpp 中的 Rope::rope(...)；

(2) Task 2：完善 rope.cpp 中的 Rope::simulateEuler(...)；

(3) Task 3：完善 rope.cpp 中的 Rope::simulateVerlet(...)；

2 实验过程与结果

2.1 Task 1：完善 rope.cpp 中的 Rope::rope(...)

rope::rope() 函数主要用于创建新的绳子对象。我们首先需要基于起始位置 *start* 和结束位置 *end*，在中间等距创建 *num_mass* 个节点，存储在 vector 类型的 *masses* 中。每个节点拥有的质量为 *node_mass*。然后，我们在节点之间创建 Spring 对象，存储在 vector 类型的 *springs* 中。Spring 两端各连接一个节点，弹性系数为 k。最后，我们将固定节点的 *pinned* 属性设为 *true*，最终完成这个 rope 的创建。

基于上述思想，完成代码如下所示：

```
Rope::Rope(Vector2D start, Vector2D end, int num_nodes, float node_mass, float k, vector<
    int> pinned_nodes)
{
    // TODO (Part 1): Create a rope starting at 'start', ending at 'end', and containing '
        num_nodes' nodes.
    Mass *lMP=nullptr;  // 上一个质点
    Mass *cMP=nullptr;  // 当前质点
    for (int i=0;i<num_nodes;i++)
    {
        Vector2D CurrentPosition;// 当前质点位置
        if (num_nodes==1)
            CurrentPosition=start;
        else
            CurrentPosition=start+i*(end-start)/(num_nodes-1);
        cMP=new Mass(CurrentPosition,node_mass,false);// 创建质点
        masses.push_back(cMP);
        if (i>0)
        {
            Spring *cSP=new Spring(lMP,cMP,k);// 创建质点间的绳子
            springs.push_back(cSP);
        }
        lMP=cMP;
    }
    for (auto &i : pinned_nodes)
        masses[i]->pinned = true;// 固定相应质点
}
```

运行程序，结果如图1所示。可见，我们得到了一条黄色的有 3 个节点的绳子。

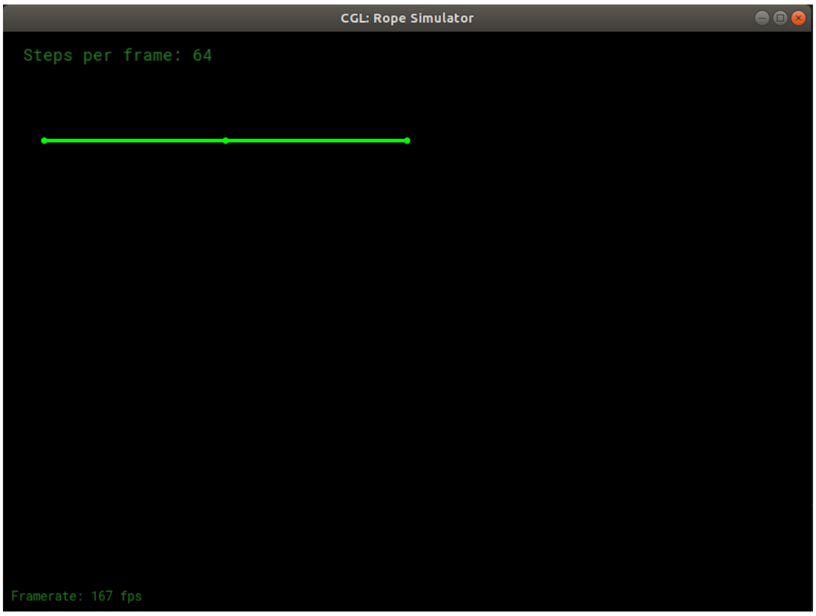

图 1: 运行结果 1

由于另一条蓝色的绳子隐藏在黄色绳子下，故我们在 application.cpp 文件中调整蓝色绳子的 *end* 位置为 $(400, 200)$。运行程序，结果如图2所示。

图 2: 运行结果 2

2.2 Task 2：完善 rope.cpp 中的 Rope::simulateEuler(...)

2.2.1 质点数量的修改

首先，我们在 application.cpp 中将欧拉绳子的质点数量修改为 16，代码如下所示：

```
namespace CGL {
......
void Application::init() {
  // Enable anti-aliasing and circular points.
```

```
    ......
    // Create two ropes
    ropeEuler = new Rope(Vector2D(0, 200), Vector2D(400, 200), 16, config.mass,
                         config.ks, {0});
    ropeVerlet = new Rope(Vector2D(0, 200), Vector2D(-400, 200), 16, config.mass,
                         config.ks, {0});
}
......
```

修改完毕后运行程序，运行结果如图3所示，可见两条绳子都有 16 个节点。

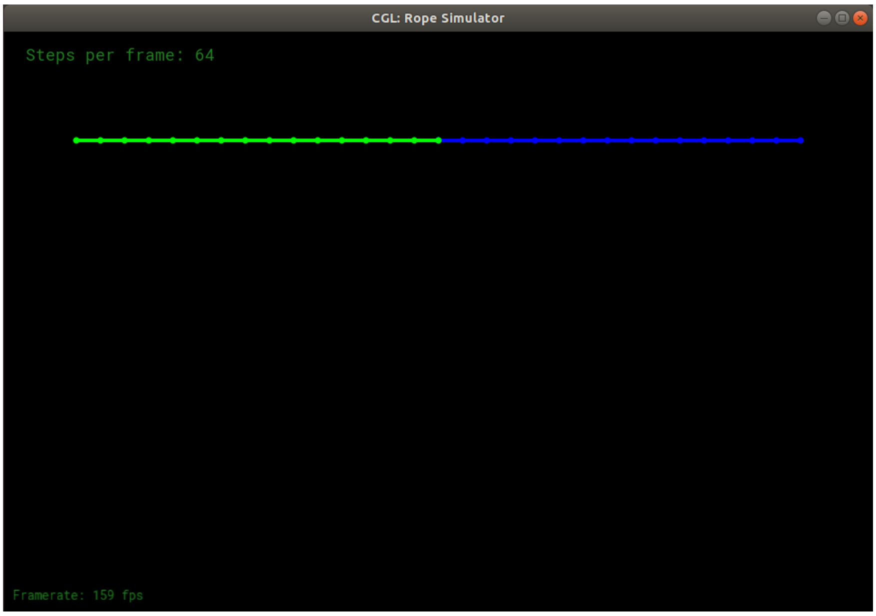

图 3: 运行结果 3

2.2.2 显式欧拉法和半隐式欧拉法的实现

然后我们需要实现显式欧拉法和半隐式欧拉法。这两种方法都需要计算每个质点受到的合力。合力由三部分组成：弹簧力、重力和阻力。弹簧力由胡克定律计算得到。假设两个质点的位置分别为 $\mathbf{a}$ 和 $\mathbf{b}$，之间弹簧的原长为 l，弹性系数为 k_s，那么我们有：

$$\mathbf{f}_{\mathbf{b}\to\mathbf{a}} = -k_s \frac{\mathbf{b}-\mathbf{a}}{\|\mathbf{b}-\mathbf{a}\|}(\|\mathbf{b}-\mathbf{a}\| - l) \tag{1}$$

此外，假设一个质点当前运动速度为 $\mathbf{v}$，阻尼系数为 k_d，那么其受到的阻力为：

$$\mathbf{f}_d = -k_d \mathbf{v} \tag{2}$$

当计算得到一个质点受到的外力合力后，我们就可以通过牛顿第二定律得到该质点的加速度 $\mathbf{a}$。对于显式欧拉法而言，其先更新质点的位置，再更新质点的速度：

$$\mathbf{x}(t+1) = \mathbf{x}(t) + \mathbf{v}(t)\mathrm{d}t \tag{3}$$

$$\mathbf{v}(t+1) = \mathbf{v}(t) + \mathbf{a}(t)\mathrm{d}t \tag{4}$$

对于半隐式欧拉法而言，其先更新质点速度，再更新质点位置：

$$\mathbf{v}(t+1) = \mathbf{v}(t) + \mathbf{a}(t)\mathrm{d}t \tag{5}$$

$$\mathbf{x}(t+1) = \mathbf{x}(t) + \mathbf{v}(t+1)\mathrm{d}t \tag{6}$$

最终实现的代码如下所示：

```cpp
void Rope::simulateEuler(float delta_t, Vector2D gravity)
{
    // 计算每个质点受到的弹簧力
    for (auto &s : springs)
    {
        // TODO (Part 2): Use Hooke's law to calculate the force on a node
        Vector2D a2b=s->m2->position - s->m1->position;
        Vector2D f=s->k * (a2b/a2b.norm()) * (a2b.norm()-s->rest_length);// 弹簧力
        s->m1->forces+=f;
        s->m2->forces-=f;
    }
    for (auto &m : masses)
    {
        if (!m->pinned)
        {
            // TODO (Part 2): Add gravity and global damping, then compute the new velocity and position
            m->forces+=gravity * m->mass;// 重力
            float k_d=0.1;
            m->forces+=-k_d*m->velocity; // 阻力
            Vector2D a=m->forces/m->mass;// 加速度
            if (false)// 显式欧拉法
            {
                m->position+=m->velocity*delta_t;
                m->velocity+=a*delta_t;
            }
            else    // 半隐式欧拉法
            {
                m->velocity+=a*delta_t;
                m->position+=m->velocity*delta_t;
            }
        }
        // Reset all forces on each mass
        m->forces = Vector2D(0, 0);
    }
}
```

设置阻尼系数 $k_d = 0$，编译后采用默认参数 "./ropesim" 运行程序。显式欧拉法运行结果如图4所示，半隐式欧拉法运行结果如图5所示。

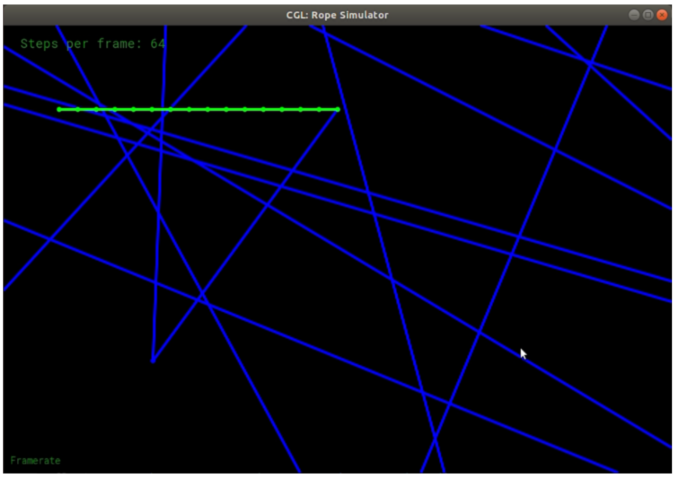

图 4: 显式欧拉法，$k_d = 0$, 步数 $= 64$

图 5: 半隐式欧拉法，$k_d = 0$, 步数 $= 64$

可见，两种方法都出现了质点乱飞出显示区域、消失不见的情况。这是因为在程序中，$\mathrm{d}t$ 的计算方式如下所示：

$$\mathrm{d}t = \frac{1}{Steps\_per\_frame} \tag{7}$$

这就导致了步数越小时，$\mathrm{d}t$ 值越大。在初始情况下，受到重力和弹簧力的影响，质点的加速度较大，故 $\mathrm{d}t$ 较大时会产生较大的速度，从而造成质点出现较大的位移。过大的位移又会进一步造成弹簧力的失衡，这就是为什么会出现质点乱飞、在屏幕消失的原因。

我们依然设置阻尼系数 $k_d = 0$，调整步数为 65536，显示欧拉法运行结果如图6和7所示，可见这次运行正常。

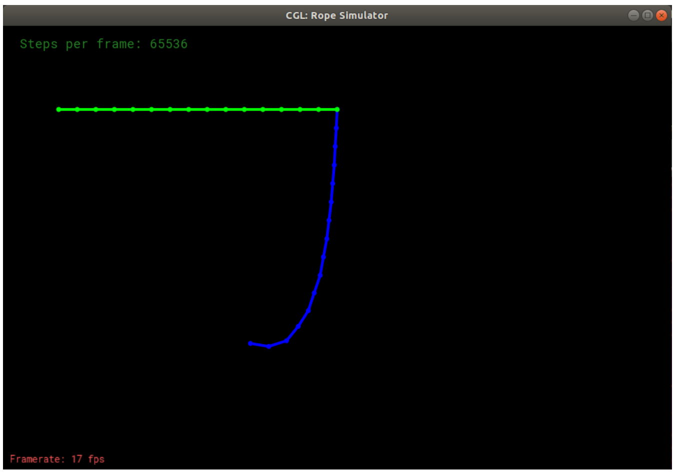

图 6: 显式欧拉法，$k_d = 0$, 步数 $= 65536$

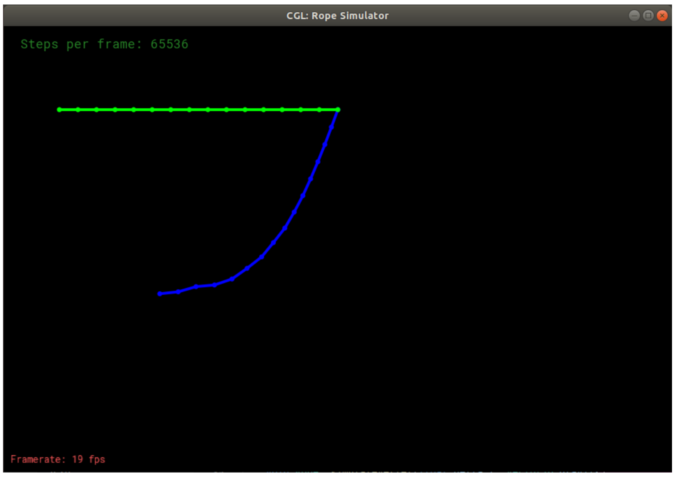

图 7: 半隐式欧拉法，$k_d = 0$, 步数 $= 65536$

2.2.3 比较不同步数下欧拉绳子的摆动情况

统一设置阻尼系数为 0.01，分别设置步数为 1024、4096、16384 和 65536，显式欧拉法运行结果如图8、9、10和11所示。

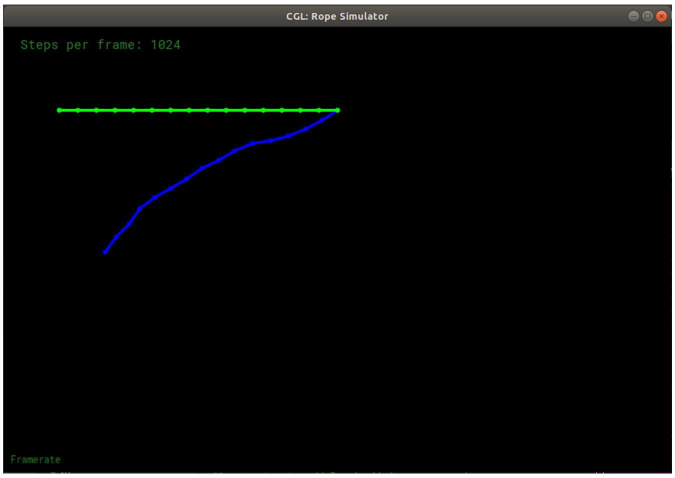

图 8: 显式欧拉法，$k_d = 0.01$, 步数 $= 1024$

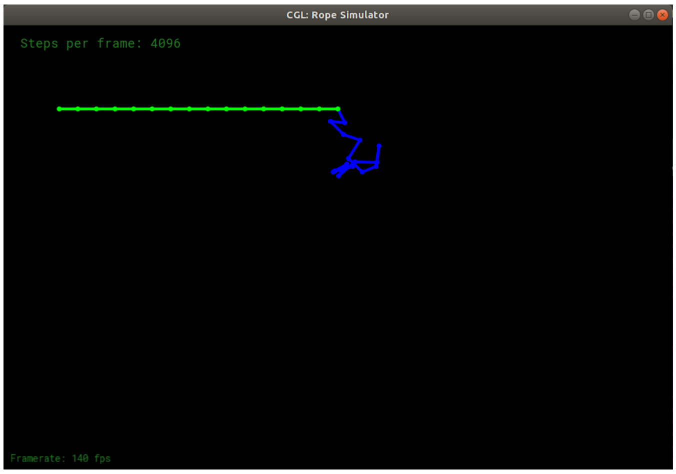

图 9: 显式欧拉法，$k_d = 0.01$, 步数 $= 4096$

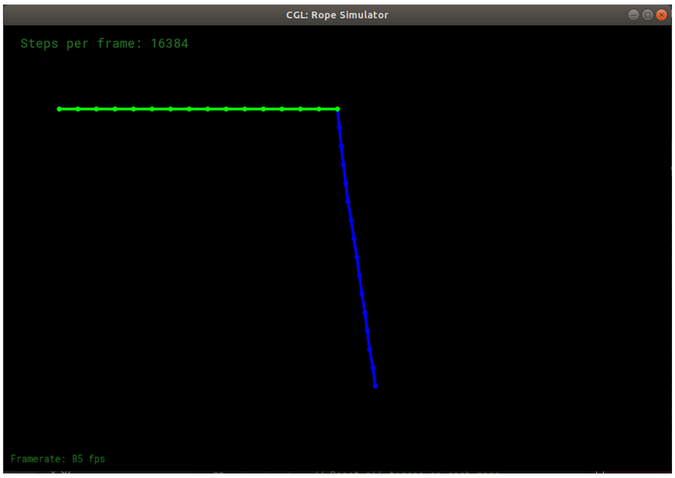

图 10: 显式欧拉法，$k_d = 0.01$, 步数 $= 16394$

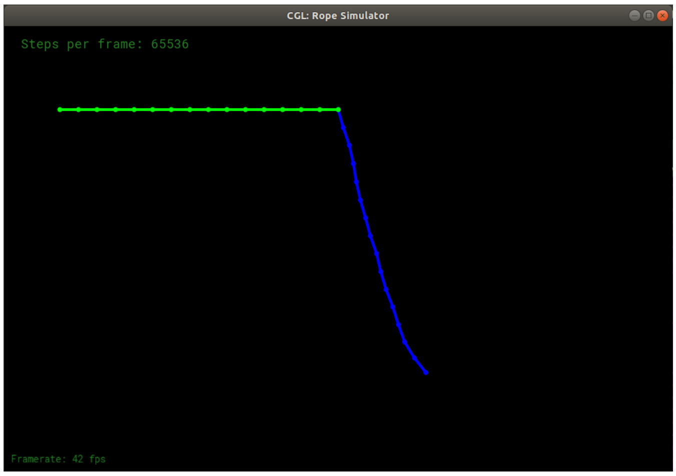

图 11: 显式欧拉法，$k_d = 0.01$, 步数 $= 65536$

当步数为 1024 时，绳子摆动几下后质点就失去控制、乱飞出屏幕。当步数为 4096 时，绳子摆动时间有所增长，但在即将垂直、停止摆动时缩成一团，“蠕动” 一段时间后乱飞出屏幕；当步数为 16394 时，绳子正常摆动、直至垂直静止。当步数为 65536 时，绳子依然正常摆动、直至垂直静止，相比步数为 16394 的绳子摆动更慢、更自然。

半隐式欧拉法运行结果如图12、13、14和15所示。

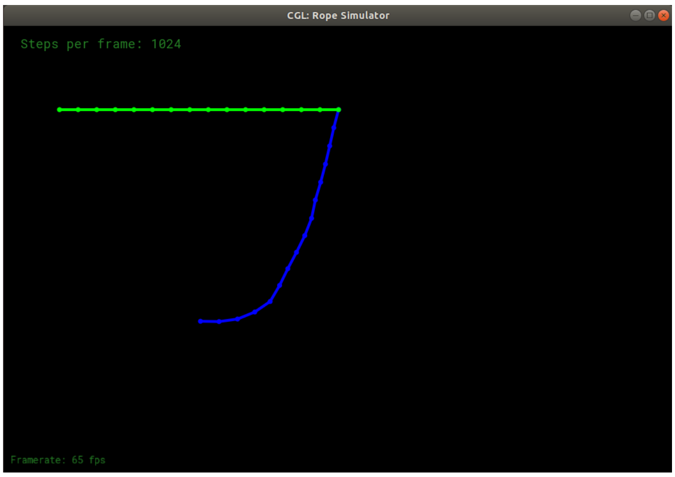

图 12: 半隐式欧拉法，$k_d = 0.01$, 步数 $= 1024$

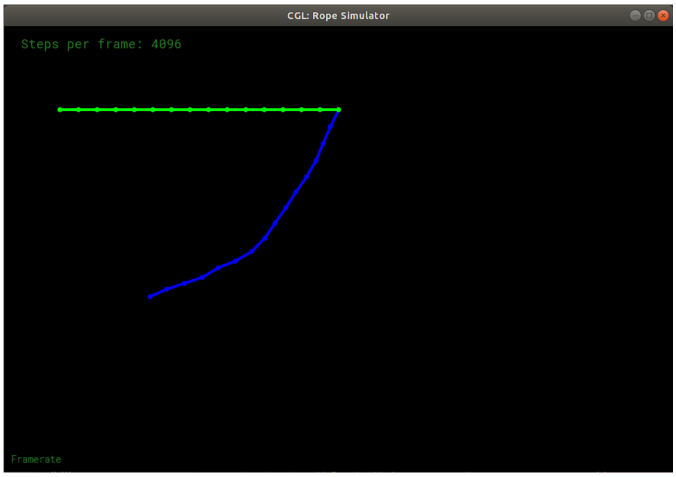

图 13: 半隐式欧拉法，$k_d = 0.01$, 步数 $= 4096$

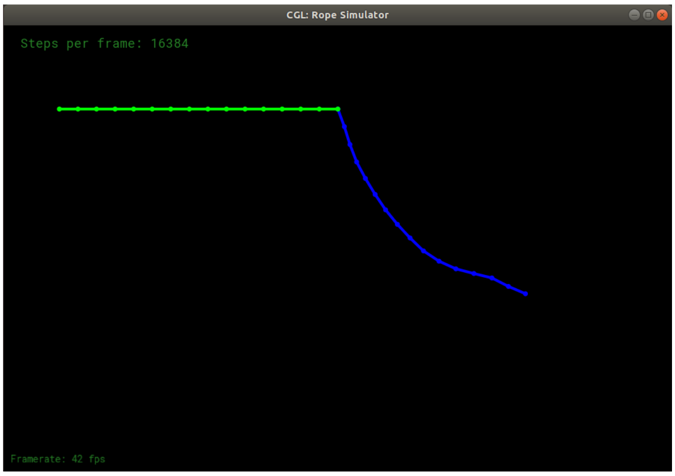

图 14: 半隐式欧拉法，$k_d = 0.01$, 步数 $= 16394$

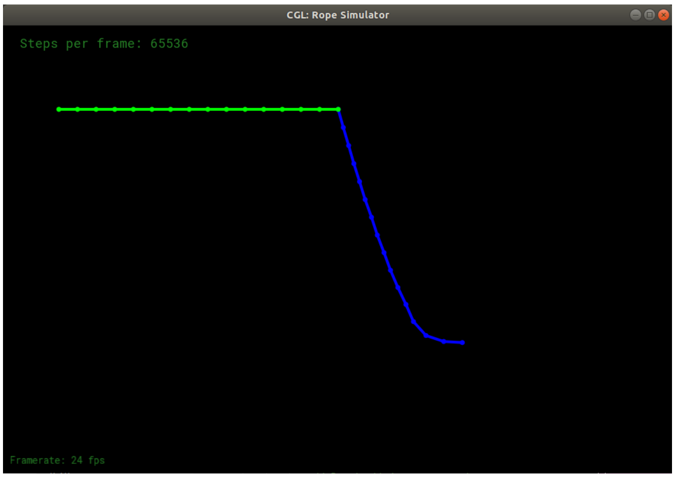

图 15: 半隐式欧拉法，$k_d = 0.01$, 步数 $= 65536$

在不同步数下，半隐式欧拉法的绳子均能正常摆动、直至垂直静止。当步数为 1024 时，绳子摆动较快、形变十分僵硬。随着步数的提高，绳子的摆动逐渐自然，但帧率也逐渐降低、摆动得"越来越慢"。

之所以在步数较低的情况下显式欧拉法的绳子出现质点乱飞、而半隐式欧拉法的绳子能够正常摆动，个人猜测原因可能在于速度更新的时间不同。在半隐式欧拉法中，先更新速度再更新位置的方式使得各质点能够以实时的速度去移动。在显式欧拉法中，速度的更新比位置的更新"慢一拍"，而绳子摆动时各质点的初始加速度较高、后逐渐降低，这就导致了质点以未更新的"较快"速度进行位移，而较大的位移进一步产生更大的弹簧力、造成更大的加速度。这种"正反馈"就会造成质点最终乱飞的结果。

2.2.4 比较不同阻尼系数下欧拉绳子的摆动情况

当步数为 65536 时，分别设置阻尼系数为 0、0.001、0.01 和 0.1，显式欧拉法运行结果如图16、17、18和19所示。

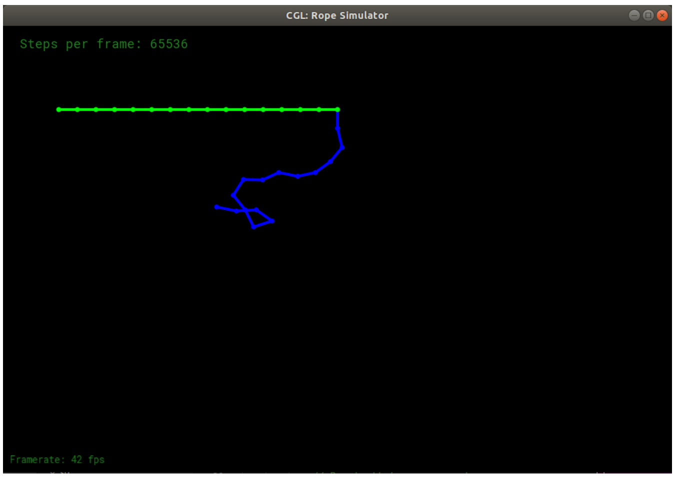

图 16: 显式欧拉法，$k_d = 0$, 步数 $= 65536$

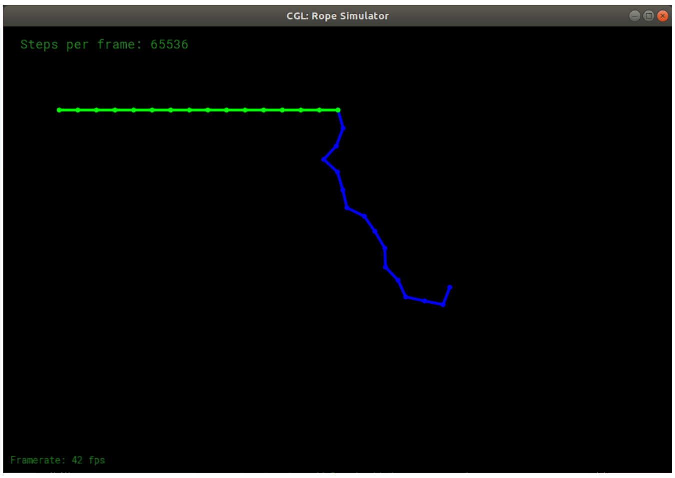

图 17: 显式欧拉法，$k_d = 0.001$, 步数 $= 65536$

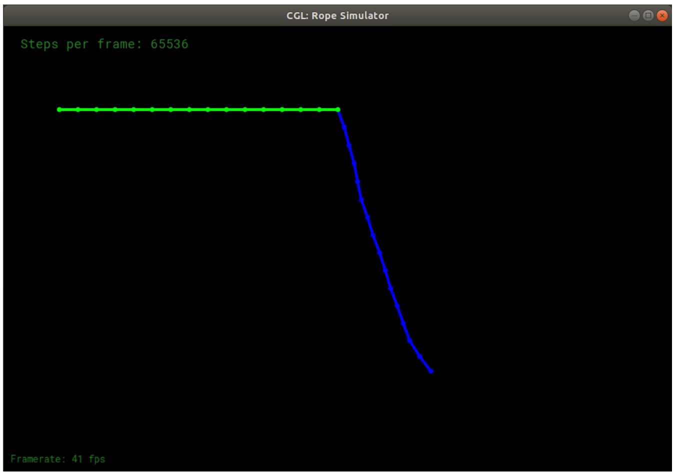

图 18: 显式欧拉法，$k_d = 0.01$, 步数 $= 65536$

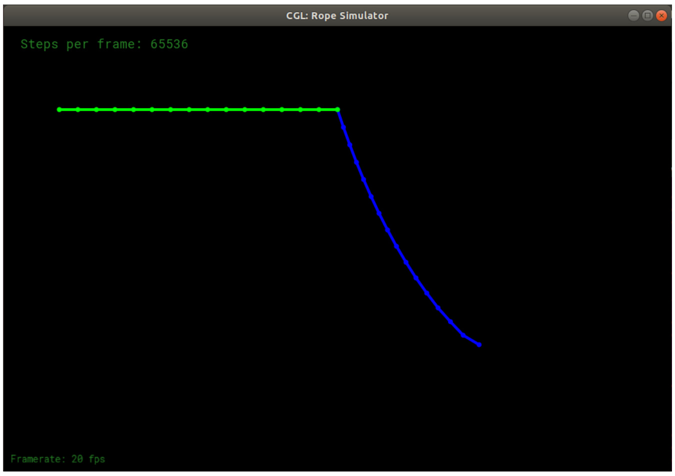

图 19: 显式欧拉法，$k_d = 0.1$, 步数 $= 65536$

当阻尼系数为 0 和 0.001 时，绳子不会停止，而是在自然摆动几下后缩成一团不断加速“蠕动”。个人猜测原因同上述分析一致。当阻尼系数为 0.01 时，绳子自然摆动、幅度越来越小直至停止垂直。当阻尼系数为 0.1 时，绳子没有摆动，而是从水平位置垂下后直接静止。

半隐式欧拉法运行结果如图20、21、22和23所示。

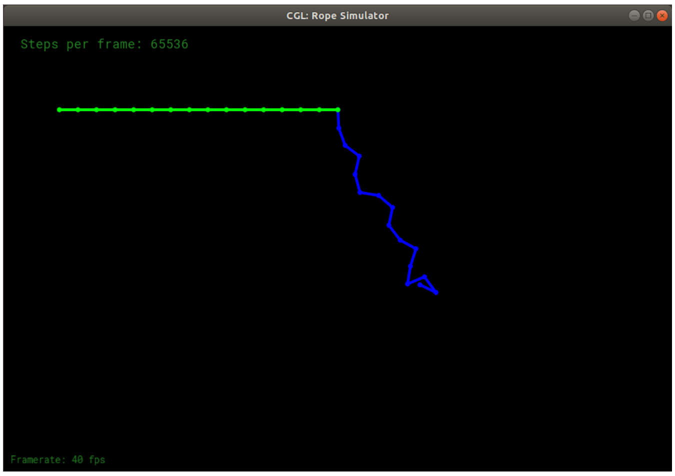

图 20: 半隐式欧拉法，$k_d = 0$, 步数 $= 65536$

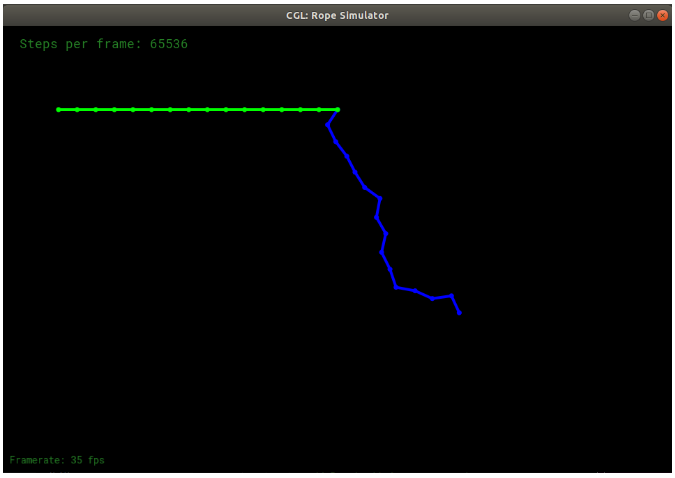

图 21: 半隐式欧拉法，$k_d = 0.001$, 步数 $= 65536$

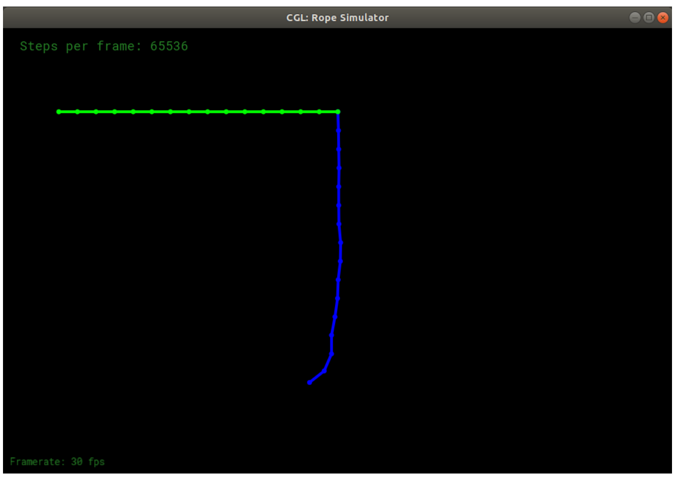

图 22: 半隐式欧拉法，$k_d = 0.01$, 步数 $= 65536$

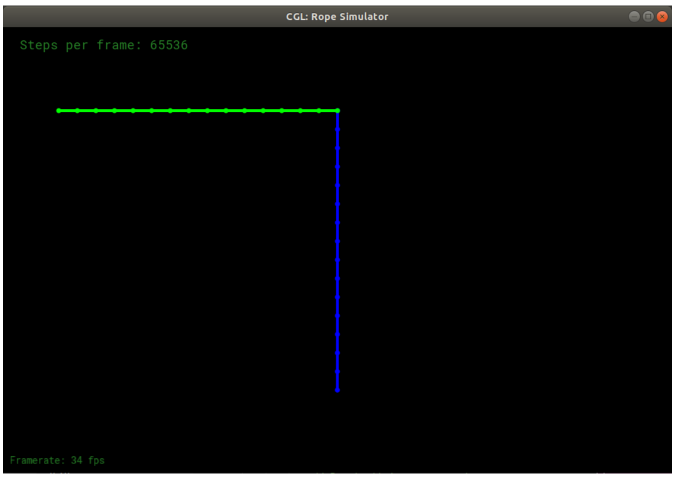

图 23: 半隐式欧拉法，$k_d = 0.1$, 步数 $= 65536$

当阻尼系数为 0 时，绳子不会停止，摆动几下后在垂直位置上下抽动。阻尼系数为 0.001 时绳子会自然摆动、直至摆动幅度逐渐缩小到垂直位置。但绳子不会完全静止，而是在垂直位置轻轻晃动。当阻尼系数为 0.01 时，绳子会在摆动几下后静止在垂直位置。当阻尼系数为 0.1 时，绳子会从水平位置直接回落到垂直位置、不发生任何摆动。

2.3 Task 3：完善 rope.cpp 中的 Rope::simulateVerlet(...)

2.3.1 将 Verlet 绳子的质点数量修改为 16

我们已在2.2.1实现了质点数量的修改，运行结果见图3。

2.3.2 实现显式 Verlet 法

与欧拉法不同，显式 Verlet 法通过式8来更新位置：

$$\mathbf{x}(t+1) = \mathbf{x}(t) + [\mathbf{x}(t) - \mathbf{x}(t-1)] + \mathbf{a}(t) * \mathrm{d}t * \mathrm{d}t \tag{8}$$

加速度 **a** 通过计算质点受到的合力、然后利用牛顿第二定律得到。我们可以使用2.2.2中同样的方式得到质点重力、弹簧力和阻力的合力计算得到加速度。此外，当 **a** 特指重力加速度时，我们可以通过解约束的方法来进一步更新质点位置，而阻尼系数 k_d 的引入需要直接修改式8：

$$\mathbf{x}(t+1) = \mathbf{x}(t) + (1-k_d)[\mathbf{x}(t) - \mathbf{x}(t-1)] + \mathbf{a}(t) * \mathrm{d}t * \mathrm{d}t \tag{9}$$

基于上述过程，当 **a** 特指重力加速度、绳子长度固定时，实现代码如下所示：

```cpp
void Rope::simulateVerlet(float delta_t, Vector2D gravity)
{
    for (auto &s : springs)
    {
        // TODO (Part 3): Simulate one timestep of the rope using explicit Verlet (
            solving constraints)
        // 在此进行质点的位置调整，维持弹簧的原始长度
        Vector2D a2b=s->m2->position - s->m1->position;
        float L=a2b.norm() - s->rest_length;
        if (s->m1->pinned)
        {
            if (s->m2->pinned)// 两端质点均固定，不调整
                continue;
            else              // 只有一端质点固定，移动L
                s->m2->position-=L*a2b.unit();
        }
        else
        {
            if (s->m2->pinned)// 只有一端质点固定，移动L
                s->m1->position+=L*a2b.unit();
            else              // 两端质点均不固定，各移动L/2
            {
                s->m1->position+=L/2*a2b.unit();
```

```
                s->m2->position-=L/2*a2b.unit();
            }
        }
    }
    for (auto &m : masses)
    {
        if (!m->pinned)
        {
            Vector2D a=gravity;// 重力加速度
            Vector2D temp_position = m->position;
            // TODO (Part 3): Set the new position of the rope mass
            // 在此计算重力影响下，质点的位置变化
            float damping_factor=0;// 阻尼系数
            m->position=m->position+(1-damping_factor)*(m->position-m->last_position)+a*
                delta_t*delta_t;
            m->last_position=temp_position;
        }
    }
}
```

设置阻尼系数为 0，步数为 64，运行结果如图24所示，可见黄色绳子正常摆动，不会停止。

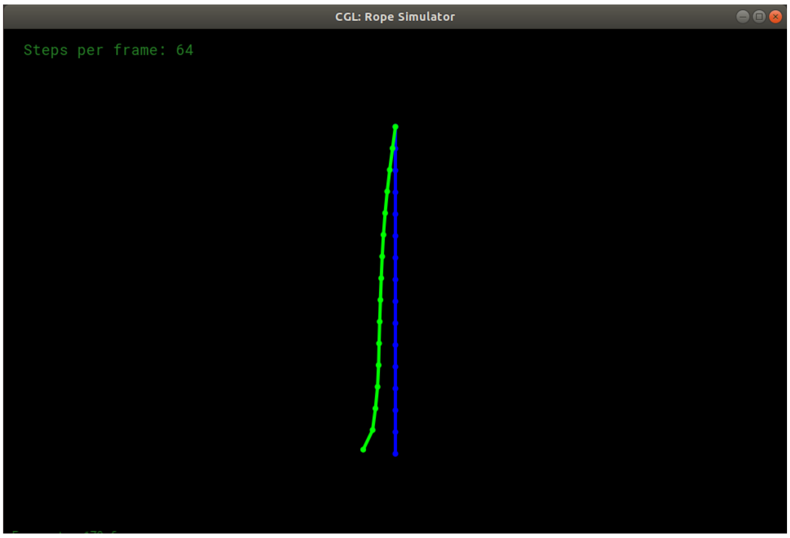

图 24: 显式 Verlet 法 1，$k_d = 0$, 步数 $= 64$

若通过计算质点的全部合力来得到 **a**，实现代码如下所示：

```
void Rope::simulateVerlet(float delta_t, Vector2D gravity)
{
    for (auto &s : springs)
    {
        // TODO (Part 3): Simulate one timestep of the rope using explicit Verlet (
            solving constraints)
        // 在此进行质点的位置调整，维持弹簧的原始长度
        Vector2D a2b=s->m2->position - s->m1->position;
        Vector2D f=s->k * (a2b/a2b.norm()) * (a2b.norm()-s->rest_length);// 弹簧力
```

```
            s->m1->forces+=f;
            s->m2->forces-=f;
        }
        for (auto &m : masses)
        {
            if (!m->pinned)
            {
                m->forces+=gravity * m->mass;// 重力
                float k_d=0.1;// 阻尼系数
                m->forces+=-k_d*m->velocity; // 阻力
                Vector2D a=m->forces/m->mass;// 加速度
                Vector2D temp_position = m->position;
                // TODO (Part 3): Set the new position of the rope mass
                m->position=m->position+(m->position-m->last_position)+a*delta_t*delta_t;
                m->last_position=temp_position;
            }
            m->forces=Vector2D(0,0);
        }
}
```

设置阻尼系数为 0，步数为 64，运行结果如图25所示，可见黄色绳子正常摆动，不会停止。

图 25: 显式 Verlet 法 2，$k_d = 0$, 步数 $= 64$

2.3.3 比较不同步数下 Verlet 绳子的摆动情况

设置阻尼系数为 0.00005，分别设置步数为 16、64、256 和 1024。固定绳子长度、加速度只考虑重力加速度的显式 Verlet 法如图26、27、28和29所示。

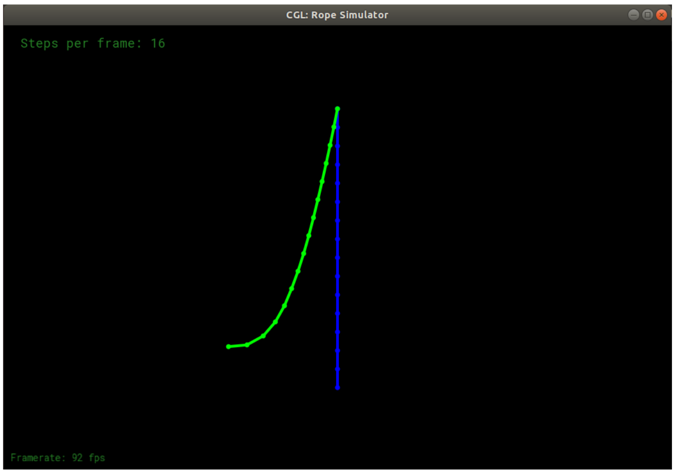

图 26: 显式 Verlet 法 1，$k_d = 0.00005$, 步数 $= 16$

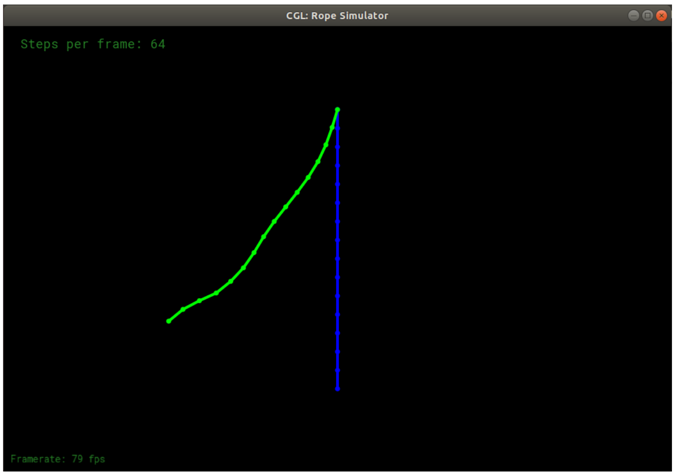

图 27: 显式 Verlet 法 1，$k_d = 0.00005$, 步数 $= 64$

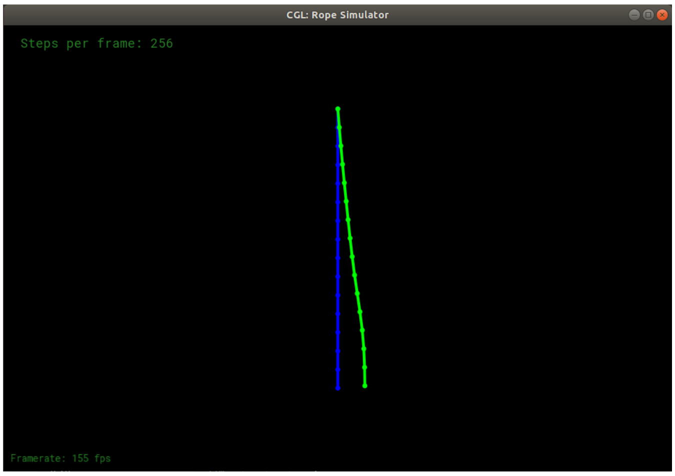

图 28: 显式 Verlet 法 1，$k_d = 0.00005$, 步数 $= 256$

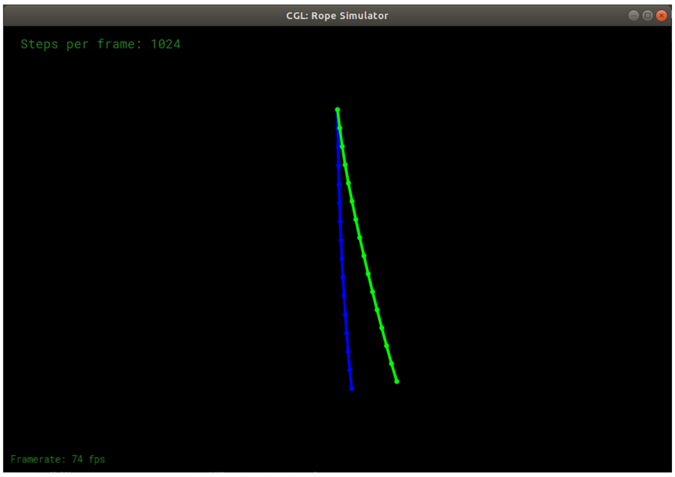

图 29: 显式 Verlet 法 1，$k_d = 0.00005$, 步数 $= 1024$

随着步数逐渐增大，绳子摆动愈加自然。此外，当步数较大时，绳子会更快静止，这时因为随着 dt 的减小，绳子每次更新时的位移更小，更有可能到达垂直位置不再摆动，而不会出现每次更新位置时都会越过垂直位置、从而左右摆动的情况。

设置阻尼系数为 0.001，分别设置步数为 16、64、256 和 1024。加速度考虑所有合力的加速度的显式 Verlet 法如图30、31、32和33所示。

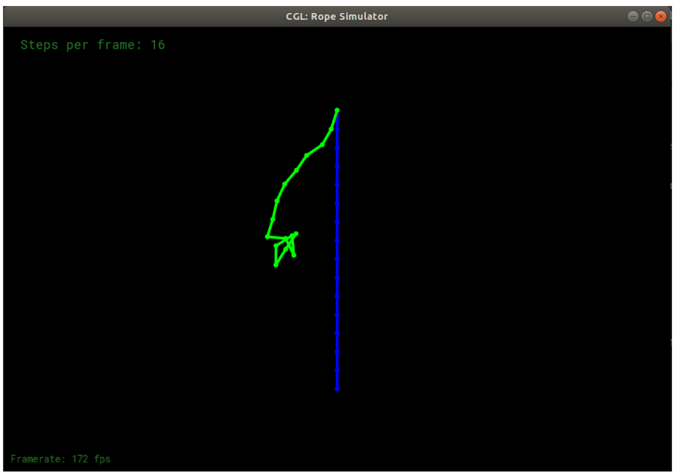

图 30: 显式 Verlet 法 2，$k_d = 0.001$, 步数 $= 16$

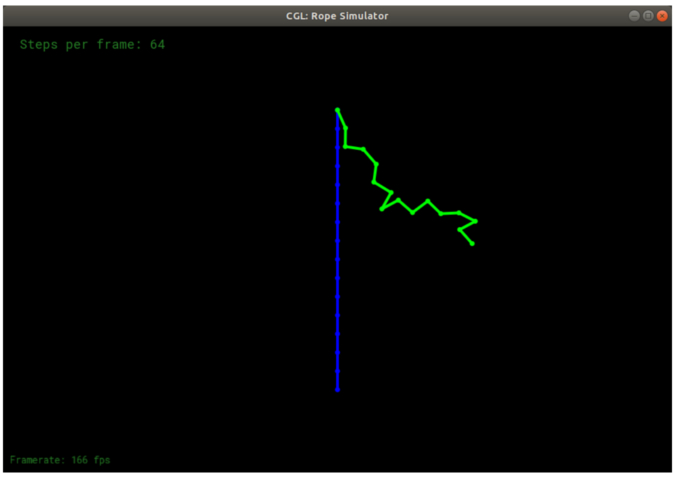

图 31: 显式 Verlet 法 2，$k_d = 0.001$, 步数 $= 64$

图 32: 显式 Verlet 法 2，$k_d = 0.001$, 步数 $= 256$

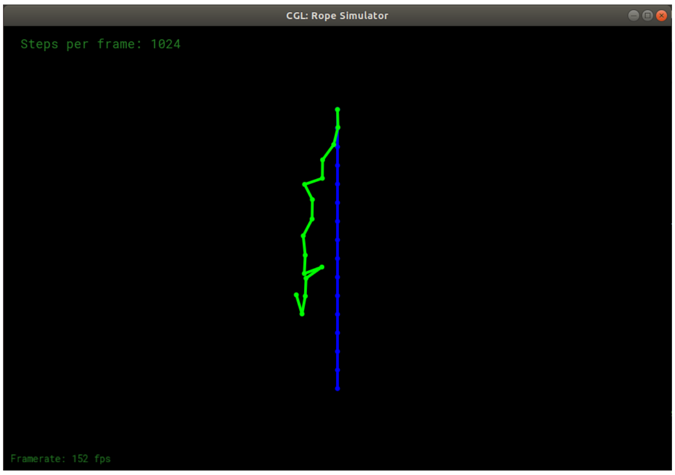

图 33: 显式 Verlet 法 2，$k_d = 0.001$, 步数 $= 1024$

同样，随着步数增大，绳子摆动逐渐自然。由于这种实现方式允许弹簧力的存在和绳子长度的变化，故质点的运动会显得有些凌乱。

2.3.4 比较不同阻尼系数下 Verlet 绳子的摆动情况

设置步数为 256，分别设置阻尼系数为 0、0.00005、0.0005 和 0.005。固定绳子长度、加速度只考虑重力加速度的显式 Verlet 法如图34、35、36和37所示。

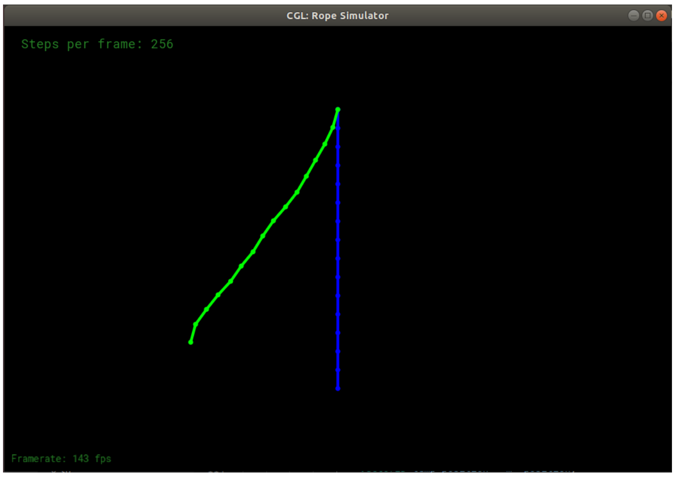

图 34: 显式 Verlet 法 1，$k_d = 0$, 步数 $= 256$

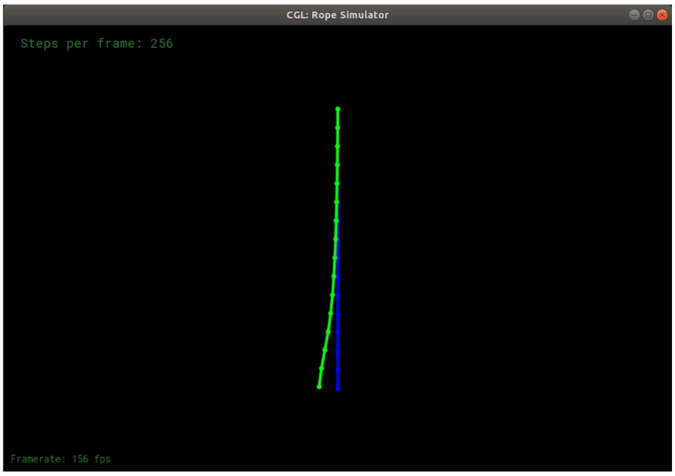

图 35: 显式 Verlet 法 1，$k_d = 0.00005$, 步数 $= 256$

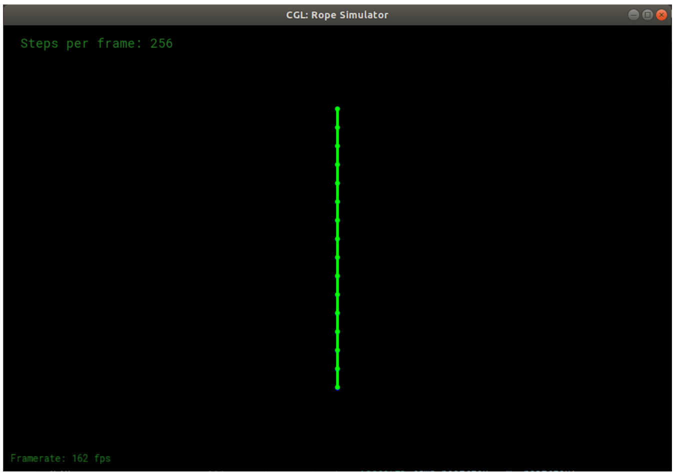

图 36: 显式 Verlet 法 1，$k_d = 0.0005$, 步数 $= 256$

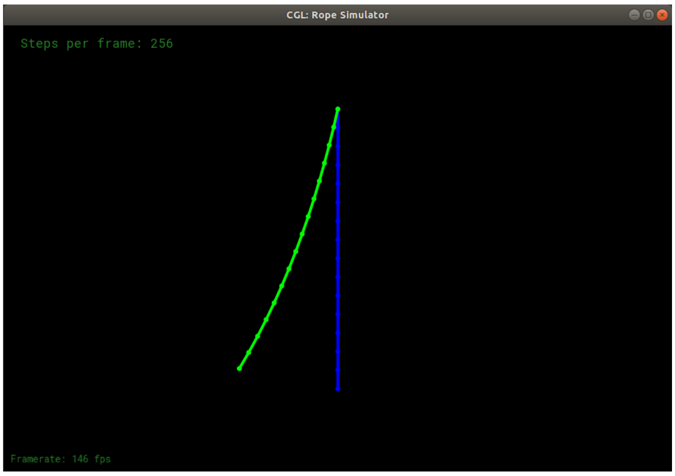

图 37: 显式 Verlet 法 1，$k_d = 0.005$, 步数 $= 256$

当阻尼系数为 0 时，绳子不会停下、一直在左右摆动；当阻尼系数为 0.00005 时，绳子在左右摆动一段时间后停下；当阻尼系数为 0.0005 时，绳子会从水平处自然落至垂直位置、不会发生摆动；当阻尼系数为 0.005 时，绳子同样会从水平处自然落至垂直位置，但速度比阻尼系数为 0.0005 的绳子慢，这时因为阻力增大、导致下落速度减少。

设置步数为 256，分别设置阻尼系数为 0、0.00005、0.0005 和 0.005。加速度考虑所有合力的加速度的显式 Verlet 法如图38、39、40和41所示。

图 38: 显式 Verlet 法 2，$k_d = 0$, 步数 $= 256$

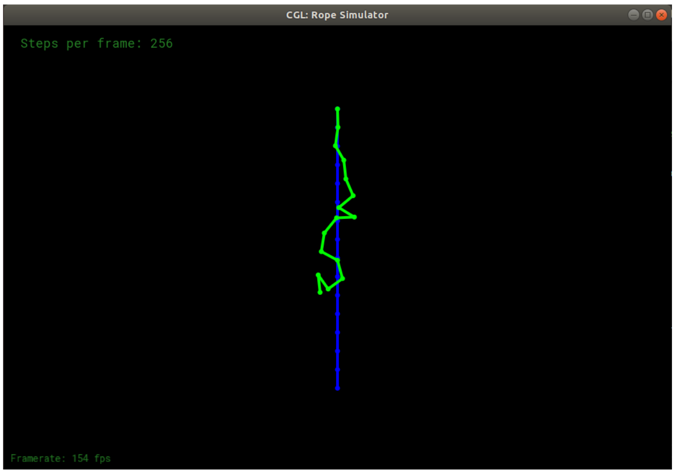

图 39: 显式 Verlet 法 2，$k_d = 0.00005$, 步数 $= 256$

图 40: 显式 Verlet 法 2，$k_d = 0.0005$, 步数 $= 256$

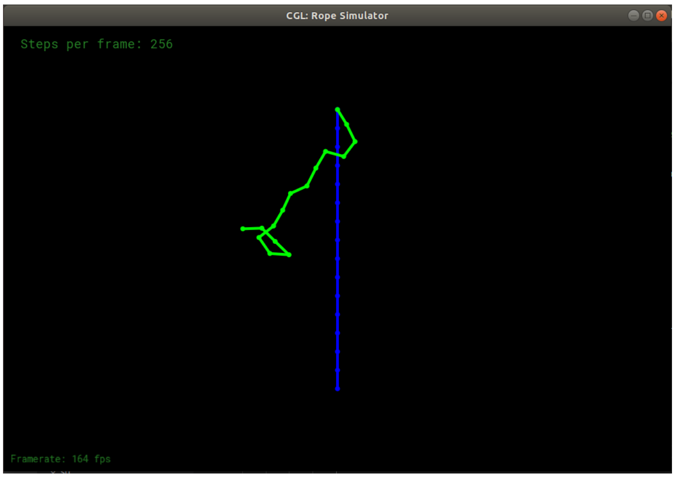

图 41: 显式 Verlet 法 2，$k_d = 0.005$, 步数 $= 256$

同样，随着阻尼系数逐渐增大，绳子恢复静止的时间逐渐缩短。

3 总结感想

本次作业是真 · 最后一次个人作业。由于在完成本次作业时还没学习弹簧质点系统的有关内容，所以我花了很长时间去掌握其中的知识点。在实现显式 Velnet 法时，就如何保持绳子长度固定的问题，我尝试了很多方法去解决，也和助教、其他同学进行了交流，并上网查阅了些资料，无果。故我还是使用2.3.2所示的代码和思想去保持绳子长度固定，但我个人认为这个方法是存在缺陷的：例如对于三个质点 $x_1x_2x_3$，为了维持 x_1x_2 的长度、调整两个质点的位置后，若再调整 x_2x_3 的长度，则会因为 x_2 位置的调整影响到绳子 x_1x_2 的长度。不过在具体实现时，发现这个问题并不明显，绳子运动比较正常。

希望我能顺利完成下一次的作业。