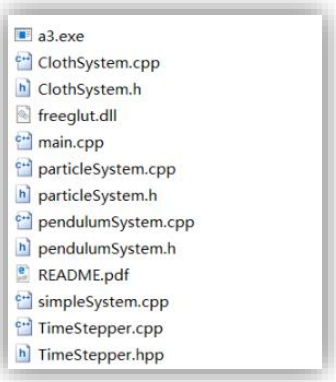
计算机图形学实习三

——Physical Simulation

一、 文件目录

这里只给出做出了修改的文件。文件目录如下：



二、 编译环境

Windows 10 + visual studio 2010

三、 实现功能

1. 常微分方程的两种积分法：Euler 法、梯形法。
2. SimpleSystem 实现，单点绕（0，0，0）旋转。
3. PendulumSystem 单摆系统和多粒子链的实现。
4. ClothSystem 粒子系统布料系统实现。
5. 三种系统之间的切换。
6. Easy 附加分：布料有风无风、光滑渲染、布料摆动、无摩擦碰撞、RK4 算法。

四、 可执行文件

1. 执行参数格式：**a3.exe [e/r/t (default r)] [步长 (default 0.04)]**。
2. 三个系统之间切换：t 键。
3. 布料系统有风无风：d 键。
4. 布料系统布料移动：s 键。
5. 布料系统布料与网格视图切换：w 键。
6. 系统重置：r 键。

五、 实现过程

1. 常微分方程的两种积分法：Euler 法、梯形法。根据给出的对应这两种算法的数值积分方程，这两个算法很容易实现。需要注意的是梯形法的提前算一步。实现体在 TimeStepper.cpp 中 ForwardEular::takeStep 和 Trapzoidal::takeStep 函数中。
2. SimpleSystem 实现，单点绕（0，0，0）旋转。这里实现的是一个绕坐标中心旋转

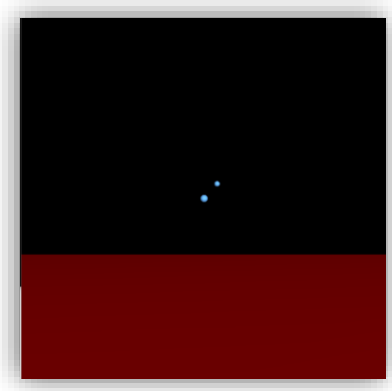
的粒子旋转运动，运动方式即

$$x=\sin t,\ y=\cos t$$

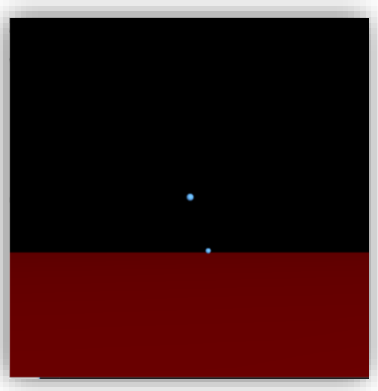
求导

$$x' = \cos t,\ y' = \sin t。即\ x' = -y,\ y' = x$$

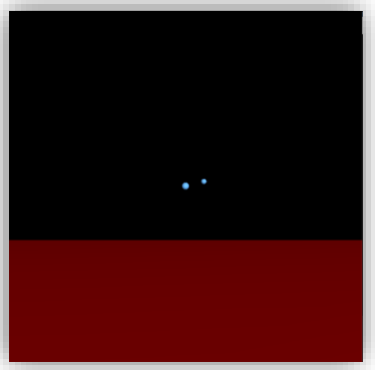
根据此实现求导函数 evalF 即可。这里比较了两种数值积分法的优劣，一段时间后的运行情况如图所示：



RK4



Euler



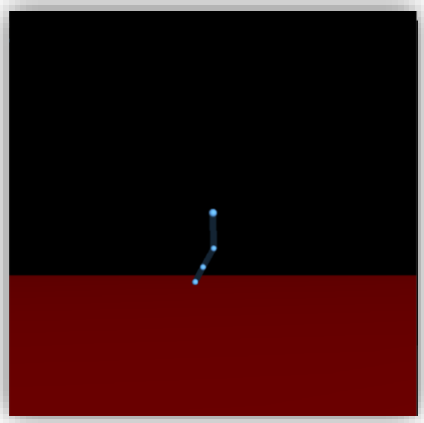
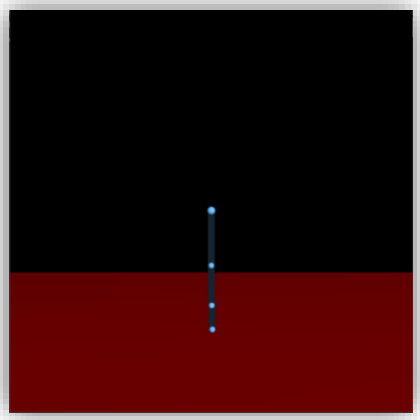
梯形

可以看出，Euler 法相对来说非常不稳定，梯形法较稳定。以下实现均基于给出的 RK4 算法函数实现。

3. PendulumSystem 单摆系统和多粒子链的实现。这个系统实现有几个困难。其一是 Position 和 velocity 的存储。对于已实现的数值积分方法，只能针对系统的 state 进行积分，因此该系统对应的 state 中必须存储点的速度 velocity 和位置 position，求导函数将对应位置的 velocity 转化为加速度 a，将对应位置的 position 转化为速度 velocity2。这里我采用了文档中的提示，在 state 中对 position 和 velocity 进行交替存储，存储和转化方式如下：

total_index	state		after evalf		after stepper
0	position[0]		volecity[0]		position[0]
1	volecity[0]		a[0]		volecity[0]
2	position[1]		volecity[1]		position[1]
3	volecity[1]		a[1]		volecity[1]
4	position[2]		volecity[2]		position[2]
5	volecity[2]	evalf	a[2]	stepper	volecity[2]
6	position[3]	--->	volecity[3]	--->	position[3]
7	volecity[3]		a[3]		volecity[3]
8	position[4]		volecity[4]		position[4]
9	volecity[4]		a[4]		volecity[4]
10	position[5]		volecity[5]		position[5]
11	volecity[5]		a[5]		volecity[5]
…	…		…		…

4. PendulumSystem 中单个质点的所受的合力，这里只考虑重力、阻尼力、弹簧弹力。单个质点质量为 mass，G=mg，这里 g 设为(0,-9.8,0)，容易得出 G。根据阻尼力公式 f = -k*v 可得出。这里封装了求出指定 total_index 的粒子所受连接到的所有弹簧合力 PendulumSystem::getSpringForce，调用该函数可获取对应质点所受弹簧力 F，则指定点所受总的合力为 **FF = G + f + F**，加速度 a=(1/m)*FF。对应求导过程参见上述第 3 点，据此实现 PendulumSsytem::evalf 函数。By the way，为了固定第一个点(0,0,0),在求导函数 evalf 中将其所受合力 FF 置零即可。
5. PendulumSystem 中质子间弹簧连接存储与单个质子弹簧力求解函数 PendulumSystem::getSpringForce。弹簧数据结构为结构体 Spring，保存该弹簧连接的两个点 leftnode,rightnode 及弹簧静态长度 static_length 和劲度系数 k_spring。粒子系统村存储弹簧的数据结构为 vector<Spring>,对应函数 addSpring 用于添加两个粒子之间的弹簧。在构造质点的过程中使用该函数将弹簧存至 vector<Spring>中。求解单个粒子所受所有弹簧的合力，即遍历所有连接到该粒子的弹簧，将其对该粒子产生的弹力累加即可。对应函数实现为 getSpringForce。
6. PendulumSystem 弹簧绘制，简单绘制粗直线即可，对应函数为 PendulumSystem::drawSpring。
7. PendulumSystem 实现了一个不动点在（0，0，0），其余点顺次在 X 轴上，重力朝 Y 轴负向竖直向下的四质点弹簧。运行过程效果如图（左图跳动中，右图静止）：

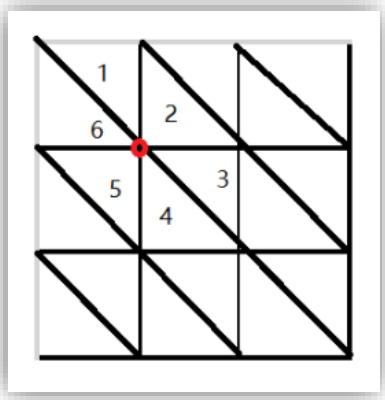



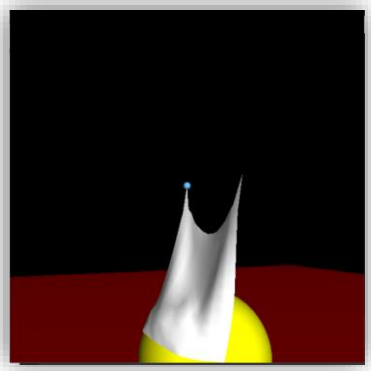
8. ClothSystem 粒子系统布料系统实现。这个系统的实现过程主要是基于原点的 pendulumSystem 系统，只是扩展到了二维网格。这里需要注意的是指定粒子的状态 position 和 velocity 仍是按照 PendulumSystem 的方式线性存储在 state 容器中的，线性坐标为 total_index，二维坐标为(i,j)，相互转换方式为 total_index=ClothSystem::indexOf(i,j)。
9. ClothSystem 中弹簧存储与弹簧力的设置。横向质点设置为固定个数(14 个)，简单改下构造函数即可扩展。构造函数中，结构弹簧、抗剪弹簧、抗弯弹簧依次添加入存储弹簧的 vector 中，按照 横向质点个数*横向质点个数 的质点网格进行设置。为了简化之后针对单个粒子寻找所连接的所有弹簧的搜索过程，这里用一张二维表 vector<vector<int>> p2p_spring 记忆化存储，p2p_spring[i]即为 total_index=i 的点所连接的弹簧数组（数组中所存 int 值为存储弹簧的线性 vector 下标）。这样一来，在 getSpringForce 函数中不必搜索所有弹簧，仅通过查表就可以获得单个粒子连接到的所有弹簧，遍历累加即可。

point	spring[0]	spring[1]	spring[2]	spring[3]
0	0	1	2	
1	9	3		
2	4			
3	5	3	4	2
4	5	6		
5	7	0	1	
6	9	7		

10. ClothSystem::evalf 函数实现过程与 pendulumSystem 实现类同。后续有附加功能的扩展，另外，在此也将布料上端的两个固定点所受合力置为 0.
11. 附加功能实现一：光滑渲染。这里我采用的是同学提供的方法来调整单个质点的法向量。步骤为：求得所有面的正朝向的法向量并归一化，单个质点的法向量为该点所在所有平面求得的法向量求和并归一化，这样取得的均值可以使效果非常光滑。示例网格如下，图中点的法向量为所在的 6 个面的所有归一化的法向量累加之和并再次归一化，即

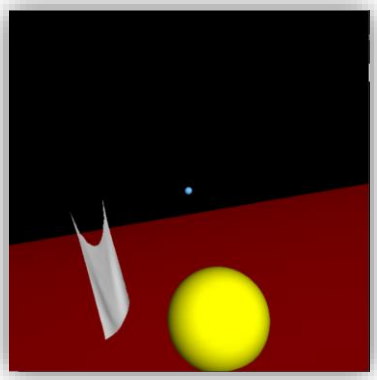
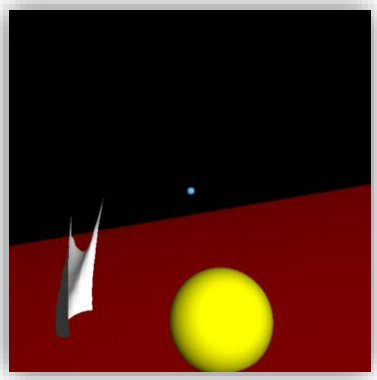
$$N = (N_1.\text{normalized} + N_2.\text{normalized} + N_3.\text{normalized} + N_4.\text{normalized} + N_5.\text{normalized} + N_6.\text{normalized}).\text{normalized}$$



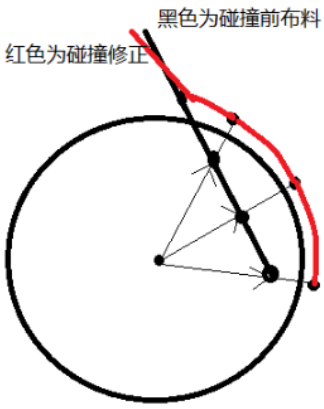


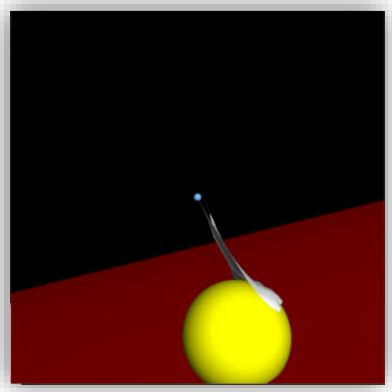
相关实现为 clothSystem::drawclothes()。

12. 附加功能实现二：布料有风无风。这个功能是利用 open_wind 指示变量并初始化为 false，在 OpenGL 键盘监视函数中监听 d 键并更改 open_wind 的值。每次调用 evalf 函数求解合力时随机给网格中的一些点添加朝向固定的“风力”，因为在一定范围内随机，会对布料产生抖动效果。效果展示（左边无风静止，右边有风抖动）：

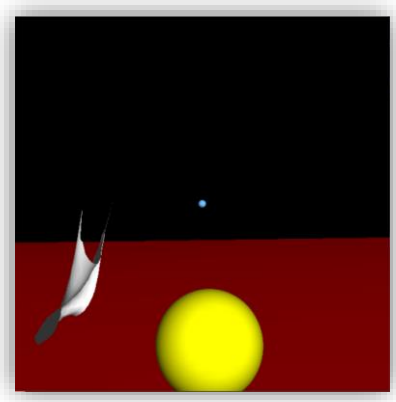



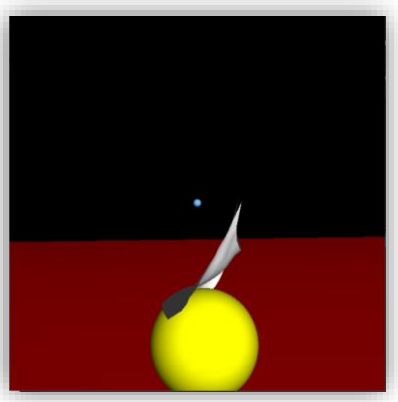
13. 附加功能实现三：布料无摩擦碰撞。这里实现的是示例的与球体无摩擦碰撞。布料上的点在球体之内时做出向球表面的映射，即将原先从球心到布料质点的向量延长至球半径 R+无穷小值 e 即可。相应实现为 clothSystem::draw 函数中对应部分。原理和效果如图：

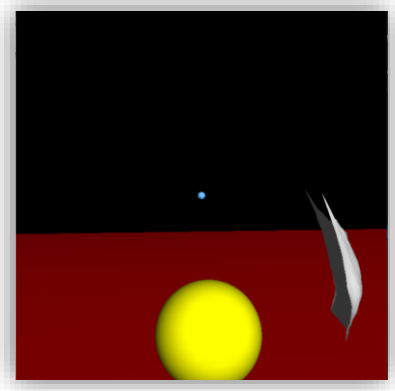




14. 附加功能实现四：布料摆动。在求导时为布料的上部两个固定点设置速度即可，此外设置固定范围相应调整速度的方向，在 OpenGL 键盘监听事件中监听 s 键开启移动，相应实现在 evalF 函数中。效果如下：







15. RK4 算法实现。相应实现部分为 TimeStepper.cpp 中 RungeKutta4::takeStep 函数。（不够稳定，未完美实现）。

六、问题与解决

1. 布料系统绘制速度较慢，release 编译下速度较快，此外，因为 windows 10 系统在电量有限时会降低显卡性能，这个也会影响到布料系统的绘制速度。这个是我之前的一个认识误区。
2. 光滑绘制布料。这里要感谢任哲旋同学给我提供的算法思路。
3. 自己实现的 RK4 算法不够稳定。
4. Euler 法和梯形法没有找到合适的参数稳定展示。
5. 碰撞检测时布料下方点存在“抖动”情况，未解决。

作者：AnDJ