可视计算与交互概论大作业报告

2023 年 1 月 10 日

1 选题介绍 (D.3)

Fast Simulation of Mass-Spring Systems

http://tiantianliu.cn/papers/liu13fast/liu13fast.html

该论文使用加速弹簧质点解算快速稳定模拟弹簧质点系统，与隐式欧拉方法相比，能够得到更高的帧率，视觉效果上达到与真实相近的结果，兼顾了隐式欧拉方法的稳定的同时计算成本更低.

2 实现思路

显示欧拉算法实验结果稳定性比较差, 而隐式积分方法需要求解大型方程组, 论文将隐式欧拉方法转化为对隐式欧拉积分的优化公式进行最小化的过程, 具体如下:

考虑具有 n 个点的三维 Mass Spring 系统，我们使用 $\mathbf{q} \in \mathbb{R}^{3n}$ 表示某时刻点的坐标，$\mathbf{v} \in \mathbb{R}^{3n}$ 表示速度，根据经典运动学规律，在 t 时刻，我们可以将位置以及速度的更新记为:

$$\begin{aligned} \mathbf{v}_{t+\Delta t} &= \mathbf{v}_t + \Delta t\ \mathbf{M}^{-1}\ \mathbf{f}(\mathbf{q}_t) \\ \mathbf{q}_{t+\Delta t} &= \mathbf{q}_t + \Delta t\ \mathbf{v_t} \end{aligned} \tag{1}$$

其中 $\mathbf{f}_t \in \mathbb{R}^{3n}$ 为系统所受合力，$\mathbf{M} \in \mathbb{R}^{3n \times 3n}$ 是一个对角矩阵，给定第 i 个节点（$i = 1, 2, 3, ..., n$ ），其质量为 m_i ，那么矩阵 $\mathbf{M}$ 可表示为（$\otimes$ 表示克罗内克积）:

$\mathbf{M} = diag\{m_1, m_2, ..., m_n\} \otimes \mathbf{I}_3$

式 (1) 为显式欧拉积分的方法，即当前时刻 $t+\Delta t$ 的状态只依赖于上一时刻 t ，然而这种显示积分方法数值误差较大，尤其在步长 Δt 较大的情况下。因此需要使用隐式积分的方法进行求解，即：

$$\begin{aligned}\mathbf{v}_{t+\Delta t} &= \mathbf{v}_t + \Delta t\ \mathbf{M}^{-1}\ \mathbf{f}(\mathbf{q}_{t+\Delta t}) \\ \mathbf{q}_{t+\Delta t} &= \mathbf{q}_t + \Delta t\ \mathbf{v}_{\mathbf{t}+\mathbf{t}}\end{aligned} \tag{2}$$

消去 $v_{t+\Delta t}$ 整理得:

$\mathbf{q}_{t+\Delta t} = \mathbf{q}_t + \Delta t\ \mathbf{v}_{\mathbf{t}} + \Delta t^2\ \mathbf{M}^{-1}\mathbf{f}(\mathbf{q}_{t+\Delta t})$

继续利用 $\Delta t\ \mathbf{v}_t = \mathbf{q}_t - \mathbf{q}_{t-\Delta t}$，可得：

$\mathbf{q}_{t+\Delta t} - 2\mathbf{q}_t + \mathbf{q}_{t-\Delta t} = \Delta t^2\mathbf{M}^{-1}\ \mathbf{f}(\mathbf{q}_{t+\Delta t})$

记 $\mathbf{x} := \mathbf{q}_{t+\Delta t}$ ， $\mathbf{y} := 2\mathbf{q}_t - \mathbf{q}_{t-\Delta t}$ ，可将上式整理为：

$$\mathbf{M}(\mathbf{x}-\mathbf{y}) = \Delta t^2\mathbf{f}(\mathbf{x}) \tag{3}$$

我们需要求方程 (3) 中的变量 $\mathbf{x}$，而这个方程中的 $\mathbf{f}(\mathbf{x})$ 包含的非线性部分（例如弹簧的弹性力）使得隐式欧拉十分消耗计算资源。我们假设 $\mathbf{f}$ 中只包含保守力，那么 $\mathbf{f}$ 应该负向于系统能量 $\mathbf{E}$ 的梯度，即：$\mathbf{f}(\mathbf{x}) = -\nabla E(\mathbf{x})$

考虑如下方程 $\mathbf{g}(\mathbf{x})$ 和其梯度 $\nabla_{\mathbf{x}}\mathbf{g}(\mathbf{x})$ ：

$$\begin{aligned}\mathbf{g}(\mathbf{x}) &= \frac{1}{2}(\mathbf{x}-\mathbf{y})^T\mathbf{M}(\mathbf{x}-\mathbf{y}) + \Delta t^2 E(\mathbf{x}) \\ \nabla\mathbf{g}(\mathbf{x}) &= \mathbf{M}(\mathbf{x}-\mathbf{y}) - \Delta t^2\mathbf{f}(\mathbf{x})\end{aligned} \tag{4}$$

$\nabla_{\mathbf{x}}\mathbf{g}(\mathbf{x}) = \mathbf{0}$ 等价于 (3) 式，而 $\nabla_{\mathbf{x}}\mathbf{g}(\mathbf{x}) = \mathbf{0}$ 意味着求解 $\mathbf{g}(\mathbf{x})$ 的局部极值点，这里需要注意，实际上 $\nabla_{\mathbf{x}}\mathbf{g}(\mathbf{x}) = \mathbf{0}$ 并不意味着得到 $\mathbf{g}(\mathbf{x})$ 的全局极值点，这需要对 $\mathbf{g}(\mathbf{x})$ 的 Hessian 矩阵做出限制（即半正定或半负定），而弹簧质点模型的 Hessian 矩阵和弹簧的的参数是有关系的，在调整参数的情况下，我们可以认为求解式 (3) 等价于最优化式 (4)：

$$\mathbf{M}(\mathbf{x}-\mathbf{y}) = \Delta t^2\mathbf{f}(\mathbf{x}) \Leftrightarrow argmin_{\mathbf{x}}\mathbf{g}(\mathbf{x}) \tag{5}$$

弹簧质点模型中对弹簧弹性势能的定义如下：

$$E_{spring} = \frac{1}{2}k(||\mathbf{x}_i - \mathbf{x}_j|| - r)^2 \tag{6}$$

这个部分是 $\mathbf{g}(\mathbf{x})$ 中的非线性部分，于是我们将 mass-spring 系统的隐式积分问题转化成一个非线性优化问题。

事实上，我们需要求 (4) 的最小值，首先需要得到弹性势能的最小值：

$$(||\mathbf{x}_i - \mathbf{x}_j|| - r)^2 = min_{||\mathbf{d}||=r}||\mathbf{x}_i - \mathbf{x}_j - \mathbf{d}||^2 \tag{7}$$

可以通过三角不等式等号成立条件易得：$\mathbf{d} = r(\mathbf{x}_{ij})/\left\|\mathbf{x}_{ij}\right\|$，这一步便是 local solver 的步骤。

(4) 中 E(x) 由 E_{ext} 和 E_{spring} 组成，故式 (5) 可改写为：

$$\begin{aligned} argmin_{\mathbf{x}}\mathbf{g}(\mathbf{x}) &= argmin_{\mathbf{x}}\frac{1}{2}(\mathbf{x}-\mathbf{y})^T\mathbf{M}(\mathbf{x}-\mathbf{y}) + \Delta t^2(E_{spring} + E_{ext}) \\ &= argmin_{\mathbf{x}}(min_{\mathbf{d}}\frac{1}{2}(\mathbf{x}-\mathbf{y})^T\mathbf{M}(\mathbf{x}-\mathbf{y}) - \Delta t^2\mathbf{f}_{ext}^T\mathbf{x} \\ &+ \sum_{i=1}^{s}\frac{k_i}{2}\Delta t^2||\mathbf{x}_{i1} - \mathbf{x}_{i2} - \mathbf{d}||^2) \end{aligned} \tag{8}$$

(8) 式我们已经把能量最小值带入，只需求解这个二次优化问题，这一步骤便是 global solver。

Fast Mass Spring 算法通过 local(寻找最佳弹簧方向)-global(确定节点位置) 的交替迭代加速解算过程，同时由于线性系统的矩阵独立于当前状态，我们可以预先进行 Cholesky 分解得到稀疏矩阵，进一步加速。

2.1 算法亮点

fast mass spring 算法中 local solver 是对系统中每个弹簧系统进行约束并依次极小化的非线性优化问题求解，global solver 是在 local solver 的基础上进行二次优化，考虑整体弹簧质点系统在优化公式的最小化。算法实现在 lab4 中 task.cpp 文件的 AdvanceMassSpringSystem 函数中，为了讲解代码需要对 $g(x)$ 进行变形和换元，数学过程如下：

$$\begin{aligned} \mathbf{g}(\mathbf{x}) &= \frac{1}{2}(\mathbf{x}-\mathbf{y})^T\mathbf{M}(\mathbf{x}-\mathbf{y}) - \Delta t^2\mathbf{f}_{ext}^T\mathbf{x} + \sum_{i=1}^{s}\frac{k_i}{2}\Delta t^2||\mathbf{x}_{i1} - \mathbf{x}_{i2} - \mathbf{d}||^2 \\ &= \frac{1}{2}(\mathbf{x}-\mathbf{y})^T\mathbf{M}(\mathbf{x}-\mathbf{y}) - \Delta t^2\mathbf{f}_{ext}^T\mathbf{x} + \frac{\Delta t^2}{2}\mathbf{x}^T\mathbf{L}\mathbf{x} - \Delta t^2\mathbf{x}^T\mathbf{J}\mathbf{d} + constant_0 \\ &= \frac{1}{2}\mathbf{x}^T\mathbf{M}\mathbf{x} - \mathbf{x}^T\mathbf{M}\mathbf{y} - \Delta t^2\mathbf{f}_{ext}^T\mathbf{x} + \frac{\Delta t^2}{2}\mathbf{x}^T\mathbf{L}\mathbf{x} - \Delta t^2\mathbf{x}^T\mathbf{J}\mathbf{d} + constant_1 \\ &= \frac{1}{2}\mathbf{x}^T(\mathbf{M} + \Delta t^2\mathbf{L})\mathbf{x} - \Delta t^2\mathbf{x}^T\mathbf{J}\mathbf{d} - \mathbf{x}^T(\mathbf{M}\mathbf{y} + \Delta t^2\mathbf{f}_{ext}^T) + constant_1 \end{aligned} \tag{9}$$

其中矩阵 **L** 和矩阵 **J** 由下式换元得到：

$\frac{\Delta t^2}{2}\sum_{i=1}^{s} k_i||\mathbf{x}_i - \mathbf{x}_j - \mathbf{d}||^2 = \frac{\Delta t^2}{2}\mathbf{x}^T\mathbf{L}\mathbf{x} - \Delta t^2\mathbf{x}^T\mathbf{J}\mathbf{d} + constant_0$

于是我们可以得到矩阵 $\mathbf{L} = (\sum_{i=1}^{s} k_i\mathbf{A}_i\mathbf{A}_i^T) \otimes \mathbf{I}_3 \in \mathbb{R}^{3n\times 3n}$，矩阵 $\mathbf{J} = (\sum_{i=1}^{s} k_i\mathbf{A}_i\mathbf{S}_i^T) \otimes \mathbf{I}_3$，向量 $\mathbf{A}_i \in \mathbb{R}^n$ 是一个指示向量，其第 j 个分量可表示为：

$$\mathbf{A}_{i,j} = \begin{cases} 1, & j == i_1 \\ -1, & j == i_2 \\ 0, & otherwise \end{cases} \tag{10}$$

where j=0,1,2,...,n-1

向量 $\mathbf{S}_i \in \mathbb{R}^s$，其中第 j 个分量可表示为 $\mathbf{S}_{i,j} = \delta_{i,j}, j = 0, 1, 2, ..., s-1$

对于 global solver 来说，我们只需求出 $\mathbf{x}$ 其满足 $\nabla\mathbf{g}(\mathbf{x}) = \mathbf{0}$ 即可，而这个方程可变形为：

$$\begin{aligned} &\nabla\mathbf{g}(\mathbf{x}) = \mathbf{0} \\ \Leftrightarrow &(\mathbf{M} + \Delta t^2\mathbf{L})\mathbf{x} = \Delta t^2\mathbf{J}\mathbf{d} + \mathbf{M}\mathbf{y} + \Delta t^2\mathbf{f}_{ext}^T \\ \Leftrightarrow &\mathbf{Q}\mathbf{x} = \mathbf{b} \end{aligned} \tag{11}$$

其中 $\mathbf{Q} = \mathbf{M} + \Delta t^2\mathbf{L} \in \mathbb{R}^{3n\times 3n}$，$\mathbf{b} = \Delta t^2\mathbf{J}\mathbf{d} + \mathbf{M}\mathbf{y} + \Delta t^2\mathbf{f}_{ext}^T \in \mathbb{R}^{3n\times 1}$. 实际上矩阵 $\mathbf{Q}$ 在迭代的过程当中是不变的, 可以对其进行预计算, 同时 $\mathbf{Q}$ 作为实对称矩阵有非常好的一些性质, 可以对其进行 cholesky 分解, 即 $\mathbf{Q} = \mathbf{L}\mathbf{L}^T$，其中 $\mathbf{L}$ 是一个下三角矩阵，通过 cholesky 分解我们把原来解线性方程组的时间复杂度从 $\mathbf{O}(n^3)$ 降到了 $\mathbf{O}(n^2)$，这也是 Fast Mass Spring 算法效率高的一个原因。

具体到代码实现, 我们需要先计算出 $\mathbf{L}$,$\mathbf{M}$,$\mathbf{J}$, $\mathbf{f}_{ext}$, 然后计算出 $\mathbf{Q}$, 并对其进行 cholesky 分解。$\mathbf{y}$ 的计算需要用到 $\mathbf{q}_{t-\Delta t}$ 和 $\mathbf{q}_t$，在代码中我使用 prevState 和 currState 命名，其中 prevState 存储在 CaseMassSpring 这个类中，作为函数输入传递进来，currState 在函数内部定义，由 system.Position 构造即可。阻尼模型上选择了 ether drag 的模型, 将系统中的阻力通过速度的衰减率来实现到模型中, 这个速度衰减率被命名为 α, 是 MassSpringSystem 的成员变量，可以通过 GUI 界面进行调整，我们之前的推导实际上是没有考虑阻尼的，而加上阻尼只需要对 (2) 式替换 v_t 时使用 $v_t^{'} = \alpha v_t$ 即可，故之后的 $\mathbf{y} = (1+\alpha) * currState - \alpha * prevState$.

接下来就到了 local-global 循环迭代的步骤, 迭代次数 Iteration 在 GUI 界面中可以控制，local solver 通过三角不等式得到最优解 $\mathbf{d} = \frac{\mathbf{x}_i - \mathbf{x}_j}{||\mathbf{x}_i - \mathbf{x}_j||}r$，通过 currState 便可以计算出 $\mathbf{d}$，global solver 步骤中先计算 $\mathbf{b} = \Delta t^2\mathbf{J}\mathbf{d} +$

$\mathbf{My}+\Delta t^2\mathbf{f}_{ext}^T$, 然后通过 cholesky solver 解方程更新 currState 的值即可。最终迭代完成后将 currState 的值传回 system, 更新其 Position 和 Velocities。

此外, 在求解过程中需要注意的是 currState 在 local-global 循环求解的过程中会因重力影响越来越向下, 即便在每次循环中求解之后强制固定 Fixed Points 的坐标, 这种情况在 Iteration 较大的时候仍然会十分明显, 故我们需要对 $\mathbf{Qx}=\mathbf{b}$ 的求解进行修改, 将固定点前后确定在同一位置。我们可以在 (9) 式中添加一个额外的惩罚项:

$$\frac{w}{2}tr((\mathbf{Cx}-\mathbf{Cb})^T(\mathbf{Cx}-\mathbf{Cb}))=\frac{1}{2}tr(\mathbf{x}^T(w\mathbf{C}^T\mathbf{C})x)-tr(\mathbf{x}^Tw\mathbf{C}^T\mathbf{C}b)+constant \tag{12}$$

其中 w 为一个很大的常数, $\mathbf{C}\in R^{|Fix|*n}$, 每个固定点对应一行, 每行只有一个 1, 其余都是 0, 只有在对应列与点编号相等时为 1。之后我们对 (12) 进行求导, 可知 $\mathbf{Q}^*=\mathbf{Q}+\mathbf{Qp}$, 其中 $\mathbf{Qp}$ 为在每个固定点 i 对应的 (i,i) 位置为 w, 其余位置为 0 的矩阵, $\mathbf{b}$ 则需要在每个固定点 i 对应的 $3*i, 3*i+1, 3*i+2$ 的位置分别加上 $w*x[3*i], w*x[3*i+1], w*x[3*i+2]$ 即可。

2.2 编译环境

编译环境以及使用的第三方库与 lab4 的要求一致.

2.3 运行环境及运行效率

运行环境与 lab4 要求一致.

运行效率与显式欧拉 (Explicit Euler) 方法和隐式欧拉牛顿 (Newton) 法 (最大迭代次数设置为 100) 对比表格如下 (单位: s): 显式欧拉运行速

表 1: 运行时间对比

EE(1000It)	**N(100MaxIt)**	**FMS(10It)**	**FMS(100It)**	**FMS(1000It)**
0.057	5.5	0.008	0.08	0.8

度虽然较快, 但是结果的稳定性较差, 牛顿法运行速度即便是与 1000 次的 FastMassSpring 相比, 也慢了数倍, 同时在视觉效果上相差并不大。

文件夹中包含有 15, 25, 40,70,100 这五种迭代次数下的结果 (文件名对应迭代次数)。如果需要修改 Iteration 的值查看其它迭代次数下的结果可

以修改 AdvanceMassSpringSystem 函数。

3 对实验结果的思考

相较于牛顿法求精确解，Fast Mass Spring 算法在大部分情形下，可以使用短数倍的时间实现与精确解十分相近的结果，算法通过将非线性问题分解为若干个容易解的子非线性问题，再最后化成一个二次优化问题的思路，节省了大量计算资源。其与传统隐式欧拉牛顿法的思路差别主要为最小化 **E** 的思路，在求解方程处，采用了预处理 cholesky 分解来替代求 **Q** 的逆，并且考虑到真实物体在弹簧质点系统建模情况下 **Q** 的稀疏性，采用了 Eigen 的 SparseMatrix 使得时间复杂度降到 $O(n^{1.5})$，对于固定连接和点可以采用能量惩罚项实现更好地效果，与刚性物体碰撞模型或许也可以采用固定接触面的方法实现，接下来的问题或许是弹簧质点系统与非刚性材料碰撞，两个系统之间的碰撞以及弹簧质点系统与刚性物体有吸附性时碰撞(或者较大摩擦，比如湿毛巾的碰撞模拟)，要对这些碰撞产生好的效果可能需要对系统更复杂的建模，我阅读了Blender Tutorial有很多布料的模拟情形和 GPU-based simulation of cloth wrinkles at submillimeter levels 中高分辨率下的布料碰撞模拟，我认识到在本论文之后布料模拟还有很多新的发展，但某些复杂情形可能还无法在 Real Time 情况下得到好的效果。通过这次大作业，我了解了许多 Simulation 的知识，对这个 Topic 有了更深的认识和更多的兴趣。