Lab4-Report

王想 2100013146

1 Task 1: Inverse Kinematics

1.1 Sub-Task 1: Forward Kinematic

除根节点外,一个节点的全局旋转角度就是其父节点的旋转角度加上该节点的局部旋转角度, 所以

$$GlobalRotation[i] = normalize(GlobalRotation[i-1] * LocalRotation[i])$$

一个节点的全局坐标就是其父节点的全局坐标加上局部的偏移量, 而局部的偏移量就是把相对父亲的偏移量旋转父节点的全局旋转角度, 所以

$$GlobalPosition[i] = GlobalPosition[i-1] + rotate(GlobalRotation[i-1], LocalOffset[i])$$

实现效果结合CCD IK 与FABR IK展现。

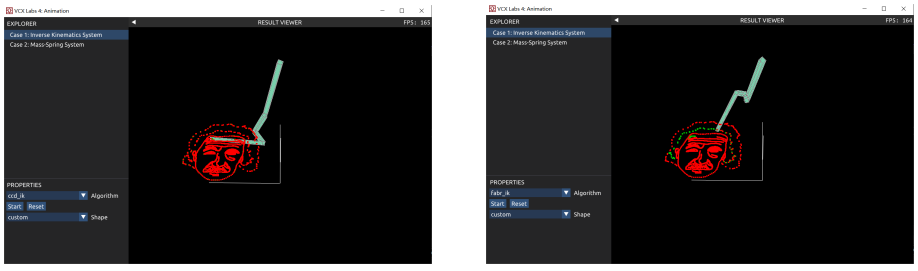
1.2 Sub-Task 2: CCD IK

从倒数第二个节点循环到第一个节点, 每个节点都计算该节点到最后一个节点的方向向量(记为toEnd)与该节点到目标的方向向量(记为toTarget), 将局部旋转设置为从原先的局部旋转角度加上从toEnd旋转到toTarget的旋转角度, 每计算完一个节点后就从该节点开始执行Forward Kinematic。

1.3 Sub-Task 3: FABR IK

FABR IK算法主要分为两个部分,从后往前"拉扯",从前往后"拉扯",但原理相似。以从后往前为例, 先把最后一个节点直接"拉"到目标位置上, 从倒数第二个节点循环到第一个节点, 每个节点都计算目标位置到该节点的方向向量, 然后把该节点的全局位置设置成从目标位置沿着方向向量偏移关节长度的距离, 把目标位置设置成此节点的位置,进行以上步骤循环。从前往后类似上述操作进行处理。最后通过全局坐标计算旋转角度,并执行一次Forward Kinematic。

效果如下[1]:



(a) **CCD IK** (b) **FABR IK**

图 1: **Results**

[1]动态展示请查看ccd.mp4和fabr.mp4

1.4 Sub-Task 4: 自定义曲线

参照注释中给出的网址,进行函数定义与曲线绘制,自定义曲线形状为五瓣花。

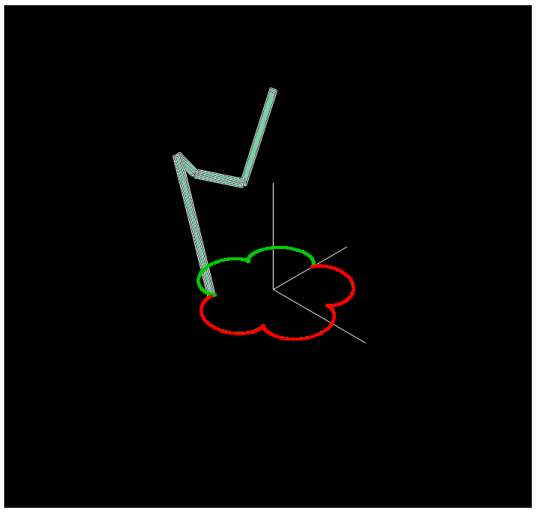
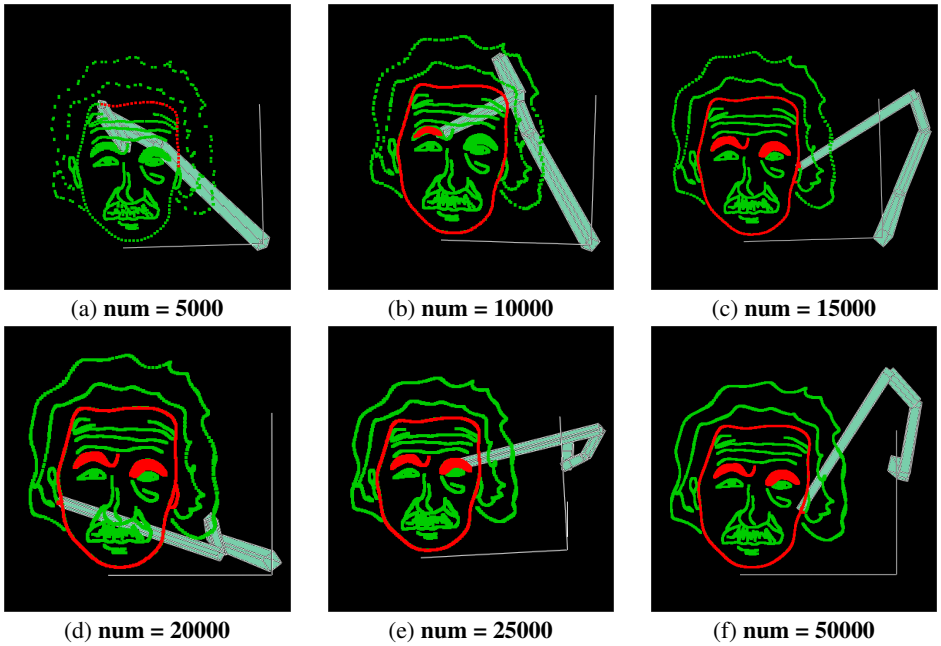


图 2: **Flower**

1.4.1 Sub-Task 4.1

当采样点足够多时,采样点不均匀的效果就不明显了,所以最简单粗暴的方法就是增加采样点数量, 可以看到随着采样点数量增加,头发逐渐变密集,采样点不均匀的效果在$num = 20000$时就已经几乎看不出来了, 当$num = 50000$时,头发部分都已经看上去是连续曲线,完全看不到采样点不均匀的效果。



(a) **num = 5000** (b) **num = 10000** (c) **num = 15000**

(d) **num = 20000** (e) **num = 25000** (f) **num = 50000**

图 3: **Results**

2 Questions

Q1 无论是CCD IK还是FABR IK,从理论上分析IK结果, 当目标距离过远时,两种算法都会驱使所有关节会完全伸直, 形成一条直线段,指向目标位置,实验也证明了上述分析结果,如下图所示。

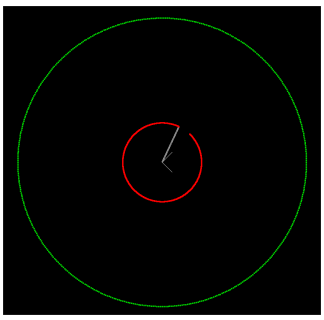


图 4: **Target too far**

Q2 实验默认设置的最大迭代次数是100,此时看不出来迭代次数对结果的影响, 将最大迭代次数设置成2之后差距便显现出来。迭代次数为2时,对于部分曲线, CCD IK会偶尔出现错误,但FABR IK仍然能够保持准确,并且FABR表现得更加稳定。从理论上分析,此现象的原因是FABR IK算法的收敛速度更快,因此FABR IK所需要的迭代次数更少。

Q3 加入限制,先记录下前一帧的关节位置、旋转等参数, 如果前后两帧的目标点位置距离很小, 那么前后两帧关节的位置、旋转等参数差距应该较小, 此时如果解出来的结果与前一帧的结果差距过大则舍弃此解, 计算新解,直到得到差距较小的解, 以此作为后一帧的关节的参数解。

3 Task 2: Mass-Spring System

按照课件介绍的算法步骤实现即可, 前面部分的公式推导与理论证明在PPT中已经呈现, 下面重点解释Hessian矩阵的公式推导与算法实现:

$$\nabla g(\mathbf{x}) = \frac{1}{h^2} M(\mathbf{x} - \mathbf{y}) - f_{int}(\mathbf{x})$$

$$f_{int}(\mathbf{x}) = \sum_{j \in \mathbf{N_i}} f_{ij}$$

所以

$$\begin{aligned} \nabla^2 g(\mathbf{x}) =& \frac{\partial(\nabla g(\mathbf{x}))}{\partial \mathbf{x}} \\ =& \frac{1}{h^2} M\mathbf{I} - \sum_{j \in \mathbf{N_i}} \frac{\partial f_{ij}}{\partial \mathbf{x}} \end{aligned} \tag{1}$$

根据以上公式构建Hessian矩阵, 但如果是一个个元素去填充矩阵会难以处理, 由于上式是累加的形式, 容易想到遍历所有的力(即遍历所有弹簧)计算相应的偏导数, 并且每计算一个就在Hessian矩阵相应位置累加上该值, 遍历完所有边后自然就得到了正确的Hessian矩阵。根据$\mathbf{x_n}$计算$\mathbf{x_{n+1}}$的算法步骤如下:

1. 初始化$\mathbf{x_n^{(0)}} = \mathbf{x_n},\ k = 0$
2. 计算一阶导数$\nabla g(\mathbf{x_n^{(k)}})$

3. 初始化Hessian矩阵为零矩阵,把对角线全部加上$\frac{1}{h^2}M$
4. 遍历所有弹簧,分别计算弹力并算出相应的偏导数,将Hessian矩阵对应位置累加上该偏导数,最终得到$g(\mathbf{x_n^{(k)}})$的Hessian矩阵($\nabla^2 g(\mathbf{x})$)
5. 解稀疏线性系统$\nabla^2\mathbf{g}(\mathbf{x_n^{(k)}})\ \mathbf{delta_x} = \nabla\mathbf{g}(\mathbf{x_n^{(k)}})$求得$\mathbf{delta_x}$
6. $\mathbf{x_n^{(k+1)}} = \mathbf{x_n^{(k)}} + \mathbf{delta\_x},\ k = k+1$
7. 重复Step2-6直到收敛,$\mathbf{x_{n+1}} = \mathbf{x_n^{(k)}}$

以上步骤中,单一弹簧的弹力对相应位置的偏导数由以下公式给出(弹力为f;x0,x1分别为弹簧两端的位置):

$$\frac{\partial f}{\partial x0} = \begin{pmatrix} xx & xy & xz \\ yx & yy & yz \\ zx & zy & zz \end{pmatrix}$$

记$distance = x0 - x1$,设弹簧弹性系数为k,上式中,

$$xx = \frac{k}{\|distance\|} * (\|distance\| - spring\_RestLength + \frac{distance.x * distance.x}{\|distance\|})$$

$$yy = \frac{k}{\|distance\|} * (\|distance\| - spring\_RestLength + \frac{distance.y * distance.y}{\|distance\|})$$

$$zz = \frac{k}{\|distance\|} * (\|distance\| - spring\_RestLength + \frac{distance.z * distance.z}{\|distance\|})$$

$$xy = yx = \frac{k}{\|distance\|} * \frac{distance.x * distance.y}{\|distance\|}$$

$$xz = zx = \frac{k}{\|distance\|} * \frac{distance.x * distance.z}{\|distance\|}$$

$$yz = zy = \frac{k}{\|distance\|} * \frac{distance.y * distance.z}{\|distance\|}$$

弹力f对位置x1求偏导可类似推导,推导结果恰好与对x0求导结果相同。

实现时本Lab中设置循环32次,保证准确性的同时也能够保证一定的帧率,效果如下[2]:

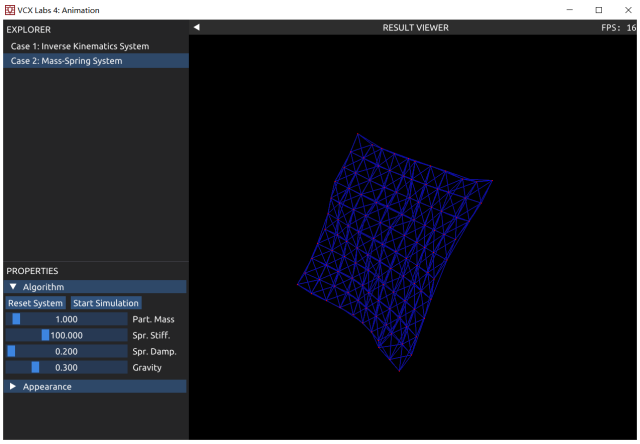


图 5: **Simulation**

[2]动态展示查看simulation.mp4