可视计算与交互概论 lab4 报告

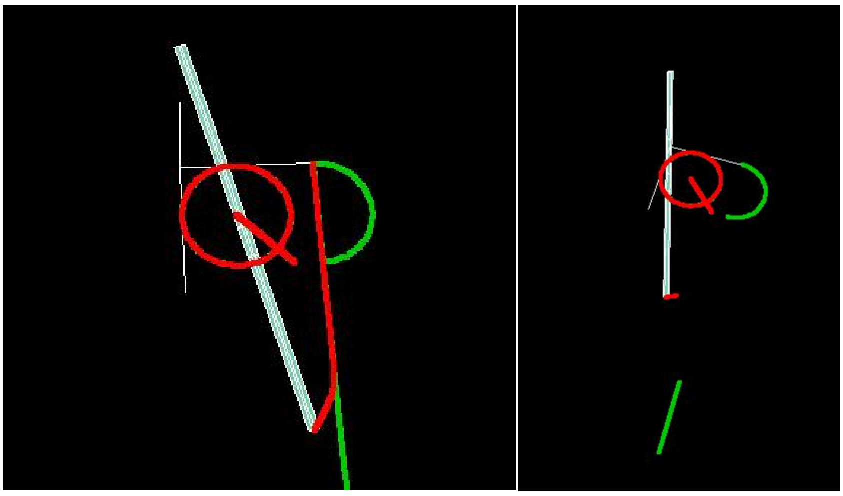
邱荻 2000012852

Task 1: Inverse Kinematics (5')

在阅读和补全代码的过程中，请在报告中回答下面的问题：

1. 如果目标位置太远，无法到达，IK 结果会怎样？

会伸直努力去画但是画不到，然后在那个方向上所能够到的最远位置去画。如：



2. 比较 CCD IK 和 FABR IK 所需要的迭代次数。

FABR IK 迭代次数比 CCD IK 少，速度更快。把迭代次数打印输出发现 CCD 几乎在 maxCCDIKIteration 内都还没收敛，而 FABR 几乎在个位数迭代次数就收敛了。

3. （选做，只提供大概想法即可）由于 IK 是多解问题，在个别情况下，会出现前后两帧关节旋转抖动的情况。怎样避免或是缓解这种情况？

目前还没有观察到这种情况。如果有的话，可以设置一下关节的移动范围使其不能突变。

思路：

正向运动学：

坐标用父关节的全局坐标加上父关节在全局旋转方向上的偏移量 offset。

旋转参照这个

旋转的串接

和基于矩阵的变换一样，多个四元数表示的旋转可以通过乘法串接旋转，例如按照q_1、q_2、q_3的次序旋转，串接后的q_{next}：

$$q_{next} = q_3 q_2 q_1$$

注意四元数的相乘次序和进行旋转的次序是相反的。

逆向运动学：

通过子骨骼的变换，从而倒推出父骨骼应该怎么变换，而父骨骼产生变换之后，此时子骨骼会受到父骨骼变换的影响，子骨骼和目标点产生新的差距，此时需要迭代使用 IK 算法，直至趋近目标点。

CCD IK（循环坐标下降逆动态学）

1.骨骼间的结构为子父层次链接（父节点的变换会影响子节点的）

2.每个骨骼都以【自身轴点到尾叶子节点的方向】旋转到【自身轴点到目标点方向】，开始趋近

3.最终效果与迭代次数成正比

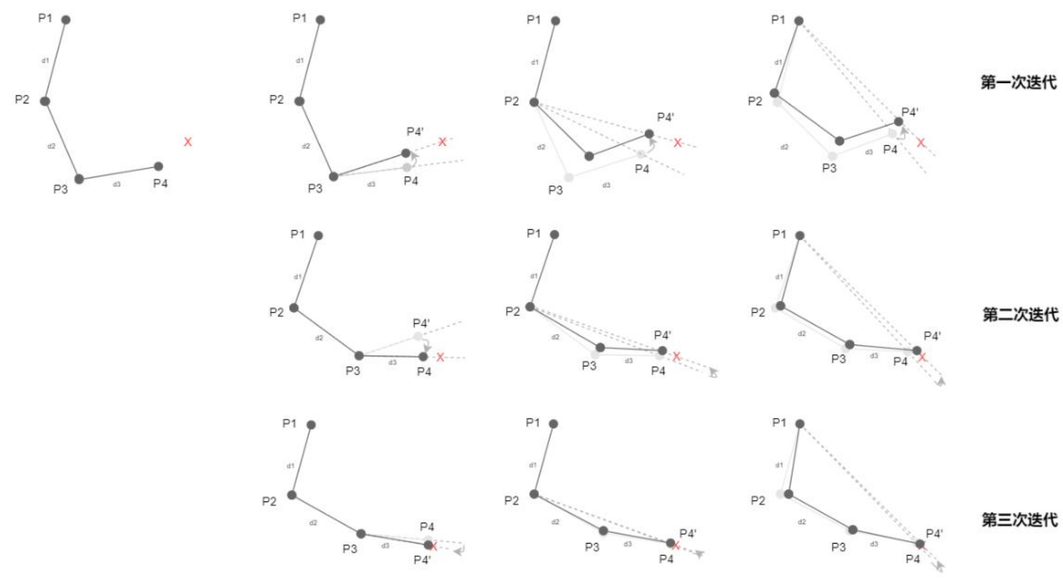
其中旋转用 glm::rotation 函数，转完之后 local 变了记得 forward 以更新全局的参数。

```
GLM_FUNC_DECL tquat<T, P> glm::rotation ( tvec3< T, P > const & orig,
                                          tvec3< T, P > const & dest
                                        )
```

Compute the rotation between two vectors.

param orig vector, needs to be normalized param dest vector, needs to be normalized

See also
GLM_GTX_quaternion



FABR IK(Forward and Backward Reaching Inverse Kinematics)

算法如下：

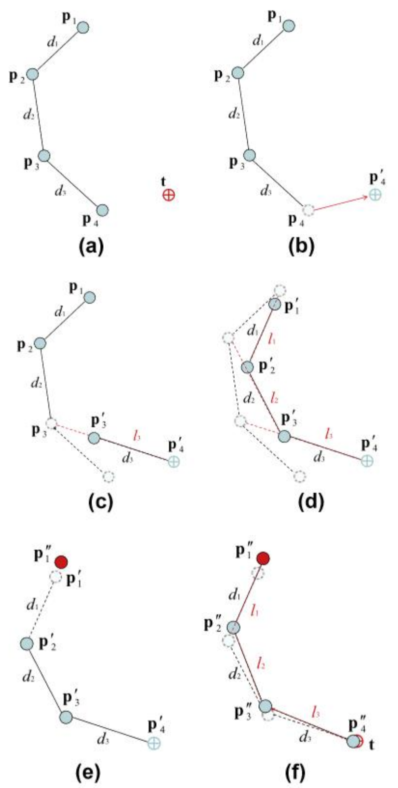
1.先从末端骨骼开始计算，先将最末端的骨骼 p4p4 移到目标位置 t 处，此时骨骼 p4p4 的位置为 p′4p4′。

2.将 p3p3 和 p′4p4′连成一条直线，通过原有的 p3p3 和 p4p4 的距离，将现在的 p3p3 拉到与 p′4p4′同样的距离处 p′3p3′。

3.以此类推，一直处理到根骨骼 p0p0

4.再从根骨骼 p′0p0′开始处理。由于根骨骼再整个迭代过程中是默认为不动的，因此再把根骨骼 p′′0p0″移到原来的位置 p0p0 处，接着使用同样的距离约束，一直处理到尾骨骼 p4p4。

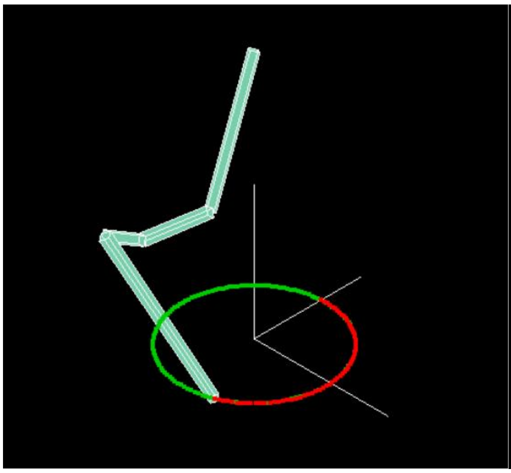
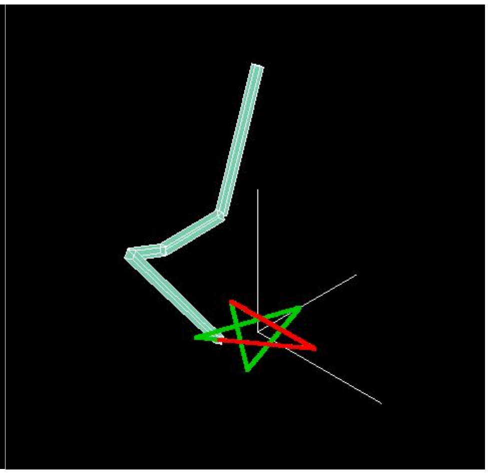
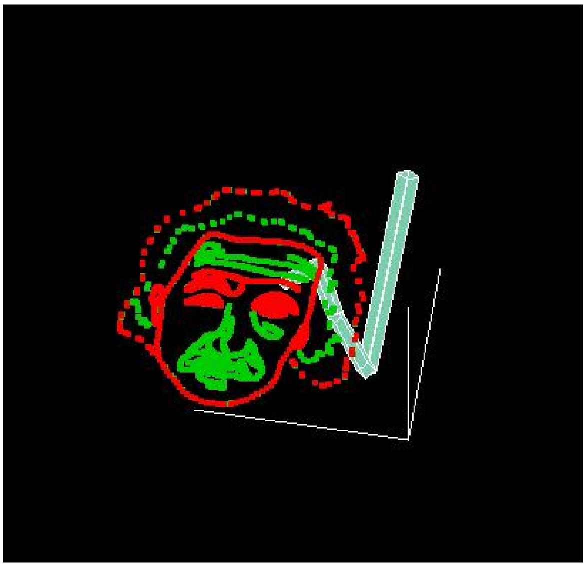
5.重复 2~5 的迭代过程，直到最终的尾骨骼位置与到达目标位置（或与目标位置距离小于某个预定值）停止。此时整个算法结束。



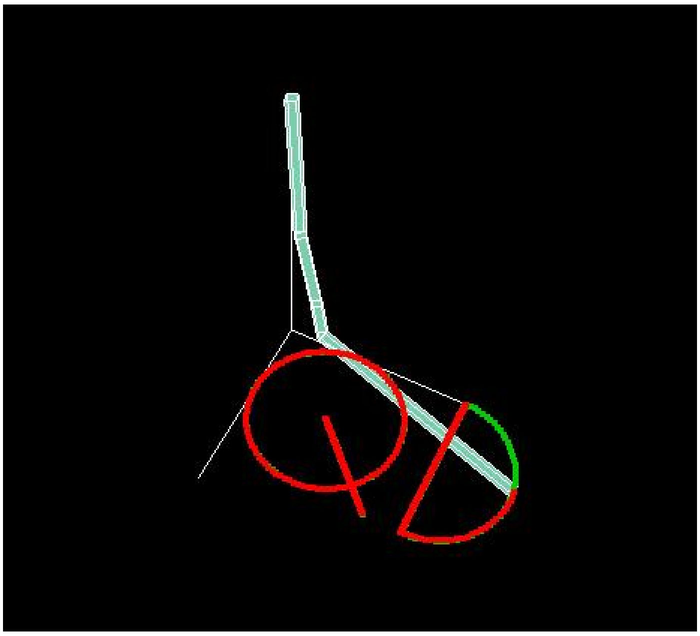
旋转的处理框架已经给处理好了，只用写位置的处理，计算出方向然后移动关节长度就行了。写的时候注意一下方向和正负号，不然会出现很多奇怪的鬼畜 bug。

最终结果：

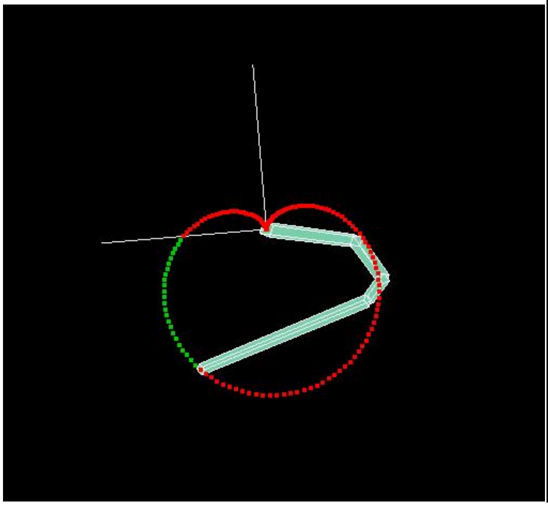
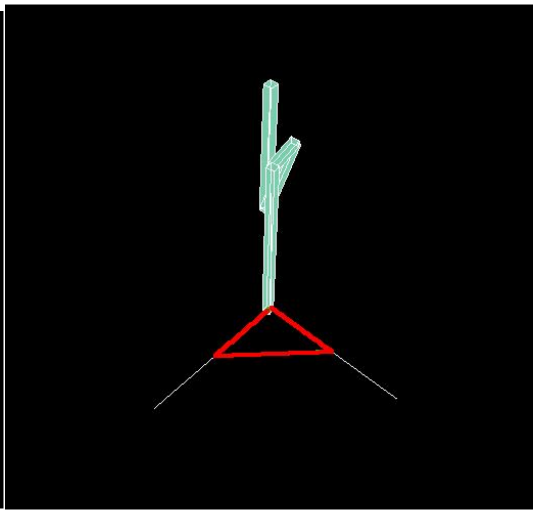
CCD IK

Custom：我的姓名缩写 QD



FABR IK

其实肉眼看感觉两种方法画出来差不多（

Task 2: Mass-Spring System

先对着代码看一下显示欧拉，还比较好理解，学习一下代码框架。

$$r(t+\Delta t) = r(t) + v(t)\Delta t$$
$$v(t+\Delta t) = v(t) + a(t)\Delta t$$

隐式欧拉方法（implicit Euler method）：

$$v(i+1) = v(i) + a(i+1)\Delta t$$
$$x(i+1) = x(i) + v(i+1)\Delta t$$

根据 PPT
即求解最优化问题

- $x_{n+1} = y + h^2 M^{-1} f_{int}(x_{n+1})$

$x_{n+1} = argmin_x\ g(x)$, for $g(x) = \frac{1}{2h^2}|x-y|_{\mathrm{M}}^2 + E(x)$

- Implicit Euler = energy minimization:

$$argmin_x \underbrace{\frac{1}{2h^2}|x-y|_{\mathrm{M}}^2}_{\text{inertia}} + \underbrace{E(x)}_{\text{elasticity}}$$

- **Stable** under **any** timestep size

$$\|x\|_{\mathrm{M}}^2 = x^{\mathrm{T}} M x$$

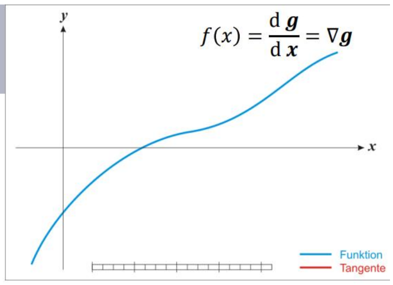
$$f_{int}(x_{n+1}) = \frac{dE(x)}{dx}$$

$$E_{ij} = \frac{1}{2} k(\|x_j - x_i\| - l_0)^2$$

$$f_{ij} = k(\|x_j - x_i\| - l_0)\frac{x_j - x_i}{\|x_j - x_i\|}$$

用牛顿迭代法求解

Numerical Solver



$$x_{n+1} = argmin_x\ g$$

- Newton's method:

Start from a guess x_1,

update with $x_{k+1} = x_k - (\nabla^2 g)^{-1} \nabla g$

- Compute Hessian matrix $\nabla^2 g \in R^{3n \times 3n}$ at every step
- Solve Matrix equation at every step (**main bottleneck**)
- Line search: prevent overshoot

Algorithm 2: Newton Solver with Backtracking Line Search

$\mathbf{x}^{(1)} := \mathbf{y}$;
$g(\mathbf{x}^{(1)}) :=$ `evalObjective`$(\mathbf{x}^{(1)})$
for $k = 1, \dots,$ numIterations **do**
  $\nabla g(\mathbf{x}^{(k)}) :=$ `evalGradient`$(\mathbf{x}^{(k)})$
  $\nabla^2 g(\mathbf{x}^{(k)}) :=$ `evalHessian`$(\mathbf{x}^{(k)})$
  $\delta\mathbf{x}^{(k)} := -\nabla^2 g(\mathbf{x}^{(k)})^{-1} \nabla g(\mathbf{x}^{(k)})$
  $\alpha := 1/\beta$
  repeat
    $\alpha := \beta\alpha$
    $\mathbf{x}^{(k+1)} := \mathbf{x}^{(k)} + \alpha\delta\mathbf{x}^{(k)}$
    $g(\mathbf{x}^{(k+1)}) :=$ `evalObjective`$(\mathbf{x}^{(k+1)})$
  until $g(\mathbf{x}^{(k+1)}) \le g(\mathbf{x}^{(k)}) + \gamma\alpha\,(\nabla g(\mathbf{x}^{(k)}))^{\mathrm{T}} \delta\mathbf{x}^{(k)}$;
end

分为以下几个小步骤：

1. 求 y

y= $(x_n + h(v_n + h\, M^{-1}\, f_{ext}))$

2. 求 g 根据这个式子

$$g(\boldsymbol{x}) = \frac{1}{2h^2}|\boldsymbol{x}-\boldsymbol{y}|_{\mathrm{M}}^2 + E(\boldsymbol{x})$$

3. 求∇g

$$\boxed{\nabla F(\mathbf{x}^{(k)}) = \frac{1}{\Delta t^2}\mathbf{M}(\mathbf{x}^{(k)} - \mathbf{x}^{[0]} - \Delta t \mathbf{v}^{[0]}) - \mathbf{f}(\mathbf{x}^{(k)})}$$

4. 求∇^2g

$$\boxed{\frac{\partial^2 F(\mathbf{x}^{(k)})}{\partial \mathbf{x}^2} = \frac{1}{\Delta t^2}\mathbf{M} + \mathbf{H}(\mathbf{x}^{(k)})}$$

5. 牛顿法解方程

Algorithm 2: Newton Solver with Backtracking Line Search

$\mathbf{x}^{(1)} := \mathbf{y};$

$g(\mathbf{x}^{(1)}) := \texttt{evalObjective}(\mathbf{x}^{(1)})$

for $k = 1, \ldots, \text{numIterations}$ **do**

  $\nabla g(\mathbf{x}^{(k)}) := \texttt{evalGradient}(\mathbf{x}^{(k)})$

  $\nabla^2 g(\mathbf{x}^{(k)}) := \texttt{evalHessian}(\mathbf{x}^{(k)})$

  $\delta\mathbf{x}^{(k)} := -\nabla^2 g(\mathbf{x}^{(k)})^{-1} \nabla g(\mathbf{x}^{(k)})$

  $\alpha := 1/\beta$

  repeat

    $\alpha := \beta\alpha$

    $\mathbf{x}^{(k+1)} := \mathbf{x}^{(k)} + \alpha\delta\mathbf{x}^{(k)}$

    $g(\mathbf{x}^{(k+1)}) := \texttt{evalObjective}(\mathbf{x}^{(k+1)})$

  until $g(\mathbf{x}^{(k+1)}) \leq g(\mathbf{x}^{(k)}) + \gamma\alpha\,(\nabla g(\mathbf{x}^{(k)}))^{\mathsf{T}}\delta\mathbf{x}^{(k)};$

end

6. 解出 x，待入得到 v=(xn+1-xn)/h

$$x_{n+1} = x_n + h\, v_{n+1}$$

效果如下：