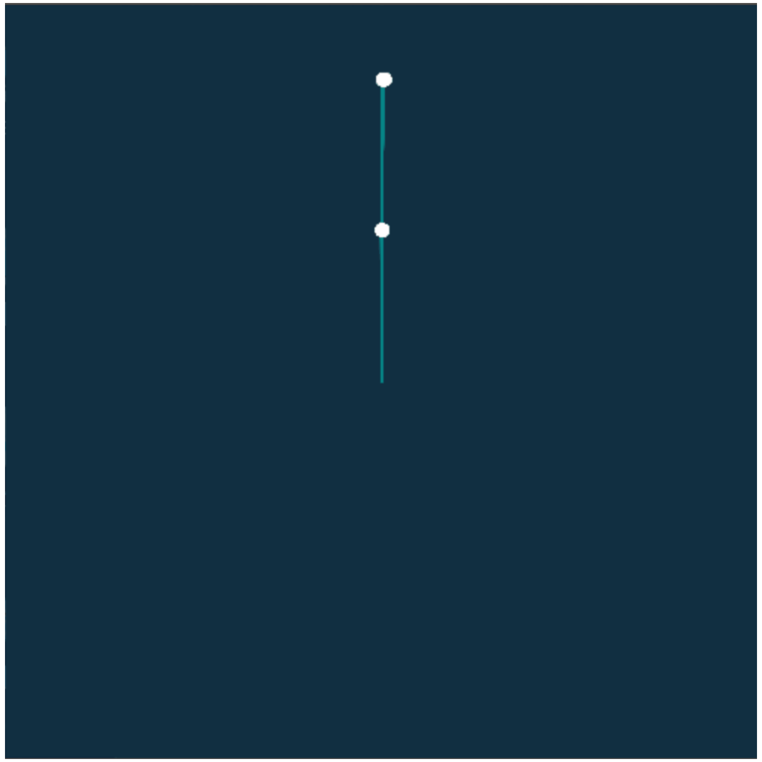
Taichi - 双摆模拟

metachow@github

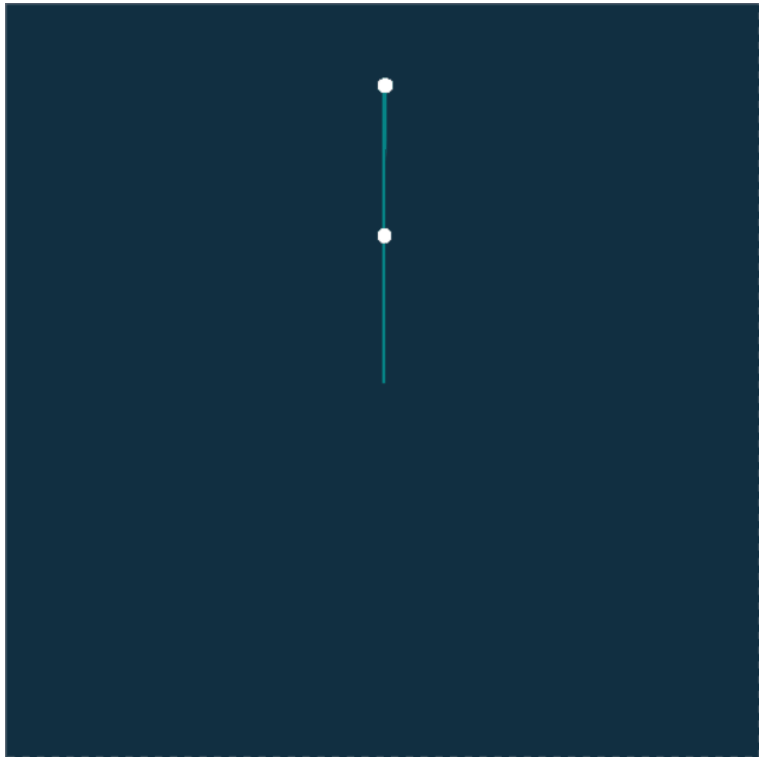
2021.11.25

内容

刚性双摆

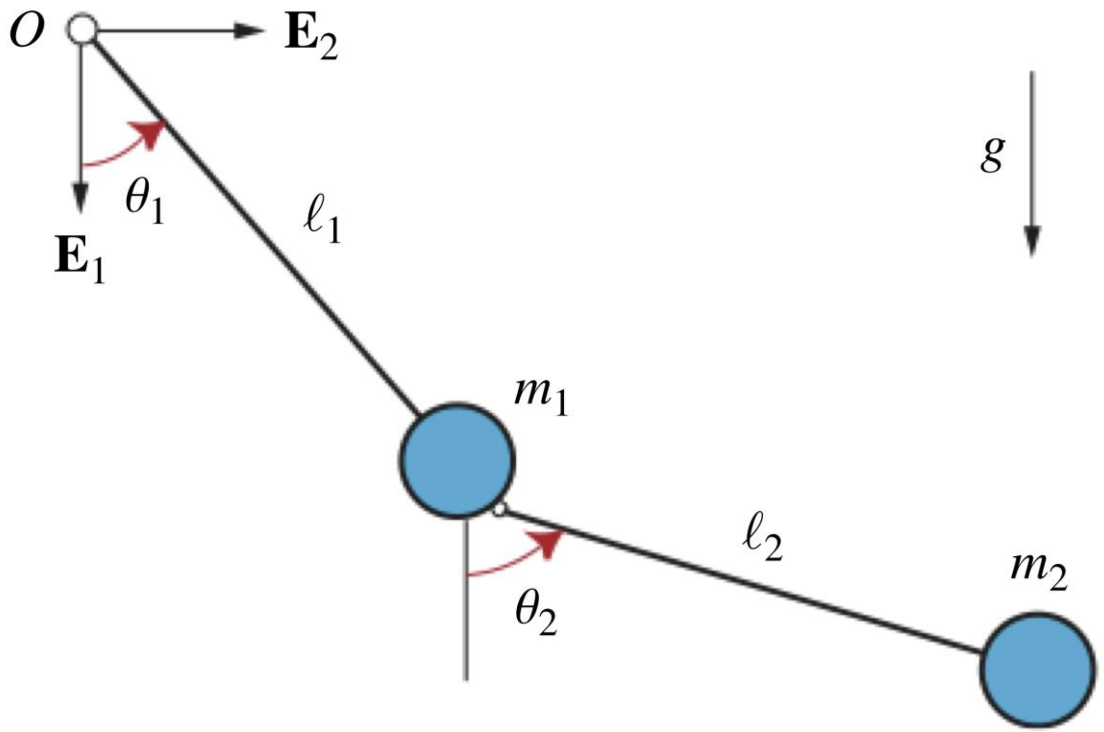


弹性双摆



https://github.com/metachow/Taichi-homework

刚性双摆 – 特性



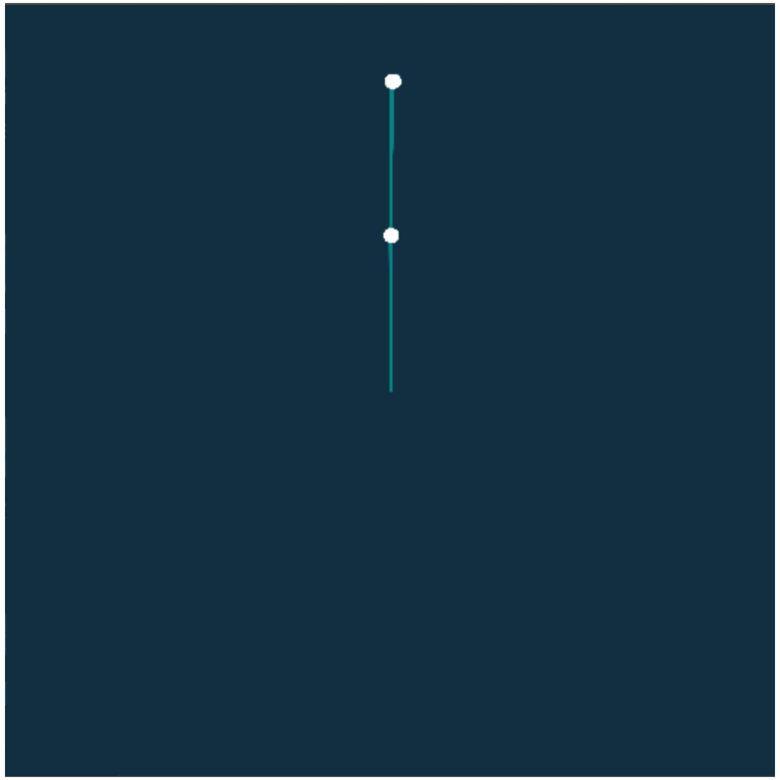
- **组成 – 简单:**
由两个单摆相接组成,连接两个质点和原点的是无质量的刚性杆(不可伸长)

- **行为 – 混沌:**
系统对初始位置极度敏感,即使是非常接近的初始位置,在运行一段时间之后轨迹也会相当不同

起始位置不同

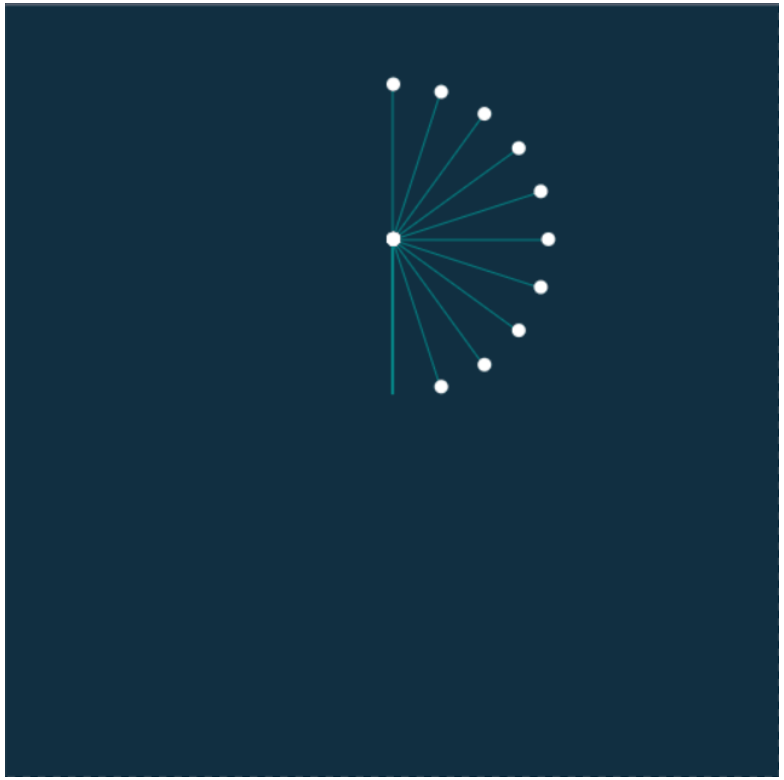
$\Delta\theta_2 = 0.0001, N = 100$

刚性

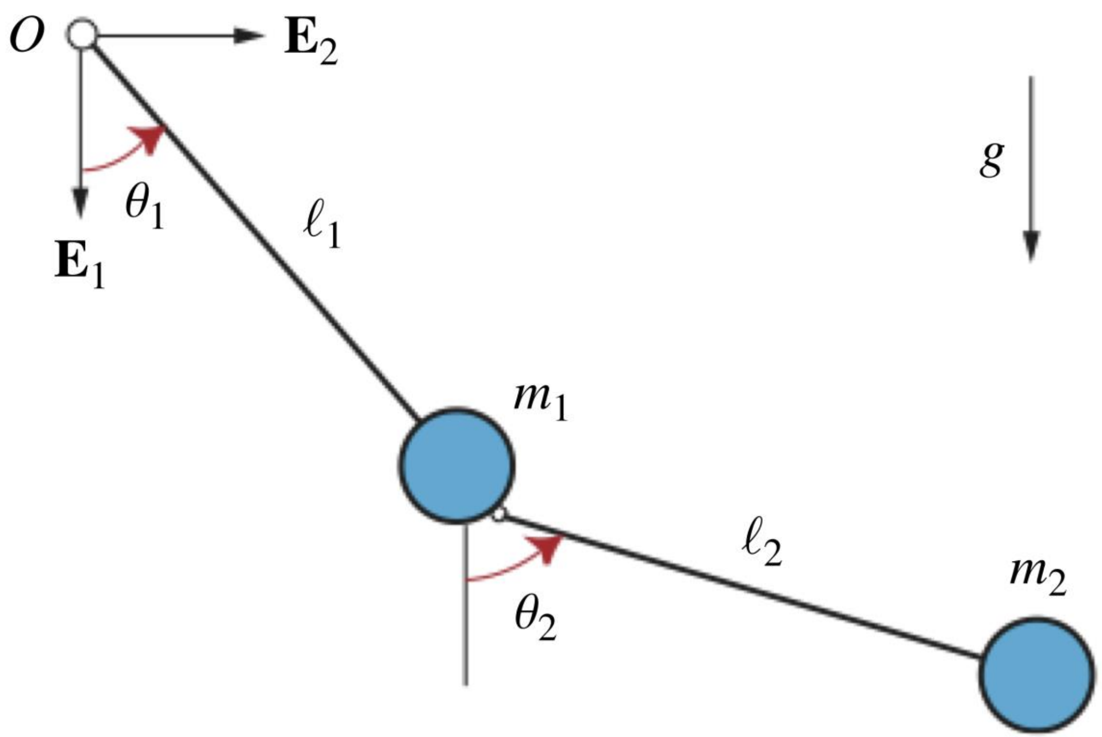


$\Delta\theta_2 = \frac{\pi}{20}, N = 10$

弹性



刚性双摆 – 物理推导

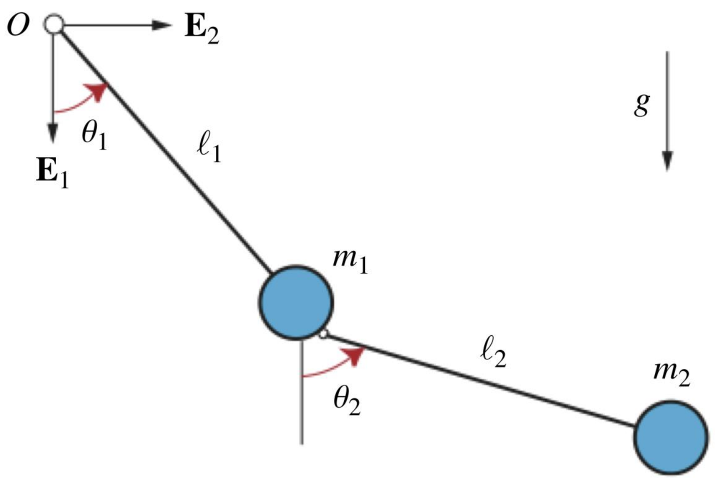


- 参考: https://web.mit.edu/jorloff/www/chaosTalk/double-pendulum/double-pendulum-en.html

- 对单个系统:
当质量与杆长都已经确定的情况下,可以仅通过$[\theta_1,\theta_2]$唯一确定整个系统的位形(广义坐标)

那么:
可以将广义坐标转换为笛卡尔坐标
$P_1 = O + [l_1 sin\theta_1, -l_1 cos\theta_1]$
$P_2 = P_1 + [l_2 sin\theta_2, -l_2 cos\theta_2]$

刚性双摆 – 物理推导



- **Position:**

$$P_1 = O + [l_1 sin\theta_1, -l_1 cos\theta_1]$$
$$P_2 = P_1 + [l_2 sin\theta_2, -l_2 cos\theta_2]$$

- **Velocity:**

$$\dot{P}_1 = [\dot{\theta}_1 l_1 cos\theta_1, \dot{\theta}_1 l_1 sin\theta_1]$$
$$\dot{P}_2 = \dot{P}_1 + [\dot{\theta}_2 l_2 cos\theta_2, \dot{\theta}_2 l_2 sin\theta_2]$$

- **Acceleration:**

$$\ddot{P}_1 = [-\dot{\theta}_1{}^2 l_1 sin\theta_1 + \ddot{\theta}_1 l_1 cos\theta_1, \dot{\theta}_1{}^2 l_1 cos\theta_1 + \ddot{\theta}_1 l_1 sin\theta_1]$$
$$\ddot{P}_1 = \ddot{P}_1 + [-\dot{\theta}_2{}^2 l_2 sin\theta_2 + \ddot{\theta}_2 l_2 cos\theta_2, \dot{\theta}_2{}^2 l_2 cos\theta_2 + \ddot{\theta}_2 l_2 sin\theta_2]$$

- **Forces:**

$$m_1 P_1 = [-T_1 sin\theta_1 + T_2 sin\theta_2, T_1 cos\theta_1 - T_2 cos\theta_2 - m_1 g]$$
$$m_2 P_2 = [-T_2 sin\theta_2, \quad T_2 cos\theta_2 - m_2 g]$$

Simplification

$$[\ddot{\theta}_1, \ddot{\theta}_2]$$

Or write out the Lagrangian directly:

$$\mathcal{L} = T - V$$

刚性双摆-物理推导

- All we need:

$$\theta_1'' = \frac{-g\,(2\,m_1 + m_2)\sin\theta_1 - m_2\,g\sin(\theta_1 - 2\,\theta_2) - 2\sin(\theta_1 - \theta_2)\,m_2\,(\theta_2'^2\,L_2 + \theta_1'^2\,L_1\cos(\theta_1 - \theta_2))}{L_1\,(2\,m_1 + m_2 - m_2\cos(2\,\theta_1 - 2\,\theta_2))}$$

$$\theta_2'' = \frac{2\sin(\theta_1 - \theta_2)\,(\theta_1'^2\,L_1\,(m_1 + m_2) + g(m_1 + m_2)\cos\theta_1 + \theta_2'^2\,L_2\,m_2\cos(\theta_1 - \theta_2))}{L_2\,(2\,m_1 + m_2 - m_2\cos(2\,\theta_1 - 2\,\theta_2))}$$

- $\ddot{\Theta}_n = [\ddot{\theta}_1, \ddot{\theta}_2]$
- $\dot{\Theta}_{n+1} \mathrel{+}= \ddot{\Theta}_n \cdot dt$
- $\Theta_{n+1} \mathrel{+}= \dot{\Theta}_{n+1} \cdot dt$(*Symplectic euler*)

$P_1 = O + [l_1 sin\theta_1, -l_1 cos\theta_1]$
$P_2 = P_1 + [l_2 sin\theta_2, -l_2 cos\theta_2]$

刚性双摆 – 代码

- *数据*
- $\Theta_N = [\theta_1, \theta_2]_N$
- $\dot{\Theta}_N = [\dot{\theta}_1, \dot{\theta}_2]_N$
- $\ddot{\Theta}_N = [\ddot{\theta}_1, \ddot{\theta}_2]_N$
- $P_1 = [x_1, y_1]_N$
- $P_2 = [x_2, y_2]_N$

```
ang = ti.Vector.field(2, ti.f32, N)
v_ang = ti.Vector.field(2, ti.f32, N)
a_ang = ti.Vector.field(2, ti.f32,N)
```

```
pos_1 = ti.Vector.field(2, ti.f32, N)
pos_2 = ti.Vector.field(2 , ti.f32, N)
```

刚性双摆 – 代码

- 1,初始化:角位移,角速度

```
@ti.kernel
def initialize():
    for i in range(N):
        ang[i] = ang_0 + ti.Vector([0, -delta*i/N])
        origin[i] = center
        v_ang[i] *= 0.0
```

刚性双摆 – 代码

- 2,根据当前角位移,角速度,杆长,和质点质量计算角加速度

```
@ti.kernel
def compute():
    for i in range(N):
        a_ang[i] = ti.Vector([(-g*(2*m_1+m_2)*ti.sin(ang[i][0])-m_2*g*ti.sin(ang[i][0]-2*ang[i][1])-2*ti.sin(ang[i][0]-ang[i][1])*\
            m_2*(v_ang[i][1]**2*l_2+v_ang[i][0]**2*l_1*ti.cos(ang[i][0]-ang[i][1])))/(l_1*(2*m_1+m_2-m_2*ti.cos(2*ang[i][0]-2*ang[i][1]))),\
            (2*ti.sin(ang[i][0]-ang[i][1])*(v_ang[i][0]**2*l_1*(m_1+m_2)+g*(m_1+m_2)*ti.cos(ang[i][0])+v_ang[i][1]**2*l_2*m_2*\
                ti.cos(ang[i][0]-ang[i][1])))/(l_2*(2*m_1+m_2-m_2*ti.cos(2*ang[i][0]-2*ang[i][1])))])
```

$$\theta_1'' = \frac{-g\,(2\,m_1 + m_2)\sin\theta_1 - m_2\,g\sin(\theta_1 - 2\,\theta_2) - 2\sin(\theta_1 - \theta_2)\,m_2\,(\theta_2'^2\,L_2 + \theta_1'^2\,L_1\cos(\theta_1 - \theta_2))}{L_1\,(2\,m_1 + m_2 - m_2\cos(2\,\theta_1 - 2\,\theta_2))}$$

$$\theta_2'' = \frac{2\sin(\theta_1 - \theta_2)\,(\theta_1'^2\,L_1\,(m_1 + m_2) + g(m_1 + m_2)\cos\theta_1 + \theta_2'^2\,L_2\,m_2\cos(\theta_1 - \theta_2))}{L_2\,(2\,m_1 + m_2 - m_2\cos(2\,\theta_1 - 2\,\theta_2))}$$

刚性双摆 – 代码

- 3,使用求得的角加速度更新角速度和角位移(*Symplectic euler method*)
- $\dot{\Theta}_{n+1} \mathrel{+}= \ddot{\Theta}_n \cdot dt$
- $\Theta_{n+1} \mathrel{+}= \dot{\Theta}_{n+1} \cdot dt$

```
@ti.kernel
def update():
    dt = h/substepping
    for i in range(N):
        v_ang[i] += a_ang[i] * dt
        ang[i] += v_ang[i] * dt
```

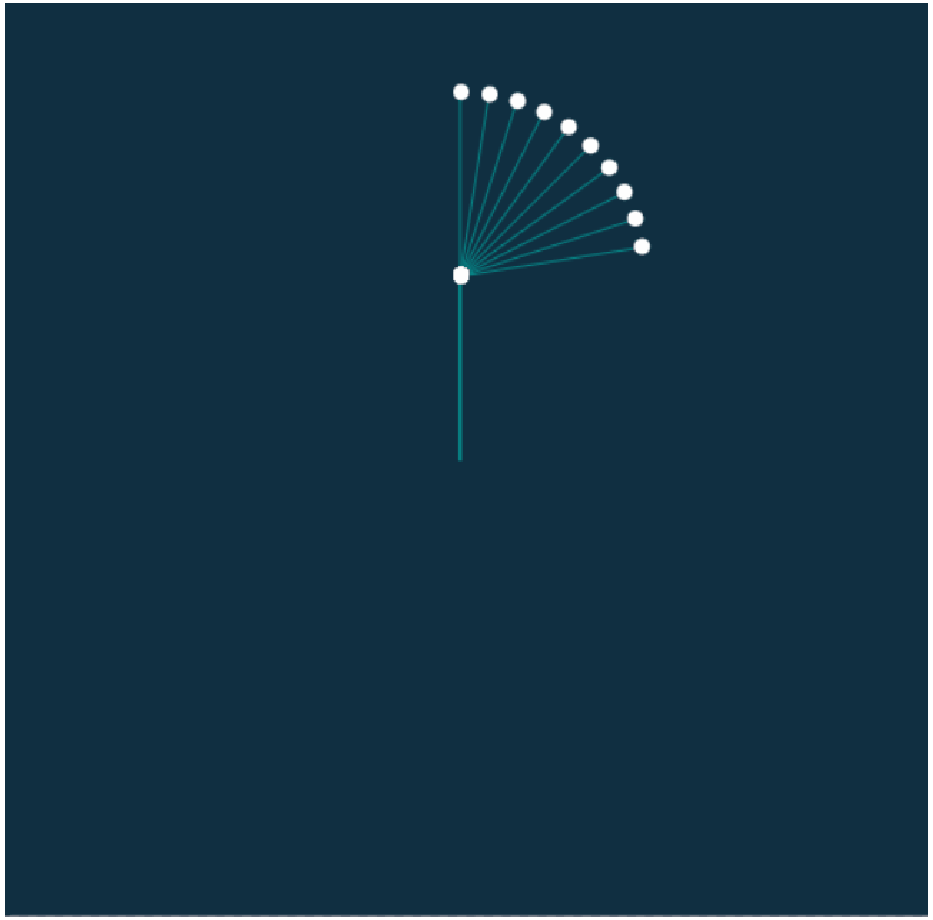
刚性双摆 – 代码

- 4,更新为直角坐标系位置

$$P_1 = O + [l_1 sin\theta_1, -l_1 cos\theta_1]$$
$$P_2 = P_1 + [l_2 sin\theta_2, -l_2 cos\theta_2]$$

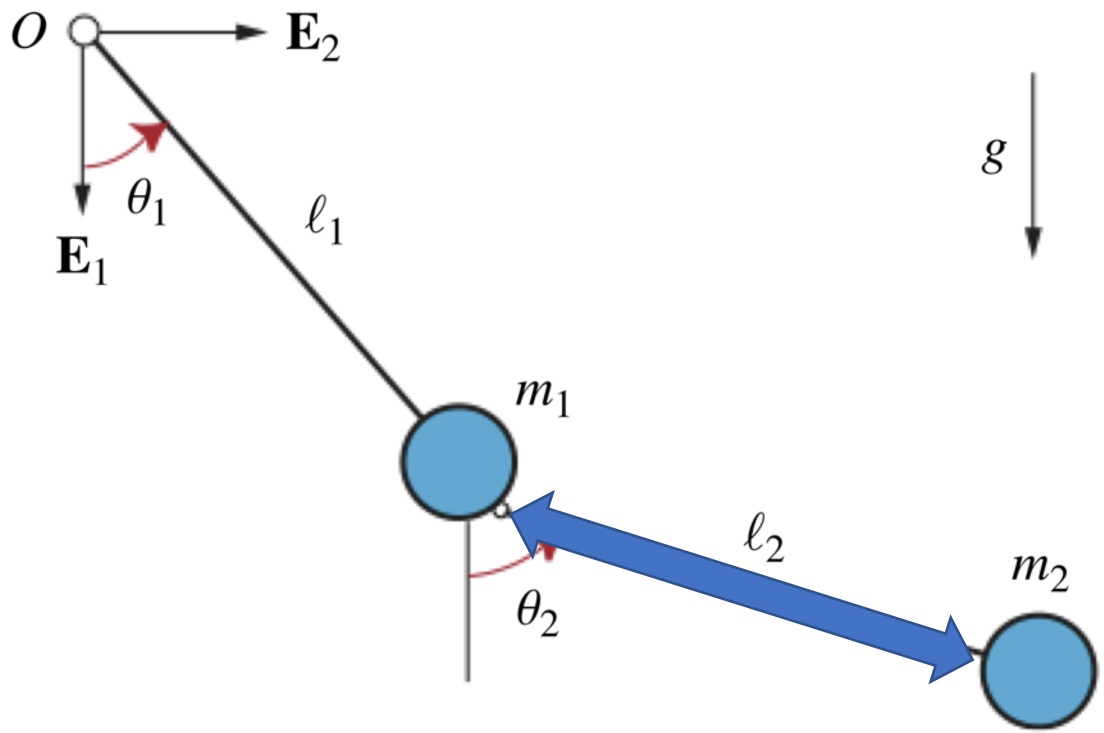
```
@ti.kernel
def set_pos():
    for i in range(N):
        pos_1[i] = center + ti.Vector([l_1 * ti.sin(ang[i][0]), -l_1 * ti.cos(ang[i][0])]) *
        pos_2[i] = pos_1[i] + ti.Vector([l_2 * ti.sin(ang[i][1]), -l_2 * ti.cos(ang[i][1])])
```

刚性双摆-最终效果



弹性双摆

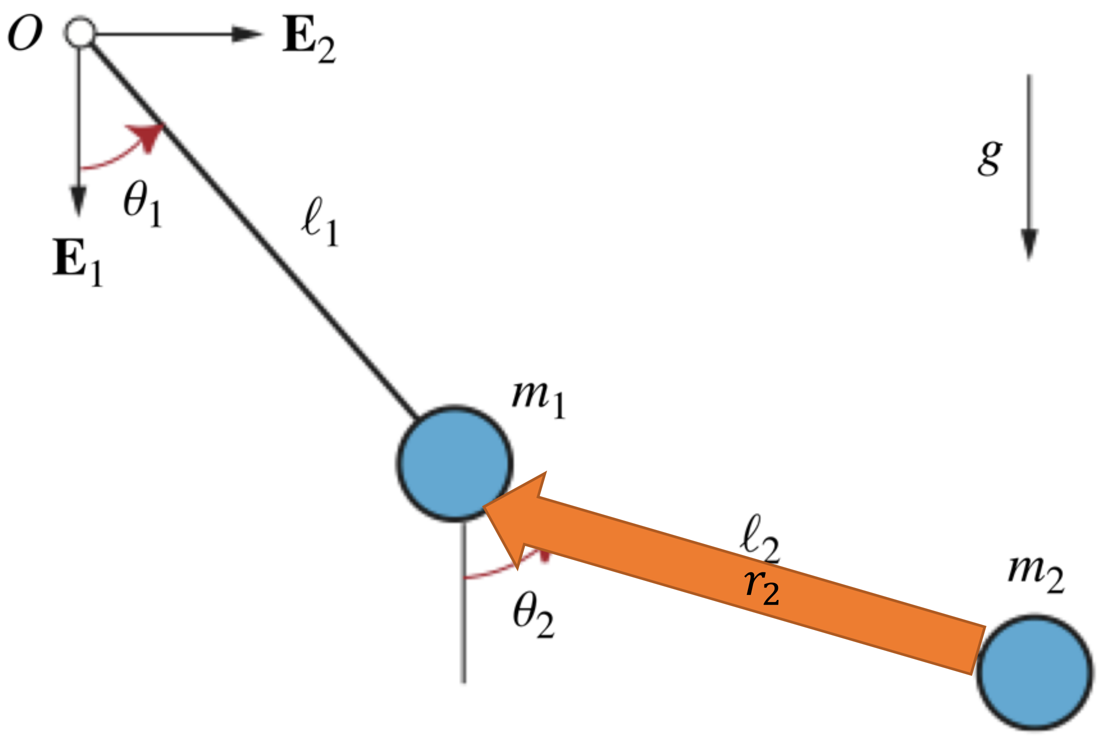
- l_1, l_2由原本的刚性轻杆变为可伸缩的轻弹簧:



- 因此就无法仅通过θ_1, θ_2这两个参数就唯一确定系统的位形.
- 但是弹簧的存在也使得问题得到了简化, 可以直接根据质点1,2的位置计算弹力的大小
- 参考了老师利用mass-spring求解系统运动的代码

弹性双摆 – 物理推导

- l_1, l_2由原本的刚性轻杆变为可伸缩的轻弹簧:



- 在任一时刻
- 先对m_2分析:

设弹簧原长为l_{20},当前长度为:

$$l_2 = \|r_2\| = \|P_1 - P_2\|$$

弹力:

$$grad_2 = -k_2(l_2 - l_{20})\, r_2/l_2$$

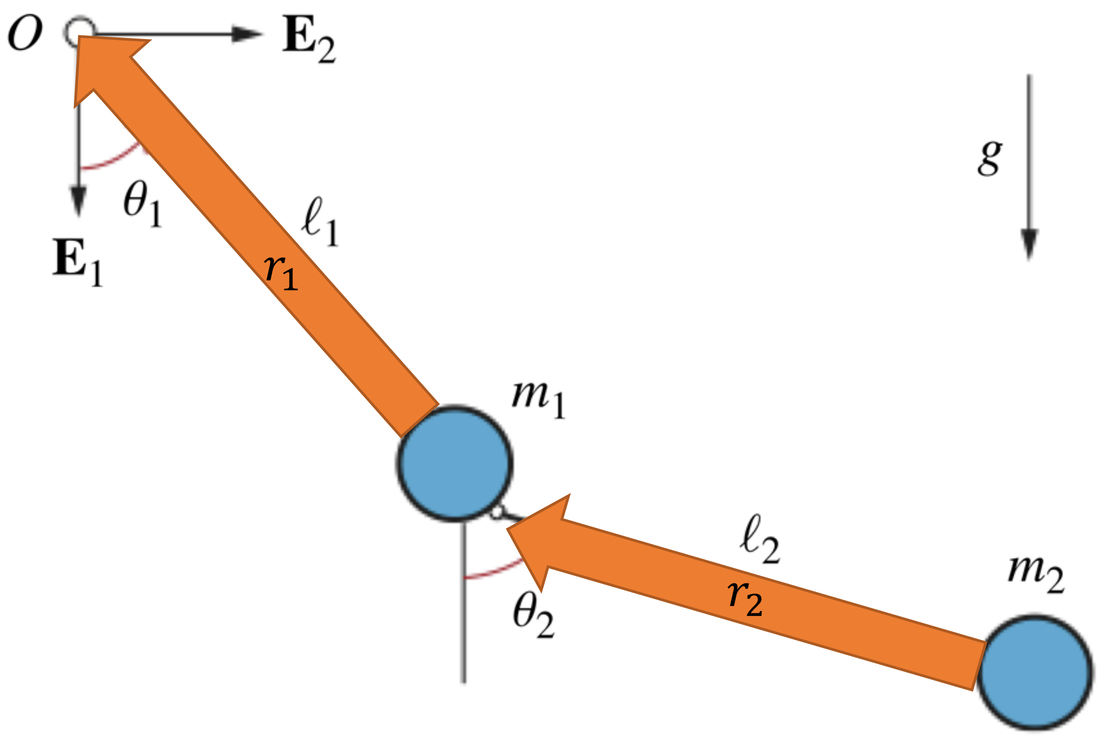
始终指向弹力增大的方向

m_2加速度:

$$a_2 = -\frac{grad_2}{m_2} - g$$

弹性双摆 – 物理推导

- l_1, l_2由原本的刚性轻杆变为可伸缩的轻弹簧:



- 再对m_1分析:

设弹簧原长为l_{10},当前长度为:

$$l_1 = \|r_1\| = \|O - P_1\|$$

弹力:

$$grad_1 \mathrel{+}= -k_1(l_1 - l_{10})\, r_1/l_1$$

$$grad_1 \mathrel{+}= \quad k_2(l_2 - l_{20})\, r_2/l_2$$

与$grad_2$大小相等,方向相反

m_1加速度:

$$a_1 = -\frac{grad_1}{m_1} - g$$

弹性双摆 – 代码

- 1,初始化
- 2, 计算弹力

$$l_1 = \|r_1\| = \|O - P_1\|$$
$$l_2 = \|r_2\| = \|P_1 - P_2\|$$

$$grad_1 \mathrel{+}= - k_1(l_1 - l_{10})\, r_1/l_1$$
$$grad_1 \mathrel{+}= \quad k_2\,(l_2 - l_{20})\, r_2/l_2$$

$$grad_2 \mathrel{+}= - k_2(l_2 - l_{20})\, r_2/l_2$$

```
@ti.kernel
def compute_gradient():
    for i in range(N):
        grad_1[i] = ti.Vector([0.0, 0.0])
        grad_2[i] = ti.Vector([0.0, 0.0])

    for i in range(N):
        r_1 = origin[i] - pos_1[i]
        r_2 = pos_1[i] - pos_2[i]
        l1 = r_1.norm()
        l2 = r_2.norm()

        k_1 = YoungsModulus[None]/l_1
        k_2 = YoungsModulus[None]/l_2

        gradient_1 = k_1*(l1-l_1)*r_1/l1
        gradient_2 = k_2*(l2-l_2)*r_2/l2

        grad_1[i] += -gradient_1
        grad_1[i] += gradient_2


        grad_2[i] += -gradient_2
```

弹性双摆 – 代码

- 3,计算质点加速度,更新速度和位移

$$a_1 = -\frac{grad_1}{m_1} - g$$

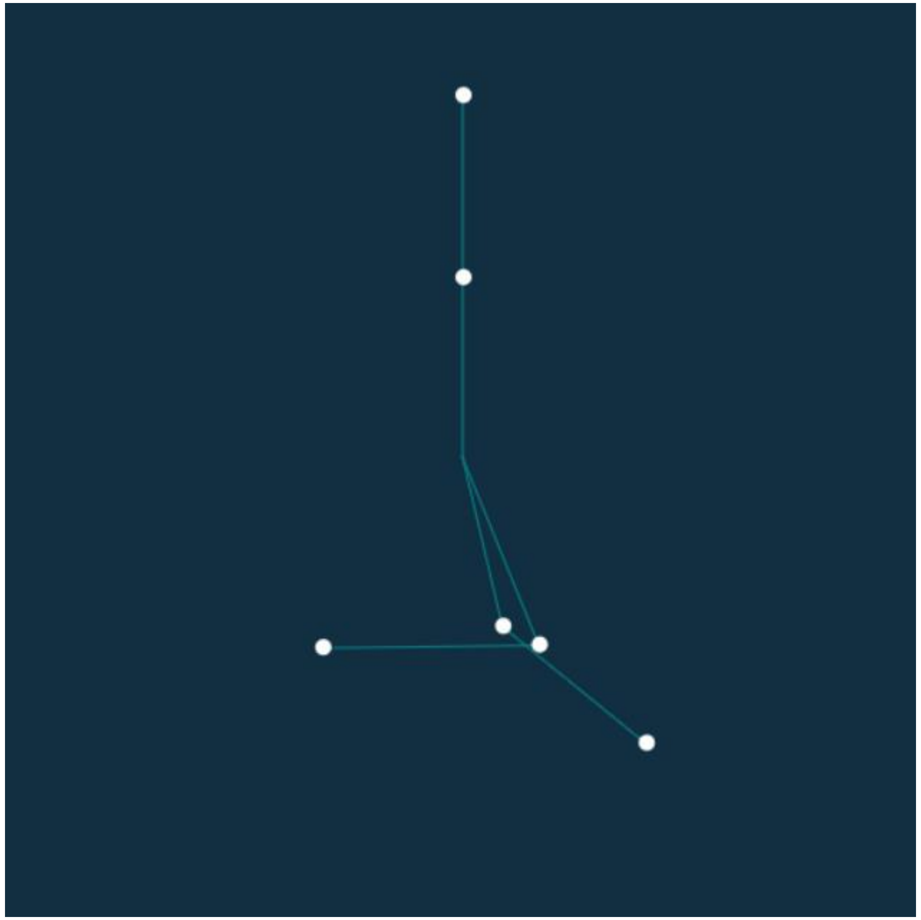
$$a_2 = -\frac{grad_2}{m_2} - g$$

$$v_1 += a_1 * dt$$
$$v_2 += a_2 * dt$$

$$\mathrm{p}_1 += v_1 * dt$$
$$\mathrm{p}_2 += v_2 * dt$$

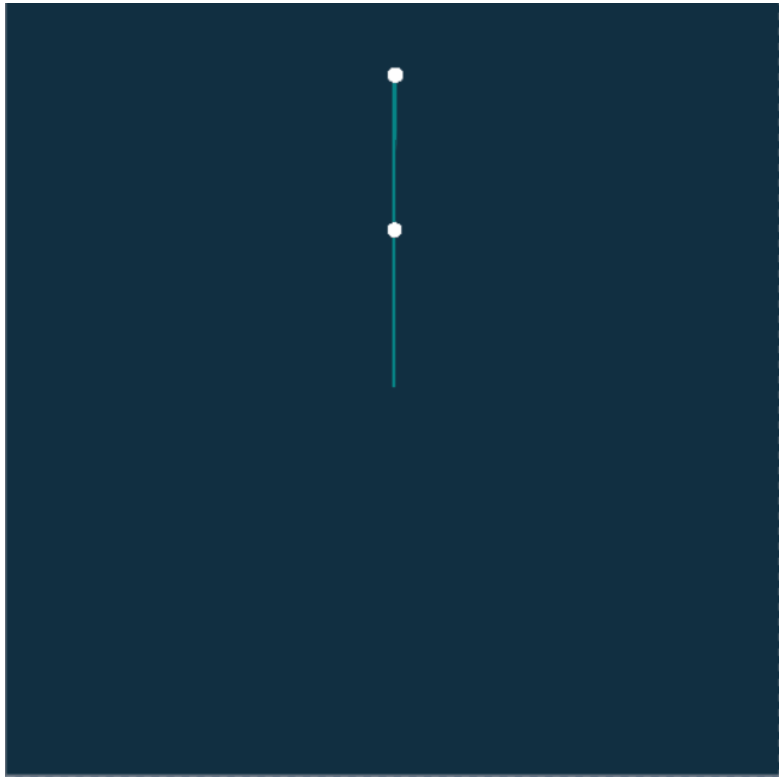
```
@ti.kernel
def update():
    for i in range(N):
        acc_1[i] = -grad_1[i]/m_1 - ti.Vector([0.0, g])
        acc_2[i] = -grad_2[i]/m_2 - ti.Vector([0.0, g])

        vel_1[i] += dh * acc_1[i]
        vel_2[i] += dh * acc_2[i]

        pos_1[i] += dh * vel_1[i]
        pos_2[i] += dh * vel_2[i]
```

弹性双摆最终效果

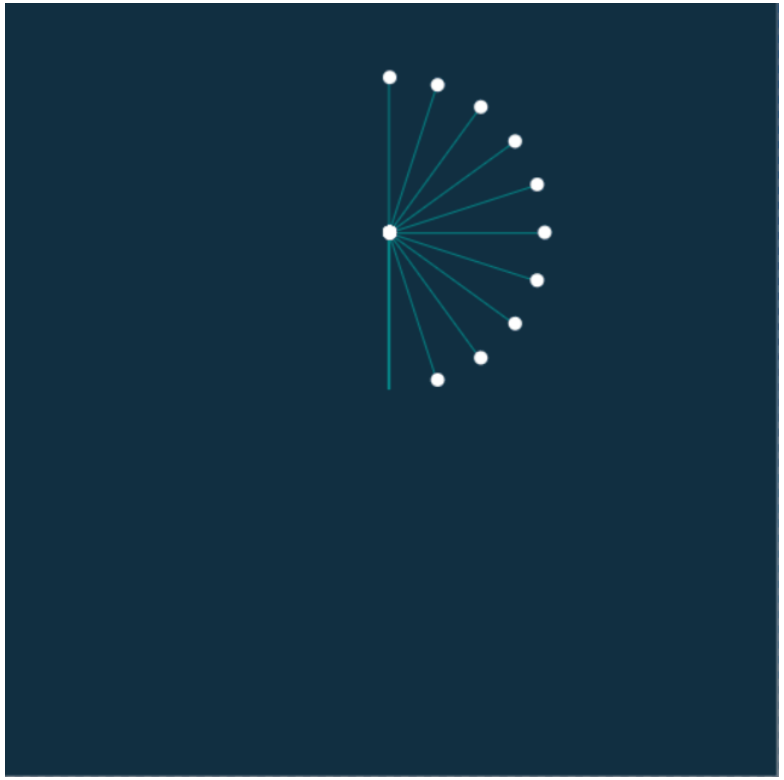


不完善的地方

刚性双摆



弹性双摆



当dt过大时，以及弹性双摆的劲度系数过大…

Gift

Thank you!