Computer Animation 2022
Exercise 2 - Constraint Dynamics

Digital ART Laboratory

Deadline:2022.3.31, 23:59

1 Goals and Rules

In this exercise, you are going to complete two tasks about constraint dynamics.

Setup Please update (`git pull`) the framework code at the git repository and follow the instruction to compile and run the code. If you meet performance problems while solving tasks, please make sure your build is release (`cmake -DCMAKE_BUILD_TYPE=RELEASE ../`).

(repo link: `http://dalab.se.sjtu.edu.cn/gitlab/courses/ca-framework-2022`)

Submission. Compress your solution and name it in form **NAME_ID_ex2.zip**, then submit it in Canvas. **You need to provide a document clarifying your numerical algorithm and parameter choices.** Note: You only need to submit the part of the code that you modified, i.e., `2-1_bead, 2-2_pendulum` directories.

2 A Bead on a Wire

This problem is the toy example in [2]. A bead with point mass m is attached to a unit circle, moving along the circle under the gravity and circle constraint. For detailed derivation, please refer to the **Constrained Dynamics** chapter of [2]. Here, we briefly summarize the equations.

The constraint is defined by $c(\mathbf{x}) = \frac{1}{2}(\mathbf{x} \cdot \mathbf{x} - 1)$. To ensure the circle constraint, we apply constraint force (also known as reaction force) $\tilde{\mathbf{f}}$ beside the gravity force $\mathbf{g}$. The constraint force should ensure $c(\mathbf{x}) = 0$ always holds, which means $\dot{c}(\mathbf{x}) = 0$ and $\ddot{c}(\mathbf{x}) = 0$.

$$\dot{c}(\mathbf{x}) = \mathbf{x} \cdot \dot{\mathbf{x}} = 0 \tag{1}$$

$$\ddot{c}(\mathbf{x}) = \frac{\partial(\mathbf{x} \cdot \dot{\mathbf{x}})}{\partial t} = \dot{\mathbf{x}} \cdot \dot{\mathbf{x}} + \mathbf{x} \cdot \ddot{\mathbf{x}} = 0 \tag{2}$$

substitute Newton's law $m\mathbf{a} = m\ddot{\mathbf{x}} = \mathbf{g} + \tilde{\mathbf{f}}$ into Equation 2, we have

$$\dot{\mathbf{x}} \cdot \dot{\mathbf{x}} + \mathbf{x} \cdot \frac{\mathbf{g} + \tilde{\mathbf{f}}}{m} = 0 \tag{3}$$

$$\mathbf{x} \cdot \tilde{\mathbf{f}} = -m\dot{\mathbf{x}} \cdot \dot{\mathbf{x}} - \mathbf{x} \cdot \mathbf{g} \tag{4}$$

Considering the constraint force shouldn't add or remove energy from the system, we further have condition for the constraint force: $\tilde{\mathbf{f}} \cdot \dot{\mathbf{x}} = 0$. Combined with Equation 1, we can rewrite constraint force as $\tilde{\mathbf{f}} = \lambda \mathbf{x}$. Substitute it into Equation 4, we have

$$\lambda = \frac{-\mathbf{g} \cdot \mathbf{x} - m\dot{\mathbf{x}} \cdot \dot{\mathbf{x}}}{\mathbf{x} \cdot \mathbf{x}}$$

So now, with the definition for constraint force, we can obtain the acceleration of the bead and update its velocity and position.

2.1 Questions (10 Points)

- (5 Points) Implement numerical algorithm to simulate the bead on a wire problem.
- (1 Point) Running the simulation, you may observe the bead slowly drifting away from the circle center. Why is it? Please provide some solutions in your document.
- (2 Points) Please implement one of your proposed solutions and give details of it in your document. NOTE: parameter tweaking (e.g. make dt smaller) doesn't count.
- (2 Points) Document.

3 Double Pendulum

The setup of the double pendulum problem is shown in Figure 1. Two particles p_1, p_2 with mass m_1, m_2 are connected with a **massless** rod with length l_2, and p_1 is connected to the origin point with a **massless** rod with length l_1. To model the motion of pendulum, we are going to use the constraint-based method. You can use *Lagrangian Equation* [3] to solve the problem, which may be simpler, though it requires you to modify the starter code a little bit.

We define two distance constraints. One is the distance between particle $p1$ and original point, and another is the distance between two particles:

$$C(\mathbf{r_1}, \mathbf{r_2}) = \begin{bmatrix} \frac{1}{2}(\mathbf{r_1} \cdot \mathbf{r_1} - l_1^2) \\ \frac{1}{2}((\mathbf{r_1} - \mathbf{r_2}) \cdot (\mathbf{r_1} - \mathbf{r_2}) - l_2^2) \end{bmatrix}$$

where $\mathbf{r_1}$ and $\mathbf{r_2}$ are position of particles p_1, p_2. Since we are simulating two objects, we concatenate their states in order to obtain a global equation for both of them:

$$\mathbf{q} = \begin{bmatrix} \mathbf{r_1} \\ \mathbf{r_2} \end{bmatrix}, \dot{\mathbf{q}} = \begin{bmatrix} \mathbf{v_1} \\ \mathbf{v_2} \end{bmatrix}, \mathbf{f} = \begin{bmatrix} \mathbf{g} \\ \mathbf{g} \end{bmatrix}, \tilde{\mathbf{f}} = \begin{bmatrix} \tilde{\mathbf{f_1}} \\ \tilde{\mathbf{f_2}} \end{bmatrix},$$

all vectors above are of shape 6×1. $\mathbf{f}$ is the gravity force and $\tilde{\mathbf{f}}$ is the constraint force we are going to solve. From [2], we have following relation between quantities:

$$J = \frac{\partial C}{\partial \mathbf{q}}, \dot{J} = \frac{\partial^2 C}{\partial \mathbf{q} \partial t} = \frac{\partial \dot{C}}{\partial \mathbf{q}} \tag{5}$$

$$\tilde{\mathbf{f}} = J^T \lambda \tag{6}$$

$$JWJ^T \lambda = -\dot{J}\dot{\mathbf{q}} - JW\mathbf{f} \tag{7}$$

where J (called *jacobian*) and $\dot{J}$ has shape 2×6, W is a 6×6 diagonal matrix with diagonal entry $[\frac{1}{m_1}, \frac{1}{m_1}, \frac{1}{m_1}, \frac{1}{m_2}, \frac{1}{m_2}, \frac{1}{m_2}]$, and λ is the unknown we are going to solve to get the constraint force.

By solving Equation 7, we the obtain constraint force and thus can update the motion of particles by Newton's law, using time integration methods like we did in *exercise 1*. We skipped many details during the derivation above. For details, please read [1] [2] (Section *The general case* in **Constraint Dynamics** chapter).

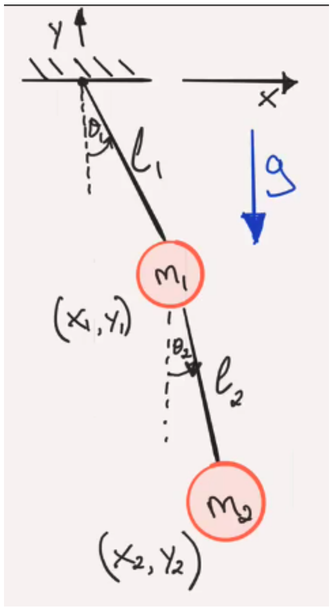

Figure 1 *Double Pendulum Problem*

3.1 Questions (10 Points)

- (5 Points) Implement double pendulum problems.
- (1 Point) Add **feedback** to your algorithm to avoid the drifting problem.
- (2 Points) Discuss how parameters (k_s, k_d) used in feedback affect the results.
- (2 Points) Document.

References

[1] David Baraff and Andrew Witkin. Physically based modeling. SIGGRAPH 2001 Course Notes. Available online at `https://graphics.pixar.com/pbm2001/`.

[2] David Baraff and Andrew Witkin. Physically based modeling: Principles and practice. SIGGRAPH 1997 Course Notes. Available online at `http://www.cs.cmu.edu/~baraff/sigcourse/`.

[3] Freeball. Equations of motion for the double pendulum (2dof) using lagrange's equations. `https://www.youtube.com/watch?v=tc2ah-KnDXw` or see Canvas `https://vshare.sjtu.edu.cn/play/8df4ac0c07f4799a5d67f6429a890e8d`.

Appendix A Math Resources

- derivative of a vector respect to a scalar is still a vector, `https://en.wikipedia.org/wiki/Matrix_calculus#Vector-by-scalar`
- derivative of a scalar respect to a vector is a vector, `https://en.wikipedia.org/wiki/Matrix_calculus#Scalar-by-vector`
- derivative of a vector respect to a vector is a matrix, normally called jacobian, `https://en.wikipedia.org/wiki/Jacobian_matrix_and_determinant`, `https://en.wikipedia.org/wiki/Matrix_calculus#Vector-by-vector`
- dot product derivative, `https://en.wikipedia.org/wiki/Dot_product#Properties`

Appendix B *Eigen* Reference - Vector/Matrix Operations

- The default vector in *Eigen* is column vector, e.g. `Eigen::Vector3d`. To obtain a row vector, declare it as `Eigen::RowVector3d` or transpose it from a column vector v, e.g. `v.transpose()`.
- To declare a matrix type in *Eigen* with custom shape, you can do it in a static way, e.g.

  `typedef Eigen::Matrix<double, 2, 6> Matrix26;`

  create a new matrix type of shape 2×6. Or you can do it in a dynamic way, e.g.

  `Eigen::MatrixXd d; d.resize(2, 6);`.
- To get the inverse of a matrix: `m.inverse()`.
- To concatenate two vectors v_1, v_2 of shape 3×1 into a single larger vector 6×1:

  `Vector6d v; v<<v1, v2`.