Computer Animation 2022
Exercise 3 - Rigid Body Dynamics

Digital ART Laboratory

Deadline:2022.04.14, 23:59

1 Goals and Rules

In this exercise, you are going to implement two ways of rigid body rotation and detect/simulate rigid body collision.

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_ex3.zip**, then submit it in Canvas. **You need to provide a document clarifying your data structure and numerical algorithm.** Note: You only need to submit the part of the code that you modified, i.e., `4-1_spinning` and `4-2_collision` directories.

2 Rigid Body Dynamics

2.1 Prerequisite

In previous exercises, we set up objects as particles with single point mass (i.e., no volumes). In this exercise, we are going to model volume objects so that it has angular velocity and could collide with each other. Please read slides and course note [2] to get familiar with concepts about the general rigid body simulation, such as `rotation`, `linear momentum`, `angular momentum` and etc. These content can be found on page **D2-D31** in the course note.

2.2 Rotation

As described in [2], the state of a rigid body can be represented by $Y(t)$ and its derivative $\dot{Y}(t)$:

$$Y(t) = \begin{bmatrix} \mathbf{x} \\ R \\ \mathbf{P} \\ \mathbf{L} \end{bmatrix}, \dot{Y} = \begin{bmatrix} \mathbf{v} \\ \Omega = \omega^* R \\ \mathbf{f} \\ \tau \end{bmatrix}, \omega^* = \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix}$$

where $Y(t)$ contains position, rotation matrix, linear momentum and angular momentum of the rigid object. To get the new orientation in a time step Δt, we could integrate using $R_{t+1} = R_t + \Delta t \Omega$

The matrix-based rotation scheme will suffer from the instability problem (and also numeric drifting). To solve this, we could replace rotation matrix R with quaternion $\mathbf{q}$, which yields:

$$\omega' = \begin{bmatrix} 0 \\ \omega \end{bmatrix}, \dot{\mathbf{q}} = \frac{1}{2}\omega' \mathbf{q}$$

where multiplication between ω' and $\mathbf{q}$ is quaternion multiplication. So the update of rotation is integrated by $\mathbf{q}^*_{t+1} = \mathbf{q}_t + \Delta t \dot{\mathbf{q}}_t$ and the final rotation is the normalized quaternion $\mathbf{q}_{t+1} = \frac{\mathbf{q}^*_{t+1}}{||\mathbf{q}^*_{t+1}||}$.

Implementation Guide The framework has provided some helper function of the rigid body, such as

- `object.getRotation()`
- `object.getRotationMatrix()`
- `object.setRotation()`

For quaternion, you can use the `Quaterniond` class provided in `Eigen`.

2.3 Collision

To detect collisions in a scene, we first need to determine all pairs of objects which have possibilities to collide in the following time step Δt. This step is known as the `Broad Phase`. Obtaining all pairs of objects, then we need to compute colliding contact points where collisions happen (e.g. vertex-face or edge-edge collision). This is called the `Narrow Phase`. With all collision contacts computed, finally, we can apply impulses to chang the dynamics of objects to reflect the effects of collisions.

A naive broad phase would be treating all pairs of objects in the scene having the possibility to collide. It actually does nothing and leaves all checking workload to the narrow phase. There are two aspects to evaluate the performance of a broad phase algorithm:

- The time complexity of the algorithm itself.
- The workload it delivered to the narrow phase algorithm. The smaller workload, the better the performance since it filters out non-collide pairs.

So the naive broad phase algorithm is inefficient in both aspects since it needs $O(n^2)$ time complexity to produce the list of paris and doesn't filter out any non-collide pairs. A better algorithm would be using the axis-aligned bounding box (AABB) to pre-exclude objects pairs that don't collide (i.e., AABB of the pair doesn't overlap). Though it is still an $O(n^2)$ algorithm, it improves performance by reducing the workload of the narrow phase. To provide even better performance, more advanced method such as `Sweep and Prune` (see [2]) or `Space Partitioning` (e.g, binary tree partitioning space) can be used.

In the narrow phase, we are dealing with two objects a and b that may have contact points. An exhausted approach finding all colliding contacts would be checking all edges and faces of these two objects. The time complexity is square to the number of faces and edges. Separating Axis Theorem (SAT) method and Gilbert-Johnson-Keerthi (GJK) can be used to improve the performance.

After the narrow phase, we should have a list of colliding contact points. The magnitude of impulse $\mathbf{j} = j\mathbf{n}$ is derived as follows:

$$j = \frac{-(1+\epsilon)\mathbf{v}^-_{rel}}{m_a^{-1} + m_b^{-1} + \mathbf{n} \cdot (I_a^{-1}(\mathbf{r}_a \times \mathbf{n})) \times \mathbf{r}_a + \mathbf{n} \cdot (I_b^{-1}(\mathbf{r}_b \times \mathbf{n})) \times \mathbf{r}_b} \tag{1}$$

Please read [1] [2] page **D40-D47** for its derivation of above equation.

If you are interested in obtaining a deeper understanding about collision physics such as collision detection algorithms and techniques, please refer to [3] (available at Canvas).

2.4 Questions (20 Points)

- Implement rigid body rotation based on rotation matrix and quaternion. (4 Points)
- Implement an improved broad phase algorithm. (5 points)
- Implement an improved narrow phase algorithm. (5 points)
- Implement the impulse-based collision. (3 points)

- Document. Besides the description of the algorithm and your implementation. Please discuss and compare two different rotation algorithms. (3 points)
- (Extra points) The impulse model of Equation 1 is frictionless. Please modify it to model friction [4].
- (Extra points) The framework doesn't include sub-step for collision detection so that larger time steps will cause the system to fail (i.e., we don't compute t_c in Figure 13 of [2]). Please implement sub-step collision detection to enable larger time step simulation.

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] Christer Ericson. *Real-Time Collision Detection*. CRC Press, Inc. (Available at Canvas), USA, 2004.

[4] Scott Lembcke. Realtime rigid body simulation using impulses. UMM CSCI Senior Seminar Conference 2006 Morris Minnesota. Available at Canvas.