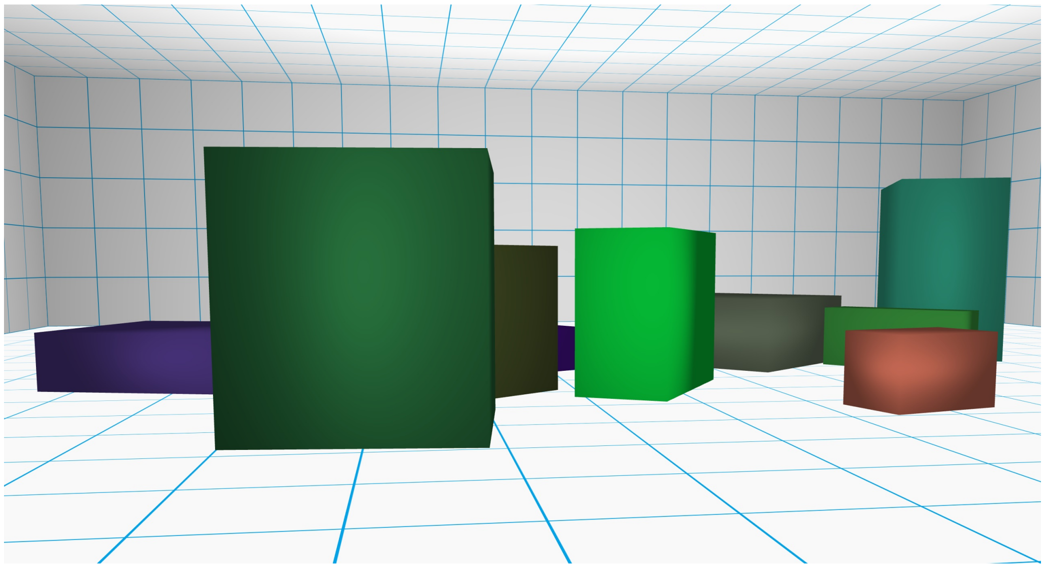
Assignment 4: Rigid Body Dynamics

Due April 14 at 11:59pm



Introduction

Rigid bodies are a fundamental element of many physical simulations, and the physics underlying their motions are well-understood. In this assignment, you will be implementing basic rigid body simulation of cuboid objects, including both dynamics and collision response.

Getting Started

The project to run for this assignment is `Assignment4`. As before, the places where you will need to fill in code are marked with a `TODO`.

As with previous assignments, the spacebar will start and stop the assignment. Gravity has already been implemented, though because the cubes themselves are not being time integrated, you will see nothing happening until you've implemented the update function.

This time, pressing 'r' will begin to apply a force that pulls all of the objects towards their average location, which has the effect of vacuuming them closer to each other. This can be helpful for testing your collision responses between objects, or simply for introducing additional forces into the system. This force should apply both linear and angular acceleration, since the force point is offset slightly from the center of mass.

Your Tasks

Part 1: Dynamics

Your first task is simply to implement the equations of motion for rigid bodies. This is not too different from implementing the basic equations of motion for particles. However, recall that rigid bodies have both linear and angular momentum; you will need to implement changes in both of these due to external forces, as detailed in the slides.

Specifically, consider a set of forces F_i acting on a rigid body at locations r_i (where r_i is given in local coordinates). Then the net force and linear acceleration on an object can be computed respectively as

$$F = \sum F_i; \qquad \dot{v} = \frac{1}{M}F.$$

The net torque and angular acceleration on the object are respectively computed as

$$\tau = \sum \tau_i = \sum r_i \times F_i; \qquad \dot{\omega} = I^{-1}(\tau - \omega \times I\omega),$$

where I is the (world-space) inertia tensor. These accelerations can then be integrated forward in time using standard integration schemes (e.g. symplectic Euler). Velocities are easily updated using a standard additive update rule.

However, recall that care must be taken when updating the rotation given the angular velocity. The angular velocity vector ω can be converted to the derivative of the rotation matrix $\dot{R}(t)$ by $\dot{R}(t) = \omega(t)_\times R(t)$, where $\omega(t)_\times$ is the skew-symmetric cross-product matrix of ω. Adding $R_{t+1} = R_t + \Delta t \dot{R}_{t+1}$ will not in general result in an orthonormal matrix. We can correct this by orthonormalizing the resulting matrix using Gram-Schmidt.

In the `MatrixUtils` class, you will find some useful functions which may assist you in these tasks.

You will need the moment of inertia of the objects we are simulating. Our simulation is limited to cuboids (i.e. rectangular prisms), for which the moment of inertia matrix is known to be

$$I = \begin{bmatrix} \frac{1}{12}m(h^2+d^2) & 0 & 0 \\ 0 & \frac{1}{12}m(w^2+d^2) & 0 \\ 0 & 0 & \frac{1}{12}m(w^2+h^2) \end{bmatrix}$$

Your tasks for this part are:

1. Implement `Cuboid::computeBodyCoordsMOI`, which computes the moment of inertia tensor in body coordinates for the given cuboid object. Then, implement the two functions `Cuboid::computeWorldCoordsMOI` and `Cuboid::computeWorldCoordsInverseMOI`, which respectively compute the moment of inertia tensor, and its inverse, in world coordinates.

2. Implement `Cuboid::applyForce`, which applies a force F_i to a cuboid at a position r_i, adding the force and torque to the accumulator variables indicated.
3. Implement `Cuboid::update`, which integrates the cuboid position and orientation forward in time.

Part 2: Impulse-Based Collision Response

It is difficult to observe anything about the dynamics of the rigid bodies without some form of collisions. The collision response model we will be following is taken from `https://en.wikipedia.org/wiki/Collision_response`.

We are given two rigid bodies B_1 and B_2 which are in collision at a point p. Let p be r_1 when expressed in the body coordinates of B_1, and r_2 when expressed in the body coordinates of B_2. Assuming we already know the magnitude of the collision response impulse j_r that must be applied, we can apply linear impulses

$$v_1' = v_1 - \frac{j_r}{m_1}\hat{n}$$
$$v_2' = v_2 + \frac{j_r}{m_2}\hat{n}$$

where v_i is the pre-collision velocity of body i, v_i' is the post-collision velocity, and m_i is the mass. $\hat{n}$ is the normal vector at the collision point; this normal is assumed to point away from B_1 and towards B_2. Similarly, we can apply angular impulses

$$\omega_1' = \omega_1 - j_r I_1^{-1}(r_1 \times \hat{n})$$
$$\omega_2' = \omega_2 + j_r I_2^{-1}(r_2 \times \hat{n})$$

where ω_i and ω_i' are the pre-collision and post-collision velocities, and I_i is the moment of inertia.

Then, it remains to compute the impulse magnitude. This can be obtained from the following equation:

$$j_r = \frac{-(1-e)v_r \cdot \hat{n}}{m_1^{-1} + m_2^{-1} + ((I_1^{-1}(r_1 \times \hat{n})) \times r_1 + (I_2^{-1}(r_2 \times \hat{n})) \times r_2) \cdot \hat{n}}$$

See the article for the full derivation. Here, v_r is the relative velocities of the two bodies *at the contact point*: $v_r = v_{r_2} - v_{r_1}$, where v_{r_2} denotes the velocity at the point r_1 when considered as a part of body 1. e is the coefficient of restitution, which is a constant.

This equation is correct for the case of two rigid bodies in collision. For one rigid body colliding with an immobile barrier such as the floor, this equation also accurately describes the response if we consider the floor to have infinite mass. Then, if we treat the floor as body 1 (so that the normal points upward out of the floor), all of the terms involving body 1 go to zero, and we are left with only the terms for body 2.

In addition to the collision response in the normal direction, there is also a frictional response in a tangential direction $\hat{t}$. To compute the direction of this response, we use direction of the tangential velocity or force at that point. Specifically, we first project the relative velocity of the two bodies v_r onto the plane orthogonal to the contact normal $\hat{n}$ by subtracting the normal component $(v_r \cdot \hat{n})\hat{n}$. If this is zero, meaning there is no relative velocity, we project the current net force instead. If there is no net force either, then there is no friction, and we consider the tangent vector to be 0. If $\hat{t}$ is non-zero at this point, we normalize it and continue.

We can then compute the friction impulse magnitude j_f as

$$j_f = \begin{cases} -(mv_r \cdot \hat{t})\hat{t} & v_r \cdot \hat{t} = 0 \quad mv_r \cdot \hat{t} \leq j_s \\ -j_d\hat{t} & \text{otherwise} \end{cases}$$

where j_s and j_d are coefficents of static and dynamic friction respectively. Once we have the impulse magnitude j_f, we apply linear and angular impulses along the direction $\hat{t}$ in a manner analogous to the normal collision response impulses.

You are highly recommended to look at `Cuboid.h` in order to see which fields and methods are available to help you.

Your tasks in this section are:

1. Implement `CollisionPlane::frictionTangent`, which, given a cuboid and a collision point, computes the tangential direction at the collision point in which the friction force will be applied.
2. Implement `CollisionPlane::applyCollisionResponse`, which does the following:
   (a) Check if the given cuboid is colliding with the plane (i.e. is intersecting the opposite side of the plane from the way the normal is pointing). If it isn't, stop; otherwise:
   (b) Apply a positional correction that pushes the rigid body out of the plane.
   (c) Compute and apply the both the normal and the frictional collision response to a cuboid that is colliding with a wall or floor (represented as an infinite plane). Assume that the plane has infinite mass.
3. Implement `ParticleSystem::applyCollisionImpulse`, which applies impulses to two cuboids that are in collision. In this part, you *only* need to implement the normal collision response. You do *not* need to implement collision detection (which is already implemented for you), or friction impulses.

Note that our collision detection algorithm between rigid bodies is only an approximation, and will not prevent all instances of interpenetration between cubes. Still, you should be able to see cubes interacting with each other and with the floor after completing these tasks.

Submission instructions

There is no extension for this assignment, so you need only submit a write up and your code. Please delete the `Assignment4/Release` and `Assignment4/Debug` directories if they are present. Then, zip **only** the `Assignment4` directory.

In order to submit, please e-mail the following items to the instructor (scoros@cmu.edu) and TA (christoy@cs.cmu.edu):

- A brief writeup (2 pages max) that describes your effort in implementing this assignment, including the extension you have implemented, what worked and what did not work, and any interesting insights you may have gained.
- A `.zip` of the contents of your `Assignment2` directory, after having deleted the `Debug` and `Release` directories inside (please be careful not to delete your code!).

Please also include the string "CS15467" in your subject line so we know to look for it.

Notes on Academic Integrity

You are allowed to collaborate on the assignment in terms of formulating ideas, developing physical models and mathematical equations. However, you must implement the code and do the write up completely on your own, and understand what you are writing. Please also list the names of everyone that you discussed the assignment with.