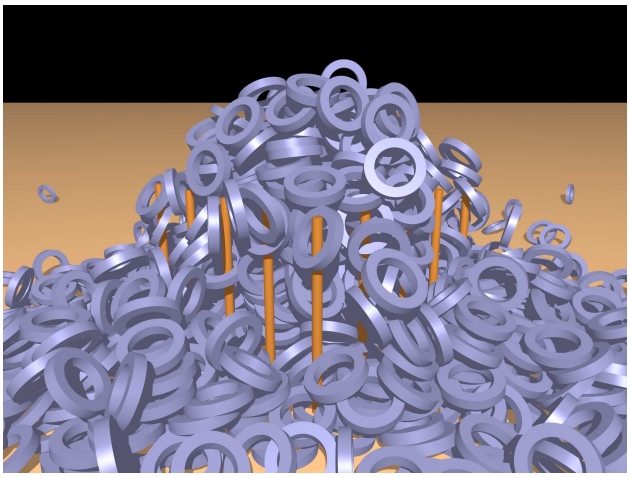
Assignment 3: Particle Systems and Rigid Body Simulation in Unity

CS 248 Winter 2017-2018

Due Date: Monday, 12 February by 6:30pm



Introduction This assignment consists of performing two types of simulation: simulating a particle system and simulating a rigid body. To this end, you will build upon a pair of `C++` libraries provided as starter code for this assignment. First, you will design your own particle system rules to simulate some sort of group behavior. Examples of systems that you may choose to explore include birds flocking together, groups of fish swimming, and crowds of people. Second, you will expand upon the provided rigid body starter code to add friction in order to bring a falling cube to rest as it slides along the ground. Finally, you will create two scenes in Unity to showcase your particle system code and your rigid body code.

Background In order to incorporate third party libraries and preexisting code, Unity supports the integration of its system with plugins written in a variety of languages including `C`, `C++`, and Objective-`C`. In this assignment, we provide `C++` starter code for simulating particle movement and rigid body motion. This code is compiled as a .dll (on Windows) or a .bundle (on OS X) and saved within the Assets hierarchy of a Unity project. Because the names of `C++` functions are mangled during compilation, we must use a `C`-style API to communicate between a Unity script and a `C++` library.

For this assignment we are looking at standard simulations that could be done using the Unity engine without incorporating our own code. The point in using our own code is to reinforce our understanding of the tools we use, as the `C++` libraries give us direct control over the details of the simulations being performed. In addition to providing a greater degree of control over the engine, plugins are useful in situations where fast simulations are needed to meet the performance requirements of a game. Furthermore, understanding plugins gives us the ability to reuse existing `C++` libraries in a game, so that we do not need to reimplement existing tools in one of Unity's scripting languages.

Implementation The two components of this assignment are related to the simulation of a particle system and the simulation of a rigid body. To begin, it will be helpful to read and understand the provided starter code.

Starter Code The starter code consists of two `C++` libraries, `ParticlePlugin` and `RigidBodyPlugin`, with projects for both Visual Studio and XCode. These are loaded in the provided Unity project by the scripts `ParticleMotionScript` and `RigidBodyMotionScript`. These scripts interact with the plugin libraries through a set of functions with `C` linkage (accomplished by wrapping the function declarations in an `extern "C"` block in the plugin code). By passing arrays of object states (e.g., positions and rotations) to these interface functions, we can update Unity game objects with the results of our simulation computations. Links to documentation pages with details on Unity plugins are provided in the References section below.

Particle System The starter `C++` library and `C#` scripts provide a basic example for creating game objects based on a prefab, and moving those game objects about a central point. Use this code to become familiar with the syntax for using plugin functions in a Unity script, and then create your own set of rules to simulate a particle system. Examples of systems you can choose to simulate include flocks of birds, schools of fish, and crowds of people. Please see the References section below for some related resources.

Rigid Body Collision Response The starter code for the rigid body scene simulates a rigid body with some initial velocity falling onto the ground plane. If you play the rigid body scene without making any modifications, you will observe that a cube falls to the ground, then slides off indefinitely. Your task for this part of the assignment is to incorporate friction into the rigid body simulation.

Begin by studying the rigid body code—in particular, you will be modifying the function `Rigid_Body::Collide_Cube_Ground`. This function currently implements a frictionless collision response. It uses an impulse factor K, which describes how much the point-wise velocity will change when an impulse is applied to the rigid body at a point. This relationship is described by

$$\Delta\vec{v} = K\vec{j},$$

where $\vec{v}$ is velocity and $\vec{j}$ is the impulse. The default implementation finds the expected velocity change $-(1+\varepsilon)\vec{v}_n$, where ε is the coefficient of restituion and $\vec{v}_n$ is the normal component of the relative point-wise velocity at the collision point. Thus, we can find the impulse $\vec{j}$ using $\vec{j} = -K^{-1}(1+\varepsilon)\vec{v}_n$, then apply that impulse to the rigid body.

After understanding the code, you will need to incorporate friction into the process. The intuition here is based on the in-class discussion of friction: first, we will attempt to apply static friction. If the static friction prevents the object from moving in the tangential direction, we are finished. Otherwise, the object is moving tangent to the colliding surface and we must apply kinetic friction.

To incorporate friction, you must take the tangential relative point-wise velocity $\vec{v}_t$ into consideration in addition to the normal component of the relative point-wise velocity. You can begin by simply zeroing out the tangential component of velocity. This is accomplished by calculating the expected velocity change as $-(1+\varepsilon)\vec{v}_n - \vec{v}_t$, then computing the sticking impulse $\vec{j}_{stick}$. You can then check whether the sticking impulse is in the friction cone, i.e., whether $|\vec{j}_{stick} - (\vec{j}_{stick}\cdot\vec{N})\vec{N}| \leq \mu\vec{j}_{stick}\cdot\vec{N}$, where $\vec{N}$ is normal of the collision surface. If so, since we have already zeroed out the tangent velocity, nothing else needs to be done. Otherwise, we need to compute the correct impulse with sliding friction, using the equations

$$j_n = -(1+\varepsilon)\vec{v_n}\cdot\vec{N}/(\vec{N}\cdot(K(\vec{N}-\mu\vec{T})))$$
$$\vec{j} = j_n\vec{N} - \mu j_n\vec{T},$$

where:

- $\vec{N}$ is the normal direction of the relative velocity (for our purposes, $\vec{N} = [0,1,0]$ because we are simply colliding with the ground in the xz-plane),

- $\vec{T}$ is the tangential direction of the relative velocity, and
- j_n is the component of the impulse $\vec{j}$ in the normal direction $\vec{N}$.

For more details about the derivation of the above equations, you can use the link in the References section below to refer to the paper "Nonconvex Rigid Bodies with Stacking."

Note To update your plugin libraries, you will need to rebuild your Visual Studio/XCode project, then replace the existing .dll/.bundle file in the Unity Assets folder with the newly built library file. In our experience, it is necessary to close and re-open Unity after replacing these files in order for their changes to take effect.

Grading The assignment will be graded out of 10 points according to the following criteria:

- 3 points: Modify (or replace) the particle system starter code with your own code to model a group of objects (e.g., birds, fish, or crowds).
- 2 points: Create a scene in Unity to showcase your modified particle system code. The objects in this scene should have their movement specified by your `C++` code.
- 3 points: Modify the rigid body starter code to incorporate friction so that the cube does not slide indefinitely on the ground.
- 2 points: Create a scene in Unity with multiple falling cubes, using the rigid body `C++` code to control their movement.

Resources

- Unity documentation on plugins: http://docs.unity3d.com/Manual/Plugins.html
- How to build a plugin for Unity: http://docs.unity3d.com/Manual/PluginsForDesktop.html
- An example simulation of flocking birds, with JavaScript source code provided: http://threejs.org/examples/#canvas_geometry_birds
- Another example simulation of flocking birds, with C++ source code: https://processing.org/examples/flocking.html
- An extremely highly-cited paper on bird (or "boid") flocking simulation: http://dl.acm.org/citation.cfm?id=37406
- Wikipedia discussion of the above boids paper: http://en.wikipedia.org/wiki/Boids
- Guendelman et. al. "Nonconvex Rigid Bodies with Stacking." http://physbam.stanford.edu/~fedkiw/papers/stanford2003-01.pdf