Aurora Motion Sim

By Jason Foat
jasonfoat@gmail.com

This is an analysis in support of the Aurora Motion Sim Take-Home Assignment. We are charged with the problem of creating a simulation of a pen thrown spinning in the air. (We make the presumption that this is a pen used for writing, and not one for corralling animals or small humans. :-) We consider something like the pen shown below… an oblong instrument that fits easily in a human hand. A worthy and fun task. The good news is that we have done this: a simulation has been created. Below we discuss the details.

Simulation?

Before I describe the details of what I've done, it's worth taking a moment to consider the title of this section itself. What is a simulation? For our purposes we presume it is an abstraction of a real thing which is represented in a computer, for the purposes of analyzing the behavior of the real thing. Hence, to make an effective simulation we must consider which behavior we are interested in analyzing.

For the purposes of this exercise, we sent our intention to model the physical motion of the pen being thrown into the air with as much physical realism as allowed with limited resources; in this case, time.

My Submission

Given the above goal, my simulation design is intended to represent our pen as a single entity, moving in air, lofted under the force of a human hand, somewhere near the surface of the earth. Therefore we ignore deformations of the pen, the rotation of the earth, and certainly any effects due to special relativity. We are therefore left with the motion of a rigid body under Newton's laws, in the presence of a constant gravitational field, possibly immersed in the fluid known as the atmosphere.

The equations for such motion are well known, and easily found elsewhere, such as on the Wikipedia page for Rigid Body Dynamics. From this reference, we are reminded that the motion of a rigid body can be broken up into two parts, the motion of the center of mass, and the rotation about the center of mass.

CM Motion

The motion of the center of mass is described easily by Newton's 2nd Law, F_{total} = ma. Considering the motion of the pen in the air, the forces acting on it are gravity, and aerodynamic drag. Considering the standard formula of aerodynamic drag,

$$F_\mathrm{d} = \frac{1}{2}\rho u^2 c_\mathrm{d} A$$

The magnitude of the aerodynamic drag for likely values of density (1 kg/m^3), velocity (1 m/s), c_d (1), and A (.05 m in length by x .005 m width, or ~ 10^{-4} m^2), is a magnitude of 10^{-4} N. Comparing this to the force of gravity, mg = .01 kg * 10 m/s ~ 0.1 N. Hence we neglect the aerodynamic drag, and conclude that for the center of mass motion the essential dynamics are the same as a simple particle moving under a constant acceleration due to gravity, dv/dt = -g. This has closed form solution:

$$\mathbf{s}(t) = \mathbf{s}_0 + \mathbf{v}_0 t + \frac{1}{2}\mathbf{a}t^2 = \mathbf{s}_0 + \frac{1}{2}\left(\mathbf{v}_0 + \mathbf{v}(t)\right)t$$

The closed form solution can be leveraged for analyzing simulation fidelity, for example, in unit tests.

Rotational Motion

Rotational motion for rigid bodies is described by Euler's Equations:

$$\begin{aligned} I_1\dot{\omega}_1 + (I_3 - I_2)\omega_2\omega_3 &= M_1 \\ I_2\dot{\omega}_2 + (I_1 - I_3)\omega_3\omega_1 &= M_2 \\ I_3\dot{\omega}_3 + (I_2 - I_1)\omega_1\omega_2 &= M_3 \end{aligned}$$

where the Is are moments of inertia in the body axis, and the omegas are the time derivatives of Euler angles in a convenient representation.

For the case of a pen as pictured above, we make the simplifying assumption that the inertial moments are well described by that of a right circular cylinder. If we choose the 3rd axis (I_3) as our axis of symmetry, then I_1 = I_2. (This assumption could be validated by discussion of Parallel Axis Theorem. We skip for brevity.)

The Ms above are the torques on the system. Gravity acts at the center of mass, so provides no torque. Similar to the above analysis for aerodynamic drag on the center of mass, we can also conclude that the effect of aerodynamic drag can be neglected for rotational motion. Therefore the M's all vanish in the Euler equations, and we are left with so-called Torque Free motion of a rotating rigid body.

Representation of State

Given the above discussion, we represent the state of the spinning pen with the six rigid body coordinates (x, y, z, theta, phi, psi), and their derivatives (vx, vy, vz, theta_dot, phi_dot, and psi_dot). Hence, we consider a 12 dimensional state vector.

Integration of Equations of Motion

The equations of motion are represented by the six 2nd order differential equations of Newton's Law and Euler's Equations, as above. Using the state vector described above, we turn this into the 12 first order differential equations of the form:

$$\dot{\boldsymbol{x}} = \boldsymbol{v}(t, \boldsymbol{x})$$
$$\boldsymbol{x}|_{t=0} = \boldsymbol{x}_0$$

Numerical integration of this system can be done by any number of means. We choose the so-called Midpoint Method, as it represents a good balance between expense and performance.

Overall Design

A modular design was chosen to maximize testability, comprehension, and extensibility. The sim itself is a class which can be instantiated as many times as desired. Separate header files are given for each class or library used in the development and test of the simulation. (DynamicEntity, Integrators, TestUtils, and Simulation.) Each class has unit tests, and these are contained in the _Test files of the same name. To extend the simulation for another type of DynamicEntity, all one needs to do is create a class that conforms to the interface expected by Integrator and Simulation. It can then be simulated.

The most important class in this design is the Simulation class. (See Simulation.h). To run a new simulation, clients need only instantiate a DynamicEntity initialized to its initial dynamic state, instantiate a Simulation with it and some parameters (such as total time of simulation, and time step, etc), and then call the Simulation::Run() method. The Simulation will run and output DynamicEntity status as specified by the client.

Trade Offs

There are few that are of note:

Polymorphism: A template-based, data driven was chosen for polymorphism. DynamicEntity objects are cheap to copy, and do not inherit from a base class. Any object conforming to the interface specified there can be used. This is done to enhance cache-friendliness, and overall speed of the simulation. However, it does lead to some duplication of code (since there are no virtual functions), a lack of encapsulation (state variables are public), and a possible proliferation of template types leading to a larger executable size.

Header-only Design: Header only library designs are considered to produce faster code, as they present the compiler with opportunities for in-lining functions, etc.. However they increase build times as the class/library must be recompiled by every client. Here we use header only simply for convenience of development.

Modular Sim: The simulation is implemented as a class, Simulation, rather than as an executable. This allows multiple Simulations to be instantiated in a given executable. However, that means individual sim steps are not interactable by a client. New elements would have to be added to the interface to allow for a client to examine variables while the sim is running.

Numerical Integration: As noted above, the Midpoint Method was chosen. This is a middle-of-the-road approach which balances speed (it's only second order, which is neither as fast as first order Euler, or as accurate as 4th order Runge-kutta.)

Physical Fidelity: As mentioned above, we chose a relatively simple approach to the dynamics, neglecting aerodynamic drag, and any non-uniformities of inertia tensor. The trade off here was speed of development vs physical fidelity.

Contents

This is a brief description of the contents of the zip file.

`build_and_run_example_sim.sh`: Shell script which builds and runs an example. Sim outputs the pen dynamic state to stdout every time step.

`build_and_run_tests.sh`: Shell script which builds and runs all unit tests. Unit test results are output to stdout.

`clean_all.sh`: Shell script which deletes all build artifacts.

`Simulation.h`: Main simulation class. See use in unit tests and ExampleSim.cpp

`Example_Sim.cpp`: As above.

`DynamicEntites.h`: Contains several dynamic entities useful in testing and developing sim, such as ConstantVelParticle, SimpleSpringMotion, CrudeSpinningPen. Defines interface required for Simulation and Integrators.

`Integrators.h`: Contains numerical integrators used in stepping simulation forward, including EulerStep and MidpointMethod.

`Simulation_Test.cpp, Integrators_Test.cpp, DynamicEntities_Test.cpp`: Unit tests for classes of corresponding names.

`TestUtils.cpp`: Test functions used by unit tests.

Next Steps:

Were I to keep working on this, here are things I'd do next:

1) Add visualization. The simplest thing to do is have the output saved out in .csv for easy plotting. Beyond that, provided per step updates would be great for possible 3D visualization. It might also be good to write the results out to a bag.
2) Add air resistance. I'm really curious how this might change the dynamics besides just retarding the motion. In particular, adding boundary layer friction to the spinning motion might give some interesting lift effects.
3) More unit tests, of course!