15-869 References 1/28

I was very excited to see the first paper listed below appear in SIGGRAPH 2000. At that time, and in part due to this paper, I started teaching dynamic simulation in detail in my courses, because to create a system like this at that time, you needed to be able to roll your own simulator. I tried for several years to make writing a physically based simulator such as this one a standard course project. What I found, however, was that to succeed at this type of programming you had to be extremely detail oriented, intuitive, and really pretty amazing at doing unit tests and debugging – frankly it was far too difficult as a standard class project. However, it is a wonderful final project if you fit the above description. Here is that motivating paper.

Laszlo, Joseph, Michiel van de Panne, and Eugene Fiume. "**Interactive control for physically-based animation**." In Proceedings of the 27th annual conference on Computer graphics and interactive techniques, pp. 201-208. ACM Press/Addison-Wesley Publishing Co., 2000.

http://www.dgp.toronto.edu/~jflaszlo/interactive-control.html

My handout “spatial.pdf” walks through how to create a forward dynamic simulation of the Luxo lamp character using Featherstone dynamics and associated Spatial Vector notation. This handout can be found linked off the course web page. There may even be starter code still available from my CS229 class offered at Brown University. I haven’t checked..

Extremely useful background on physics simulation for computer graphics in general can be found in the 1997 SIGGRAPH course notes for the course “**Physically Based Modeling: Principles and Practice**,” by David Baraff and Andrew Witkin. However, this course does not cover simulation of animated characters. http://www.cs.cmu.edu/~baraff/sigcourse/

I have found the easiest way to implement and understand forward dynamics is by implementing and using Featherstone inverse dynamics. This is what I have done in my own code, and the approach is outlined in “spatial.pdf” and is covered in class. However, there is also a more efficient recursive forward dynamics algorithm courtesy of Roy Featherstone. If you really want to understand Featherstone forward dynamics, the go-to reference for learning this algorithm from the ground up is Brian Mirtich’s 1996 dissertation, which can be found here. (See Chapter 4.) Note that the Appendix in Mirthch’s thesis also contains some very useful mathematical information and derivations.

Mirtich, Brian Vincent. "**Impulse-based dynamic simulation of rigid body systems**." PhD diss., University of California, 1996. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.108.783

The following paper is a practical overview of using the Featherstone forward dynamics algorithm from a game development perspective:

Kokkevis, Evangelos. "**Practical physics for articulated characters**." In Game Developers Conference, vol. 2004.
http://www.computerscience.nl/docs/vakken/mgp/literature/Practical%20Physics%20for%20Articulated%20Characters.pdf

If you want another view on recursive inverse dynamics, see the following tutorial from Karen Liu (Chapter 7). This tutorial is an excellent resource in general. For example, it also includes a derivation of Lagrangian dynamics, from which the familiar equations of motion can be obtained if you would like to see more detail. This tutorial was written in the context of introducing Karen Liu's simulator RTQL8.
http://www.cc.gatech.edu/~karenliu/RTQL8_files/dynamics-tutorial.pdf
http://www.cc.gatech.edu/~karenliu/RTQL8.html

Emanuel Todorov has recently introduced a simulator MuJoCo to the robotics community, which can be found using the reference and link below.

Todorov, Emanuel, Tom Erez, and Yuval Tassa. "**MuJoCo: A physics engine for model-based control**." In Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on, pp. 5026-5033. IEEE, 2012. http://homes.cs.washington.edu/~todorov/papers/TodorovIROS12.pdf

http://www.mujoco.org/

As preparation for this class, I did some overview of features of various dynamics simulators, including looking at which dynamics formulations they used. My quick notes, copied and pasted from the various web pages, are given below.

ODE http://ode-wiki.org/wiki/index.php?title=Manual:_Introduction

- Rigid bodies with arbitrary mass distribution.
- Joint types: ball-and-socket, hinge, slider (prismatic), hinge-2, fixed, angular motor, linear motor, universal.
- Collision primitives: sphere, box, cylinder, capsule, plane, ray, and triangular mesh, convex.
- Collision spaces: Quad tree, hash space, and simple.

- Simulation method: The equations of motion are derived from a Lagrange multiplier velocity based model due to Trinkle/Stewart and Anitescu/Potra.
- A first order integrator is being used. It's fast, but not accurate enough for quantitative engineering yet. Higher order integrators will come later.
- Choice of time stepping methods: either the standard ``big matrix" method or the newer iterative QuickStep method can be used.
- Contact and friction model: This is based on the Dantzig LCP solver described by Baraff, although ODE implements a faster approximation to the Coloumb friction model.

Bullet Physics Library http://bulletphysics.org/wordpress/

Some of the features include:

- Featherstone Articulated btMultiBody, with two-way interaction with regular btRigidBody. See Bullet/Demos/FeatherstoneMultiBodyDemo. Thanks to Stephen Thompson for the initial Featherstone implementation, and Martin Felis and Jakub Stepien for the discussion and information.
- New MLCP constraint solver interface for higher quality direct solvers. Dantzig (by Russel Smith OpenDE), PATH and Projected Gauss Seidel Mixed Linear Complementarity solvers, with fallback to the original Bullet sequential impulse solver. In a nutshell, direct MLCP solvers are useful when higher quality simulation is needed, for example in robotics. The performance is less than the PGS/SI solver, but it usually convergence faster to a good solution. See src/BulletDynamics/MLCPSolvers and Bullet/Demos/ForkLiftDemo for an example.
- New btFixedConstraint as alternative to a btGeneric6DofConstraint with all DOFs locked. See Demos/VoronoiFractureDemo
- Various bug fixes, related to force feedback and friction. Improved performance between btCompoundShape
  using the new btCompoundCompoundCollisionAlgorithm. See the commit log at https://code.google.com/p/bullet/source/list

RTQL8 (pronounced "articulate"): Dynamic Multibody Simulator
http://www.cc.gatech.edu/~karenliu/RTQL8.html

RTQL8 is an open source library for developing kinematic and dynamic applications in robotics and computer animation. The multibody simulator is designed to aid development of motion control algorithms. RTQL8 uses generalized coordinates to represent articulated rigid body systems and computes Lagrange's equations derived from D'Alemberts principle to describe the dynamics of motion. In contrast to many popular physics engines which view the simulator as a black box, RTQL8 provides full access to internal kinematic and dynamic states, such as mass matrix, Coriolis and centrifugal force, transformation matrices and their derivatives, etc. RTQL8 also provides efficient computation of Jacobian matrices for arbitrary points in arbitrary coordinate frames.

MuJoCo http://www.mujoco.org/

Modeling and simulation of **Mu**lti-**Jo**int dynamics with **Co**ntact

KEY FEATURES

- simulation in generalized coordinates, based on efficient recursive algorithms that avoid joint constraint violations
- velocity-stepping for contacts, joint limits and friction, using novel impulse solvers
- multiple numerical integrators, including Runge Kutta methods adapted to a mixture of smooth and impulsive dynamics
- rich models of actuators and transmissions including motors, cylinders, biological muscles, tendons, slider-cranks
- intuitive XML model format, and ability to parse URDF models