Lab project 1.3 – Pile of boxes

In this programming task you will develop a simulator that can handle a small stack of boxes. You will implement intersection tests and finds for box-box and box-plane, and resolve collisions and contacts for these objects. You can get 3 extra bonus credits on this lab project.

The following is mandatory:

1. Set up the framework you need to represent multiple rigid body boxes and a plane/halfspace.
2. Implement the intersection test/find code reusing the Box-Box and Box-Plane code that is used in ODE. See: http://ode.org and http://sourceforge.net/project/showfiles.php?group_id=24884
   Look for e.g. dCollideBoxBox and dCollideBoxPlane in the file collision_std.cpp. The code dependencies are mainly to the math library and mainly consist of macros. Use *O(N(N-1)/2)* tests, i.e. test all objects against all other objects.
   If you instead implement all intersection finds/tests yourself, you will get 1 bonus credit (see below).
3. Use *one* of the following methods to resolve collisions and contacts. The methods are described in detail further down in this document.
   a. A sequential/iterative impulse method for resolution of collisions and contacts.
   b. A penalty based method.
4. Treat the interaction with the constrained "bone" from project 1.2 in an approximate way, i.e. just do two body exchanges of collision impulses.
5. Neglect the velocity dependent forces/torques (i.e. assume the inertia tensor has spherical symmetry so that the gyroscopic term becomes zero).
6. Run the first part of the scene with the "bone" sliding on the wire, hitting the pile, resulting in falling boxes.
7. Follow the instructions on the course web and write the first half of your lab report.

It is possible to gain *maximum 3 extra bonus credits* on this lab project:

1. Instead of using the intersection test/find code from ODE, you may use the intersection test code from Eberly's book and add what you need in order to compute the contact sets. Make sure you handle special cases, like when the edges are parallel. See lecture notes and discussions in Eberly (chapter 5, second half). See also the examples and improvements suggested in Kenny Erleben's PhD thesis:
   http://www.diku.dk/~kenny/thesis.pdf (p. 178 – 183)
   It is highly recommended that you visualize your contact points and normals for sanity check and debugging! *(1 bonus credit)*
2. Test both the impulse method and the penalty based method for resolving collisions and contacts, and do a brief (visual and/or measured) comparison of precision, stability and performance for box stacking.
   (2 bonus credits)

3. Add functionality for analyzing the box-pile simulation. Compute the total energy (i.e. translational, rotational and potential) and measure it during the simulation and plot as a function of time and indicate collision events (unless you add the plot to the scene you may use e.g. gnuplot, matlab or your favourite plotting software). Discuss your findings. Also measure overlaps (or levitation!) and comment on how this can be used to analyze the quality of your simulator. If working on the impulse-based simulator, analyze the convergence rate in the iterative steps. If working on the penalty-based simulator, analyze stability and precision related to timestep, spring constant and overlap. *(2 bonus credit)*
4. In case you develop the impulse simulator, add an extra corrective round of iterations utilizing anchor points (i.e. the fixed plane) and incrementally let the impulse solver propagate from bottom to top of the stack with zero restitution coefficient (i.e. smear out errors). See lecture notes for details. *(1 bonus credit)*
5. In case you develop the spring-and-damper simulator, improve the friction spring by handling static and dynamic friction separately, i.e. if friction force is within the cone, keep the spring fixed, but otherwise move it to the appropriate new anchor point (see lecture notes). *(1 bonus credit)*

Impulse based simulation method

Assuming you have obtained a useful contact set, you will now use the Newton Coulomb Impulse method described in the lecture notes to resolve collisions and resting contacts. The simulation will have the following structure:

Advance timestep

- Do an intersection test/find to get contact set.
- Collision loop
    - Resolve all non-separating contacts using the Newton Coulomb impulse method by sequentially looping over rigid body pairs and their collision points, instantaneously updating your linear momentum, angular momentum and the corresponding linear velocities and angular velocities.
    - After each full pass, remove all separating contacts completely from the contact set used in coming passes, so that after 5-10 passes you only have a few or no non-separating contacts left.
    - Run the loop 5-10 times.
- Time integrate your linear momentum and angular momentum with external forces, i.e. gravity (only adds to the momentum) and possibly air damping (to both angular and linear momentum) and derive the updated velocities.
- From the previous iteration loop you might still have some non-separating contacts that must be resolved. In addition, the gravitational acceleration update of the velocities that you just did also might have introduced new non-separating contacts.
- Check for remaining non-separating contacts in the original contact set and treat all of these as resting contacts using Newton Coulomb impulses by sequentially looping over contact points, exactly as before. However, now incrementally change the restitution from -1 to 0 over

some 10-20 iteration passes to slow down resting contacts. Don't remove anything from the contact set this time, but let everything relax till the restitution is zero. A restitution of -1 means that impulses will be zero and your momenta and velocities aren't changed at all (so you can start with e.g. -0.9 and then do -0.8 etc. down to 0.0). A restitution of 0 means that all bounciness is removed.

- Update positions using e.g. explicit Euler (you can experiment with other methods too).
- Advance another timestep.

See lecture notes and Eberly for details about the Newton Coulomb impulse method. In particular you will need Eqn. 5.27 in Eberly for computing the magnitude of the impulse, but use the version in the lecture notes where Coulomb friction was added. The whole thing is implemented in the pseudo code on page 248 in Eberly, but there is a *typo* in this code: In the expression for `fNumer = …`, the last `kA` should be replaced by `kB`!

Be careful about initial conditions. Your simulator might not be a full general purpose simulator and therefore it might break if you e.g. start with overlapping boxes.

Comment on stability and performance.

Further information (paper, presentation, videos) about this method, developed by Guendelmann, Bridson and Fedkiw, and published at SIGGRAPH 2003 is found on their website:

http://graphics.stanford.edu/papers/rigid_bodies-sig03/

In this lab project we are using the simplest version of the methods described in this paper, where we e.g. just do one intersection test/find and reuse that during the entire timestep.

Penalty based simulation method

Assuming you have obtained a useful contact set, you will now use a spring-and-damper method to resolve both collisions and resting contacts.
Your simulator will have the following structure:

Advance timestep

- Do intersection test/find to get contact set
- Attach a damped spring at the contact point along the normal such that the force is zero for zero penetration and proportional to the penetration. Sum up forces and torques (also damping) to the involved rigid bodies.
- Attach another spring between an anchor point and the current contact point and implement the *simpler version of penalty friction*(see lecture notes). Sum up forces and torques.
- You may also interact with the "bone" using a spring-damper force.
- Integrate the system using e.g. leap-frog or your favourite integrator and increment time.
- Advance another timestep.

Comment on stability and performance and choices of damping coefficient and spring constant.