Overview

- rigid body dynamics in computer + video games
- simulation structure overview
- a selection of rarely mentioned issues and optimizations
- cutting corners and approximations
- NovodeX

Where does Physics Fit in?

layers of an interactive simulation:

- Behavior, AI: long way to go
- Physics: could be better
- Graphics, Sound: works great

state of the art:
- math + basic algs well known for a while now
- no “best” algorithm, implementation difficult
- TODO: features, robustness, performance, scalability

Isn’t Rigid Body Dynamics a solved problem?

why classic robotics research is only a start:

<table>
<thead>
<tr>
<th>Problem size</th>
<th>Robotics</th>
<th>Games</th>
</tr>
</thead>
<tbody>
<tr>
<td>~1 robot</td>
<td>virtual world</td>
<td></td>
</tr>
<tr>
<td>Configuration</td>
<td>derive motion eqs for one robot</td>
<td>very dynamic</td>
</tr>
<tr>
<td>Mechanisms</td>
<td>robot created so that motion eqs are simple</td>
<td>anything, ev. very redundant</td>
</tr>
<tr>
<td>Constraints</td>
<td>primarily equality (joints)</td>
<td>primarily inequality (contacts)</td>
</tr>
<tr>
<td>Accuracy</td>
<td>simulation</td>
<td>visually OK</td>
</tr>
</tbody>
</table>
Isn't Rigid Body Dynamics a solved problem?

- four game middleware companies use four conceptually very different simulation approaches
- I want to knock over a house made of individual bricks in real time...

State of the Art: Boxes

- universally applicable
- stress test for technology: who can make the tallest stack?
- good friction model is important

State of the Art: Cars

- boxes, plus:
  - suspension, steering, tires, aerodynamics, engine, gearbox, damage model

State of the Art: Cars

- conflict between fun factor and realism
- design controllers that override real physics
State of the Art: Bodies

Unreal Tournament 2003
2002, Epic Games

- boxes, plus:
  - joints, complex limits, joint friction

State of the Art: Bodies

Hitman 2
2002, IO Interactive

- only dead bodies for now: balancing is hard!
- augment live characters w. dynamics
  - clothes
  - secondary movement

Simulation Structure

let's write a simulator:

Find Contact Points, Normals
Compute Constraint Forces
Integrate Accelerations, Velocities

primary goal: maintain nonpenetration constraints between a group of rigid bodies

(TIP: start in 2D)

Sources of Error

why it won't work the first time:

Find Contact Points, Normals
Compute Constraint Forces
Integrate Accelerations, Velocities

an error introduced anywhere may prevent nonpenetration constraints from being satisfied
Collision Detection

- Most Collision detection research so far has dealt with:
  - determining if bodies intersect or not
  - penetration depth of convex bodies
  - distance between bodies

- We really need:
  - contact points and normals between eventually penetrating nonconvex bodies
  - lightweight data structures

Why Deal with Penetrations?

- Avoid expensive rollbacks
- Avoid simulation slowdown with continuous collision detection
- Avoid failure when user starts / puts it in a slightly nondisjoint state
- Or when forced into a nondisjoint state due to simulation error

Benefits of a Constant Time Step

- Forget about subdividing time step whenever a physics event occurs
- Physics events occur way too frequently
- But their effects are mostly negligible

Finding Good Contacts

- As few contact points as possible should be able to transmit all significant interactions between a pair of bodies
- Solution is not unique, and a good solution is increasingly difficult with high penetration depth
Baraffís Contact Constraint

\[ a_i \geq 0 \quad \text{nonpenetration constraint} \]

However, this does not uniquely determine the normal force. Additional constraints:

- \( f_i \geq 0 \) contact force is repulsive
- \( f_i a_i = 0 \) constraint force is workless

\( \Leftrightarrow \) if \( f_i \) then \( a_i = 0 \); if \( a_i \) then \( f_i = 0 \)

Stack \( n \) contacts' variables. Matrix form:

\[
\begin{bmatrix}
  a_0 \\
  a_1 \\
  \vdots \\
  a_n
\end{bmatrix}
= 
\begin{bmatrix}
  f_0 \\
  f_1 \\
  \vdots \\
  f_n
\end{bmatrix}
\]

**LCP**

LCP fun facts:
- Complexity between linear programming (LP) and quadratic programming (QP)
- Used for economics, simulation, optimization
- NP complete in general
- Fortunately, our matrix \( A \) is PSD
- This is an example of several special cases which are not NP complete
- Here, solution is found after solving a short sequence of linear equality systems of size \( n \times n \)

**Iterative vs. Pivoting Solvers**

\( \tilde{\text{LCP}} \) can be solved with either:

- Pivoting algo (like Gauss elimination)
  - They change the matrix
  - Do not provide useful intermediate result
  - May exploit sparsity well

- Iterative algo (like Conjugate Gradients)
  - Only need read access to matrix
  - Can stop early for approximate solution
  - Faster for large matrices
  - Can be warm started
Equivalent to LCP

- If you are computing contact forces to satisfy nonpenetration constraints in any way, you have written a certain kind of LCP solver.
- Even if you are using simple penalty methods, because if any of below don’t hold, you don’t have realistic motion:
  - $f > 0 \quad a >= 0 \quad f_a = 0$
- If your sim is only approximate, then the LCP solution is approximate:
  - For example, penalty methods are usually “bouncy”.
  - $|f_a| < \varepsilon$

Equivalent to LCP

- Does this mean we can’t write a better contact force solver than what is in the LCP textbooks?
  - No:
    - Matrix $A$ does not have to be explicit $n \times n$.
    - $a$ and $f$ do not have to be stored explicitly either.
    - You can work in a different space.
    - You can approximate in a wide variety of ways.
    - You can always come up with a transform of your inputs/outputs to a classic LCP formulation.
    - If you introduce more complex constraints, for the sake of realism, you may end up with a QP or NCP problem; the LCP is a special case.

Example: Configuration Space

- A matrix, while PSD, is in contact space:
  - $O(n^2)$ storage for $n$ contacts.
  - Not always sparse.
  - Ill conditioned.
- It is possible to reformulate into a configuration space problem, where $f$ is not expressed explicitly, and energy minimization constraint is on bodies’ accelerations.
  - Matrix $B$: $O(n \times m)$ storage ($m =$ no. bodies).
  - Always sparse.
  - Much better conditioning.

Friction

- Physics fact: friction forces can influence normal forces and vice-versa.
- Ignore effect of friction on normal forces and solve sequentially for best performance.
- But they have to be solved for simultaneously for best results.
### Joints

- The joint forces of an articulated system are equality constraints:
  - $a_i = 0$
  - $a = Af + b = 0$

- Solve for $f$ with any linear system solver.
- But special properties of $A$ (PSD, symmetric, sparse, etc.) make a carefully chosen solver superior.

### MLCP

- An articulated system with contact constraints results in both equality and complementarity constraints to be solved simultaneously.
- Mixed Linear Complementarity Problem.

- First $m$ rows of $A$ do not have constraint on corresponding terms of $f$, and $=0$, instead of $>0$.
- Pivoting or iterative LCP solvers can be generalized to solve MLCPs.

### Joint Limits and Actuators

- Joint limits can be modelled as contacts.
- Limits and contacts can be made "soft" by adding appropriate multipliers to the constraint equation.
- Actuators can also be formulated as equality or inequality constraints on velocity, and thus fit into the LCP scheme too.

### Force vs. Impulse

- Instead of computing contact forces, we may compute contact impulses.

- **Advantages:**
  - Reduced integration error: Impulses integrated only 1x, while forces 2x until they influence pose.
  - More control: It is OK to directly set the acceleration of objects without preventing constraints from being satisfied.

- **Disadvantage:**
  - Accelerations not necessarily continuous. (Not a problem in practice.)

- All algorithms work both with forces or impulses.
### Integration

- A classical way to cope with integration error is to choose a higher order integrator.
- Must integration and contact force determination be separate?
  - If the algorithm computing the contact forces knows about the type of integration scheme used, it can anticipate its error and compensate for it.
  - This way even fast Euler integration works great.
  - Big disadvantage: external effects not formulated as constraints have severe integration error.

### Friction Cones

- Express friction as an LCP constraint:
  - \( \mu f_i \geq r_i \): Coulomb friction law
  - \( r_i \geq 0 \): Friction force in direction of
  - \( r_i \cdot (k_i \cdot v_i) = 0 \): Sliding velocity

- Note: \( v, f, r \) are interdependent, so implementing this needs a slight generalization of LCP solver.

### Work at NovodeX

- If you are already knew all this and are interested in an exciting job or internship, contact me:
  - adam.moravanszky@novodex.com
  - or Matthias M., Iler.

### Hybrid Animation

- Dynamics needs to be able to coexist with canned animation, and kinematically controlled motion.
- Example: non-physical automatic door closing on box.
  - Mostly domain specific solutions:
    - Break box
    - Apply an arbitrary force to the box, and stall the animation of the door while box moves away.
    - Etc.
### Integration

- thus two common scenarios:
  - 1
    - try to pack all effects (friction, actuators, limits, spring and damper elements, external forces etc.) into LCP solver as some sort of constraint
    - solve whole system together
    - don’t worry much about integration
  - 2
    - implement most effects as external forces on system
    - make sure you have a good integrator!

### Example: Penalty Method

- no matrix is stored
- \( f \) is a function of interpenetration \( p \)
- but: \( p = a \) so \( f = F(a) \) still...
- after \( f \) is determined:
  - apply forces to bodies
  - integrate forward in time
  - get new \( p \)
- we encoded this \( \text{response} \) of the system as matrix \( A \)
- a good penalty method converges to a solution over time as an iterative LCP solver does