Game Physics

- An overview with focus on determinism

About me

- Joacim Jonsson,
- Started computer science at KTH 1996
  - Theoretical Computer Science
- Started working with AAA games 1997
    - Renegade Ops (2011), SEGA of America
    - Just Cause 2 (2010), Square Enix
    - Battlefield: Bad Company (2008), Electronic Arts
    - RalliSport Challenge 2 (2004), Microsoft
    - Headhunter (2001), SEGA Europe

About me

- Worked at
  - Amuze
  - Starbreeze Studios
  - Digital Illusions CE
  - Avalanche Studios
- On platforms Dreamcast and forward
- With most aspects of game engines anything from hand pipelining assembler, rendering, compression, animation, AI, network, physics, ...

What is Games Physics?

- You tell me, but simulating it usually involves
  - Rigid bodies
  - Shapes
  - Motion states
  - Constraints
  - Contacts
  - Impulses
  - ....

Time

- What is time and how do we measure it?
    - A value in the Cpu frequency counter
    - Delta time, dt, is the time-difference between two frames was presented to the viewer
    - Note: can be important where/when you do the sample

Time

- Different strategies when it comes to frame updates
  - Fixed frame rate,
    - Typically 30 or 60 hz
  - Non-fixed frame rate,
    - Measure time once per frame

“Motion picture”

- $s = v * dt$
  - Where v is the velocity
  - dt is the time between the frames are presented
  - s is the distance measured in the two frames

Update gotchas

- The order of things really matters
  - Moving gunmen problem
  - Each mistake can add one frame extra latency
  - Read player input before (!) character update
  - Set velocities before (!) physics simulation
  - Read back data after (!) physics simulation

Fixed delta time

- stall on vertical sync signal
- Simple
  - Still hard to make things 100% deterministic! (replay problem)
- Common for console games

Non-Fixed delta time

- Simple suddenly became complex
  - update (x dt); update (y dt) != update ((x+y) * dt)
  - Latency problem
  - Smoothing
  - Accumulated smoothing errors (cutscene problem)
- High end PC gamers and benchmarks expects it.

Physics simulation

NOTE: the order is not written in stone!

- Collision Detection
  - adds “contact constraints” on motion equations
- Solve
  - Adjust velocities so not to violate constraints
- Integration
  - Propagate bodies according to motions

Game Physics Evolution

- 1st gen, just does the 3 steps
  - deep penetrations (hard to solve)
  - missed collision events (run through walls)
- 2nd gen, time of impact events / backtracking
  - More accurate
  - Performance has horrible worst cases
- 3rd gen, predictive / pre stabilization..
  - Stable, good performance, some artifacts

Closer look at Collision Detection

- Separated into Broad and Narrow phases
- Narrow phase is detailed
  - Generates (potential) contact points
- Broad phase reduces workload
  - Sweepline algorithm
  - Tree(s)

Closer look at narrow phase

- Convex base primitives
- Closest distance, easy problem
  - Local minima is global minima
- Penetration, harder problem
  - Generally a simulation tries to avoid this
  - “shrinked” convexes with a tolerance radius

Broadphase/World Gotchas

- Out of broadphase performance

- Secret party at the world origin
  - Non set or local space transforms..
  - Always init transforms before adding to world

Collision Gotchas

- High Detail not always a good thing
  - Performance AND design issue
- Triangle meshes dont have a solid inside
  - Volumetric geometry better
- Small items cannot use shrink-trick

Closer look at Integration

- Body state
  - Position and orientation, velocity, angular vel, ..
- Evolves over time, differential equations of motion
- Euler forward integration, for position:

$$v = v + a\,dt$$

$$x = x + v\,dt$$

- Verlet integration
- Again: foo (x dt); foo (y dt) != foo ((x+y) * dt)

Closer look at Solve

!constraints violate constant acceleration!

- Maintains integrity of constraints
  - By applying impulses / adjusting velocities
  - Errors behave like rubber bands
  - Naive pairwise analysis result in endless jitter
- Systems of equations
  - Iterative methods

Solver gotchas

- Large relative mass differences
  - Iterative solvers converge very slow
  - Results in large errors -> rubber bands
- Chains / ropes
  - Error correction cancelling
- “Extreme” inertias
  - Inverse approx 0 and gyroscope spins
  - large errors on constraints
  - Tip: Inertia optimizing utility functions...

What about Ray Casts?

- Not really part of physics simulation
- But very useful tool for game logic
  - Bullets, “sensors”, ai, ...
- Performance often Broad phase related
  - Cast directly on bodies / shapes when you can
- Do you really need instant answers?
  - Schedule in the background when possible

Back to Time again

- Changes in dt
  - Integration somewhat sensitive
  - Constraint solving usually very sensitive
- Combine fixed and non-fixed dt?
  - Yes, at the cost of a slight latency
  - Non-fixed sections can interpolate fixed states
  - Physics dt decoupled

Design gotchas

- Never think the result of a physics setup is deterministic
  - Use fake / pre animated physics when needed

- you cannot plan the player actions in detail
  - You can only set the stage
  - more freedom -> less control
  - Dont try to make a movie

Game control gotchas!

- Set transform
    - Essentially it is rapid teleportation (!)
    - Penetrations during Collision Detection
        - Bad performance
        - Sometimes catastrophic!
    - Solver has to guess
        - Stuff end up in wrong places
- Set velocities instead!

Game control gotchas!

- Manually “attaching” objects together
  - Solver doesnt know about it
    - No force feedback

      (Infinite strength if specialized motion)

- Use shared motions or constraints!

Physics in a network environment

- Deterministic nightmare
  - Constant battle of error correction
  - Player accept errors if smooth correction
- Server based
  - One consistent “truth”
  - responsiveness
- client based
  - Security and cheat issues

Network / Multiplayer design

- Separate into classes
  - effects, debris, ..
  - vehicles, character, barrels, ..
  - collapsing buildings, “game-changing” events
- Use mixture of client-only, client-server, and pre-animated physics where appropriate!

Time saver

- VISUAL debugging is priceless
  - Stop guessing what is happening
  - Your visual cortex is amazing
    at analyzing information presented in a visual form

Questions?