Fixed Timestep

Why Do Physics Engines Use a Fixed Timestep?

Programming – Physics for Games

SPECIALIST EDUCATORS IN
GAMES, ANIMATION & FILM VFX

Contents

- Variable Timesteps and Delta Time
- Fixed Timesteps
- Temporal Aliasing

Delta Time

- By now, we should all be familiar with what the Delta Time is, and why we use it
  - Or you may want to review the *Frame Independent Movement* session in *Math for Games*
- Delta Time is a variable timestep
  - Its amount will change per frame
  - It is used to provide consistent timing for animations, and other time-sensitive aplicaitons

SPECIALIST EDUCATORS IN
GAMES, ANIMATION & FILM VFX

Delta Time

- There is a huge problem with using delta time for physics simulations
- The behaviour of the physics simulation will depend on the value of delta time we pass in
  - At best, the game could ‘feel’ different according to its framerate
  - At worst, spring simulations could explode and fast moving objects start moving through walls

Delta Time

- Floating point arithmetic has limited precision
  - When performing physics calculations, the floating point values are rounded
  - While small, this produces a small amount of error
    - What we simulate may be slightly different from what we predict via calculation
  - For example, for a velocity v, $(v * dt) + (v * dt)$ will be different to $2 * v * dt$

- If we want to ensure our physics simulations are reproducible, we can’t simulate using delta time
  - Its nice to know the simulation runs the same from one execution to the next
  - But its essential when trying to network using deterministic lockstep

Fixed Timestep

- The solution is to use a fixed timestep
  - But we want to keep the ability to render at different framerates

- We need to decouple the physics from the rendering framerate

Fixed Timestep

- Each frame we get a variable chunk of time (*frameTime*)
  - this is the delta time
- We advance the simulation in fixed time increments
  - We do as many updates as we can within *frameTime*
  - Any remaining time is added to the variable *accumulatedTime* and used in the next frame

```
set fixedTimeStep = 0.01
set currentTime to system time
set accumulatedTime to 0

while game has not exited
   set newTime to system time
   set frameTime to currentTime - newTime
   set currentTime to newTime

   add frameTime to accumulatedTime

   while the accumulatedTime is >= fixedTimeStep
       updatePhysics(fixedTimeStep)
       accumulatedTime -= fixedTimeStep;

   render()
```

Temporal Aliasing

- When the previous code runs, we can get the occasional jitter
  - In general, all frames will have some small remainder of frame time left in the *accumulatedTime* variable
    - This can’t be simulated because its less than the fixed timestep value
  - This means we display the state of the physics simulation at a time slightly different to the render time
    - This is what causes the physics simulation to occasionally stutter

Temporal Anti-aliasing

- One solution is interpolate between the last and current state:
- For each physics object
  - Track it’s last state
  - Calculate it’s current state
  - Interpolate between the two states based the value of *accumulatedTime*
    - *alpha = accumulatedTime / fixedTimeStep;*

aie
SPECIALIST EDUCATORS IN
GAMES, ANIMATION & FILM VFX

Summary

- A variable timestep is not appropriate for physics simulations
- To ensure replicability, we need to use a fixed timestep for physics
- A fixed physics timestep may cause a problem known as temporal aliasing
  - Temporal aliasing can be solved by interpolating between the past and present state of a physics object

References

- Gaffer on Games. 2017. *Fix Your Timestep!*. [ONLINE] Available at: http://gafferongames.com/game-physics/fix-your-timestep/. [Accessed 13 June 2017].