Pausing a Real-time Simulation

Real-time Simulation

Consider a simulation that demonstrates something in real time. For simplicity, it could be a console application that simply reports the number of steps that have occurred every hundred steps or so. More generally, it could be an animation of a machine, or even an entire factory.

In order to isolate the topic of discussion, suppose the timer is decoupled from the simulation as shown in the C++ code below.

```
class SimulationBase
{
public:
        virtual unsigned int GetTimeStepInMicroseconds() = 0;
        virtual void Start() = 0;
        virtual bool Step() = 0; // return true to stop
};

class Timer
{
        ...
        SimulationBase* pSimulationBase = 0;
        ...
};
```

The simulation tells the timer its desired time step, Δt. For an animation, one might have $\Delta t = 16666$ microseconds, about one sixtieth of a second. (Working with a whole number of small time units, as opposed to a fraction of a second, avoids floating point error.)

The timer starts the simulation and calls Step() every Δt microseconds. If t is the total time that has passed since the beginning of the simulation, and n is the number of steps, then the time equation is

$$t = n\Delta t.$$

An algorithm for the timer is shown in Algorithm 1.

Algorithm 1

Let $n = 0$
Loop:

Accurately update t using the system clock.
If $n\Delta t < t$ call Step() and increment n.

Pausing the Simulation

A simulation may be paused in one of three ways:

1. at the user's request
2. when a step takes longer than Δt
3. when a stop is reached while debugging the simulation

Algorithm 1 behaves the same way in all three cases. When the simulation resumes, $n\Delta t$ will be far less than t and steps will occur in rapid succession until the simulation has caught up. If, for example the simulation involves a car moving at constant speed, on resume, the car will move much faster until it reaches the position it would have reached had the pause not occurred. This is because actual time t carries on while the simulation is paused.

A more natural behavior, is for the simulation to resume as if the pause had not occurred. One cannot, however, pause actual time. This motivates the introduction of a delay value d which is the amount of actual time elapsed during the pause (or the sum of the durations of many pauses). The time equation becomes

$$t = d + n\Delta t.$$

The resulting algorithm is Algorithm 2.

Algorithm 2

Let $n = 0$
Let $d = 0$
Loop:

Accurately update t using the system clock.
If $d + n\Delta t < t$ call Step() and increment n.
If $d + n\Delta t < t$ (still), update $d = t - n\Delta t$.

Simulated Time

The simulation itself is oblivious to whether a pause occurs or not. If the quantity $n\Delta t$ is computed within the simulation, it is the amount of *simulated* time. Indeed,

the timer (with yet another algorithm), may present the simulation at double speed or perhaps as fast as possible, and this in no way changes the succession of states within the simulation. Contrast this with doubling the speed of a simulation in Unity by doubling the velocities used within the Update() calls.