Game Physics

Game and Media Technology
Master Program - Utrecht University

Dr. Nicolas Pronost

Numerical Integration

Updating position

- Recall that $FORCE = MASS \times ACCELERATION$
  - If we assume that the mass is constant then
$$F(p_o, t) = m * a(p_o, t)$$
  - We know that $v'(t) = a(t)$ and $p_o{}'(t) = v(t)$
  - So we have $F(p_o, t) = m * p_o{}''(t)$

- This is a differential equation
  - Well studied branch of mathematics
  - Often difficult to solve in real-time applications

Taylor series

- Taylor expansion series of a function can be applied on p_0 at $\mathrm{t} + \Delta t$

$$p_o(t+\Delta t) = p_o(t) + \Delta \mathrm{t} * p_o{}'(t) + \frac{\Delta t^2}{2} p_o{}''(t) + \cdots + \frac{\Delta t^n}{n!} p_o{}^{(n)}(t)$$

- But of course we don't know the values of the entire infinite series, at best we have $p_o(t)$ and the first two derivatives

First order approximation

- Hopefully, if Δt is small enough, we can use an approximation

$$p_o(t + \Delta t) \approx p_o(t) + \Delta \mathrm{t} * p_o{}'(t)$$

- Separating out position and velocity gives

$$v(t + \Delta t) = v(t) + a(t)\Delta t = v(t) + \frac{F(t)}{m}\Delta t$$
$$p_o(t + \Delta t) = p_o(t) + v(t)\Delta t$$

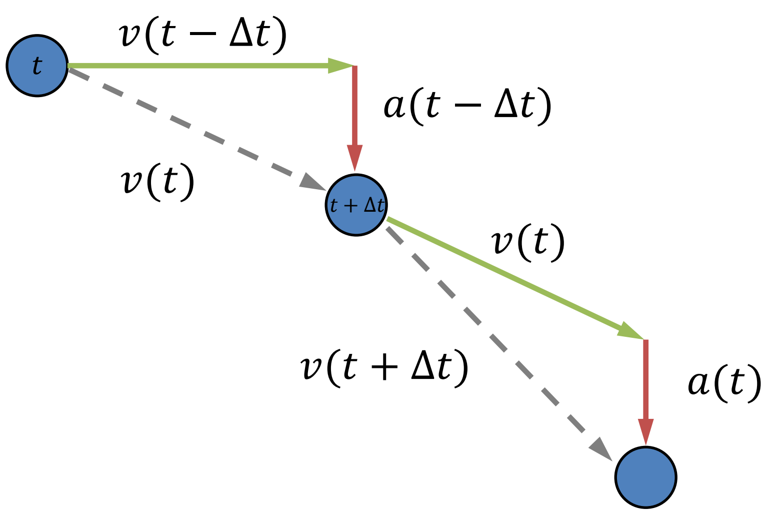
Euler's method

5.1

- This is known as Euler's method

$$v(t + \Delta t) = v(t) + a(t)\Delta t$$

$$p_o(t + \Delta t) = p_o(t) + v(t)\Delta t$$



Euler's method

- So by assuming the velocity is constant for the time Δt elapsed between two frames
  - We compute the acceleration of the object from the net force applied on it

$$a(t) = F(t)/m$$

  - We compute the velocity from the acceleration

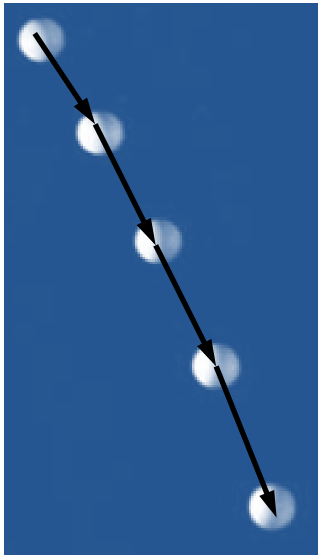
$$v(t + \Delta t) = v(t) + a(t)\Delta t$$

  - We compute the position from the velocity

$$p_o(t + \Delta t) = p_o(t) + v(t)\Delta t$$

Issues with linear dynamics

- We only look at a sequence of instants without meaning
  - *E.g.* little chance that we see the precise instant of bouncing



- Trajectories are treated as piecewise lines
  - we assume constant velocity and acceleration in-between frames

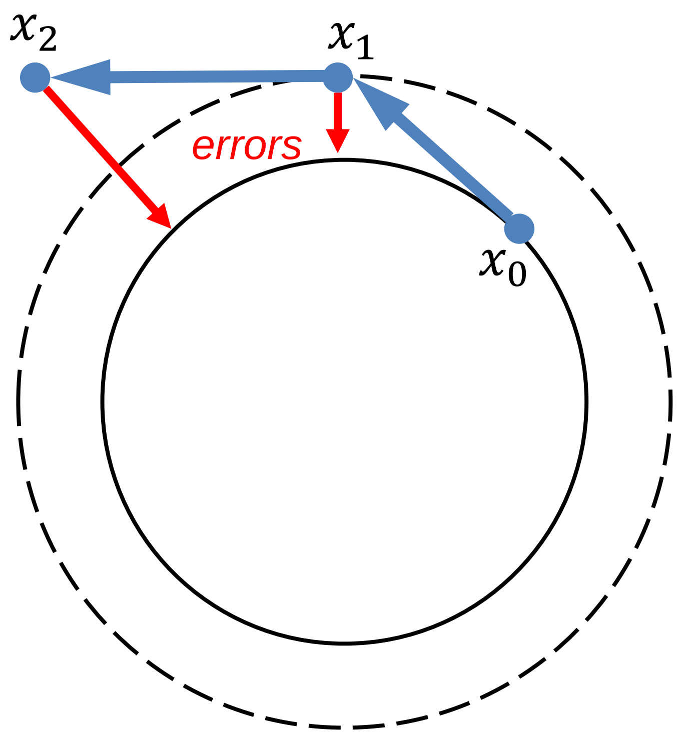
Time step

- The smaller Δt, the closer to $p_o(s) = \int_0^s v(t)\,dt$ the approximation, and so the more we can ignore these issues
- So the classic solution is to reduce Δt as much as possible
  - Usually frame rate of the game loop is enough
  - But sometimes more steps are needed (especially if frame rate drops)
    - we perform more than one integration step per frame
    - each step is called an iteration
    - if h is the length of the frame and n the number of iterations, then $\Delta t = h/n$ for each iteration of a step

Time step

- However, our assumption is that the slope at a current point is a good estimate for the slope over the entire time interval Δt
- If not, the approximation can drift off the function, and the farther it drifts the worse the tangent approximation can get

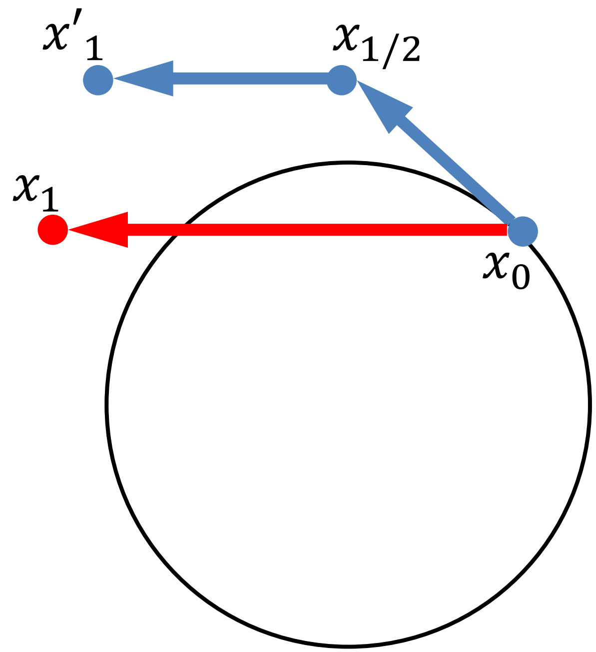
Error accumulation



- Accuracy is increased by taking the smallest step as possible, however more rounding errors occur and it is computationally expensive

Midpoint method

- In the midpoint method we calculate the tangent in the middle of the interval
  - using Euler's method on half of the desired time step
- And apply it to our point across the entire interval



Midpoint method

- The position of the point is given by

$$p_o(t + \Delta t) = p_o(t) + \Delta t * v\left(t + \frac{\Delta t}{2}, p_o + \frac{\Delta t}{2} v(t, p_o)\right)$$

- The order of the error is dependent on the square of the time step $O(\Delta t^2)$ which is better than Euler's method ($O(\Delta t)$) when $\Delta t < 1$
- Approximate the function with a quadratic curve instead of a line
- But still can drift off the function

Improved Euler's method

- The improved Euler's method considers the tangent lines to the solution curve at both ends of the interval
- It takes the average of two points, one overestimating the ideal velocity and one underestimating it
  - defined by the up/down concavity of the curve (not known in advance)
  - reduces Euler's method error as 'move back' the point towards the curve
- The order of the error is again $O(\Delta t^2)$ as the measure of the final derivative is still inaccurate

Improved Euler's method

- Velocity to the first point (Euler's prediction)

$$v_1 = v(t) + \Delta t * a(t, v)$$

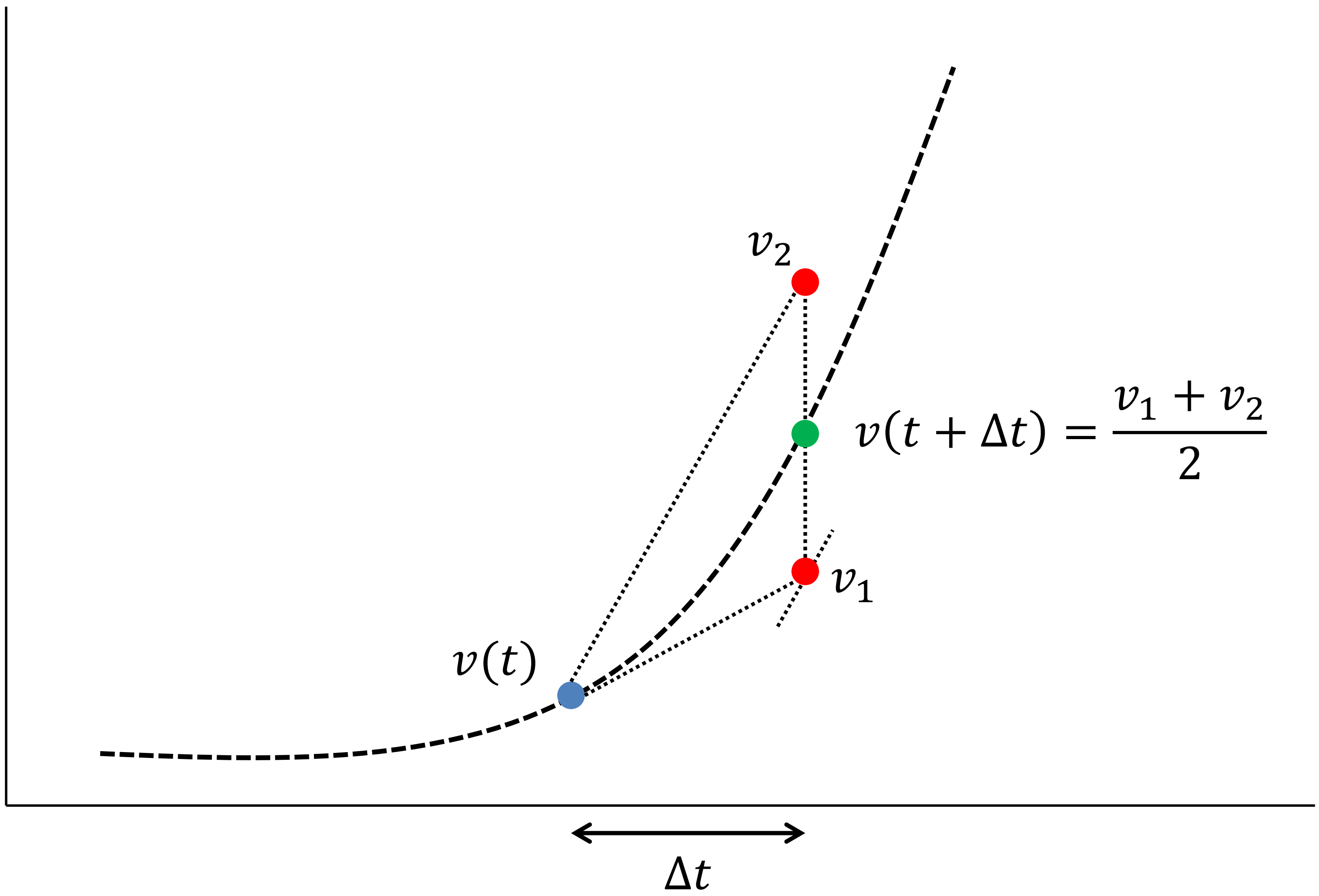
- Velocity to the second point (correction point)

$$v_2 = v(t) + \Delta t * a(t + \Delta t, v_1)$$

- Improved Euler's velocity

$$v(t + \Delta t) = \frac{v_1 + v_2}{2}$$

Improved Euler's method



Runge-Kutta method

- Hopefully there exist methods that give better results than a quadratic error
- The Runge-Kutta order four method (RK4) is for example $O(\Delta t^4)$
- It can be seen as a combination of the midpoint and modified Euler's methods where we give higher weights to the midpoint tangents than to the endpoints tangents

RK4

- We calculate the four following tangents

$$v_1 = \Delta t * a(t, v(t))$$

$$v_2 = \Delta t * a\left(t + \frac{\Delta t}{2}, v(t) + \frac{1}{2}v_1\right)$$

$$v_3 = \Delta t * a\left(t + \frac{\Delta t}{2}, v(t) + \frac{1}{2}v_2\right)$$

$$v_4 = \Delta t * a(t + \Delta t, v(t) + v_3)$$

- And weight them as follows

$$v(t + \Delta t) = v(t) + \frac{v_1 + 2v_2 + 2v_3 + v_4}{6}$$

RK4

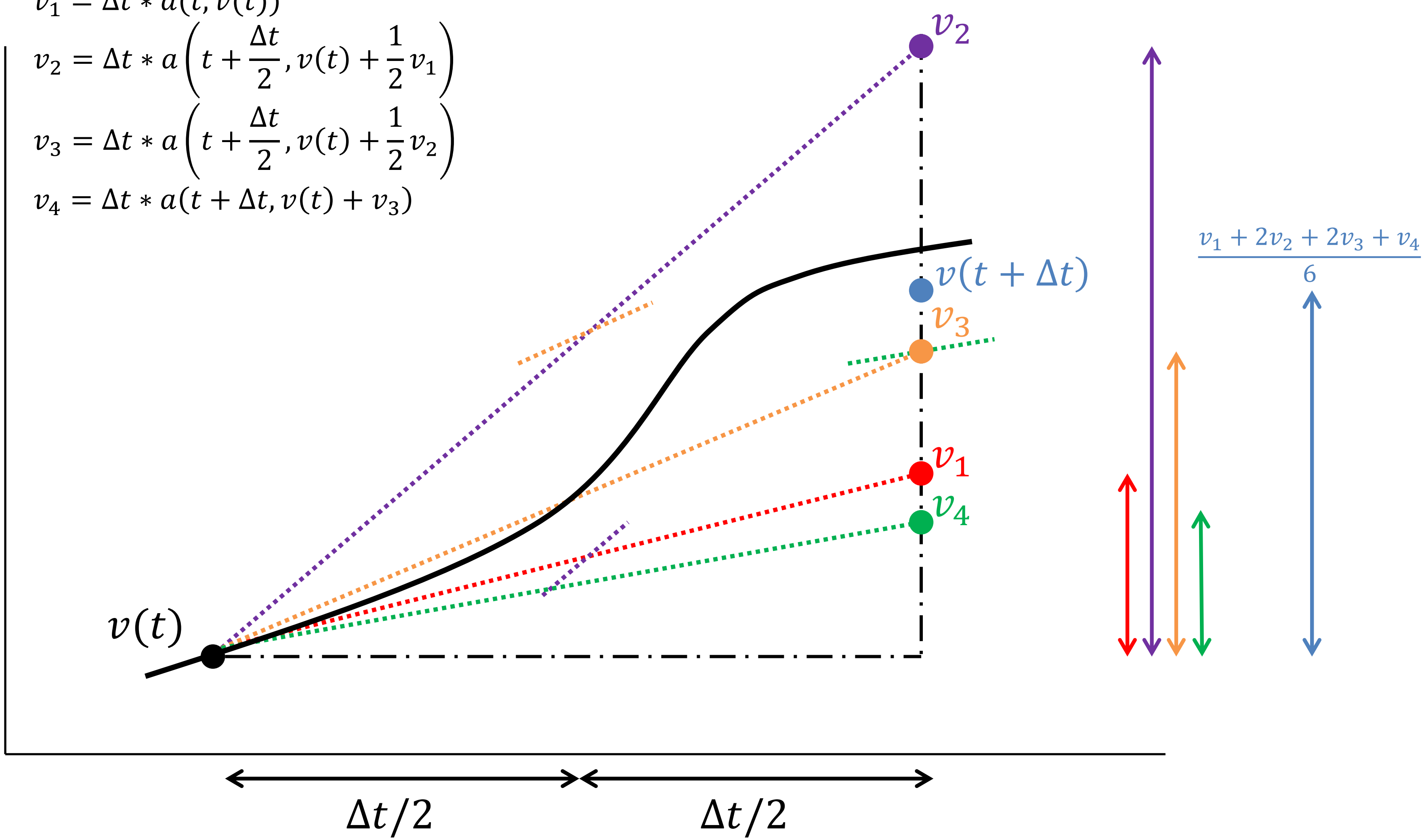
5.2

$$v_1 = \Delta t * a(t, v(t))$$

$$v_2 = \Delta t * a\left(t + \frac{\Delta t}{2}, v(t) + \frac{1}{2}v_1\right)$$

$$v_3 = \Delta t * a\left(t + \frac{\Delta t}{2}, v(t) + \frac{1}{2}v_2\right)$$

$$v_4 = \Delta t * a(t + \Delta t, v(t) + v_3)$$



Verlet integration

- The Verlet integration method is based on the sum of the Taylor expansion series of the previous time step and the next one

$$
\begin{aligned}
p_o(t+\Delta t) + p_o(t-\Delta t) \\
&= p_o(t) + \Delta t * p_o{}'(t) + \frac{\Delta t^2}{2} * p_o{}''(t) + \cdots \\
&\quad + p_o(t) - \Delta t * p_o{}'(t) + \frac{\Delta t^2}{2} * p_o{}''(t) - \cdots
\end{aligned}
$$

Verlet integration

- Solving for the current position gives us

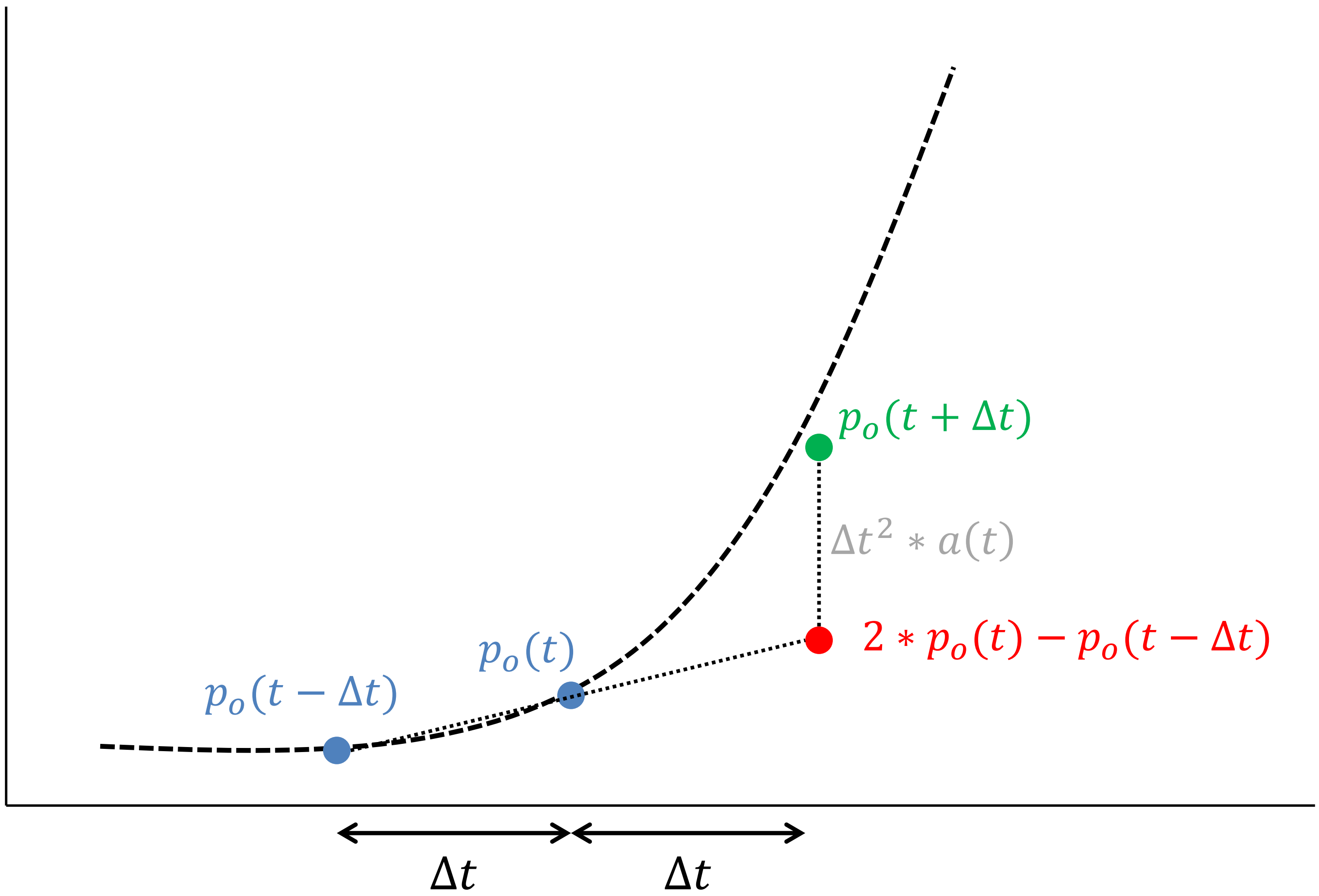
$$p_o(t+\Delta t) = 2p_o(t) - p_o(t-\Delta t) + \Delta t^2 p_o''(t) + \cdots$$

- If the higher terms in $O(\Delta t^4)$ are neglected again we get

$$p_o(t+\Delta t) = 2p_o(t) - p_o(t-\Delta t) + \Delta t^2 p_o''(t)$$

- Note that we do not explicitly use velocities

Verlet integration



Verlet integration

- It gives an order of error in $O(\Delta t^2)$
- Very stable and fast as does not need to estimate velocities
- But we need an estimation of the first $p_o(t - \Delta t)$
  - Usually obtained from one step of Euler's or RK4 method
- And more difficult to manage velocity related forces such as drag or collision

Implicit methods

- Every method so far used the current position $p_o(t)$ and velocity $v(t)$ to calculate the next position and velocity
  - this is referred to as explicit methods
- In implicit methods, we make use of the quantities from the next time step!

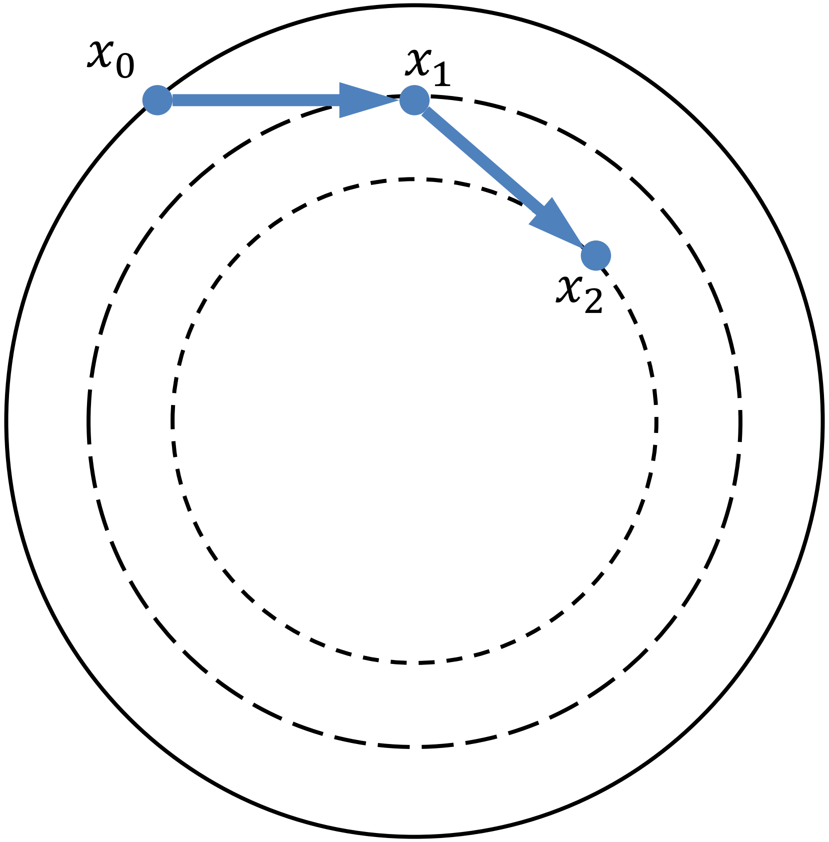
$$p_o(t + \Delta t) = p_o(t) + \Delta t * v(t + \Delta t)$$

  - this particular one is called backward Euler
  - the goal is to find the position $p_o(t + \Delta t)$ for which we would end up at p_o by running the simulation backwards

Implicit methods

- Implicit methods do not guarantee more accuracy than explicit methods
- But at least they do not add energy to the system, they lose some
- Since we usually want a damping of the position anyway (*e.g.* to simulate drag force or kinetic friction), it's a lesser evil

Backward Euler



Backward Euler

- But how do we calculate the velocity at a position we don't know yet?

- If we know the forces applied we can calculate it directly
  - For example if a drag force $F_D = -b * v$ is applied
    $$v(t + \Delta t) = v(t) - \Delta t * b * v(t + \Delta t)$$
  - And therefore
    $$v(t + \Delta t) = \frac{v(t)}{1 + \Delta t * b}$$

Backward Euler

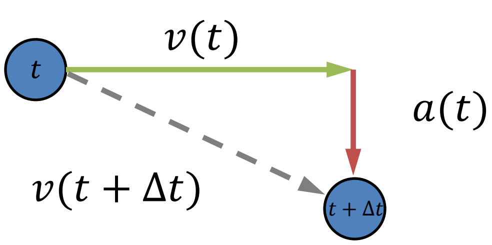
- If we don’t know the forces in advance (that happens continuously in a game) or if solving the previous equation is not possible, we use a predictor-corrector method
  - one step of explicit Euler’s method
  - use the predicted position to calculate $v(t + \Delta t)$
- More accurate than explicit method but twice the amount of calculation

Semi-implicit method

- The semi-implicit method provides simplicity of explicit Euler and stability of implicit Euler
- Runs an explicit Euler step for velocity and then an implicit Euler step for position

$$v(t+\Delta t) = v(t) + \Delta t * a(t) = v(t) + \Delta t * F(t)/m$$

$$p_o(t+\Delta t) = p_o(t) + \Delta t * \cancel{v(t)} = p_o(t) + \Delta t * v(t+\Delta t)$$



Semi-implicit method

- The position update in the second step uses the next velocity and the implicit method
  - good for position-dependent forces
  - and conserves energy over time, so very stable
- Usually not as accurate as RK4 because order of error is still $O(\Delta t)$ but cheaper and similar stability
- Very popular choice for game physics engine

Summary

- Many integration methods exist, each with its own properties and limitations
  - First order methods
    - Euler method, Backward Euler, Semi-implicit Euler, Exponential Euler
  - Second order methods
    - Verlet integration, Velocity Verlet, Trapezoidal rule, Beeman's algorithm, Midpoint method, Improved Euler's method, Heun's method, Newmark-beta method, Leapfrog integration
  - Higher order methods
    - Runge-Kutta family methods, Linear multistep method

Concluding remarks

- Dimension
  - We have shown integration methods for 1D variables
  - However, every dimension can be calculated separately using vector based structures
- Rotational motion
  - The integration methods work exactly the same for angular displacement θ, velocity ω and acceleration α
- Evaluation of all dimensions and variables should be done for the same simulation time t

End of
Numerical Integration

Next

Collision detection