CPG 7

Dr Wojciech Palubicki

Single Particle

glm::vec3 Pos

glm::vec3 Vel

glm::vec3 Force/m

Rigid Bodies

glm::vec3 Pos
glm::quat Rot

glm::vec3 Vel
glm::vec3 AngVel

Rigid Bodies

glm::vec3 Pos
glm::quat Rot

glm::vec3 Vel
glm::vec3 AngVel

glm::vec3 Force/m
glm::vec3 Torque/I

Rigid Bodies

glm::vec3 Pos
glm::quat Rot

glm::vec3 Vel
glm::vec3 AngVel

glm::vec3 Force/m
glm::vec3 Torque/I

glm::vec3 Pos
glm::quat Rot

glm::vec3 LinMom
glm::vec3 AngMom

glm::vec3 Force
glm::vec3 Torque

Differential equation

- Equations of the form $\dot{x} = f(x, t)$
- x is the position state vector of the system
- f is the function that computes the derivative of x with respect to time

For rigid bodies

- x f (x)

x	$f(x)$
position	velocity
velocity/linear momentum	acceleration/force
rotation	*angular velocity* $(\omega \star R)$
angular velocity/angular momentum	angular acceleration/torque

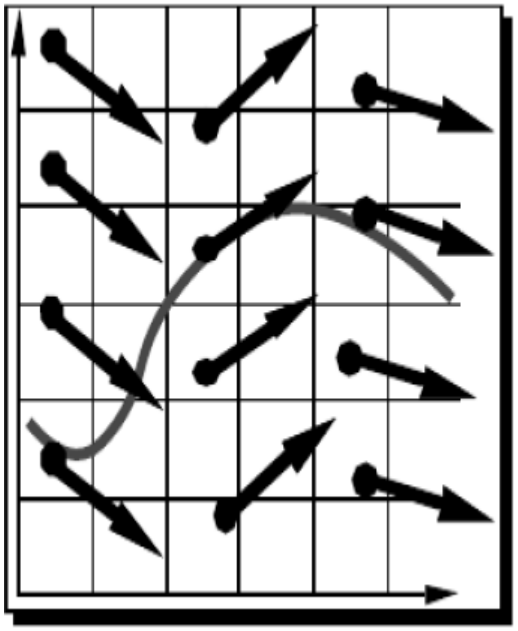
Numerical solutions

- We cannot just compute the state at time t
- There is no closed form for it, unless the problem is really simple
- We look for an **approximate** solution

Field of derivatives

- We know how to compute f
- This means that we can compute the **field of slopes** of x

Slope field



The derivative function

$$\dot{x} = f(x,t)$$

forms a vector field.

Position state vector of rigid bodies

$$\dot{\mathbf{X}}(t) = \frac{d}{dt}\begin{pmatrix} \mathbf{x}(t) \\ \mathbf{q}(t) \\ P(t) \\ L(t) \end{pmatrix}$$

$\mathbf{x}(t)$, $\mathbf{q}(t)$ → Spatial information

$P(t)$, $L(t)$ → Velocity information

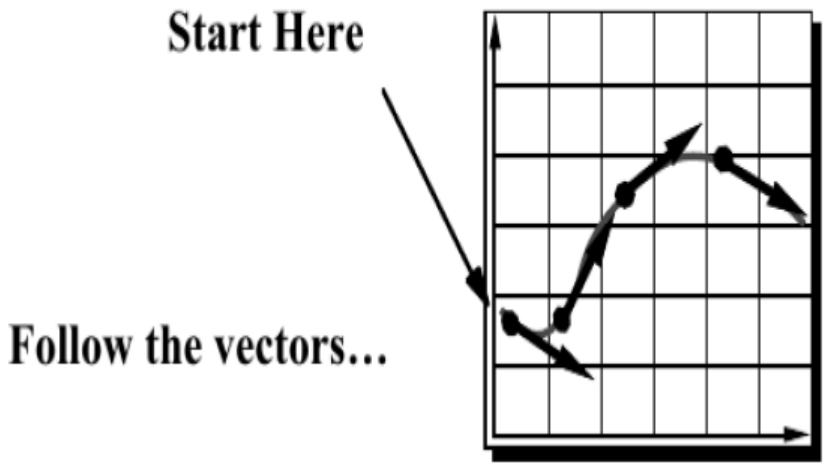
- Size of the vector: (3+4+3+3)N = 13N

Velocity state vector of rigid bodies

$$\dot{\mathbf{X}}(t) = \frac{d}{dt}\begin{pmatrix} \mathbf{x}(t) \\ \mathbf{q}(t) \\ P(t) \\ L(t) \end{pmatrix} = \begin{pmatrix} \mathbf{v}(t) \\ \frac{1}{2}\omega(t)\mathbf{q}(t) \\ F(t) \\ \tau(t) \end{pmatrix} = \begin{pmatrix} \frac{P(t)}{M} \\ \frac{1}{2}I^{-1}L(t)\mathbf{q}(t) \\ F(t) \\ \tau(t) \end{pmatrix}$$

Follow the slope

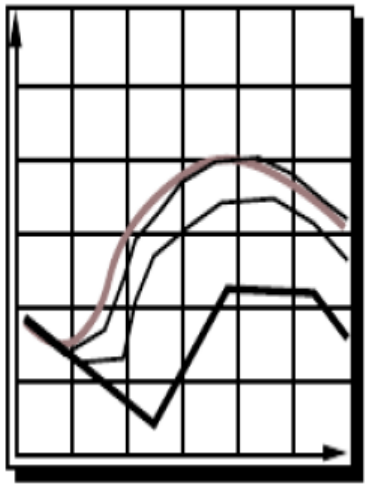
- Start at x_0
- Follow the slope
- Discrete steps



Euler's method

- Big, discrete steps along the slope: $x(t + h) = x(t) + h\dot{x}(t)$
- Works correctly and may be acceptable in some cases
- Simple to code and fast to run
- Otherwise it requires a very small time-step to compensate

Euler's method

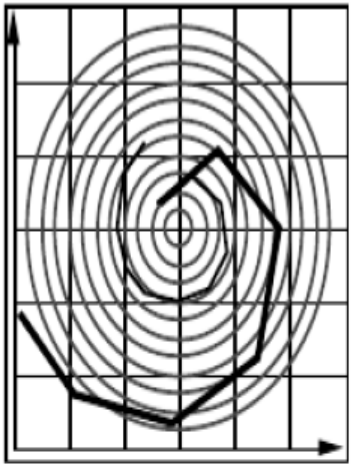


- Simplest numerical solution method
- Discrete time steps
- Bigger steps, bigger errors.

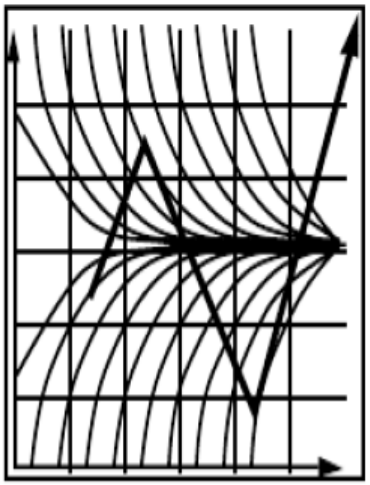
Euler's method

- Why may Euler's method not work?
- Euler's method is inaccurate - it may jump from one trajectory to another
- Not stable - it may increase the energy of the state where it should decrease it

Euler's method



Inaccuracy:
Error turns x(t) from a circle into the spiral of your choice.



Instability: off to Neptune!

Taylor series

Taylor series

- Any function can be expressed as the infinite sum of its derivatives

$$x(t + h) = x(t) + h\dot{x}(t) + \frac{h^2}{2}\ddot{x}(t) + \cdots + \frac{h^n}{n!}\frac{d^n}{dt^n}x(t) + \cdots$$

- We can **truncate** this summation to **approximate** the original function
- Most differential equations solvers are based on this technique

Deriving Euler's method

- $x(t + h) = x(t) + h\dot{x}(t) + \frac{h^2}{2}\ddot{x}(t) + \cdots + \frac{h^n}{n!}\frac{d^n}{dt^n}x(t) + \cdots$

Deriving Euler's method

- $x(t\ +\ h) =\ x(t) +\ h\dot{x}(t) + \frac{h^2}{2}\ddot{x}(t) + \cdots +$ $\frac{h^{\mathrm{n}}}{n!}\frac{d^n}{dt^n}x(t) + \cdots$
- $x(t\ +\ h) \sim x(t) +\ h\dot{x}(t)$

Deriving the midpoint method

- $x(t + h) \sim x(t) + h\dot{x}(t) + \frac{h^2}{2}\ddot{x}(t)$

Deriving the midpoint method

- $x(t + h) \sim x(t) + h\dot{x}(t) + \frac{h^2}{2}\ddot{x}(t)$
- We must now compute $\ddot{x}(t)$
- $\ddot{x}(t) = \frac{d}{dt}\dot{x} = \frac{d}{dt}f\big(x(t)\big) = f'\big(x(t)\big)\dot{x} = f'\big(x(t)\big)f(x(t))$

Deriving the midpoint method

- $\ddot{x}(t) = f'\big(x(t)\big)f(x(t))$

Deriving the midpoint method

- $\ddot{x}(t) = f'\big(x(t)\big)f(x(t))$
- We now approximate f with Euler's method
- $f\ (x_0\ + \Delta x)\ =\ f\ (x_0)\ + \Delta x f'_0(x_0)$

Deriving the midpoint method

- $f\ (x_0\ + \Delta x)\ =\ f\ (x_0)\ + \Delta x f'_0(x_0)$
- We (arbitrarily) choose $\Delta x\ = \frac{h}{2}\ f\ (x_0)$
- Substitute the approximation of f
- $f\ (x_0\ + \frac{h}{2}\ f\ (x_0))\ =\ f\ (x_0)\ + \frac{h}{2}\ f\ (x_0) f'_0(x_0)$

Deriving the midpoint method

- $f\ (x_0\ + \Delta x)\ =\ f\ (x_0)\ + \Delta x f'_0(x_0)$
- We (arbitrarily) choose $\Delta x\ = \frac{h}{2}\ f\ (x_0)$
- Substitute the approximation of f
- $f\ (x_0\ + \frac{h}{2}\ f\ (x_0))\ =\ f\ (x_0)\ + \frac{h}{2}\ f\ (x_0) f'_0(x_0)$
- We multiply both sides by h:
- $h(f\ (x_0\ + \frac{h}{2}\ f\ (x_0)) - f\ (x_0))\ =$
  $\frac{h^2}{2}\ f\ (x_0) f'_0(x_0)$

Deriving the midpoint method

- $f\ (x_0\ + \Delta x)\ =\ f\ (x_0)\ + \Delta x f'_0(x_0)$
- We (arbitrarily) choose $\Delta x\ = \frac{h}{2}\ f\ (x_0)$
- Substitute the approximation of f
- $f\ (x_0\ + \frac{h}{2}\ f\ (x_0))\ =\ f\ (x_0)\ + \frac{h}{2}\ f\ (x_0) f'_0(x_0)$
- We multiply both sides by h:
- $h(f\ (x_0\ + \frac{h}{2}\ f\ (x_0)) - f\ (x_0))\ =$
  $\frac{h^2}{2}\ f\ (x_0) f'_0(x_0) = \frac{h^2}{2}\ddot{x}$

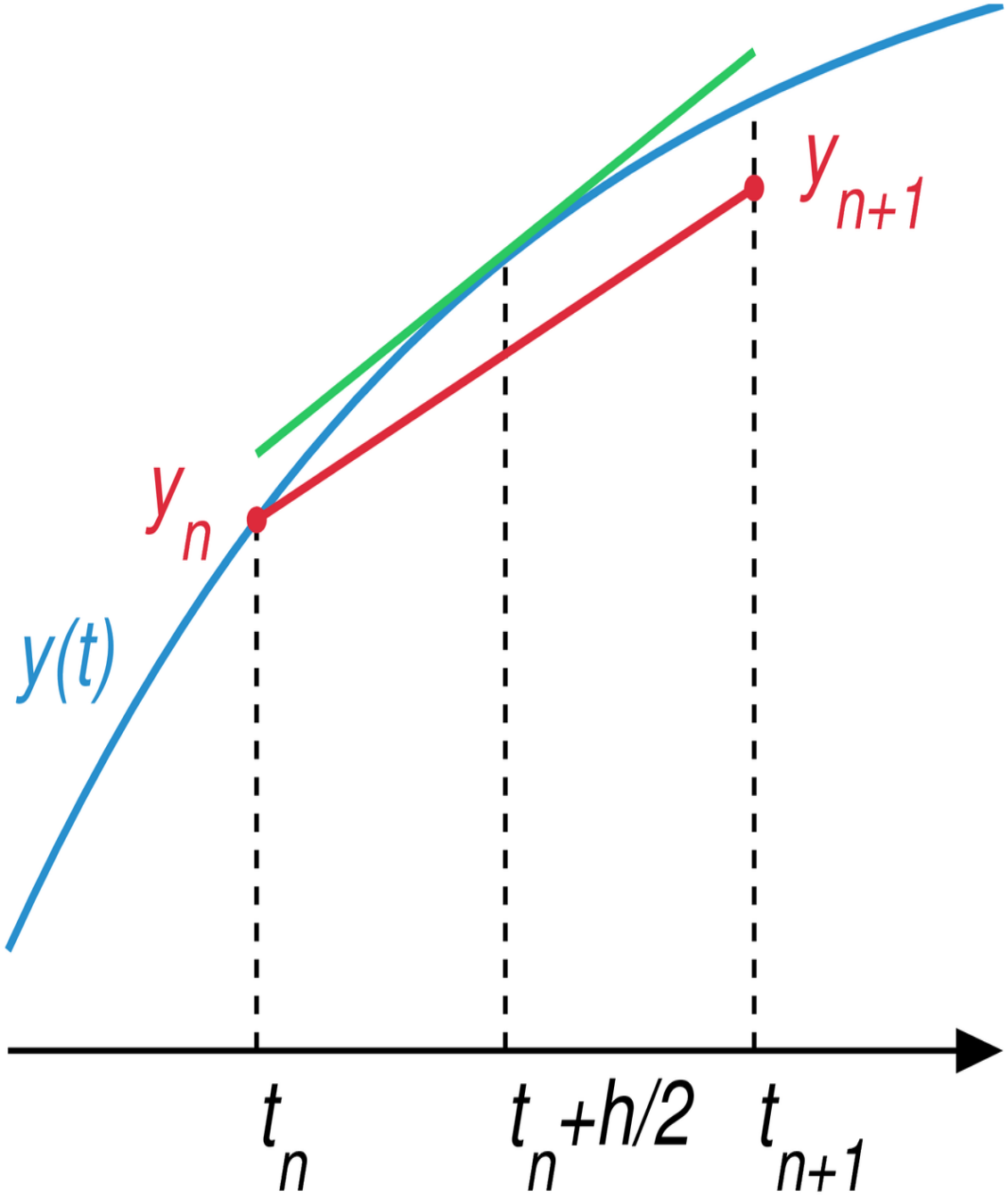
Deriving the midpoint method

- $x(t\ +\ h) \sim x(t) +\ h\dot{x}(t) + \frac{h^2}{2}\ddot{x}(t)$
- $\ddot{x}(t) = f'\big(x(t)\big)f(x(t))$
- $h\left(f\ \left(x_0\ + \frac{h}{2}\ f\ (x_0)\right) - f\ (x_0)\right) =$
  $\frac{h^2}{2} f'\ (x_0) f\ (x_0) = \frac{h^2}{2}\ddot{x}$
- $x(t\ +\ h) \sim x(t) +\ h\dot{x}(t) + h(f\ (x_0\ +$
  $\frac{h}{2}\ f\ (x_0)) - f\ (x_0))$

Deriving the midpoint method

- $x(t + h) \sim x(t) + h\dot{x}(t) + h(f(x_0 + \frac{h}{2} f(x_0)) - f(x_0))$

- We sample the derivative at the midpoint of the step, hence the name


y(t)
y n
y n+1
t n
t n+h/2
t n+1

Midpoint method

- $x(t + h) \sim x(t) + h\dot{x}(t) + h(f(x_0 + \frac{h}{2} f(x_0)) - f(x_0))$

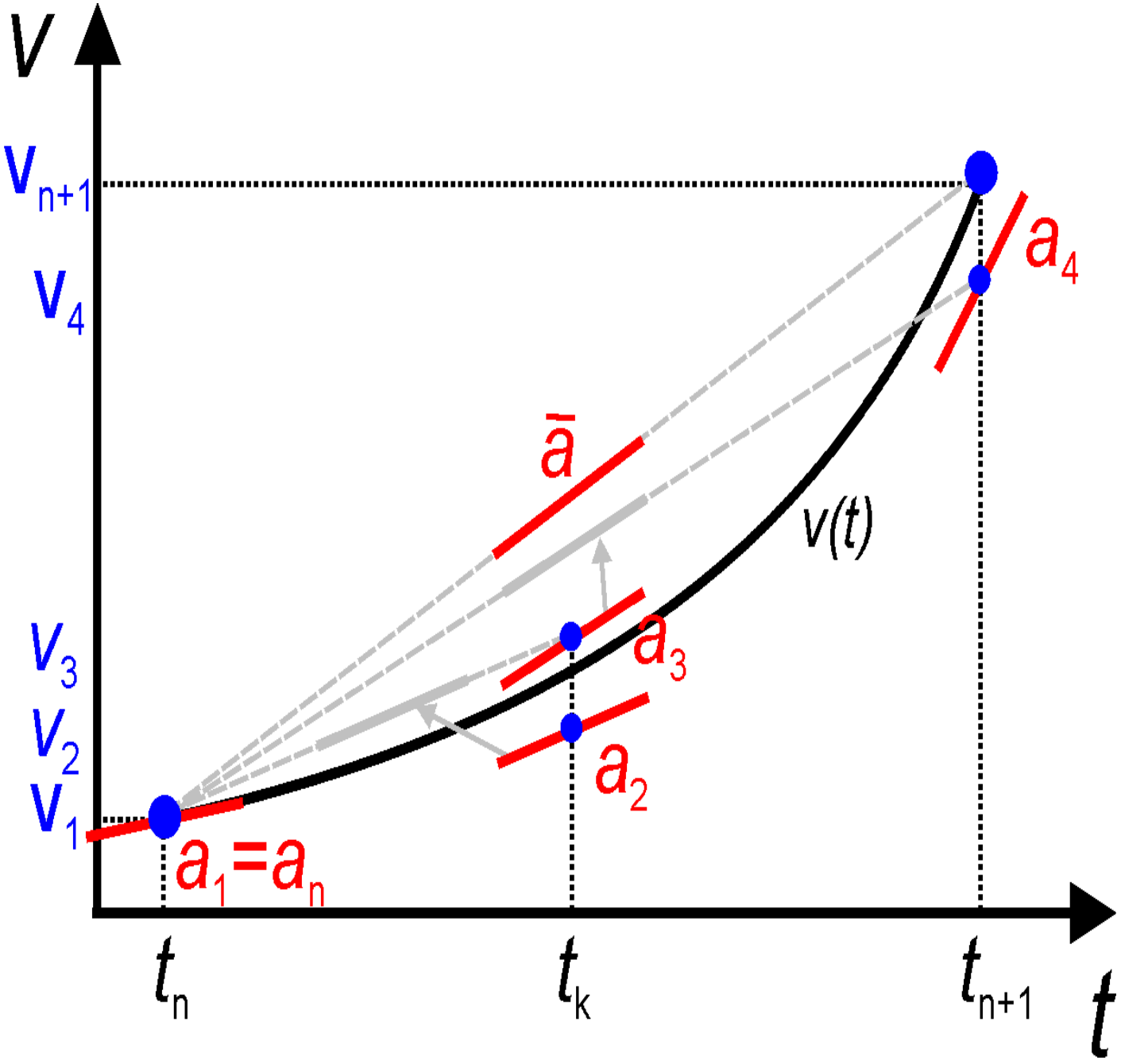
- We compute f twice; in general, the higher number of evaluations of f in advanced methods is more than offset by the far smaller time step needed

Runge-Kutta: RK4

- The most widely used numerical method in game physics is RK4

- $k_1 = hf(x_0), k_2 = hf\left(x_0 + \frac{k_1}{2}\right), k_3 = hf\left(x_0 + \frac{k_2}{2}\right), k_4 = hf\left(x_0 + \frac{k_3}{2}\right)$

$$x(t+h) = x_0 + \frac{1}{6}k_1 + \frac{1}{3}k_2 + \frac{1}{3}k_3 + \frac{1}{6}k_4$$

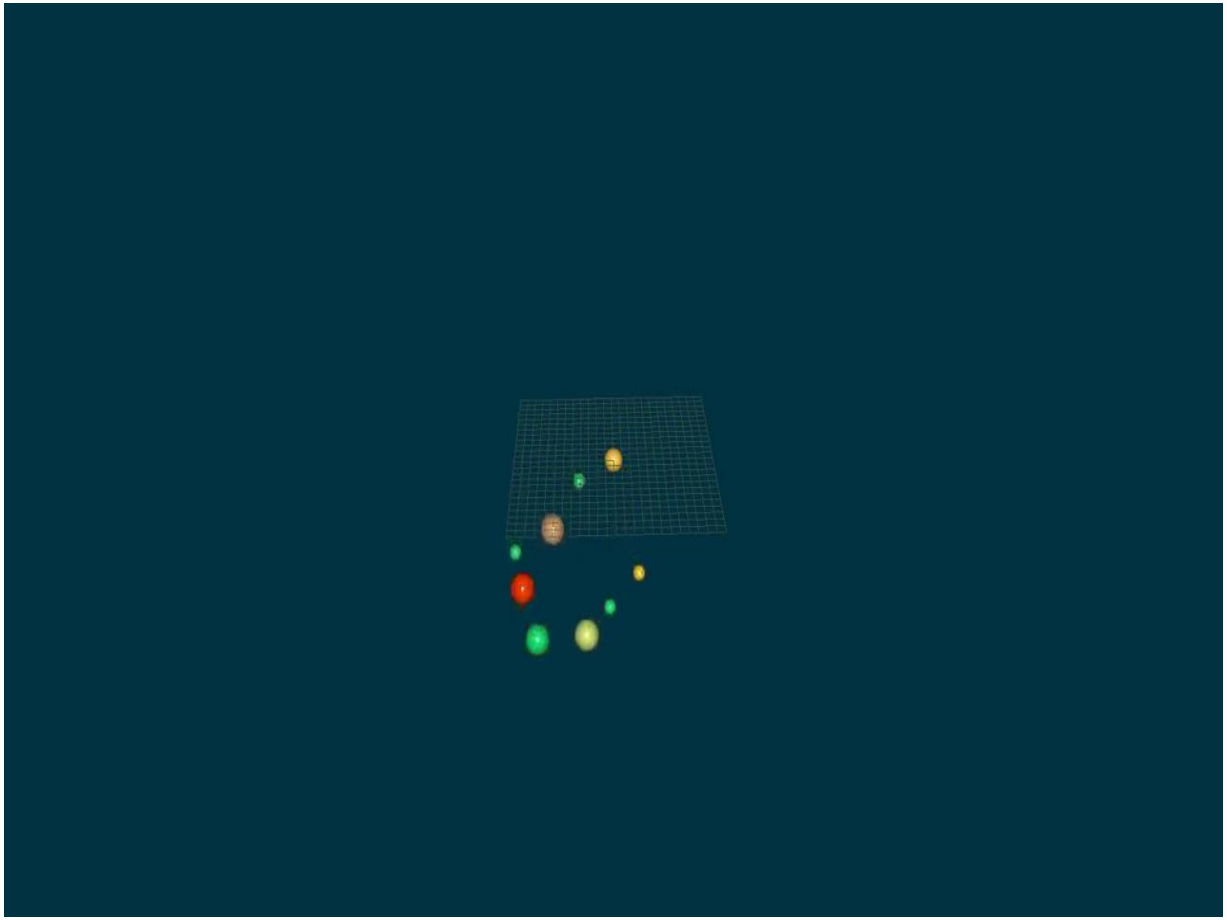

v
v_{n+1}
v_4
v_3
v_2
v_1
$\bar{a}$
v(t)
a_4
a_3
a_2
$a_1=a_n$
t_n
t_k
t_{n+1}
t

How many points?

- Why not use RK5, RK6, or even more?
- Is it not more precise after all?

How many points?

- Why not use RK5, RK6, or even more?
- Is it not more precise after all?
- By approximating higher order derivatives with first order derivatives after a while we are not inserting any new information



Integration and rotation

- We integrate L from τ
- We integrate ω from L and I^{-1}
- We integrate R from ω

Problem

- Recomputing I from every particle of the body and at every update step is slow

Caching moment of inertia I

- Compute I in body space I_{body} and then transform to world space as required
  - I(t) varies in world space, but I_{body} is constant in body space for the entire simulation

Caching moment of inertia I

- Compute I in body space I_{body} and then transform to world space as required
  - I(t) varies in world space, but I_{body} is constant in body space for the entire simulation

Body Space

World Space

Caching moment of inertia I

- Compute I in body space I_{body} and then transform to world space as required
  - I(t) varies in world space, but I_{body} is constant in body space for the entire simulation
- Transform $\omega(t)$ to body space, apply inertia tensor in body space and transform back to world space

$$L(t) = I(t)\omega(t) =$$

Caching moment of inertia I

- Compute I in body space I_{body} and then transform to world space as required
  - I(t) varies in world space, but I_{body} is constant in body space for the entire simulation
- Transform $\omega(t)$ to body space, apply inertia tensor in body space and transform back to world space

$$L(t) = I(t)\omega(t) = R(t)\, I_{body}\, R^T(t)\, \omega(t)$$

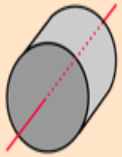
Caching moment of inertia I

- Compute I in body space I_{body} and then transform to world space as required
  - I(t) varies in world space, but I_{body} is constant in body space for the entire simulation
- Transform $\omega(t)$ to body space, apply inertia tensor in body space and transform back to world space

$$L(t) = I(t)\omega(t) = R(t)\, I_{body}\, R^T(t)\, \omega(t)$$
$$I^{-1}(t) = R(t) {I_{body}}^{-1}\, R^T(t)$$

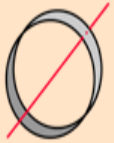
Common Moments of Inertia

Solid cylinder or disc, symmetry axis



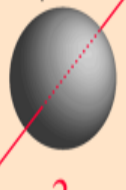
$$I = \frac{1}{2}MR^2$$

Hoop about symmetry axis



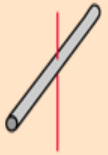
$$I = MR^2$$

Solid sphere



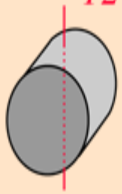
$$I = \frac{2}{5}MR^2$$

Rod about center



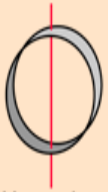
$$I = \frac{1}{12}ML^2$$

$$= \frac{1}{4}MR^2 + \frac{1}{12}ML^2$$



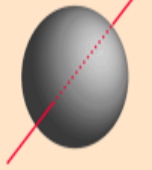
Solid cylinder, central diameter

$$I = \frac{1}{2}MR^2$$



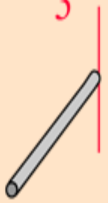
Hoop about diameter

$$I = \frac{2}{3}MR^2$$



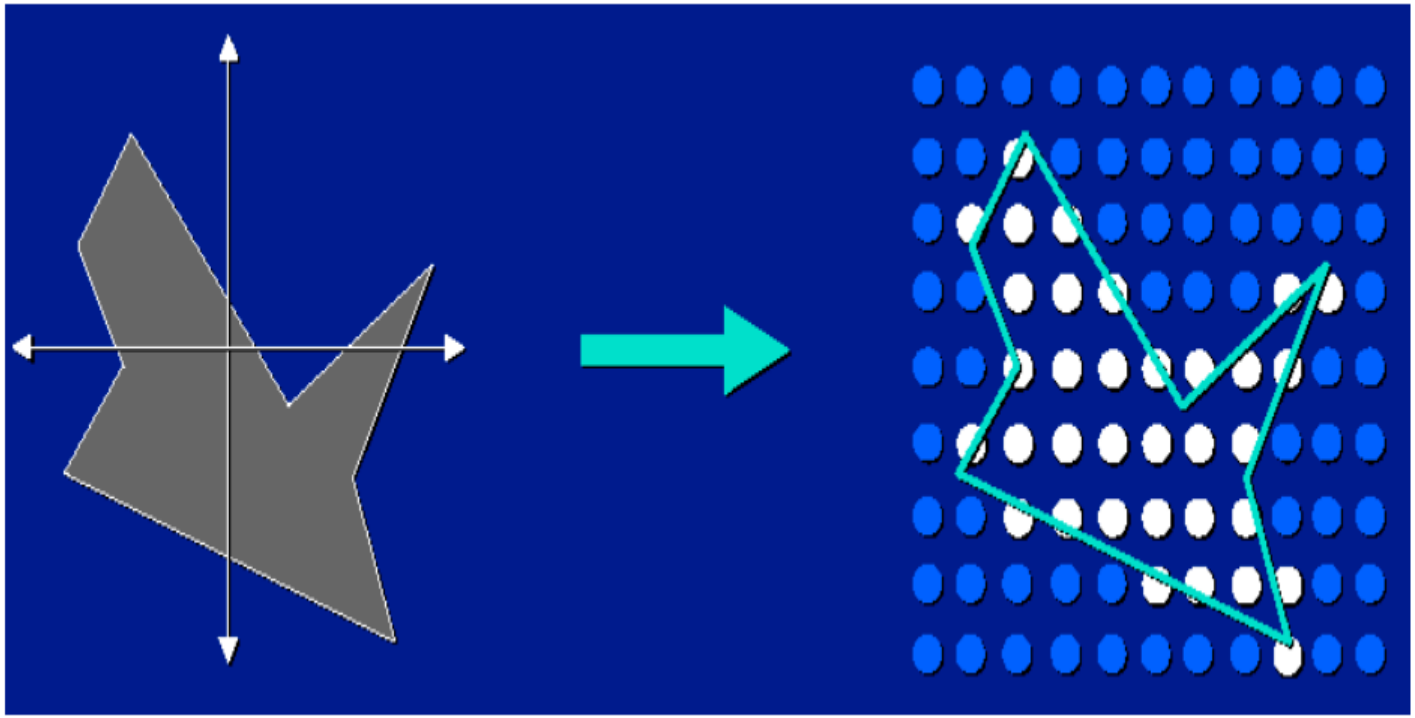
Thin spherical shell

$$I = \frac{1}{3}ML^2$$



Rod about end

Inertia tensor - approximation with points

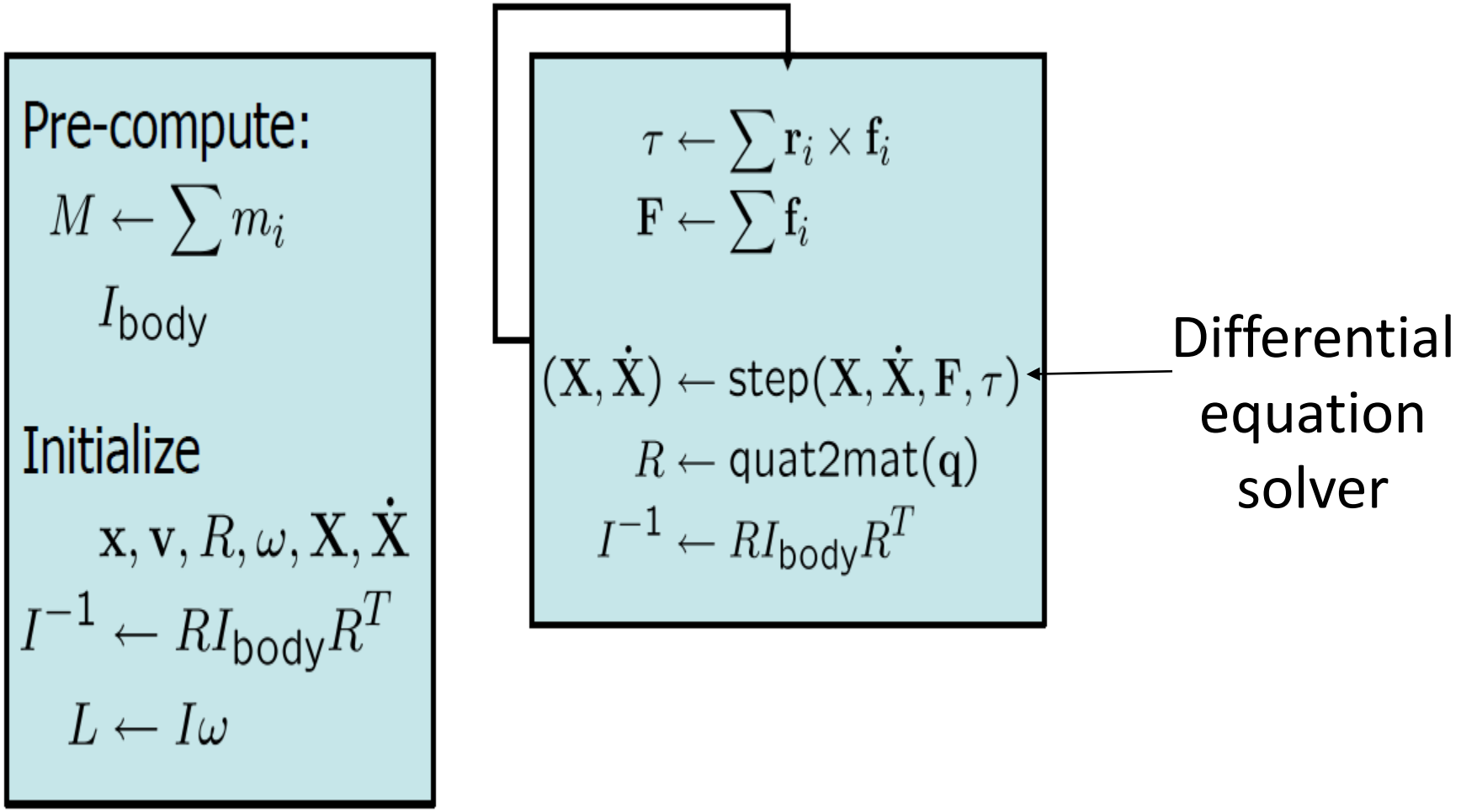


Source code and description of algorithm here (Mirtich 1996)

Simulation overview

$$\dot{\mathbf{X}}(t) = \frac{d}{dt}\begin{pmatrix} \mathbf{x}(t) \\ \mathbf{q}(t) \\ P(t) \\ L(t) \end{pmatrix} \begin{matrix} \} \rightarrow \text{Spatial information} \\ \\ \} \rightarrow \text{Velocity information} \end{matrix}$$

Simulation overview



Exam

Pre-compute:

$$M \leftarrow \sum m_i$$

$$I_{\text{body}}$$

Initialize

$$\mathbf{x}, \mathbf{v}, R, \omega, \mathbf{X}, \dot{\mathbf{X}}$$

$$I^{-1} \leftarrow R I_{\text{body}} R^T$$

$$L \leftarrow I\omega$$

$$\tau \leftarrow \sum \mathbf{r}_i \times \mathbf{f}_i$$

$$\mathbf{F} \leftarrow \sum \mathbf{f}_i$$

$$P \leftarrow P + \Delta t \mathbf{F}$$

$$L \leftarrow L + \Delta t \tau$$

$$\omega \leftarrow I^{-1} L$$

$$\mathbf{x} \leftarrow \mathbf{x} + \Delta t \frac{\mathbf{P}}{\mathbf{M}}$$

$$\mathbf{q} \leftarrow \mathbf{q} + \Delta t \frac{1}{2} \omega \mathbf{q}$$

$$R \leftarrow \text{quat2mat}(\mathbf{q})$$

$$I^{-1} \leftarrow R I_{\text{body}} R^T$$

Projects Half-Term (20% of total mark)

- Tuesday, 18.12.2018, 12.00 D-1
- Create a pretty, interactive water scene
- Use the following techniques:
  - Draw models using vertex array objects
  - Use environment, normal and shadow mapping
  - Microfacet BRDF, model different materials
  - Particle system
- Present your application AND one article from GPU Gems of your choice, 3 + 12 minutes (powerpoint)

Exerc

- Add an orientation attribute to the particle data class
- Initialize fish with orientations
- Calculate rotation matrix R with parallel transport frame method (R = glm:rotate(angle_deg, norm_axis))
- Convert to quaternion and add the derivative of the quaternion resulting from fish movement to obtain the new orientation: q = q + ½*quat_cast(R)
- Fish should be oriented now in the direction they are moving