Computer Animation and Simulation

Physically Based Animation

Overview

- Motivation
- Rigid Body Simulation
- Bodies in Collision
- Particle Systems
- Flexible Objects
- Cloth Simulation

Motivation

- How to create realistic/believable animation
  - key frame animation
    - precise control over timing, poses, positions, orientations… frame by frame
    - labour intensive and cumbersome to create realistic motions
  - physically based animation
    - makes use of forces
    - less control over position and orientation and… of each object in each frame
    - physically realistic motions

Rigid Body Simulation

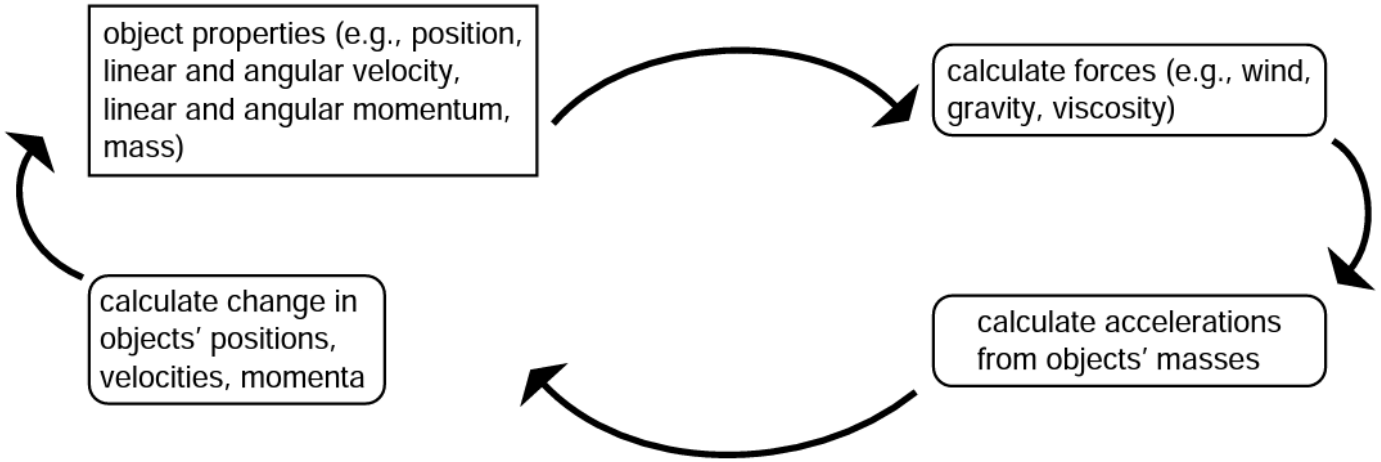
Rigid Body

- solid body in which deformation is absent or neglectable

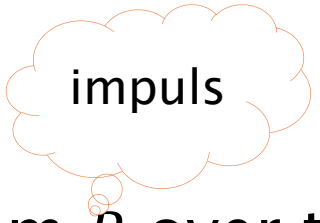
- realistic motion as a result of physically based reaction of rigid bodies to commonly encountered forces
  - gravity
  - viscosity
  - friction
  - collisions
  - …

Rigid Body Simulation

- forces applied to objects induce accelerations
  - linear (which is related to object's mass)
  - angular (which is related to mass distribution)
- accelerations cause changes in velocities
- velocities change object's position and orientation
- forces to be simulated are modelled in the system
  - in computer animation the concern is with modelling the motion of objects at discrete time steps



Motion of a Point in Space



- linear force F equals change in linear momentum P over time
  - linear momentum $P = mv$
  - force $F = \frac{\Delta P}{\Delta t} = ma$
- linear force applied to point induces linear acceleration

  $a = F / m$
- linear acceleration causes change in linear velocity

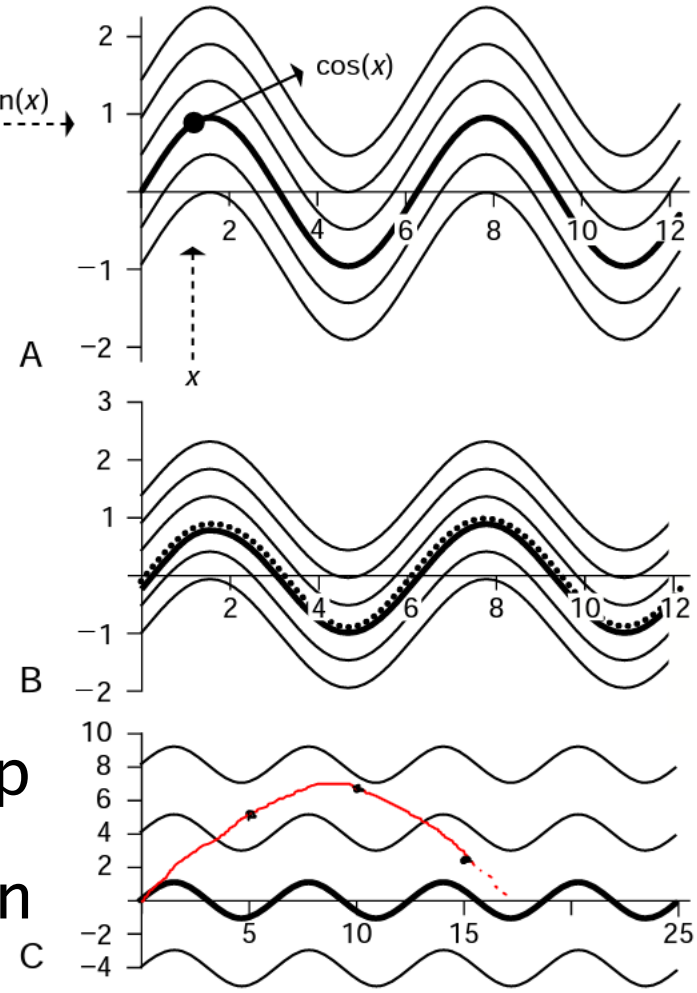
  $v' = v + a\Delta t$
- linear velocity changes points's position

  $$x' = x + v\Delta t + \frac{1}{2}a\Delta t^2$$
- functions of time to update the position of a point over time
  - position $x(t)$
  - linear velocity $v(t)$
  - linear acceleration $a(t)$

$$x(t+\Delta t) = x(t) + v(t)\Delta t + \frac{1}{2}a(t)\Delta t^2$$

Numeric Approximation

- $x(t+\Delta t) = x(t) + v(t)\Delta t + \frac{1}{2}a(t)\Delta t^2$
- size of the time step affects accuracy
  - too large: numerically approximated path deviates from the ideal continuous path
  - too small: computationally expensive
- acceleration often varies over time step
- thus, use better methods of integration
  - ODE solver



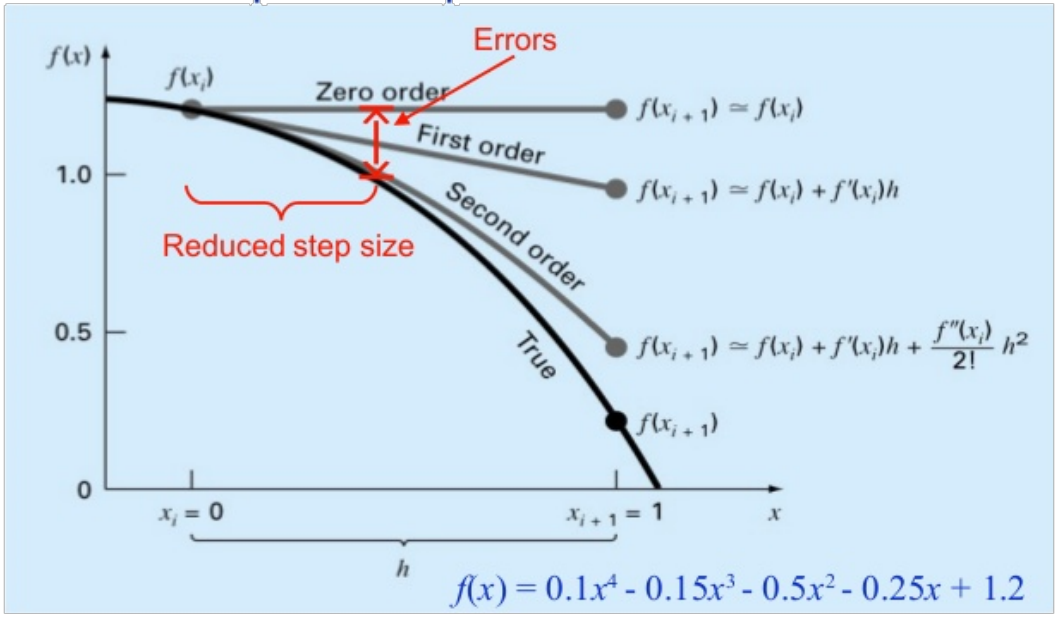
Approximating the sine curve by stepping in the direction of its derivative. (a) In this example, the sine function is the underlying (unknown) function. The objective is to reconstruct it based on an initial point and knowledge about the derivative function (cosine). Start out at any (x, y) location and, if the reconstruction is completely accurate, the sinusoidal curve that passes through that point should be followed. In (b) and (c), the initial point is (0, 0) so the thicker sine curve should be followed. (b) Updating the function values by taking small enough steps along the direction indicated by the derivative generates a good approximation to the function. In this example, $\Delta x = 0.2$. (c) However, if the step size becomes too large, then the function reconstructed from the sample points can deviate widely from the underlying function. In this example, $\Delta x = 5$.

Numerical Integration

- assuming $x(t)$ is smooth, we can express its value at the end of the step as an infinite sum involving the value and derivatives at the beginning
  - *Taylor series*
  - $$x(t_0+h) = x(t_0) + h\dot{x}(t_0) + \frac{h^2}{2!}\ddot{x}(t_0) + \frac{h^3}{3!}\dddot{x}(t_0) + \ldots + \frac{h^n}{n!}\frac{\partial^n x}{\partial t^n} + \ldots$$

Taylor Series Approximation Example:

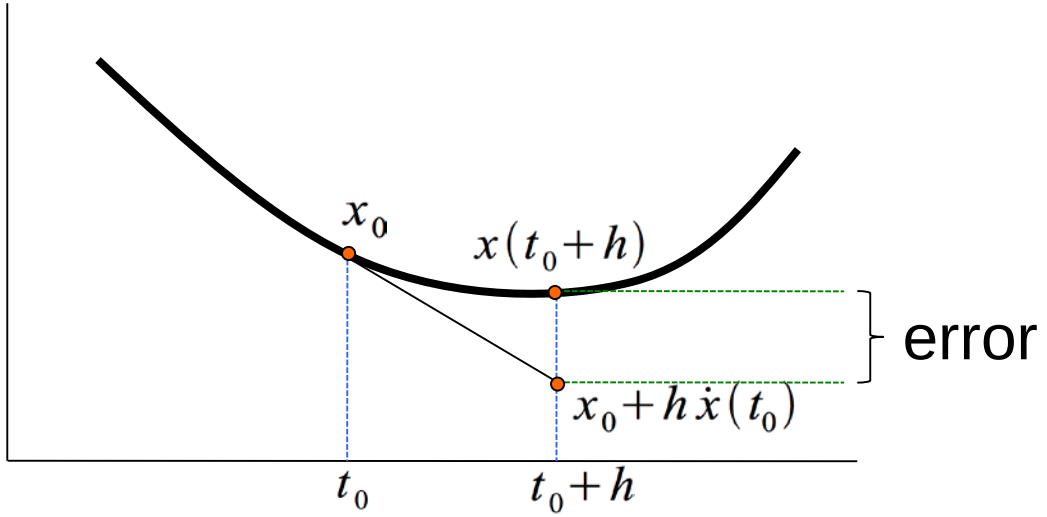
Smaller step size implies smaller error



Numerical Integration

Euler Method

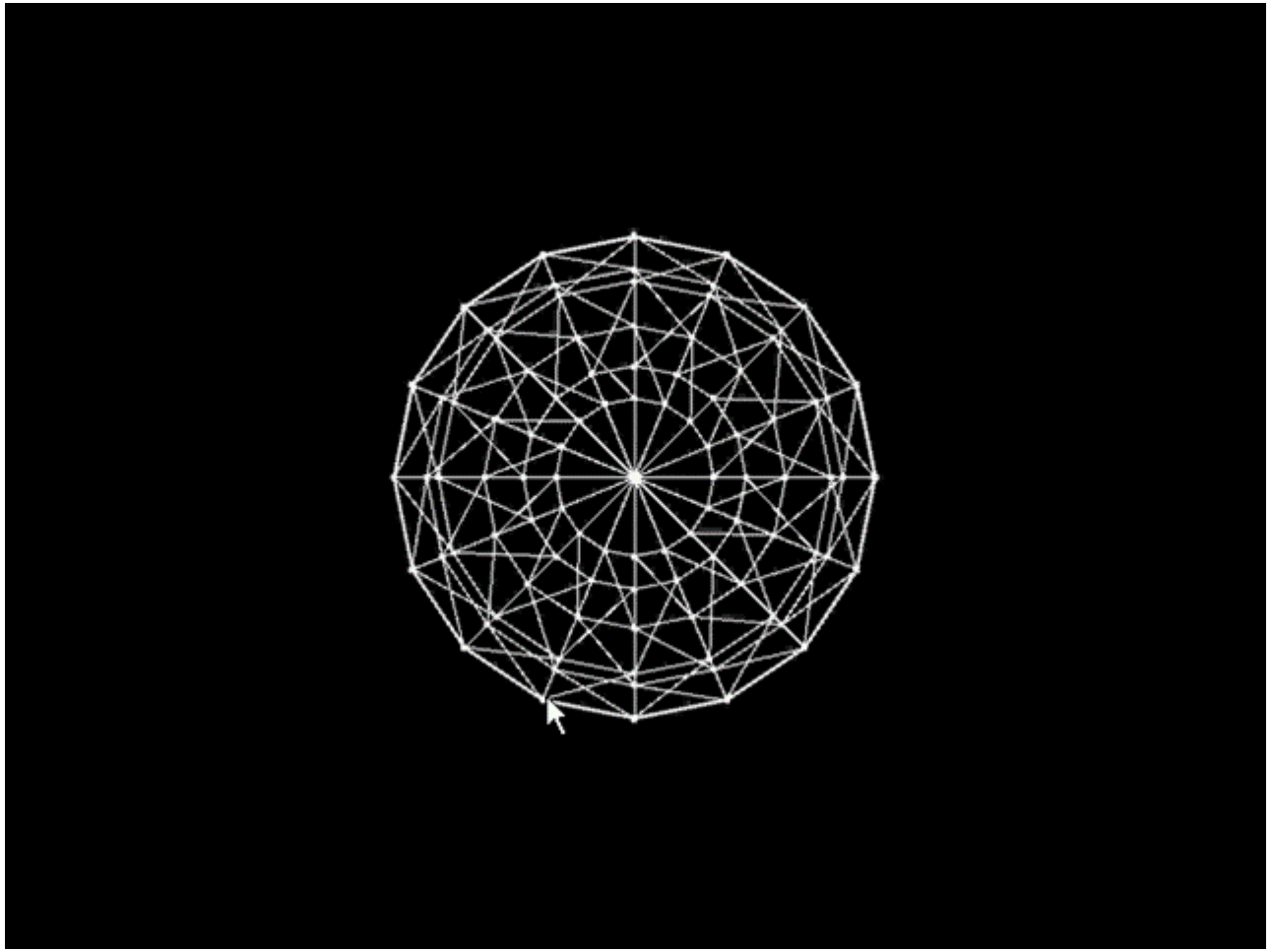
- $x(t_0+h) = x(t_0) + h\dot{x}(t_0) + \frac{h^2}{2!}\ddot{x}(t_0) + \frac{h^3}{3!}\dddot{x}(t_0) + \ldots + \frac{h^n}{n!}\frac{\partial^n x}{\partial t^n} + \ldots$
  - simple
    - derivative at start of each step is extrapolated to find next value
  - inaccurate (only correct when $x(t)$ is linear)
  - unstable (can diverge)



  - example: $x' = x + v\Delta t$

Numerical Integration

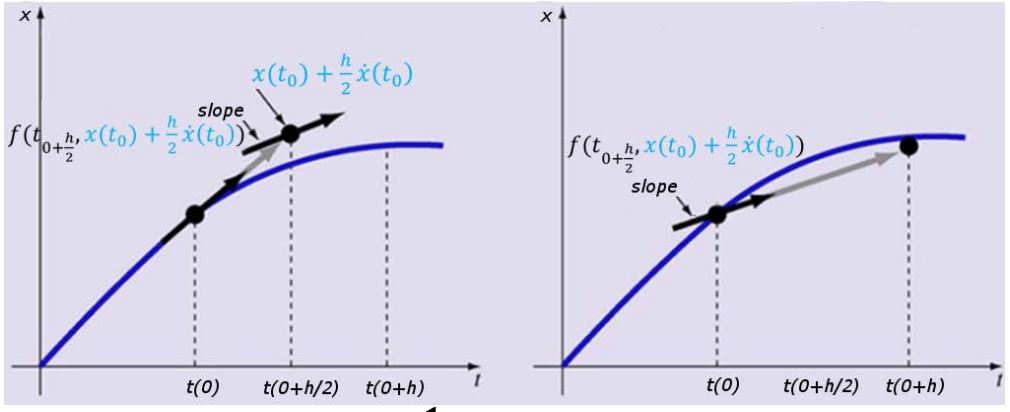
Euler Method: divergence example



Numerical Integration

Midpoint Method

- $$x(t_0+h) = x(t_0) + h\dot{x}(t_0) + \frac{h^2}{2!}\ddot{x}(t_0) + \frac{h^3}{3!}\dddot{x}(t_0) + \ldots + \frac{h^n}{n!}\frac{\partial^n x}{\partial t^n} + \ldots$$
  - rewrite as $x(t_0+h) = x(t_0) + hf\left(t_{0+\frac{h}{2}}, x(t_0) + \frac{h}{2}\dot{x}(t_0)\right)$
  $$= x(t_0) + h\dot{x}\left(t_{0+\frac{h}{2}}\right)$$
    - first, use Euler method to predict value at midpoint of interval
    - then, take a step using the slope at the midpoint



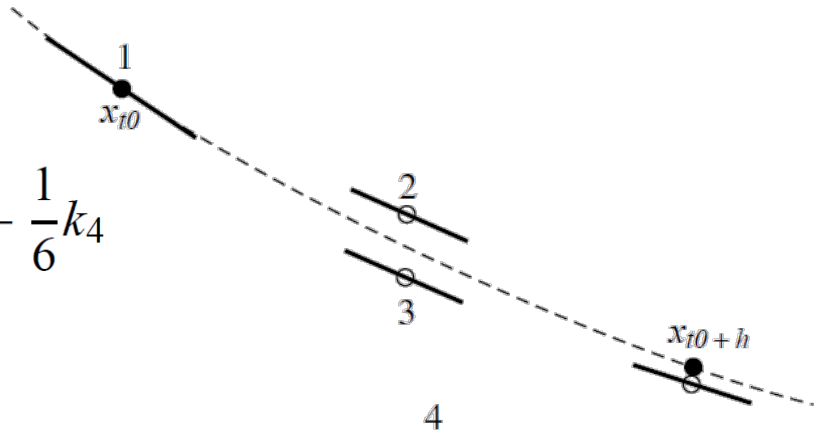
  - example: $x' = x + v\Delta t + \frac{1}{2}a\Delta t^2$

Numerical Integration

Runge-Kutta Methods

- family of higher order methods
  - second-order Runge-Kutta = midpoint method
  - fourth-order Runge-Kutta

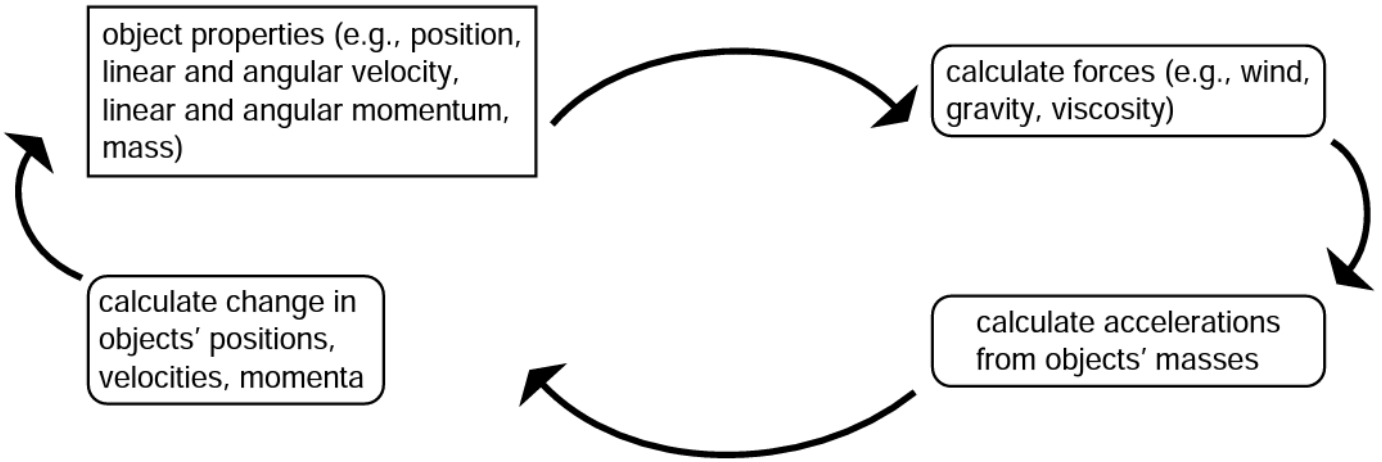
$$
\begin{aligned}
k_1 &= hf(\mathbf{x}_0, t_0) \\
k_2 &= hf(\mathbf{x}_0 + \frac{k_1}{2}, t_0 + \frac{h}{2}) \\
k_3 &= hf(\mathbf{x}_0 + \frac{k_2}{2}, t_0 + \frac{h}{2}) \\
k_4 &= hf(\mathbf{x}_0 + k_3, t_0 + h) \\
\mathbf{x}(t_0 + h) &= \mathbf{x}_0 + \frac{1}{6}k_1 + \frac{1}{3}k_2 + \frac{1}{3}k_3 + \frac{1}{6}k_4
\end{aligned}
$$



Fourth-order Runge-Kutta method. In each step the derivative is evaluated four times: once at the initial point, twice at trial midpoints, and once at a trial endpoint. From these derivatives the final function value (shown as a filled dot) is calculated.

Motion of a Rigid Body in Space

- forces applied to objects induce accelerations
  - !!! linear
    (due to change in linear momentum, related to **object's mass**)
  - !!! angular
    (due to change in angular momentum, related to **mass distribution**)
- accelerations cause changes in velocities
- velocities change object's position and orientation
- forces to be simulated are modelled in the system



Mass of a Body

- Total mass of a body
  - assume body is made up of a large number of small particles q_i
  - m_i are individual masses at points q_i
  - total mass $M = \sum m_i$

massamiddelpunt

- Center of mass
  - $q_i(t)$ is location of mass point q_i in world space
  - center of mass $x(t) = \frac{\sum m_i q_i(t)}{M}$

Linear Movement

impuls

- total linear momentum of rigid body is the same as the linear momentum of a point with mass M and velocity $v(t)$
  - linear momentum $P = Mv$
  - force $F = \frac{\Delta p}{\Delta t} = Ma$
- so, we use following functions of time to update the position of a point over time

$$a = F / M$$

$$v' = v + a\Delta t$$

$$x' = x + v\Delta t + \frac{1}{2} a\Delta t^2$$

- numerical integration

$$x(t_0 + h) = x(t_0) + h\dot{x}(t_0) + \frac{h^2}{2!}\ddot{x}(t_0) + \frac{h^3}{3!}\dddot{x}(t_0) + \ldots + \frac{h^n}{n!}\frac{\partial^n x}{\partial t^n} + \ldots$$

Rotational Movement

impuls moment

- torque (i.e. rotational force) τ equals change in angular momentum L over time
  - when force is applied not directly in line with center of mass!
    - google "Angular Momentum & the Bottle Flip Challenge"
  - angular momentum $L = I\omega$ (with I being the *inertia tensor*)
  - torque $\tau = \frac{\Delta L}{\Delta t}$

traagheidsmoment

- torque applied to object induces angular acceleration
- angular acceleration causes change in angular velocity
- angular velocity changes object's orientation
  - orientation represented by rotation matrix $R(t)$
  - angular velocity $\omega(t)$
    - direction ~ orientation of axis
    - magnitude ~ speed
  - angular acceleration $\alpha(t)$

$$q(t) = R(t)q + x(t)$$

$$\dot{q}(t) = \omega(t)^{*}R(t)q + v(t)$$

$$\dot{q}(t) = \omega(t) \times (q(t) - x(t)) + v(t)$$

Rotational Movement

traagheidsmoment

- inertia tensor $I(t)$
  - 3x3 matrix needed to describe the resistance to change due to distribution of mass of the object
  - dependent on time, not on mass
  - I_{object} is inertia tensor for untransformed object
    - let $q_i = (x_i, y_i, z_i)$ be the displacement/density of the ith particle
    - $$I_{object} = \begin{bmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{xy} & I_{yy} & I_{yz} \\ I_{xz} & I_{yz} & I_{zz} \end{bmatrix}$$

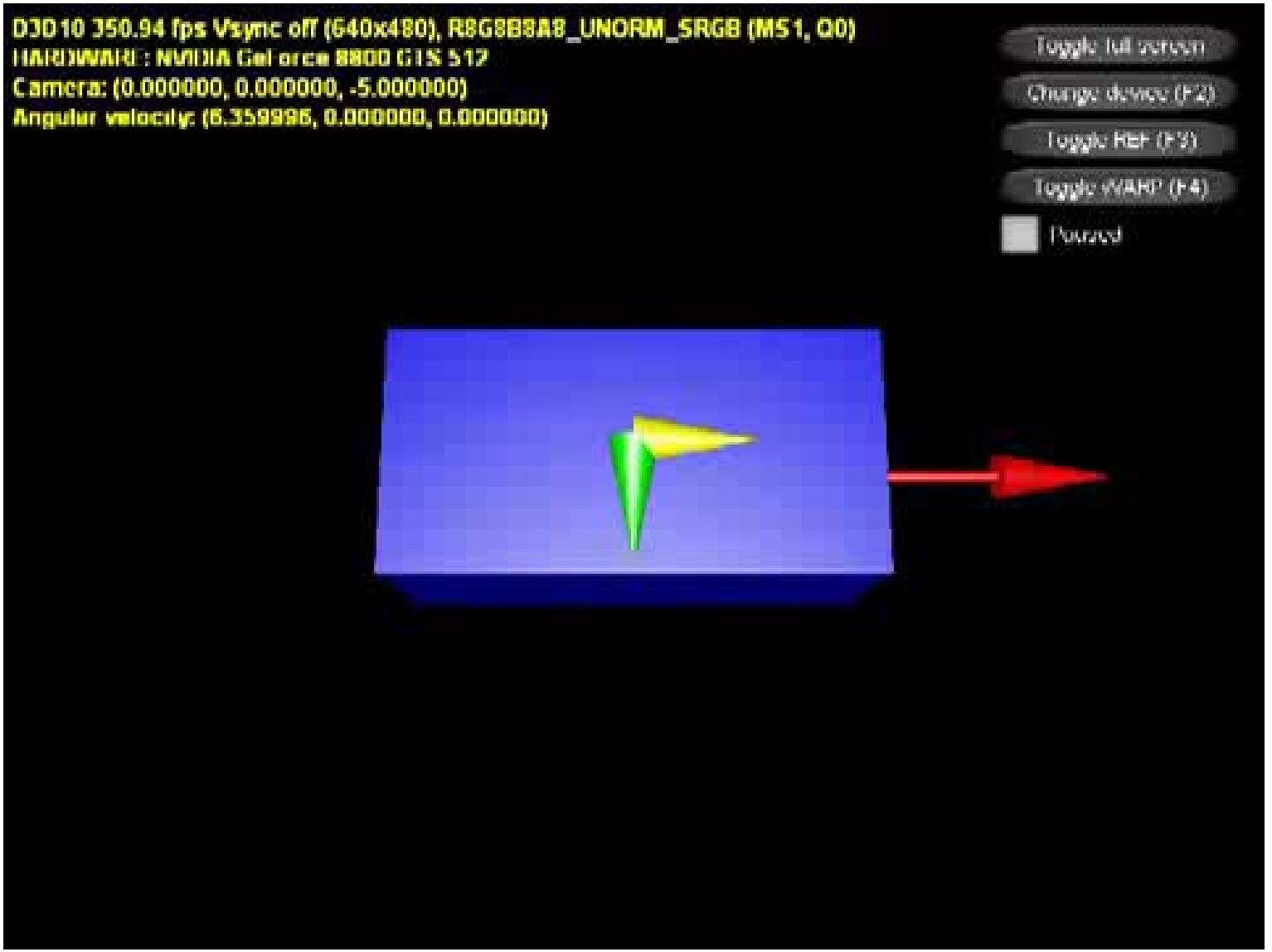
$$I_{xx} = \sum m_i(y_i^2 + z_i^2) \quad I_{xy} = \sum m_i x_i y_i$$
$$I_{yy} = \sum m_i(x_i^2 + z_i^2) \quad I_{xz} = \sum m_i x_i z_i$$
$$I_{zz} = \sum m_i(x_i^2 + y_i^2) \quad I_{yz} = \sum m_i x_i z_i$$

  - $I(t) = R(t) I_{object} R(t)^T$

Rigid Body Simulation

- initialize
  - position x, rotation mat R, linear momentum P, angular momentum L
- calculate attributes which do not change over time
  - mass M and object space inertia tensor I_{object}
- keep state of object in state vector $S(t) = [x(t), R(t), P(t), L(t)]$
- for each time step h
  - compute auxiliary variables
    - velocity $v(t) = \frac{P(t)}{M}$
    - inertia tensor $I(t) = R(t) I_{object} R(t)^T$
    - angular velocity $\omega(t) = I(t)^{-1} L(t)$
  - compute force F and torque τ due to gravity, wind, collisions…)
  - $\frac{d}{dt} S$ is known: $\frac{d}{dt} x(t) = v(t), \frac{d}{dt} R(t) = \omega(t) \times R(t), \frac{d}{dt} P(t) = F(t), \frac{d}{dt} L(t) = \tau(t)$
  - use ODE solver to update S with $\frac{d}{dt} S(t + h)$

Rigid Body Simulation



Bodies in Collision

Colliding Bodies

- all types of contact between bodies require calculation of forces to simulate the reaction

- two issues must be addressed
  1. collision detection
     - either at specific instance in time, or
     - during finite time interval

  2. collision response
     - localized forces at specific points on the object impart linear and rotational forces onto the other objects involved

Time of Collision

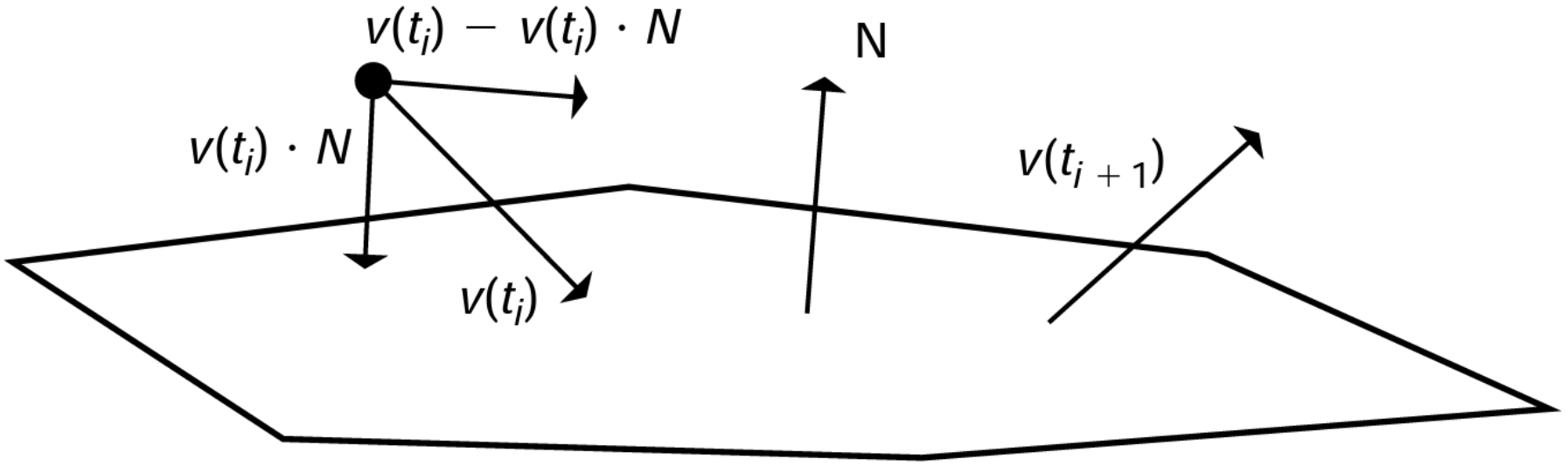
- proceed from this point in time by calculating appropriate reaction to the current situation by the particle involved in the collision
  - allows penetration of particle before collision reaction which might be visually notable

- back up time t_i to the first instant that a collision occurred and determine appropriate response

Collision Response

Particle-plane collision and kinematic response

- quick and easy
- good visual results for particles and spherically shaped objects

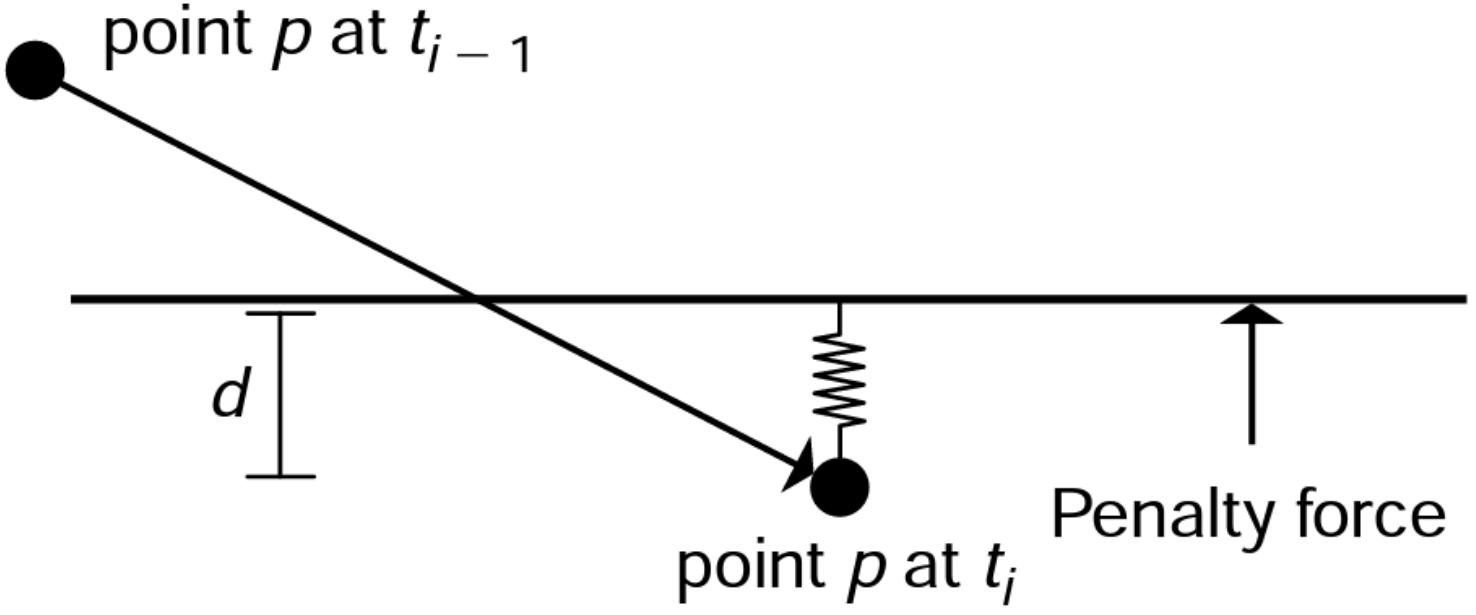


$$\begin{aligned} \mathrm{v}(t_{t+1}) &= \mathrm{v}(t_i) - (\mathrm{v}(t_i) \cdot \mathrm{N})\mathrm{N} - k(\mathrm{v}(t_i) \cdot \mathrm{N})\mathrm{N} \\ &= \mathrm{v}(t_i) - (1+k)(\mathrm{v}(t_i) \cdot \mathrm{N})\mathrm{N} \end{aligned}$$

Collision Response

Penalty method

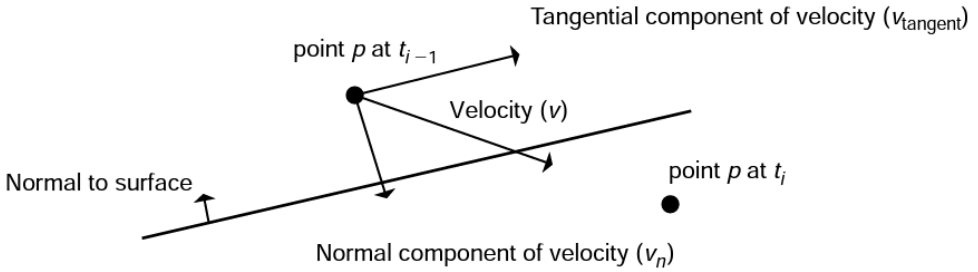
- when objects penetrate due to temporal sampling, we introduce temporary, non-physically based force in order to restore non-penetration



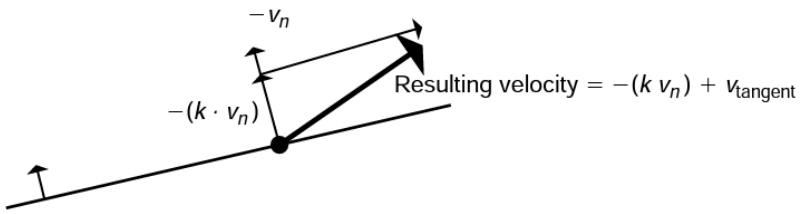
Collision Response

Impulse force of collision

- more precise way of introducing an impulse into the system: large force acting over short time interval
- used when time is backed up



Components of a particle's velocity colliding with a plane



Computing velocity resulting from a collision (k is the coefficient of restitution)

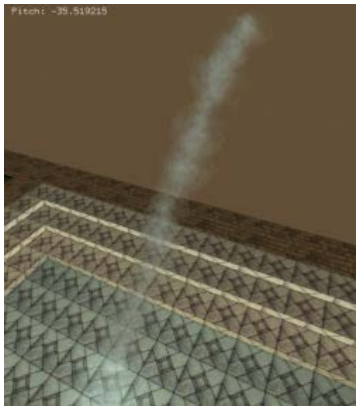
Impact response of a point with a plane.

Collision Detection and Response

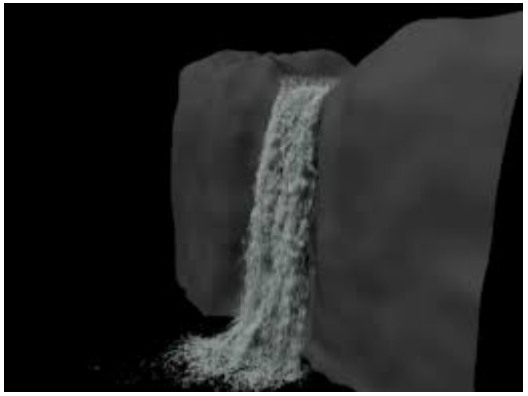
Particle Systems

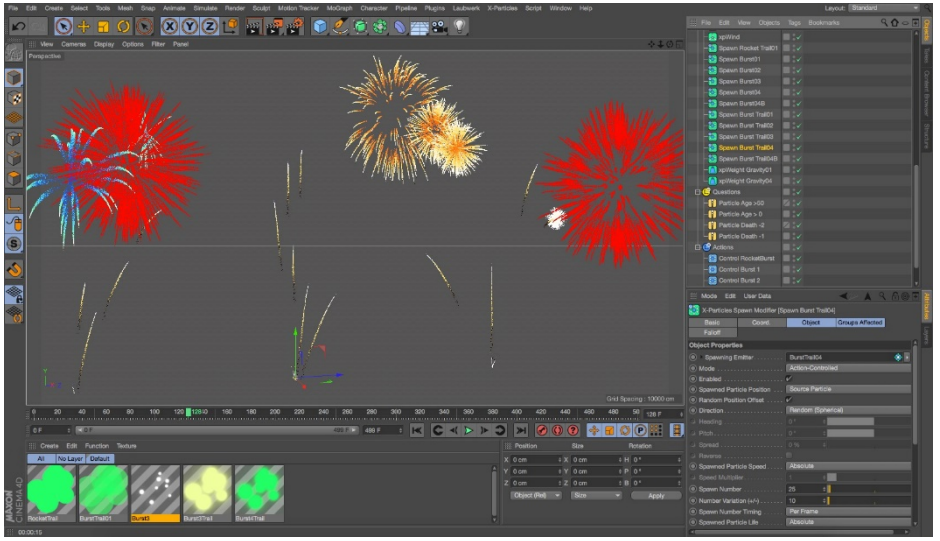
Particle System

- particle system
  - collection of a large number of point-like elements
  - animated as simple physical simulation



- assumptions
  - particles do not collide with each other
  - particles do not cast shadows on each other
  - particles do not reflect light

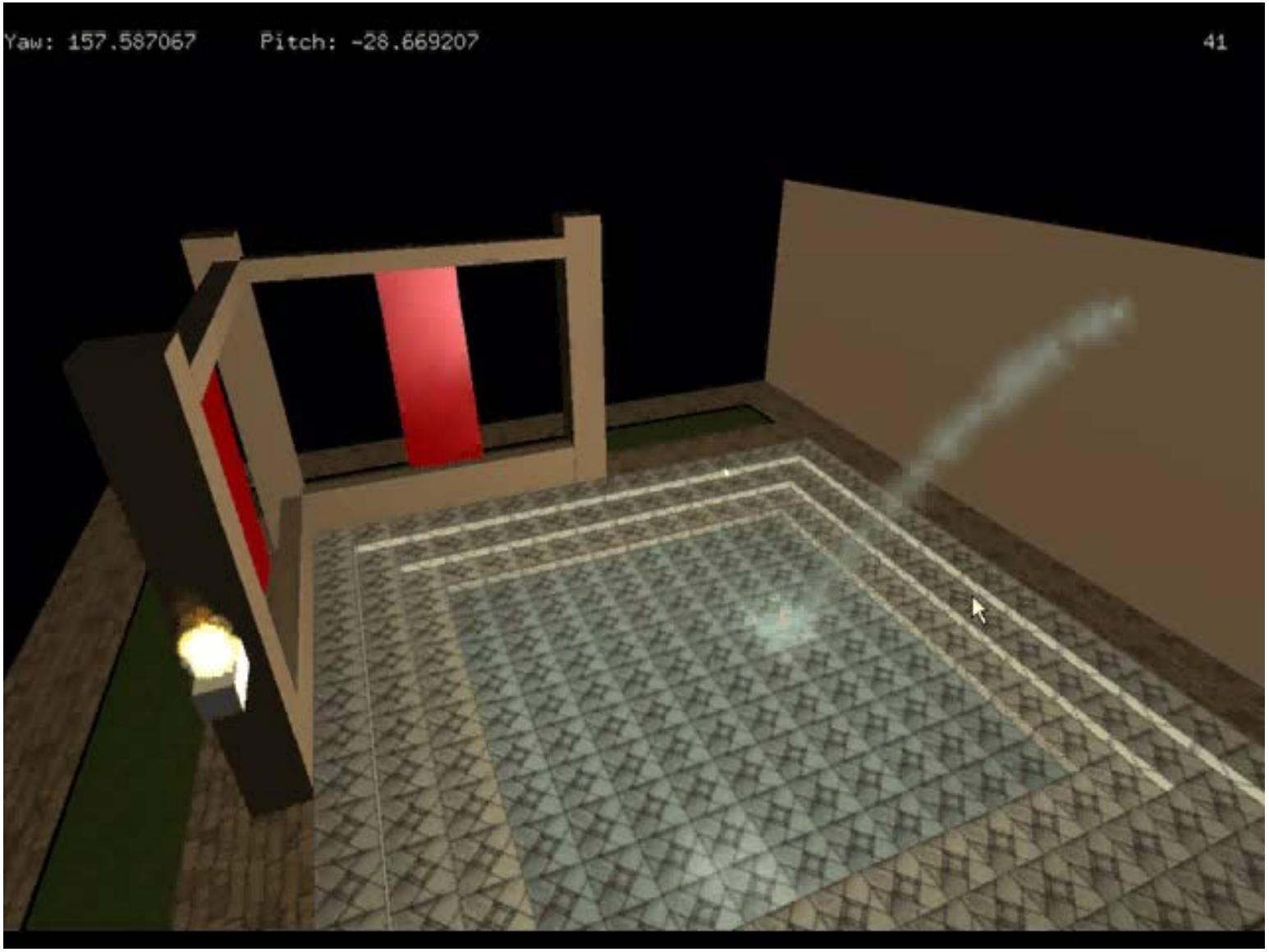




Particle

- particle generation
  - particles are generated according to a stochastic process
- particle attributes
  - position, velocity, shape parameters, colour, transparency
  - lifetime which gets decremented at each new frame
- particle animation, for each frame
  - new particles are born and are assigned attributes
  - particles that have exceeded their lifetime are terminated
  - particles get animated (position, velocity… are updated)
  - particles are rendered

Particle Animation



Particle Animation

Flexible Objects

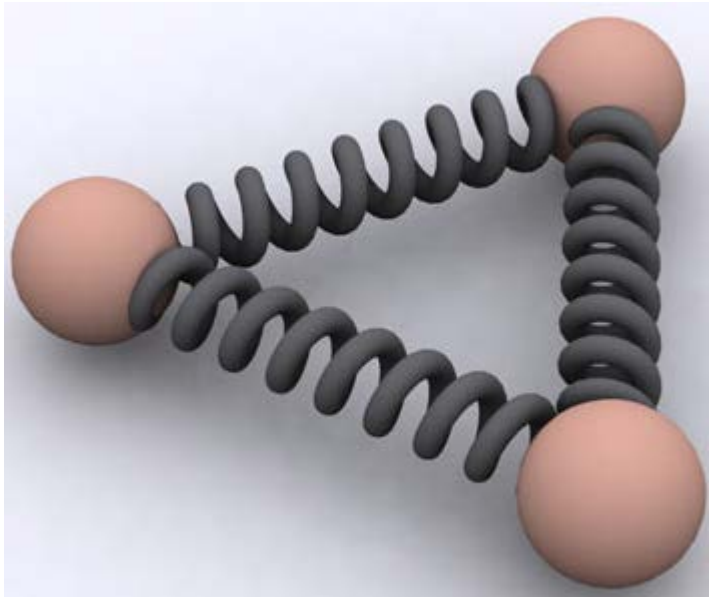
Flexible Objects

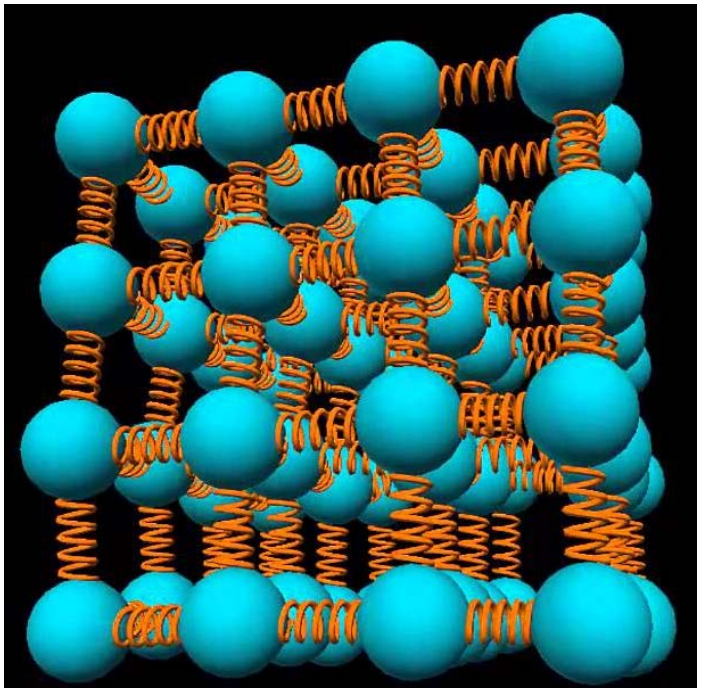
- kinematic approach
  - animator is responsible for source and target shapes
    - manual
    - FFD
    - …
  - interpolate between source and target shapes
  - Chapter 4: Interpolation-Based Animation

- physically-based approach
  - simulate reaction of body to external forces
  - mass-spring-damper system

Mass-Spring-Damper System

- modelling
  - model each vertex of an object as a **point mass**
    - distribute mass of object (un)evenly among vertices
  - model each edge of the object as a **spring**
    - rest length is set equal to original length of the edge
    - spring is paired with damper (resistance) for better control

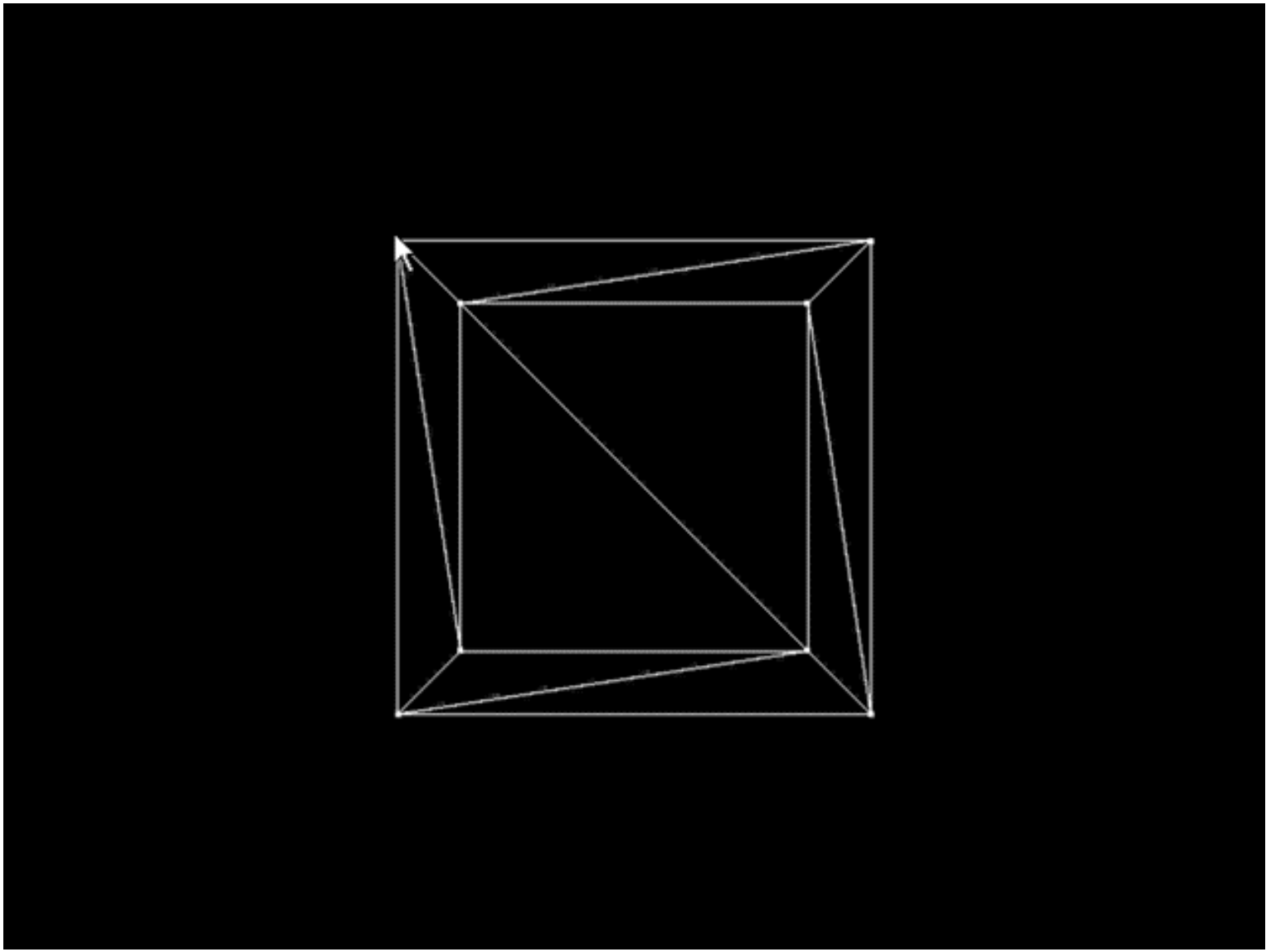




Mass-Spring-Damper System

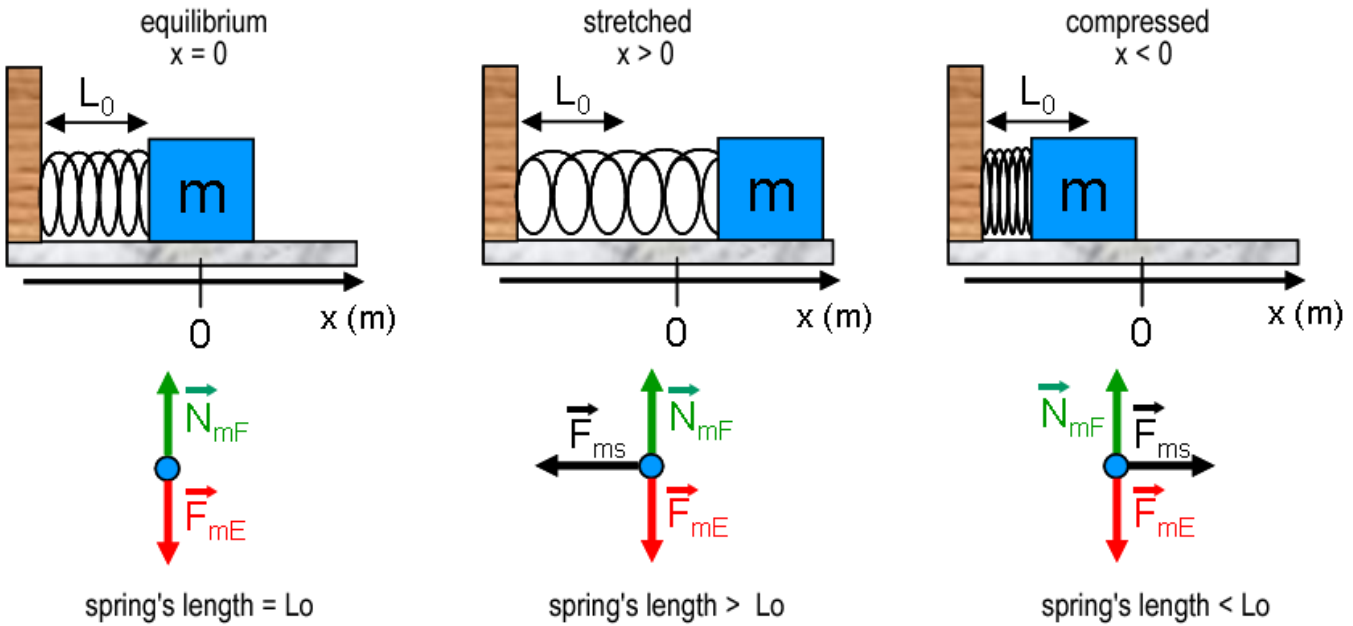
- animation
  - external forces are applied to specific vertices
  - vertices will be displaced relative to other vertices
  - displacements induce spring forces
  - spring forces impart forces to adjacent vertices as well as reactive forces back to the initial vertex
    - vertices will be displaced relative to other vertices
      - and so on…
        - and so on…

Mass-Spring-Damper System



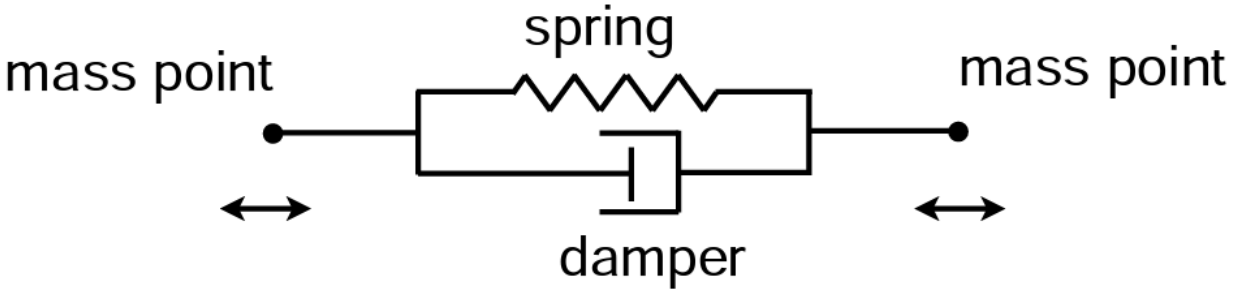
Mass-Spring-Damper Model

- Hooke's Law
  - $f_{ms} = -k_s x$
  - $\boxed{f_{ms} = -k_s(L - L_0)}$
    - k_s is *stiffness* parameter and determines how much a spring reacts to a change in length
    - L_0 is spring's rest length



Mass-Spring-Damper Model

- Spring between two particles p_1 and p_2
  - $f_s = -k_s(L - L_0)\left(\frac{p_2 - p_1}{\|p_2 - p_1\|}\right)$
    - spring force is applied to both ends of the spring in opposite directions
  - $f_d = -k_d(\dot{p}_2 - \dot{p}_1)\left(\frac{p_2 - p_1}{\|p_2 - p_1\|}\right)\left(\frac{p_2 - p_1}{\|p_2 - p_1\|}\right)$
    - *damper* force is negatively proportional to the velocity of the spring v_s
    - k_d is damper constant and determines how much resistance there is to a change in spring length

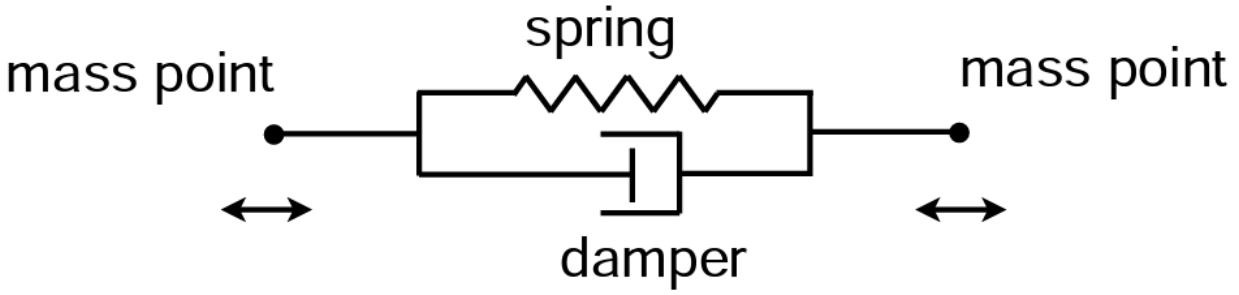


Mass-Spring-Damper Model

- Spring-damper pair
  - resulting force on p_2
  - $f = f_s - f_d$

$$= -k_s(L - L_0)\left(\frac{p_2 - p_1}{\|p_2 - p_1\|}\right) + k_d(\dot{p}_2 - \dot{p}_1)\left(\frac{p_2 - p_1}{\|p_2 - p_1\|}\right)\left(\frac{p_2 - p_1}{\|p_2 - p_1\|}\right)$$

$$= \left(-k_s(L - L_0) + k_d(\dot{p}_2 - \dot{p}_1)\left(\frac{p_2 - p_1}{\|p_2 - p_1\|}\right)\right)\left(\frac{p_2 - p_1}{\|p_2 - p_1\|}\right)$$



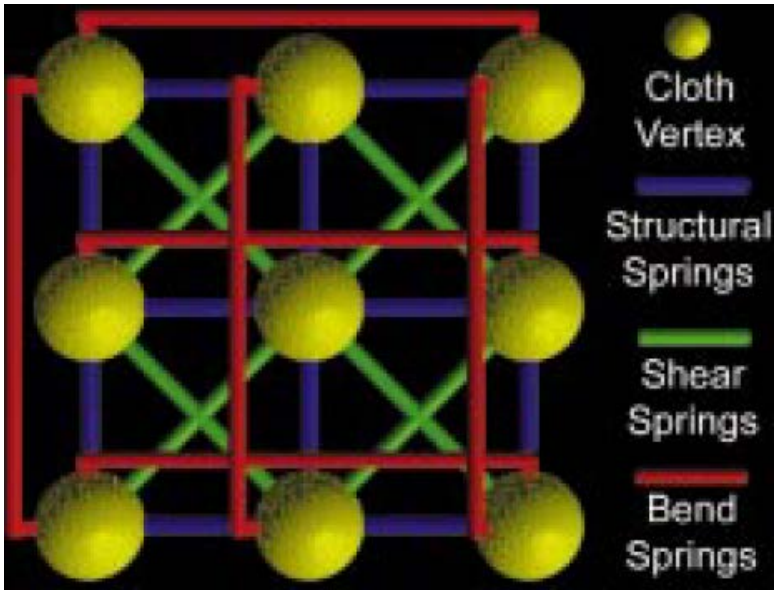
Cloth Simulation

Physically-based Model

- cloth characteristics
  - cloth weave = quadrilateral mesh mimicking the warp and weft of the threads
  - ability to stretch, bend and skew

Physically-based Model

- modeling
  - quadrilateral mesh
  - connect each point to neighboring points with springs
    - structural strings to mimic the weft of the threads
    - shear springs to preserve space between diagonal elements
    - bend springs to prevent the model from folding along the edges of the structural springs
      - real threads run the length of the fabric



Physically-based Cloth Model