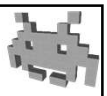
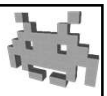
Course Plan



lec. 1: Introduction
lec. 2: Mathematics for 3D Games
lec. 3: Scene Graph
lec. 4: Game 3D Physics
lec. 5: Game Particle Systems
lec. 6: Game 3D Models
lec. 7: Game Textures
lec. 8: Game Materials
lec. 9: Game 3D Animations
lec. 10: 3D Audio for 3D Games
lec. 11: Networking for 3D Games
lec. 12: Interactive Agents for 3D Games
lec. 13: Rendering Techniques for 3D Games

33

Newtonian Dynamics (for particles)



describes the forces
given all the particle positions (and more)

$$\vec{f}(t) = \text{function}(\mathbf{p}(t), \ldots)$$

derivative w.r.t. time

$$\vec{v}(t) = \dot{\mathbf{p}}(t)$$

$$\vec{a}(t) = \ddot{\mathbf{p}}(t) = \frac{\vec{f}(t)}{m}$$

$$\dot{\mathbf{p}}(0) = \vec{v}_0$$

$$\mathbf{p}(0) = p_0$$

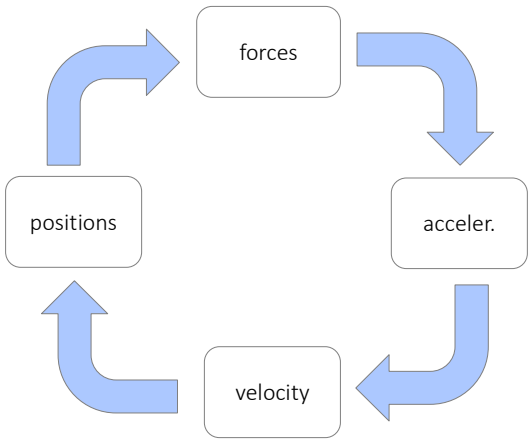
34

Newtonian Dynamics (an equivalent formulation)

$$\begin{cases} \vec{f}(t) = \text{function}(\,\mathbf{p}(t)\,,\ldots) \\ \vec{a}(t) = \dfrac{\vec{f}(t)}{m} \\ \vec{v}(t) = \vec{v}_0 + \displaystyle\int_{t'=0}^{t} \vec{a}(t') \cdot dt' \\ \mathbf{p}(t) = \mathbf{p}_0 + \displaystyle\int_{t'=0}^{t} \vec{v}(t') \cdot dt' \end{cases}$$

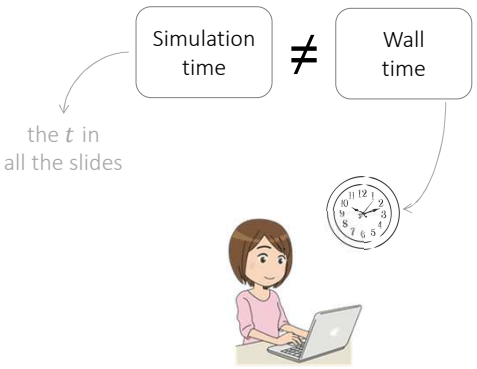
35

Dynamics (Newtonian)



36

An obvious remark, but

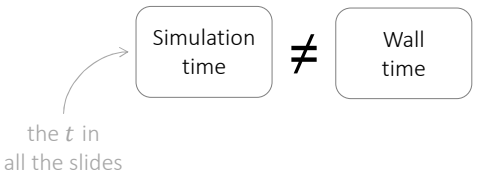


They are just artificially made to flow in sync… usually

- But (e.g.) not when:
  game is paused (t is constant), replays, fast forwards, reverses…

37

An obvious remark, but



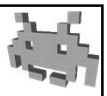
Occasionally, the difference is spectacularly exploited by clever gameplay designs!

PoP – the sands of times
(Ubisoft, 2003)

The longing
(Studio Seufz, 2020)

41

Computing physics evolution



- **Analytical** solutions:

  state = function(t)

  Given force functions (and acc), find the functions (pos, vel,…) in the specified relations:

$$\begin{cases} \vec{f}(t) = \text{function}(\mathbf{p}(t), \dots) \\ \vec{a}(t) = \dfrac{\vec{f}(t)}{m} \\ \vec{v}(t) = \vec{v}_0 + \displaystyle\int_{t'=0}^{t} \vec{a}(t') \cdot dt' \\ \mathbf{p}(t) = \mathbf{p}_0 + \displaystyle\int_{t'=0}^{t} \vec{v}(t') \cdot dt' \end{cases}$$

- **Numerical** solutions:

  1. state$_{(t=0)}$ ← init
  2. state$_{(t+1)}$ ← do_1_step(state$_t$)
  3. goto 2

42

Analytical solutions

that is, a trajectory: a position over time

Find the positions as a function $\mathbf{p}(t)$ of time t such that…

a given function

$$\begin{cases} \ddot{\mathrm{p}}(t) = \mathrm{forces}(\,\mathrm{p}(t), \dots)/\mathrm{m} \\ \dot{\mathrm{p}}(0) = \vec{\mathrm{v}}_0 \\ \mathrm{p}(0) = \mathrm{p}_0 \end{cases}$$

derivative w.r.t. time

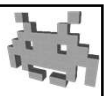
sometimes, it's a function of other things too (e.g. velocity, time…). Harder to solve!

the initial conditions (for speed and position)

A system of ODE
(Ordinary Differential Equations)

44

Analytical solutions

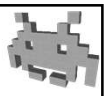


- Difficult to find
  - we need to find a function such that…
- Often, they doesn't even «exist»
  - in a form that we can write using common notations such as polynomials, exponentials, trigonometry, algebraic functions in general
- But when and if they exist, they are very convenient to use
  - we can find the position / the velocity for any given *t*
  - we can predict the status of the simulation for any given time in the past or future
- Examples of setups that admit an analytical solution:
  - the ones with a force function is constant w.r.t. positions & velocities (solution: just find its integral, twice)
  - two bodies (no more than two!), subject to reciprocal gravity force
  - a single pendulum oscillating back and forth, if one accepts an approximation (which is only precise for small oscillations)
- Most other physical setups don't!

45

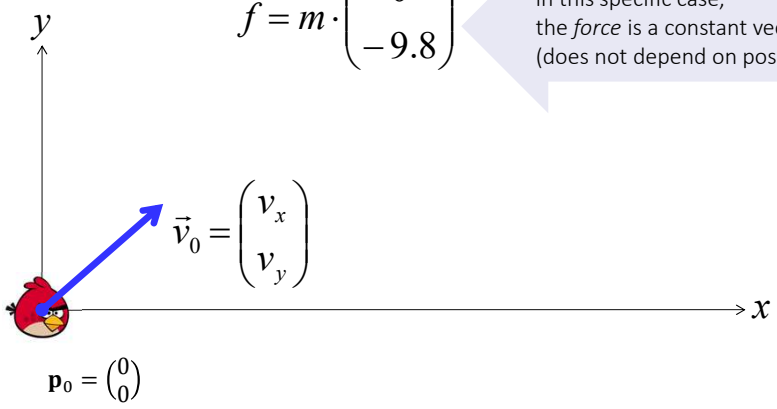
A simple example: analytical solution for…

«ballistic shooting» of a mass, ignoring friction. Here, shown in 2D (3D is similar)



$$\vec{f} = m \cdot \begin{pmatrix} 0 \\ -9.8 \end{pmatrix}$$

in this specific case, the *force* is a constant vector (does not depend on positions)



46

Simple example: analytical solution for…

$$\vec{f}(t) = m \cdot \vec{\mathbf{G}}$$

$$\vec{a}(t) = \frac{\vec{f}(t)}{m} = \vec{\mathbf{G}}$$

$$\vec{v}(t) = \vec{v}_0 + \int_{t'=0}^{t} \vec{\mathbf{G}} \cdot dt'$$

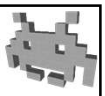
$$= \vec{v}_0 + t\ \vec{\mathbf{G}}$$

$$\mathbf{p}(t) = \mathbf{p}_0 + \int_{t'=0}^{t} \vec{v}(t') \cdot dt'$$

$$= \mathbf{p}_0 + \int_{t'=0}^{t} \left(\vec{v}_0 + t\ \vec{\mathbf{G}}\right) dt'$$

$$= \mathbf{p}_0 + t\ \vec{v}_0 + \frac{1}{2} t^2$$

$$\begin{cases} \vec{f}(t) = \text{function}(\mathbf{p}(t), \dots) \\ \vec{a}(t) = \dfrac{\vec{f}(t)}{m} \\ \vec{v}(t) = \vec{v}_0 + \displaystyle\int_{t'=0}^{t} \vec{a}(t') \cdot dt' \\ \mathbf{p}(t) = \mathbf{p}_0 + \displaystyle\int_{t'=0}^{t} \vec{v}(t') \cdot dt' \end{cases}$$

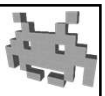


Where $\vec{\mathbf{G}}$ is the gravity constant on the planet.
On Earth, using meters and seconds as unity of measures for space and time, assuming conventions (Y = up)

$$\vec{\mathbf{G}} = \begin{pmatrix} \mathbf{0} \\ \mathbf{-9.8} \\ \mathbf{0} \end{pmatrix}$$

47

Simple example: analytical solution for…



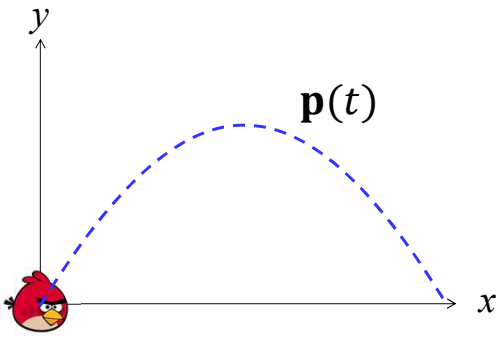
Example in 2D, with $\mathbf{p}_0 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$

$$\vec{f}(t) = m \cdot \begin{pmatrix} 0 \\ -9.8 \end{pmatrix}$$

$$\vec{a}(t) = \begin{pmatrix} 0 \\ -9.8 \end{pmatrix}$$

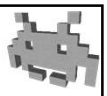
$$\vec{v}(t) = \begin{pmatrix} v_x \\ v_y - 9.8 \cdot t_C \end{pmatrix}$$

$$\mathbf{p}(t) = \begin{pmatrix} v_x \cdot t \\ v_y \cdot t - 9.8/2 \cdot t^2 \end{pmatrix}$$



49

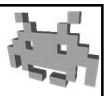
Numerical integration

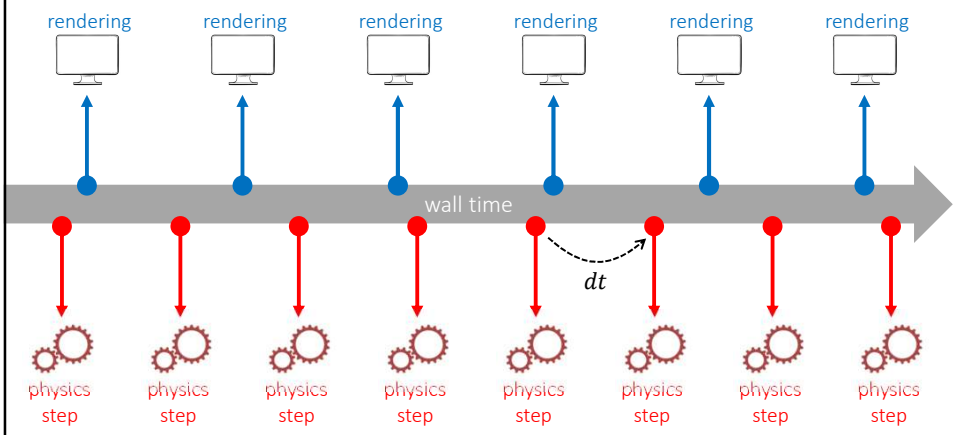


- A numerical integrator computes the integral as summed area of small rectangles
  - For a physics engine, this means just updating velocity and positions at each physics step
- A crucial parameter is the width of the rectangles i.e. *dt* = the duration of the physics step (in virtual time)
  - If physics system perform *N* steps per second: *dt* = 1.0 sec / N
  - *N* is not necessarily same rendering frame rate e.g.: rendering 30 FPS but physics: 60 steps per seconds
  - *dt* is not necessarily constant during the simulation (but in most system, it is)

50

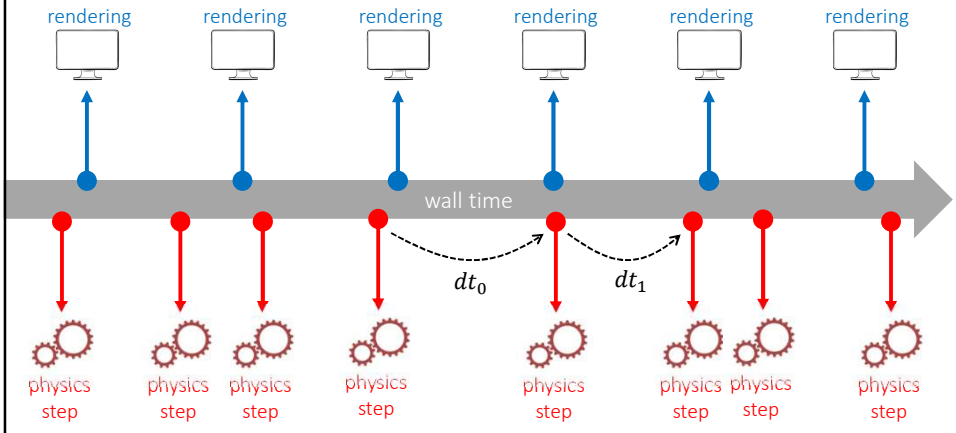
Rendering *Frames-per-Seconds* (FPS) vs Physics *Steps-per-Seconds*





51

In principle, timestep duration can be different in each step



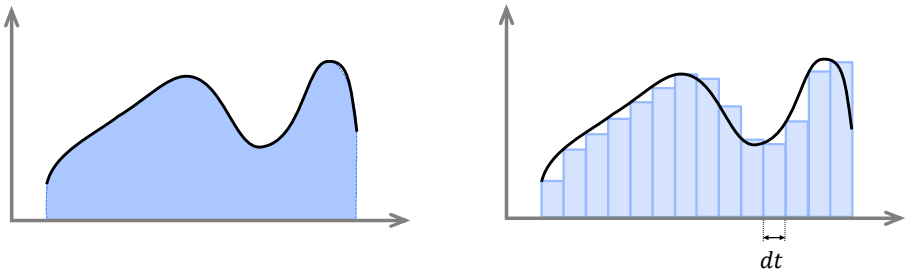
53

Numerical methods: features

- How efficient / expensive
  - must be at least soft real-time
    - (if from time to time computation delayed to next frame, ok)
- How accurate
  - must be at least plausible
    - (if stays plausible, differences from reality are acceptable)
- How robust
  - rare completely wrong results
    - (and never crash)
- How general
  - Which phenomena / constraints / object types is it able to recreate?
  - requirements depend on the context (ex: gameplay)

54

Numerical integration (concept)



area of each rectangle

$$\int \vec{v}(t) \cdot dt \Rightarrow \sum \vec{v}(t) \cdot dt$$

discretization step

55

Euler integration methods

For each step:

$\vec{f} = fun(\mathrm{p}, \ldots)$ → (1) Evaluate the combined **force** on each particle as a function of **positions** (of this and/or other particles) and maybe anything else, if needed

$\vec{a} = \vec{f}/m$ → (2) **acceleration** of each particle given by: total **force** acting on it divided by its mass

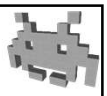
$p = p_0 + \int \vec{v} \cdot dt$ → (3) Update **position** with **velocity**

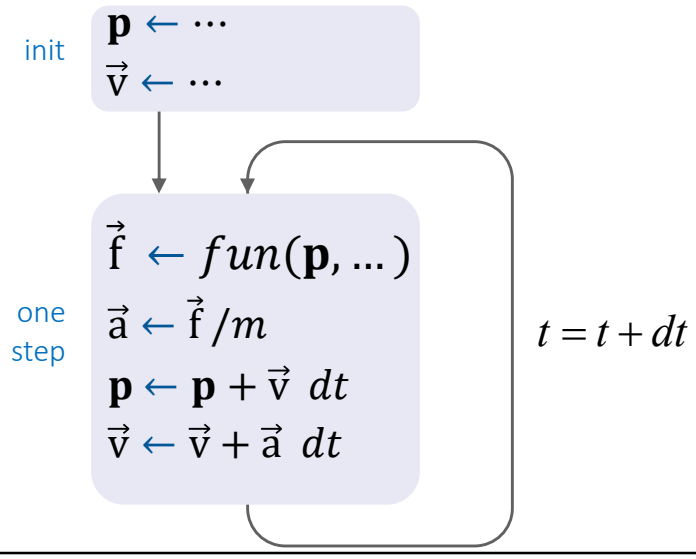
$\vec{v} = \vec{v}_0 + \int \vec{a} \cdot dt$ → (4) Update **velocity** with **acceleration**

green = state variables
blue = temp variables

56

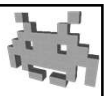
Euler integration methods





57

Forward Euler *pseudo code*



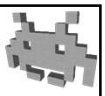
```
Vec3 position = …
Vec3 velocity = …

void initState(){
   position = …
   velocity = …
}

void physicStep( float dt )
{
   Vec3 acceleration = compute_force( position ) / mass;
   position += velocity     * dt;
   velocity += acceleration * dt;
}

void main(){
  initState();
  while (1) do physicStep( 1.0 / FPS );
}
```

Equivalent to…

$$\vec{f}_i = function(p_i, \ldots)$$
$$\vec{a}_i = \vec{f}/m$$
$$\vec{v}_{i+1} = \vec{v}_i + \vec{a}_i \cdot dt$$
$$p_{i+1} = p_i + \vec{v}_i \cdot dt$$

58

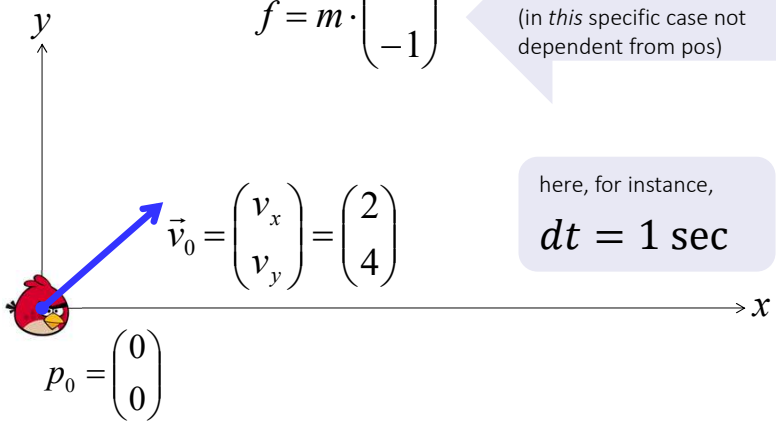
Simple example: numerical solution



Same phenomenon of previous example with G = (0, -1)

$$\vec{f} = m \cdot \begin{pmatrix} 0 \\ -1 \end{pmatrix}$$

constant
(in *this* specific case not dependent from pos)



$$\vec{v}_0 = \begin{pmatrix} v_x \\ v_y \end{pmatrix} = \begin{pmatrix} 2 \\ 4 \end{pmatrix}$$

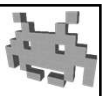
here, for instance,

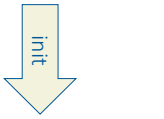
$$dt = 1 \text{ sec}$$

$$p_0 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

59

Simple example: numerical solution (with dt=1 sec)





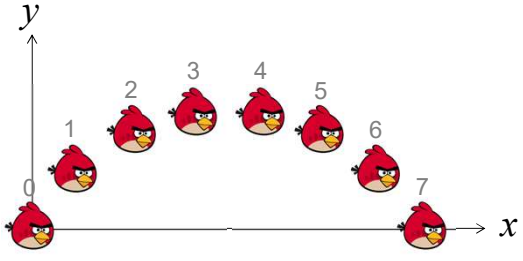
Time:	0	1	2	3	4	5	6	7	...
vel:	(2,3)	(2,2)	(2,1)	(2,0)	(2,-1)	(2,-2)	(2,-3)	(2,-4)	...
pos:	(0,0)	(2,3)	(4,5)	(6,6)	(8,6)	(10,5)	(12,3)	(14,0)	...

$$\vec{f} \leftarrow m \cdot \begin{pmatrix} 0 \\ -1 \end{pmatrix}$$

$$\vec{a} \leftarrow \vec{f}/m$$

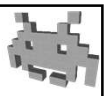
$$\vec{v} \leftarrow \vec{v} + \vec{a} \cdot dt$$

$$\mathbf{p} \leftarrow \mathbf{p} + \vec{v} \cdot dt$$



60

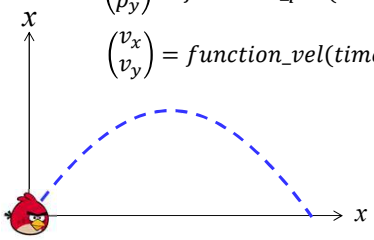
Physics evolution computation



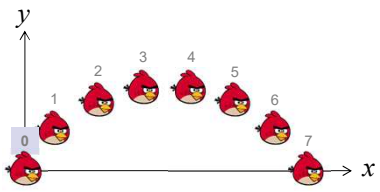
- Analytical solutions:

$$\begin{pmatrix} p_x \\ p_y \end{pmatrix} = function\_pos(time)$$

$$\begin{pmatrix} v_x \\ v_y \end{pmatrix} = function\_vel(time)$$

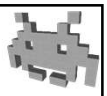


- Numerical solutions:



61

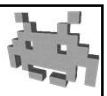
Physics evolution computation



- Analytical solutions:
  - Super efficient!
    - Close form solution
  - Accurate
  - Only simple systems
  - Formula must be found case by case
    (often they don't even exist)
  - NOT USED
    (but, for instance, useful to to make predictions for, e.g. A.I.)
- Numerical solutions:
  - Expensive (iterative)
    - but *interactive*
  - Integration errors
  - Flexible
  - Generic
  - USED FOR DYNAMICS COMPUTATIONS

62

Integration errors

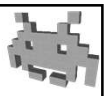


- A numerical integrator only approximates the actual value of the integrals
- The discrepancy (simulation errors) accumulates with virtual time during all the simulation
- How much error is accumulated?
- It depends on dt
  - smaller $dt \Rightarrow$ smaller error (simulation is more accurate)

  but, clearly
  - smaller $dt \Rightarrow$ more steps are needed (for simulate the same virtual time)
    $\Rightarrow$ simulation is more computationally expensive, but smaller errors,

63

The integration step dt of any numerical methods (summary)



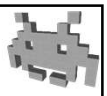
dt : delta of virtual time from last step
- the "temporal resolution" of the simulation!

number of physics steps per sec, or «physics FPS»

- if large: more efficiency
  - fewer steps to simulate same amount of virtual time
- if small: more accuracy
  - especially with strong forces and/or high velocities
- Common values: 1 sec / 60 … 1 sec / 30
  - i.e. a step simulates around 16 … 32 msec. of virtual time
  - note: it's not necessarily the same refresh rate of rendering (FPS of rendering ≠ FPS of physics. Rendering can be *less*!)
  - note: dt is not necessarily the same in all physics steps (need more accuracy *now*? Decrease *dt*)

64

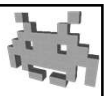
Order of convergence (of an integration method)



- How fast does the total error decrease as *dt* decreases?
  - That's called the Order of the simulation
  - 1st order: the total error can be as large as O(dt^1)
    - "if the number of physics steps doubles (physical computation effort doubles) *dt* becomes halves and errors can be expected to halve"
    - The error introduced by each single step is O(dt^2),
  - The Euler seen is 1st order
    - This is not too good, we want better
    - Note: The error is usually not that bad as linear with *dt*, but they *can* be

65

Effect of integration errors of System Energy



- Because of integration errors: simulated solutions ≠ "real" solutions
- In a real system, the total energy can never increase
  - typically, it *decreases* over time, due to dissipations
  - that is, attrition turns *dynamic energy* into *heat*
- Therefore, a particularly nasty integration error is when the total energy of the system *increases* over time
  - e.g.: a pendulum swings wider and wider
- Particularly bad because:
  - compromises stability (velocity = big, displacements = crazy, error = crazy)
  - compromises plausibility (we can *see* it's wrong)
- A simple way to avoid this: make sure the simulation always includes attritions
  - makes simulation more stable + robust

66

Forces: examples

$$\dots$$
$$\vec{f} = \text{function}(\mathbf{p}, \dots)$$
$$\dots$$

Primarily, a function of the positions

But not always, and sometimes not only of positions. Also: velocities? Global time? Buttons being pressed? Etc.

- Gravity
  - Constant · *m*, near the surface of a planet
  - Function of positions in a space simulation
- Wind pressure
  - Depends on the area exposed in the wind direction
- Electrical / magnetic forces
- Buoyancy *(ita: forza di Archimede)*
  - Depends on the weight of the submerged volume
- Mechanical springs
  - simple model: Hooke's law
- Shock waves (explosions)
- Fake / "Magic" control forces
  - added for controlling the evolution of the system, not physically justified

67

Example of forces: gravitational forces (near a planet)

- Given a particle in $\mathbf{p}$ with (gravitational) mass m

Gravity acceleration constant (a scalar, depends on the planet)

Downward versor

$$\vec{f} = m\, g\, \hat{d}_{Down}$$

$\vec{G} = g\ \hat{d}_{Down}$

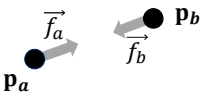
- Note: does not depend on position
- Note: if this is the only force, acceleration is just $\vec{a} = \frac{m}{m} g\, \hat{d}_{Down}$
- We already seen this example with $\vec{G} = g\ \hat{d}_{Down}$

Gravitational mass

Inertial mass

68

Example of forces: gravitational forces (in open space)



- Given two particles in $\mathbf{p}_a$ and $\mathbf{p}_b$ with (gravitational) masses m_a and m_b

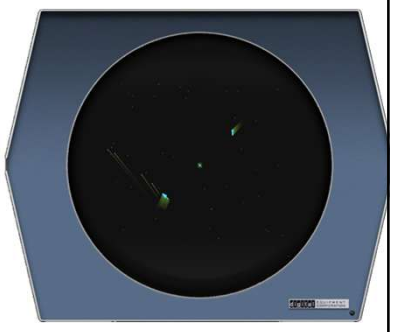
a global constant

$$\vec{f_a} = \underbrace{\frac{G\, m_a\, m_b}{\|\mathbf{p}_b - \mathbf{p}_a\|^2}}_{\text{force magnitude (a scalar)}} \underbrace{\frac{\mathbf{p}_b - \mathbf{p}_a}{\|\mathbf{p}_b - \mathbf{p}_a\|}}_{\text{force direction (a versor)}}$$

$$= \frac{G\, m_a\, m_b}{\|\mathbf{p}_b - \mathbf{p}_a\|^3} (\mathbf{p}_b - \mathbf{p}_a)$$

$$\vec{f_b} = -\vec{f_a}$$

As used by the first videogame: **spacewars!**



this image from the simulation at https://www.masswerk.at/spacewar/

70

Example of forces: gravitational force (in open space)

- A particle A in pos $\mathbf{p}_a$ with (gravitational) mass m_a is attracted by a particle B in pos $\mathbf{p}_b$ with (gravitational) mass m_b by a force

some global constant dependent on… the universe

$$\vec{f_a} = \underbrace{\frac{\mathbf{p}_b - \mathbf{p}_a}{||\mathbf{p}_b - \mathbf{p}_a||}}_{\text{force direction, from A to B (versor)}} \underbrace{\frac{G\, m_a\, m_b}{||\mathbf{p}_b - \mathbf{p}_a||^2}}_{\text{force magnitude (positive scalar)}} =$$

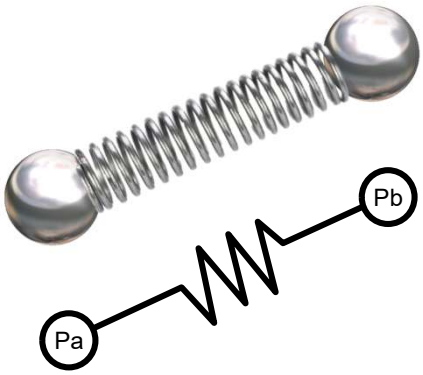
$$= (\mathbf{p}_b - \mathbf{p}_a) \frac{G\, m_a\, m_b}{||\mathbf{p}_b - \mathbf{p}_a||^3}$$

note: and B is also attracted by A, by exactly the opposite force $\vec{f_b} = -\vec{f_a}$

as seen in Space Wars, 1962

71

Forces: Springs



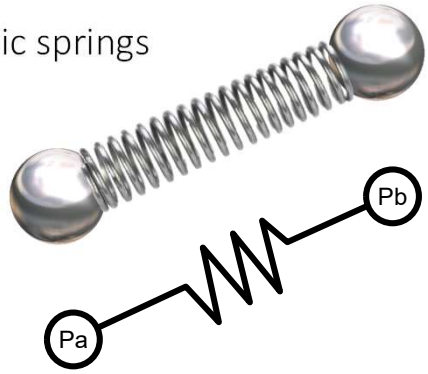
Hooke's law: $\overrightarrow{f_a} = k(\|\mathbf{p}_b - \mathbf{p}_a\| - \ell)\frac{\mathbf{p}_b - \mathbf{p}_a}{\|\mathbf{p}_b - \mathbf{p}_a\|}$

force direction (versor)

72

Forces: Springs (Hooke's law)

- Simplified model for elastic springs
- One spring connects two particles in $\mathbf{p}_a$ and $\mathbf{p}_b$
- Characterized by:
  1. Rest length ℓ
  2. Stiffness k
- Spring force: counteracts expansion and compression

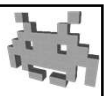


$$\overrightarrow{f_a} = k(\|\mathbf{p}_b - \mathbf{p}_a\| - \ell)\frac{\mathbf{p}_b - \mathbf{p}_a}{\|\mathbf{p}_b - \mathbf{p}_a\|}$$

$$\overrightarrow{f_b} = -\overrightarrow{f_a}$$

73

Forces: Springs (Hooke's law)



$$\overrightarrow{\mathrm{f}_a} = \underbrace{k(\|\mathbf{p}_b - \mathbf{p}_a\| - \ell)}_{\text{force magnitude (scalar) (positive or negative)}} \underbrace{\frac{\mathbf{p}_b - \mathbf{p}_a}{\|\mathbf{p}_b - \mathbf{p}_a\|}}_{\text{force direction (versor)}}$$

elongation (if positive) OR compression (if negative)

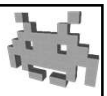
← force to be applied to particle a

$$\overrightarrow{\mathrm{f}_b} = -\overrightarrow{\mathrm{f}_a}$$

← force to be applied to particle b

74

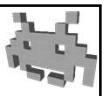
The "third law of dynamics" (a minidigression from high school)



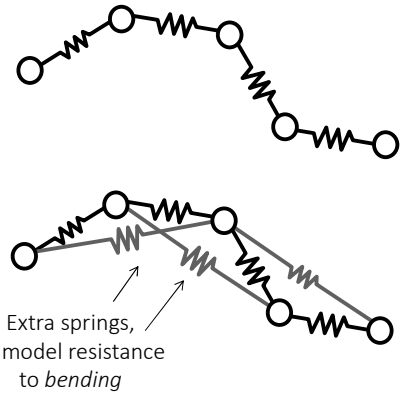
- Note: $\overrightarrow{\mathrm{f}_b} = -\overrightarrow{\mathrm{f}_a}$
- This (which you can compute independently) is just an example of the "3rd law of dynamics":
  for every action (force) in nature, there is an equal and opposite reaction
- "If particle attracts/repulses particle b with a force, particle b does the same with a force with the same magnitude and opposite verse"
- That is: the sum of all forces of the system $\overrightarrow{\mathrm{f}_a} + \overrightarrow{\mathrm{f}_b}$ is zero
- Rember forces = derivative of momentum.
  So: the total momentum (in absence of external forces) is constant

75

Mass and Spring systems

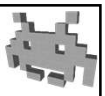


- Useful for deformable objects
- for instance: elasitic ropes (or hairs)

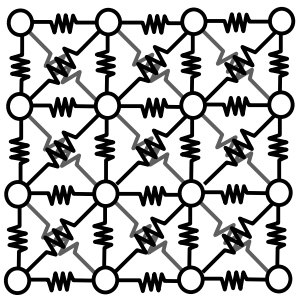


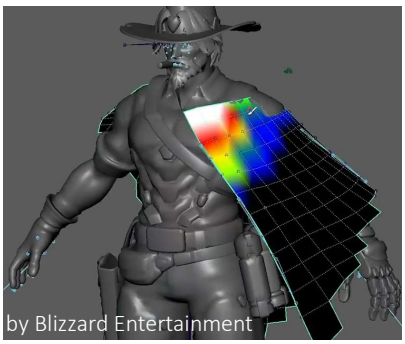
78

Mass and Spring systems



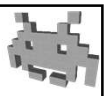
- For instance: cloth





79

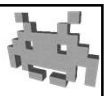
Mass and Spring systems can model…



- Elastic deformable objects (aka “soft bodies”)
  - Elastic = they go back to original shape if no ext. force is applied
  - Easily modelled as compositions of (ideal) springs.
- Plastic deformable objects? (yes, but not easy)
  - Plastic = deformed pose is kept permanently
  - Dynamically change rest-length L in response to large compression/stretching, in certain conditions
- Rigid bodies / inextensible ropes ? (no, they can’t)
  - Increase spring stiffness? $k \rightarrow \infty$
  - Makes sense, physically, but…
  - Large $k \Rightarrow$ large $\mathbf{f} \Rightarrow$ instability $\Rightarrow$ unfeasibly small dt needed
  - Doesn’t work. How to do them, then? see in a few lectures!

80

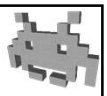
Forces: control forces



- Example: the player pressing the forward button
  $\Rightarrow$ a forward force is applied to their avatar
  - no physical justification
  - “Don’t ask questions, physics engine”
- According to many:
  it’s better when that’s not done too much
  - the more physically justified the forces, the better
  - for example: does the car accelerate…
    because a torque is applied to its two traction wheels VS
    because a force is applied to its body
  - can be harder to control
  - see also: gameplay VS cosmetics, control VS realism, emerging behaviors

81

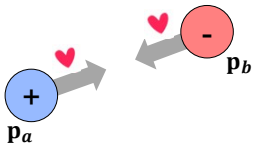
Examples of forces: electric forces



- Given two charged particles in $\mathbf{p}_a$ and $\mathbf{p}_b$ with positive or negative charges q_a and q_b

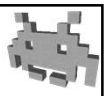
some global constant

$$\overrightarrow{f_a} = \underbrace{\frac{-K\, q_a\, q_b}{\|\mathbf{p}_b - \mathbf{p}_a\|^2}}_{\text{force magnitude (scalar) positive or negative}} \underbrace{\frac{\mathbf{p}_b - \mathbf{p}_a}{\|\mathbf{p}_b - \mathbf{p}_a\|}}_{\text{force direction (versor)}} = \frac{-K\, q_a\, q_b}{\|\mathbf{p}_b - \mathbf{p}_a\|^3} (\mathbf{p}_b - \mathbf{p}_a)$$

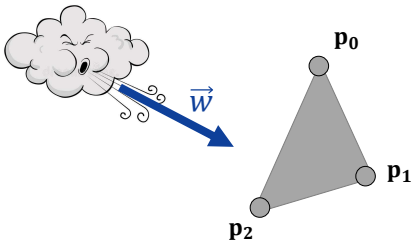


83

Examples of forces: wind pressure



- Wind is a force acting on surfaces
- The larger the exposed surface to the wind, the STRONGER / MORE INTENSE the force
- The more orthogonal the surface to the wind direction, the larger the force
- The stronger the wind pressure $\vec{w}$ (a vector), the larger the force



area vector

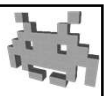
$$\vec{f} = \underbrace{\left|\frac{\mathbf{1}}{\mathbf{2}}(\mathbf{p_1} - \mathbf{p_0}) \times (\mathbf{p_2} - \mathbf{p_0}) \cdot \vec{w}\right|}_{\text{force magnitude (scalar)}} \underbrace{\frac{\vec{w}}{\|\vec{w}\|}}_{\text{force direction (versor)}}$$

(apply 1/3 of $\vec{f}$ on each particle)

84

Examples of forces: buoyancy

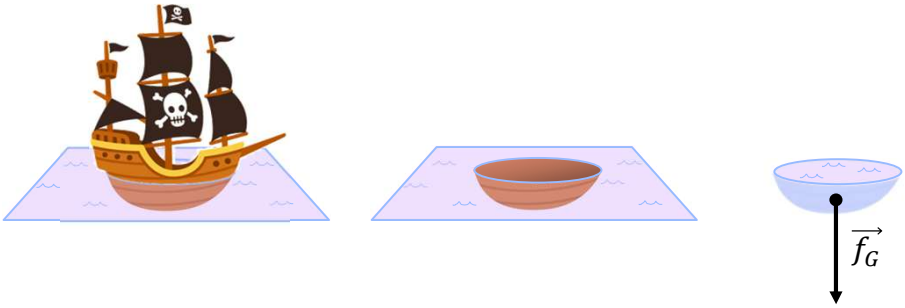
Italian: "forza di Archimede"



- Opposite of gravity force $\overrightarrow{f_G}$
  - of the submerged part... if it was made of water
  - mass of the submerged part = its volume *times* density of water

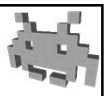
or whichever liquid

mass/volume
aka "specific mass"



85

Forces



- Remember all forces acting on a particle add up!
  (it's a vector summation)
- Only the resulting force is what counts

one step

$$\vec{f} \leftarrow fun(\mathbf{p}, \dots)$$
$$\vec{a} \leftarrow \vec{f}/m$$
$$\mathbf{p} \leftarrow \mathbf{p} + \vec{v} \cdot dt$$
$$\vec{v} \leftarrow \vec{v} + \vec{a} \cdot dt$$

$t = t + dt$

86