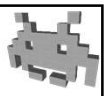
Course Plan



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. 9: Game **Materials**
lec. 8: Game **3D Animations**
lec. 10: **Networking** for 3D Games
lec. 11: **3D Audio** for 3D Games
lec. 12: **Rendering Techniques** for 3D Games
lec. 13: **Artificial Intelligence** for 3D Games

92

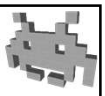
Example of forces: etc

- Remember all forces acting on a particle add up! (vector summatory)

one step

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

94



Example of forces: gravitational forces (near a planet)

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

Gravity acceleration constant (scalar,depends on the planet)

Downward versor

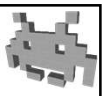
$$\vec{f} = g\, m\, \hat{d}_{Down}$$

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

Gravitational mass

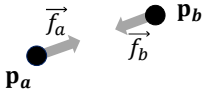
Inertial mass

97



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



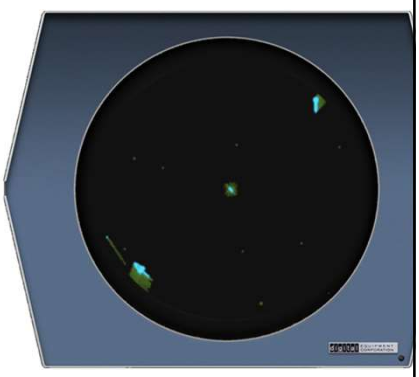
a global constant

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

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

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

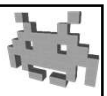
As used by the first videogame: **spacewars!**



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

98

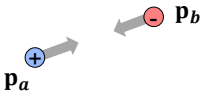
Example of forces: electric forces



- 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

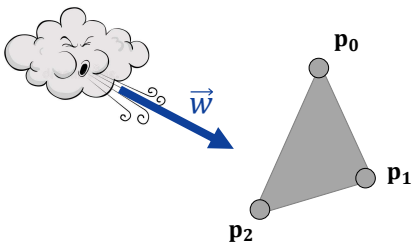
$$\overrightarrow{f_a} = \underbrace{\frac{-K\, q_a\, q_b}{\|\mathbf{p}_b - \mathbf{p}_a\|^2}}_{\text{force magnitude (scalar) positive or negative}} \underbrace{\frac{\mathbf{p}_b - \mathbf{p}_a}{\|\mathbf{p}_b - \mathbf{p}_a\|}}_{\text{force direction (versor)}} = \frac{-K\, q_a\, q_b}{\|\mathbf{p}_b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



100

Example 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



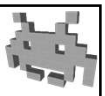
area vector

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

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

101

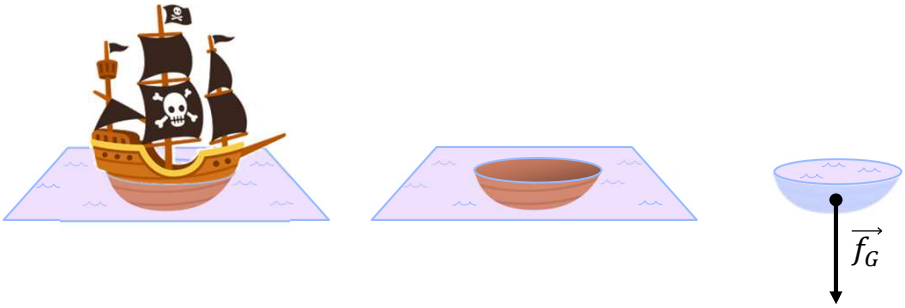
Example of forces: buoyancy



- 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

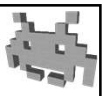
or whichever liquid

mass/volume
aka "specific mass"



102

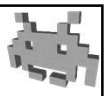
Attrition (or friction) forces



- *Isotropic* friction forces :
  - a force that opposes any motion, regardless of its direction
  - direction: always opposite of current velocity direction
  - magnitude: proportional to the speed (= magnitude of velocity vector)
  - note: this force depends on velocity, not positions.
  - models the effect of the medium where the motion happens (air, water, thin space…)
  - the denser the medium, the stronger the force (water >> air >> thin space)
- Planar friction forces:
  - A force that happens when things slide against each other
  - Always parallel to the contact plane (orthogonal to the normal)

103

Attrition (or friction) forces: simulate them with *velocity damping*



- A useful trick to simulate isotropic friction: "velocity damping"
  - simply reduce all velocity vectors by a fixed proportion
  - for example: scale velocity down by 2% per second ("drag factor" = 0.02 / sec)
    (that is, scale velocity vectors by a factor 0.98)
  - Why it makes sense:
    Higher speed = more attrition = more loss of speed.
    So, attrition = a "fixed tax" (in %) on speed.
- For planar friction:
  - Split velocity into parallel / orthogonal parts
  - Apply different Drag factors to each parts

104

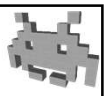
Velocity Damping: how to (in one example)

- Objective: "reduce speed by 1.5% *every second*"
  Drag factor: 0.015
- So:
  - After 1 second: $\vec{v} \leftarrow (1.0 - 0.015)\,\vec{v}$
  - After 2 seconds? $\vec{v} \leftarrow (1.0 - 0.015)^2\,\vec{v}$
  - After k seconds? $\vec{v} \leftarrow (1.0 - 0.015)^k\,\vec{v}$
  - After dt seconds? $\vec{v} \leftarrow (1.0 - 0.015)^{dt}\,\vec{v}$

    e.g 1/60 = 0.17 sec
  - Which can be approximated with $\vec{v} \leftarrow (1.0 - 0.015 \cdot dt\,)\vec{v}$
  - The approximation is good when *this* is small

105

Velocity Damping: pseudo-code



```
Vec3 position = …
Vec3 velocity = …

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

void physicStep( float dt )
{
   Vec3 acceleration = force( positions ) / mass;
   position += velocity * dt;
   velocity += acceleration * dt;
   velocity *= (1.0 – DRAG * dt);
}

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

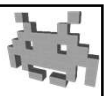
106

Velocity Damping: notes

- Velocity Damping is useful for robustness,
  - Prevents the energy to ever increase
- Problems of Velocity Damping
  - it may exaggerate frictions of, e.g., air, especially in absence of contacts
  - it's a crude approximation: attrition forces are not really *linear* with speed
- In practice:
  - low drag: hardly noticeable (in the short run), increases robustness
  - high drag: everything feels like to be moving in molasses; (ita: *melassa*); everything quickly grinds to a halt
  - super high drag: (e.g. 10% per sec) basically, no inertia anymore. May be useful to converge to (local) minimal energy states: your simulator is basically solver for statics not dynamics

107

Continuity of pos and vel



- In real Newtonian physics the state (pos and vel) can only change *continuously*
  - No sudden jump!
- In practice, sometimes is useful to artificially break continuity in the simulations
- Discontinuous changes:
  - for positions: “teleports”
  - for velocity: “impulses”
  - In the real world, those variations can well be consequences of forces, but these forces are not modelled as such, in our system

108

Dynamics displacements VS kinematic

a **discontinuous** change of state (position)

$$\ldots$$
$$\mathrm{p} = \mathrm{p} + \vec{\mathrm{v}} \cdot dt$$
$$\ldots$$

aka dynamic displacements

Justified by physics

$$\ldots$$
$$\mathrm{p} = \mathrm{p} + d\mathrm{p}$$
$$\ldots$$

aka Kinematic displacements

Just “teleportation”

109

Impulses VS Forces

a **discontinuous**
change of state (velocity)

…

$$\vec{v} = \vec{v} + \left(\vec{f} / m\right) \cdot dt$$

…

- Forces (continuous)
  - Continuous application
  - every frame

…

$$\vec{v} = \vec{v} + \left(\vec{i} / m\right)$$

…

- Impulses
  - Infinitesimal time
  - una tantum

they model very intense but
short forces
(such as impacts)

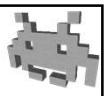
110

Impulses VS Forces

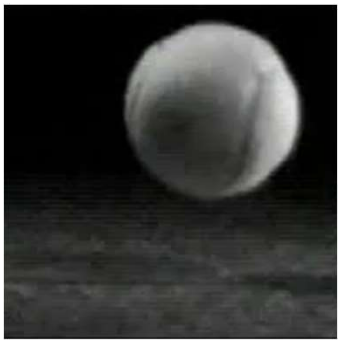
- Force :
  - it determines an acceleration
  - acc determines a (continuous!) change of vel
  - physically correct
- Impulse :
  - a (discontinuous!) change of vel
  - useful to control a simulation (direct change of velocity)
  - a physical interpretation: a force with:
    - application time approaching **zero**
    - magnitude approaching **infinity**
  - Useful to model phenomena with a time scale << *dt*
    - ex: a tennis ball rebounding against a tennis racket

111

Impulses VS Forces



- what does *truly* happen when it bounces off the ground?



- very strong forces (but not infinite)
- applied for a very short time (but not instantaneous)
- see *collision response* later for details about the impulse-based approximations

112

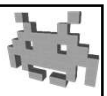
Impulses VS Forces

- what does *truly* happen when it bounces off the ground?

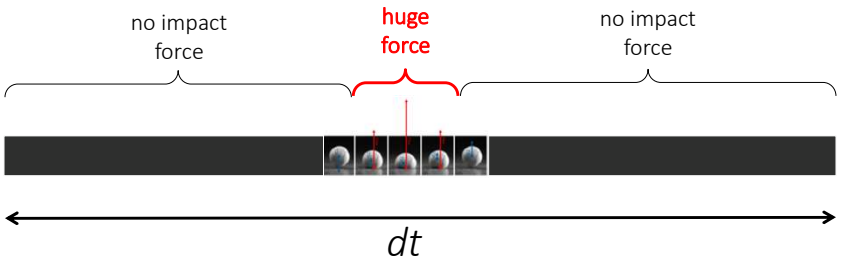
- very strong forces (but not infinite)
- applied for a very short time (but not instantaneous)
- see *collision response* later for details about the impulse based approximations

113

Impulses VS Forces



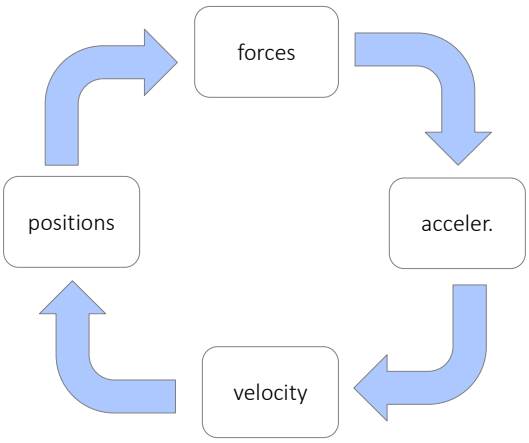
- what does *truly* happen when it bounces off the ground?



- This can only be modelled as an *impulse*, not a force
- *See also collision response, later*

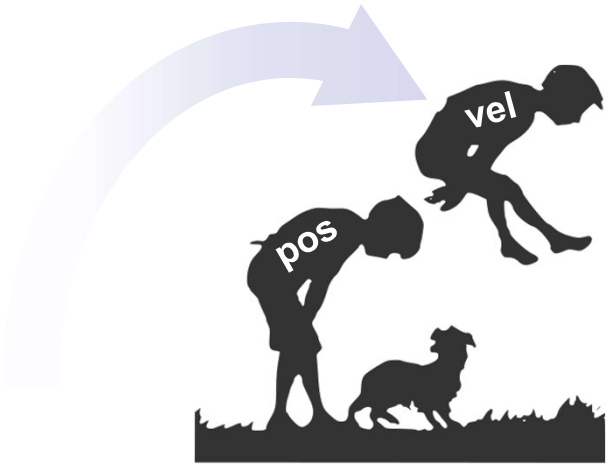
114

Next: better integration methods for (Newtonian) dynamics



115

Leapfrog Integration Method



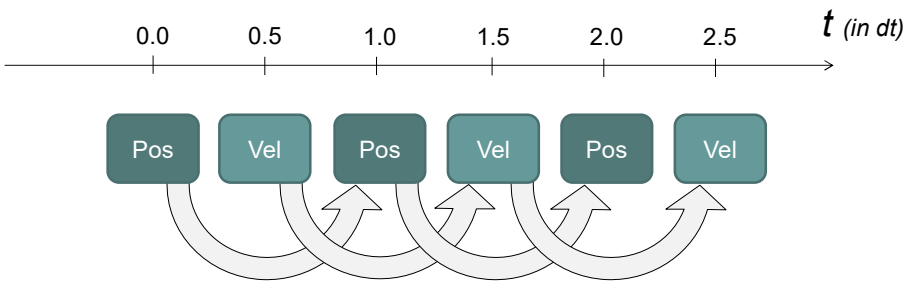
116

Leapfrog Integration Method

- Basic Idea:
  store positions at time $k \cdot dt$ (0, dt, 2dt, 3dt…)
  but store velocities at time $k \cdot dt + ½\ dt$
- Equivalent to use a summatory of
  the areas of trapezoids,
  (having base dt and height $v(t)$)
  not rectangles, to compute the integral

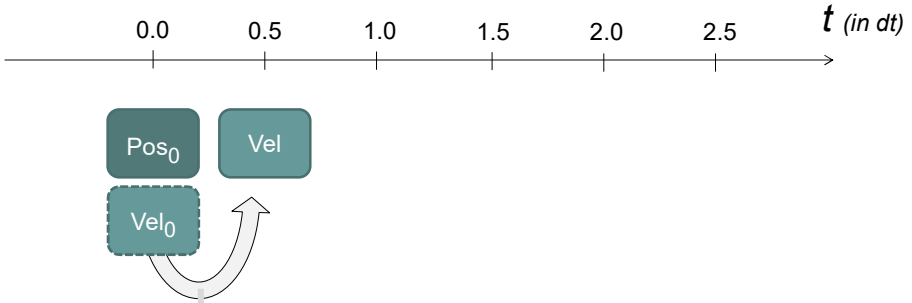
117

Leapfrog Integration Method



118

Leapfrog Integration first step

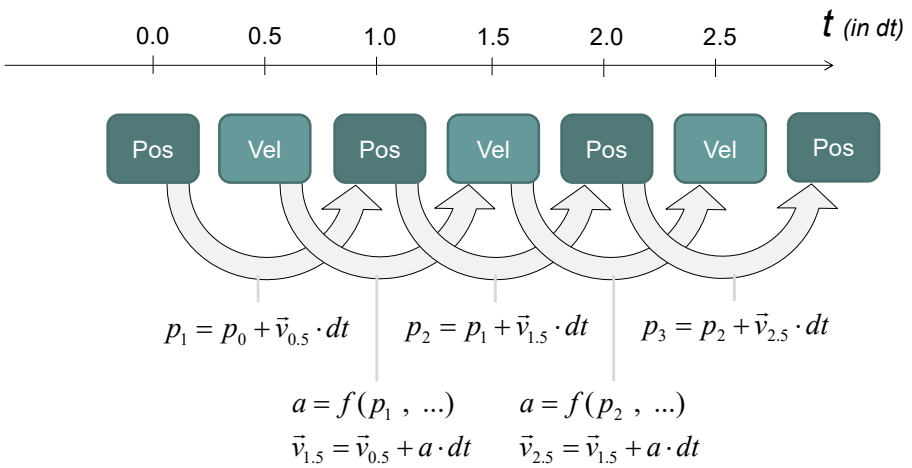


$$a = f(p_0\ ,\ ...)$$

$$\vec{v}_{0.5} = \vec{v}_0 + \vec{a} \cdot dt\,/\,2$$

119

Leapfrog Integration



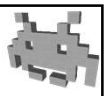
120

Leapfrog method: pros and cons

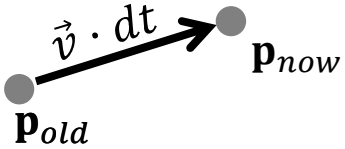
- Same cost as Euler – and basically same code
  - Velocity stored in status = velocity "half a *dt* ago" (and after updating it: "half a frame in the future")
  - Only real difference: the initialization of speed
- Better theorical accuracy, for the same *dt*
  - better asymptotic behavior: it's a "second order" system instead of first!
  - cumulated error: proportional to dt^2 instead of *dt*
  - error per frame: proportional to dt^3 instead of dt^2
- Bonus: fully reversible!
  - in theory only. Beware numerical errors.
- But: requires fixed *dt* during all the simulation
  - for the theory to work as advertised

121

Verlet integration method



- Idea: remove velocity from state
  Instead, store previous position
- Velocity is now implicit
- It's defined by:
  - current pos $\mathbf{p}_{now}$
  - last pos $\mathbf{p}_{old}$
    which we need to record



$$\mathbf{p}_{now} = \mathbf{p}_{old} + \vec{v} \cdot dt$$ ← Euler & variants

$$\Longrightarrow$$

$$\vec{v} = (\mathbf{p}_{now} - \mathbf{p}_{old})/dt$$ ← Verlet

122

Verlet integration method: (modifying Euler integration…)

init state

$$\mathbf{p}_{now} = \dots$$
$$\mathbf{p}_{old} = \dots$$

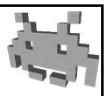
one step

$$\vec{f} = funct(\mathbf{p}_{now}, \dots)$$
$$\vec{a} = \vec{f}/m$$
$$\vec{v} = (\mathbf{p}_{now} - \mathbf{p}_{old})/dt$$
$$\vec{v} = \vec{v} + \vec{a} \cdot dt$$
$$\mathbf{p}_{\text{next}} = \mathbf{p}_{now} + \vec{v} \cdot dt$$

expanding this…

123

Verlet integration method (modifying Euler integration…)



init state

$\mathbf{p}_{now} \leftarrow \cdots$
$\mathbf{p}_{old} \leftarrow \cdots$

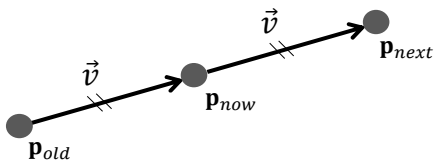
one step

$\vec{f} \leftarrow funct(\mathbf{p}_{now})$
$\vec{a} \leftarrow \vec{f}/m$
$\mathbf{p}_{next} \leftarrow 2\mathbf{p}_{now} - \mathbf{p}_{old} + \vec{a} \cdot dt^2$

$\mathbf{p}_{old} \leftarrow \mathbf{p}_{now}$
$\mathbf{p}_{now} \leftarrow \mathbf{p}_{next}$

124

Verlet integration method: geometric interpretation



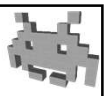
$\mathbf{p}_{next} = 2 \cdot \mathbf{p}_{now} - 1 \cdot \mathbf{p}_{old}$

$\mathbf{p}_{next}$ can be written as an extrapolation of $\mathbf{p}_{now}$, $\mathbf{p}_{old}$:

$\mathbf{p}_{next} = mix(\mathbf{p}_{old}, \mathbf{p}_{now}, 2)$

125

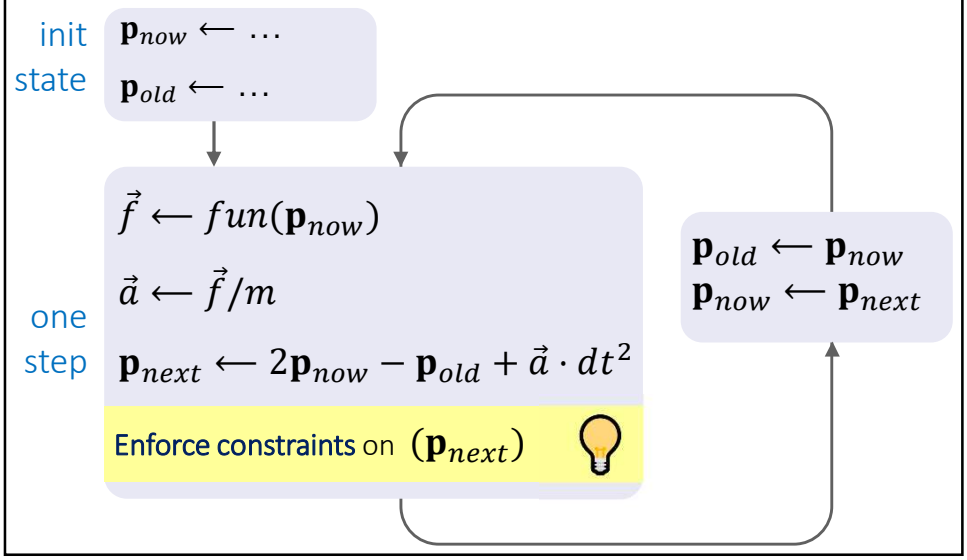
Verlet: characteristics



- Velocity is kept implicit
  - but that doesn't save RAM:
    we need to store previous position instead
  - (a point instead of a vector: same memory)
- Good efficiency / accuracy ratio
  - Per-step error: linear with *dt*
  - accumulated error: order of dt^2 (second order method)
- Extra bonus: reversibility
  - it's possible to go backward in *t* and
    reach the initial state from any state
  - only in theory… careful with implementation details

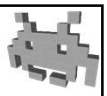
126

Verlet integration + "Position Based Dynamics" (PBD)



127

Position Based Dynamics (PDB)



- A positional constraint is an equality/inequality (a formula with '=' '>' '<' etc.) involving the *positions* of particles.
  - Useful, for example, to model consistency conditions
  - Like "*solid objects don't compenetrate each other*", or "*steel bars won't become shorter or longer than they are*"
  - We will see many examples

We enforce (impose) positional constraint directly by displacing the *positions* of particles

- Thanks to Verlet: this displacement automatically causes some appropriate update of the velocity!
- it's not necessarily correct, but it's plausible and robust

128

Example of a positional constraint

«I want all particles to stay above ground (that is, their y must never be negative) »

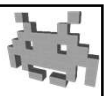
Enforce this constraint: trivial!

```
for (each particle i)
{
  if (p[i].y < 0) p[i].y = 0;
}
```

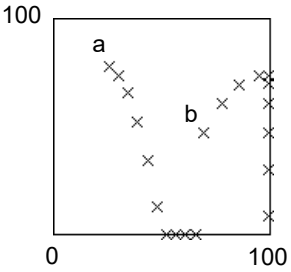
Imposing constraints like this one is a first part of collision response. For re-bounces, impulses must still be added (see collisions).

129

Example of a positional constraint (here, in 2D physics)



«I want particles to stay inside a 2D box [0 – 100] x [0 – 100] *»*



Enforce this constraint: simple clamp!

```
for (each particle i)
{
  p[i].x = clamp( p[i].x, 0, 100 );
  p[i].y = clamp( p[i].y, 0, 100 );
}
```

Imposing constraints like this one is a first part of collision response. For re-bounces, impulses must still be added (see collisions).

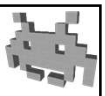
130

Verlet + Position Based Dynamics. Advantages

- flexibility: different constraints can be used to model many different phenomena
  - Useful constraints are straightforward to define
  - They are easy to impose (they involve only few particles)
  - They can be used to model many possible phenomena
  - See following slides for examples
- robustness : plausibility is ensured by *explicitly* enforcing the conditions we want to see
  - For example: a ball won't ever be seen outside the box containing it – and it will also recover from mistakes
- No forces / impulses are needed to enforce any such consistency conditions

131

How to enforce positional constraiont?
(see next lecture for the anwser)



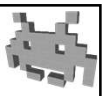
When enforcing constraints…

- If a constraint is valid, no problem.
- If it doesn't, there can be many way to change particle POSITIONS, so that it does

⚠ Which one to pick?

132

Verlet: *caveats*
(see next lecture for solutions)



⚠ it assumes a constant *dt* (time-step duration)
- if *dt* varies: corrections are needed! (how?)

⚠ Q: how to act on velocity (which is now implicit)?
- for example, how to apply impulses ?
- A: change $\mathbf{p}_{old}$ instead (how?)

⚠ Q: how to act of positions w/o impacting velocity?
- for example, to apply teleports / kinematic motions ?
- A: change both $\mathbf{p}_{new}$ and $\mathbf{p}_{old}$ (how?)

⚠ Q: how to apply velocity damps?

133