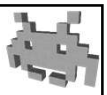
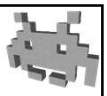
Course Plan



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

104

Verlet integration + "Position Based Dynamics" (PBD)



init state

$\mathbf{p}_{now} \leftarrow \dots$

$\mathbf{p}_{old} \leftarrow \dots$

one step

$\vec{f} \leftarrow fun(\mathbf{p}_{now})$

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

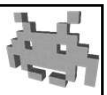
$\mathbf{p}_{next} \leftarrow 2\mathbf{p}_{now} - \mathbf{p}_{old} + \vec{a} \cdot dt^2$

Enforce constraints on $(\mathbf{p}_{next})$

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

105

Position Based Dynamics



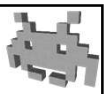
- A positional constraint is an equation/inequality involving the *positions* of particles.
  - Useful, for example, to model consistency conditions
  - Like "*solid objects don't compenetrate each other*", or "*steel bars won't become shorter or longer*"
  - We will see many specific examples

a formula with '=' '>' '<' etc.

- 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, it's plausible and robust

106

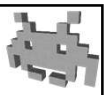
Verlet + Position Based Dynamics. Advantages



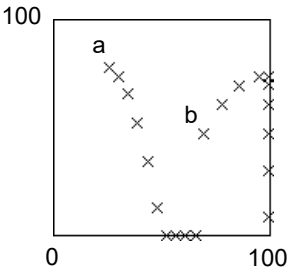
- flexibility: different constraints can be used to model many different phenomena
  - Useful constraints are straightforward to define
  - They are easy to impose (they involve only few particles)
  - They can be used to model many possible phenomena
  - See following slides for examples
- robustness : 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 (at lest, not for many frames)
- No forces / impulses are needed to enforce the same consistency condition
  - Which would be what happens in reality, but also much more difficult to do robustly

107

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



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



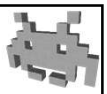
Imposing constraint: simple clamp !
ex:

```
for(int i=0; i<NUM_PARTICLES; i++)
{
  p[i].x = clamp( p[i].x, 0, 100 );
  p[i].y = clamp( p[i].y, 0, 100 );
}
```

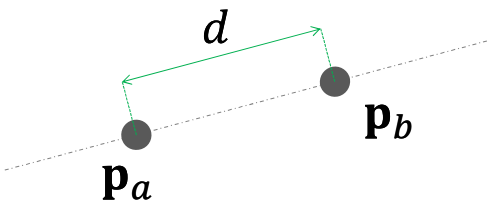
Imposing constraints like this is a first part of collision response.
For re-bounces, impulses must still be added (see collisions).

108

Example of positional constraint: equidistance constraint



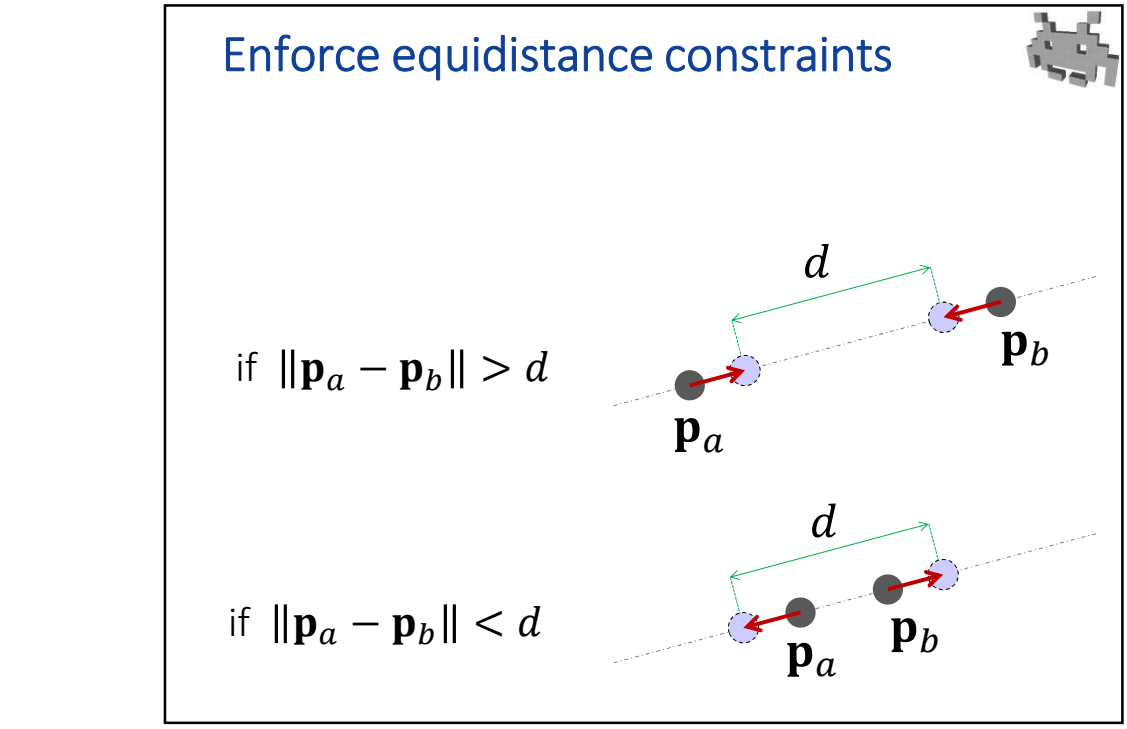
«Particles $\boldsymbol{a}$ *and* $\boldsymbol{b}$ *must stay at a fixed distance* $\boldsymbol{d}$ *»*



I want that... $\|\mathbf{p}_a - \mathbf{p}_b\| = d$

109

Enforce equidistance constraints



110

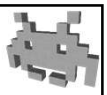
Enforce equidistance constraints: pseudo code

```
Vector3 pa, pb; // curr positions of a,b
float d;        // distance (to enforce)

Vector3 d = pa - pb;
float currDist = v.length;

d /= currDist;  // normalization of d

float delta = currDist - d ;

pa += ( 0.5 * delta) * d;
pb -= ( 0.5 * delta) * d;
```

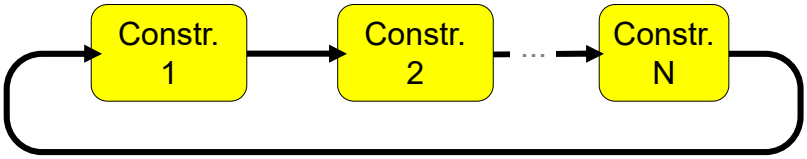
assuming equal mass, we move each particle *half the way*
(see later for the more general case)

111

Enforcing sets of constraints



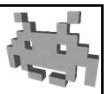
- There are many constraints to impose:
  when you solve one → maybe you break another!
- Simultaneous enforcement: computationally expensive
- Practical & easy solution: enforce them in cascade
  (Gauss-Seidel fashon):



Repeat until convergence (= max error below threshold)
…but at most for *N* times! (always remember: it's *soft* real-time)
(or, just repeat a fixed number *N* of times)

112

Enforcing sets of constraints



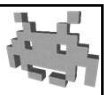
- Note:
  - The whole loop for imposing the constraints happen in the constraint enforcement phase on one physics step!
- Convergence:
  - if constraints are not contradictory
  - if convergence not reached (or solution doesn't exist): never mind, next frames will fix it (it's fairly robust)
  - needed iterations (typically): 1 ~ 10 (efficient!).
  - Optimization (to decrease number of needed iterations): solve the most unsatisfied constraints first

⚠ Problem: it's a sequential approach! ☹

  - parallelized versions (similar to Jacobi) are possible
  - they have a worse convergence in practice
    (they require more iterations)

113

Equidistance constraints VS springs



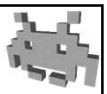
- They are similar
  - they both mean:
    these 2 particles "want to be" at *this* distance (not more, not less)

some constant scalar parameter L

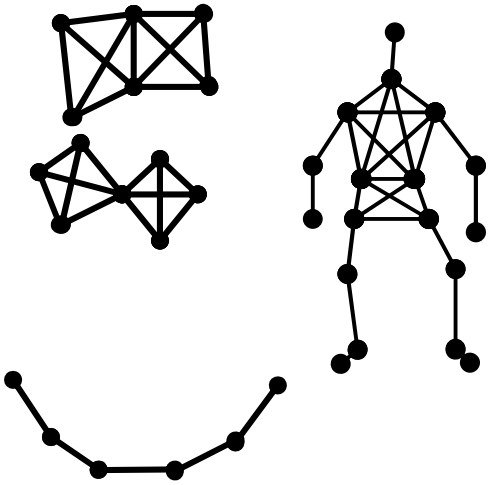
- Differences:
  - equidistance constraint:
    - applied during constraint enforcement
      - directly affecting positions
    - models a rigid rod between the two particles
      - of a given length
    - sometimes called an "HARD" constraint
  - spring:
    - applied during force evaluation step
      - affecting forces, therefore accelerations
    - models a deformable spring between the two particles
      - of a given length
    - sometimes called a "SOFT" constraint
- A physic engine can combine them in one object!

114

We can combine equidistance constraints to obtain…

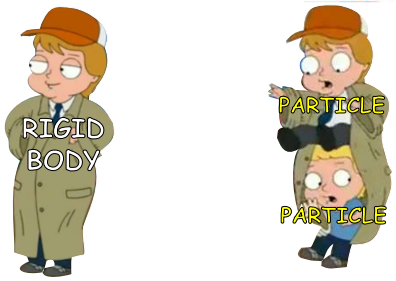


- Rigid bodies
- Articulated bodies
- Ragdolls
- Cloth
- Non-elastic ropes
- And more



115

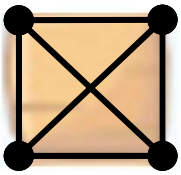
Compounds of particles + constraints disguised as rigid bodies



116

Combining equidistance constraints we obtain rigid objects

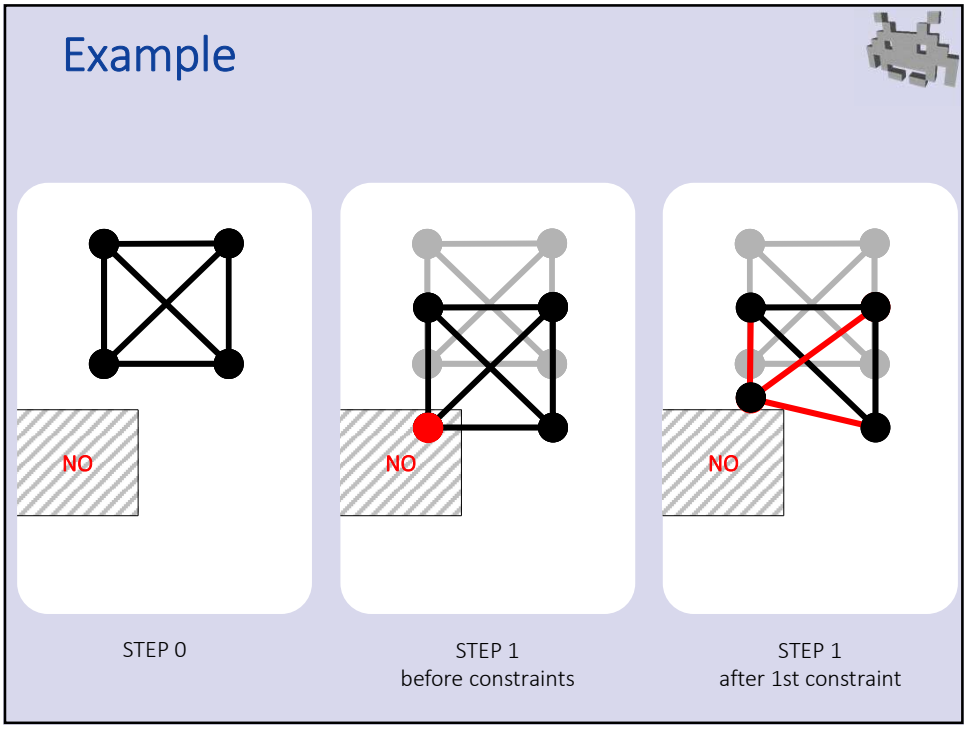
- Rigid body dynamics as emerging behavior
  - without explicitly keeping track their orientation, angular vel, angular acc., etc.



A box?
(rigid object)
In 2D a configuration of:
- 4 particles
- 6 equidistance constraints

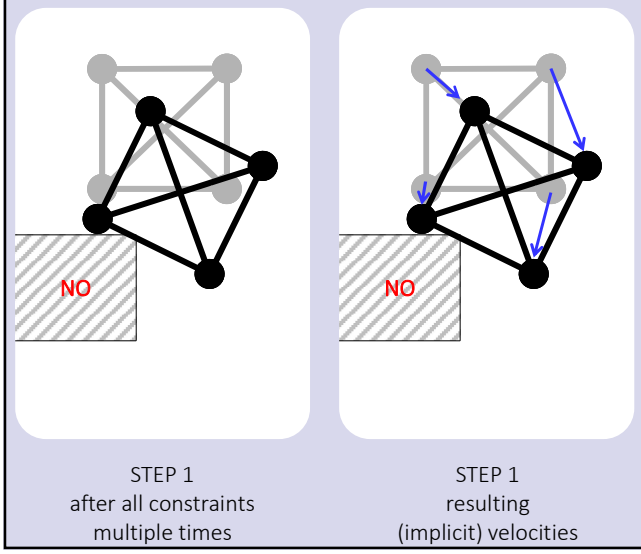
117

Example



118

Example



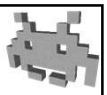
In total: the "box",
under gravity + collision
- had rotated
- gained angular velocity
  (will keep rotating by inertia)

even the system does not (explicitly) handle rotations or
angular velocities

(works in 3D as well!)

119

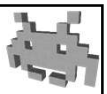
Enforcing a positional constraint: the general case.



- Check: does the equation/inequality hold?
- If so, nothing to do!
- Else:
  - All positions must be displaced a bit, so that it does
  - Infinite ways to achieve this. Which one to pick?
  - Answer:
    minimize the sum of *squared* displacements
    (with respect to current position)
    weighted by particle masses
  - Find the minimizer by analytically
    solving simple math problems
    ("analytically" = in closed form = "on paper")

120

Enforcing positional constraints in the general case: formal problem definition



- We want to enforce a constraint $\mathcal{C}$ on particles a , b , c,…
  with have mass m_a, m_b, m_c …
- $\mathcal{C}$ defined as an equation/inequality of their positions $\mathbf{p}_\mathrm{a}$, $\mathbf{p}_\mathrm{b}$, $\mathbf{p}_\mathrm{c}$, …
- We must apply the displacements $\overrightarrow{d_\mathrm{a}}$, $\overrightarrow{d_\mathrm{b}}$, $\overrightarrow{d_\mathrm{c}}$
  which minimize:

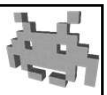
$$\underset{\overrightarrow{d_\mathrm{a}},\overrightarrow{d_\mathrm{b}},\overrightarrow{d_\mathrm{c}},\ldots}{\operatorname{argmin}} \left(m_\mathrm{a}\left\|\overrightarrow{d_\mathrm{a}}\right\|^2 + m_\mathrm{b}\left\|\overrightarrow{d_\mathrm{b}}\right\|^2 + m_\mathrm{c}\left\|\overrightarrow{d_\mathrm{c}}\right\|^2 + \cdots \right)$$

$$\text{such that } \mathcal{C}\left(\mathbf{p}_\mathrm{a} + \overrightarrow{d_\mathrm{a}} \ , \mathbf{p}_\mathrm{b} + \overrightarrow{d_\mathrm{b}} \ , \mathbf{p}_\mathrm{c} + \overrightarrow{d_\mathrm{c}} \ , \ldots\right)$$

among all the choices that satisfy this,
we want the one which minimizes this

121

Example: the equidistance constraint



- To enforce the constraint
  "particles a and b must stay at distance k"
  - input: current positions $\mathbf{p}_a$, $\mathbf{p}_b$
  - input: masses m_a, m_b
- We need to the the displacements $\overrightarrow{d_a}$, $\overrightarrow{d_b}$ found by minimizing:

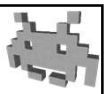
$$\underset{\overrightarrow{d_a},\overrightarrow{d_b}}{\operatorname{argmin}} \left(m_a \left\| \overrightarrow{d_a} \right\|^2 + m_b \left\| \overrightarrow{d_b} \right\|^2 \right)$$

$$\text{such that } \left\| \left(\mathbf{p}_a + \overrightarrow{d_a} \right) - \left(\mathbf{p}_b + \overrightarrow{d_b} \right) \right\| = k$$

- And the solution (in closed form) is...

122

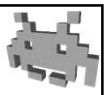
Equidistance constraints: solution for non-equal masses



```
Vector3 pa, pb; // curr positions of a,b
float ma, mb;   // masses of a,b
float d;        // distance (to enforce)

Vector3 v = pa - pb;
float currDist = v.length;

v /= currDist;  // normalization of v

float delta = currDist - d ;

/* solutions of the minimization: */
pa += ( mb/(ma+mb) * delta) * v;
pb -= ( ma/(ma+mb) * delta) * v;
```

123

Positional constraint example: "please don't sink under a plane"



- We want to enforce the constraint
  "particle a must be above a given constant plane "
  - Given: position of the particle $\mathbf{p}_a$ and its mass m_a
  - Point on a plane $\mathbf{p}_q$ and its normal (unit vec) $\hat{n}_q$
- We need to apply the displacement $\overrightarrow{d_a}$ found by minimizing:

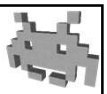
$$\underset{\overrightarrow{d_a},\overrightarrow{d_b}}{\operatorname{argmin}} \left(m_a \left\| \overrightarrow{d_a} \right\|^2 \right)$$

$$\text{such that } \left\| (\mathbf{p}_a - \mathbf{p}_q) \cdot \hat{n}_q \right\| > 0$$

- And the solution (in closed form) is, trivially…

124

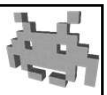
In pseudocode



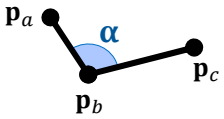
```
Vector3 pa; // curr positions of a
float ma;   // mass (no effect here)
Vector3 pq; // point on the plane
Vector3 nq; // normal of the plane (unit)

Vector3 v = pa - pq;
float currDist = Vector3.dot( v , n );

if (currDist < 0.0)
   pa -= currDist * n; // just project!
else {} // constrain holds, do nothing
```

125

More examples of positional constraints

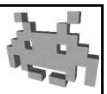


- Preserve volume of some object: *«Volume is v_C »*
  - How to impose it:
  1. Estimate current total volume v
  2. uniform scaling of the entire object of $\sqrt[3]{v_C / v}$
- Fixed positions: *«particle a stays in $\mathbf{p}_a$ »*
  - particles «pinned in position»
  - trivial to impose, but useful!
- Angle constraints, e.g. $\alpha < \alpha_{max}$
  - e.g. on joints: *«elbows cannot bend backward»*
- Coplanarity / collinearity
- Non interpenetration
  - this is part of collision handling – see collisions later



126

Rigid objects as compounds of constrained particles: advantages



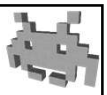
- Interesting/rich/useful set of "emerging behaviors" (i.e. effects with "just automatically happens") :
  - rigid, deformable, jointed objects
    - made of particles + hard constraints
  - their angular velocities
    - rotation around proper axis
  - their barycenter
  - their momentum of inertia
    - angular velocity is maintained
  - somewhat believable bounces on "impacts"
    - for more control: impact impulses can be added (see collisions)

you don't need to compute or store these

consequence of constraints disallowing compene-tration

127

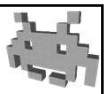
Particles + constraint, or rigid bodies?



- Rigid-body based systems:
  - explicitly compute dynamics for rigid bodies
  - updating their rotation, angular speed,…
- Particles-based systems:
  - only compute dynamics for particles
  - rigid (or deformable, or jointed) bodies as an emerging behavior
- Mixed systems:
  - use both
  - may even dynamically swap between the two representations for rigid bodies

129

Rigid body as particles + constraints: Challenges



- Approximations are introduced
  - e.g.: mass is concentrated in a few locations
- Scalability issues
  - many constraints to enforce, many particles to track
- Some of the info which is kept *implicit* is needed by the rest of the game engine
  - and must therefore be extracted ☹
  - example: the transform (position + orientation) of the "rigid body" is needed to render the associated mesh
  - similarly: angular speed, barycenter pos, velocity…

130