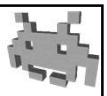
Course Plan

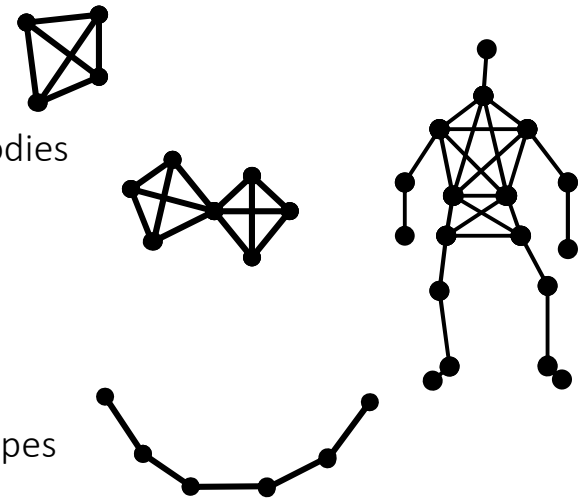


lec. 1: Introduction
lec. 2: Mathematics for 3D Games
lec. 3: Scene Graph
lec. 4: Game 3D Physics +
lec. 5: Game Particle Systems
lec. 6: Game 3D Models
lec. 7: Game Textures
lec. 8: Game 3D Animations
lec. 9: Game 3D Audio
lec. 10: Networking for 3D Games
lec. 11: Artificial Intelligence for 3D Games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

123

We can combine equidistance constraints to obtain…

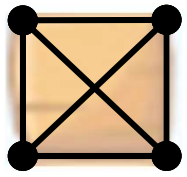
- Rigid bodies
- Articulated bodies
- Ragdolls
- Cloth
- Non-elastic ropes
- …and more



124

Combining equidistance constraints we obtain rigid objects

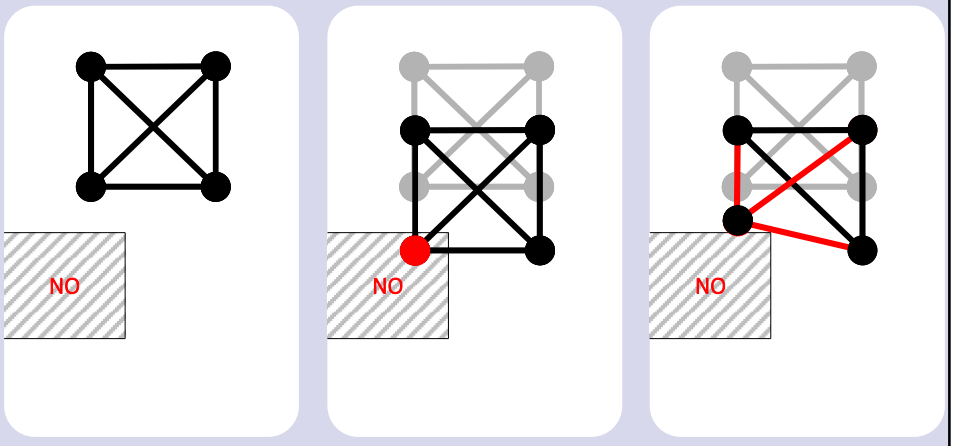
- Rigid body dynamics as emerging behavior
  - without explicitly keeping track their orientation, angular vel, angular acc., etc.



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

125

Example



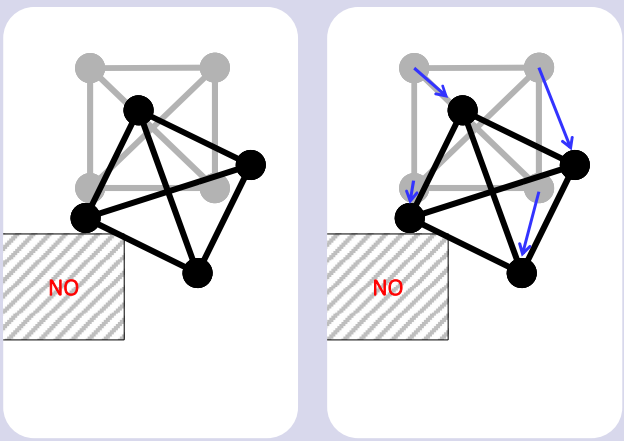
FRAME 0

FRAME 1
before constraints

FRAME 1
after 1st constraint

126

Example



FRAME 1
after all constraints
multiple times

FRAME 1
resulting
(implicit) velocities

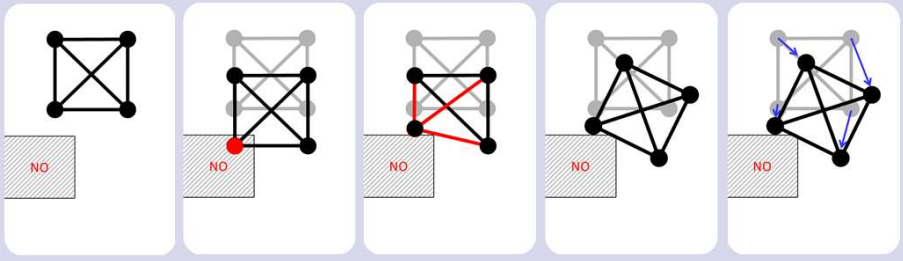
In total: the "box",
under gravity + collision
- had rotated
- gained angular velocity
  (will keep rotating by inertia)

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

(works in 3D as well!)

127

Example



FRAME 0

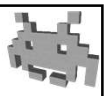
FRAME 1
dynamics

FRAME 1
constraints solving
(repeat until
convergence)

FRAME 1
final
-
implicit velocities
shown in blue

128

Enforcing a positional constraint: the general case with masses.



- Check: does the equality/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")

129

Enforcing positional constraints in the general case: formal problem definition

- We want to enforce a constraint $\mathcal{C}$ on particles $\mathrm{a}, \mathrm{b}, \mathrm{c}, \ldots$
  in positions $\mathbf{p}_\mathrm{a}, \mathbf{p}_\mathrm{b}, \mathbf{p}_\mathrm{c}$ and with masses $m_\mathrm{a}, m_\mathrm{b}, m_\mathrm{c}, \ldots$
- $\mathcal{C}$ defined as an equality/inequality of $\mathbf{p}_\mathrm{a}, \mathbf{p}_\mathrm{b}, \mathbf{p}_\mathrm{c}, \ldots$:

$$\mathcal{C}: (\mathbf{p}_\mathrm{a}, \mathbf{p}_\mathrm{b}, \mathbf{p}_\mathrm{c}, \ldots) \rightarrow \{ true, false \}$$

- We must apply the displacements $\vec{d_\mathrm{a}}$, $\vec{d_\mathrm{b}}$, $\vec{d_\mathrm{c}}$ found by:

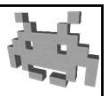
$$\underset{\vec{d_\mathrm{a}}, \vec{d_\mathrm{b}}, \vec{d_\mathrm{c}}, \ldots}{\operatorname{argmin}} \left(m_\mathrm{a} \left\| \vec{d_\mathrm{a}} \right\|^2 + m_\mathrm{b} \left\| \vec{d_\mathrm{b}} \right\|^2 + m_\mathrm{c} \left\| \vec{d_\mathrm{c}} \right\|^2 + \cdots \right)$$

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

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

130

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...

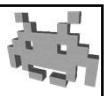
131

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;
```

132

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…

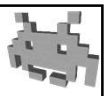
133

In pseudocode

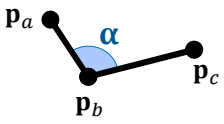
```
Vector3 pa; // curr positions of a
float ma;   // mass (no effect here)
Vector3 pq; // point on the plane
Vector3 nq; // normal of the plane (unit)

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

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

134

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. $\boldsymbol{\alpha} < \boldsymbol{\alpha}_{\max}$
  - e.g. on joints: *«elbows cannot bend backward»*
- Coplanarity / collinearity
- Non interpenetration
  - this is part of collision handling – see collisions later



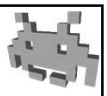
135

Position Based Dynamics (PBD)

- A set of ideas for computing physics (dynamics)
- Ingredients:
  1. Use Verlet integration
     - velocity is implicit
     - changes in positions induce changes in velocity
  2. Implement positional constraints on particles (e.g., equidistance hard constraint) to model:
     - Rigid bodies
     - Articulated bodies
     - Impacts (maybe, add collision impulses, see later)

136

Rigid-bodies as compounds of particles + constraints



- Interesting/rich/useful set of "emerging behaviors" (they just automatically happen) :
  - 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

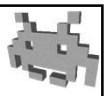
137

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…

138

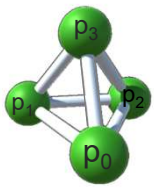
Particles + constraint, or rigid bodies?

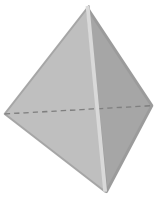


- Rigid-body based systems:
  - explicitly compute dynamics for rigid bodies
  - also store their current orientation + angular velocity
  - update them (just like position + velocity)
- Particles-based systems with PBD:
  - only compute dynamics for particles
  - rigid (or deformable, or jointed) bodies as an emerging behavior
- Mixed systems:
  - dynamically swap between the two representations for rigid bodies
  - how to pass from one to the other?

139

Examples: how to extract...





Particle Compound

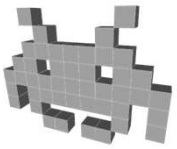
Rigid Body

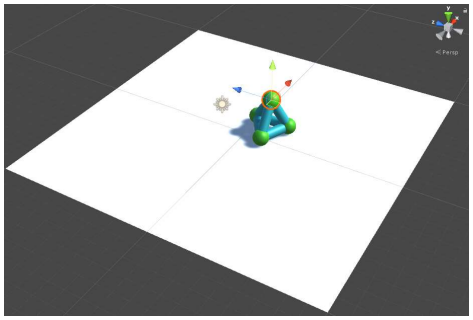
STATIC (on init)	masses $m_0...m_3$ initial positions $r_0...r_3$	? →	mass m barycenter b moment of inertia I (matrix)
DYNAMIC (during sim.)	positions $p_0...p_3$ velocities $v_0...v_3$	? → ← ?	position p (of barycenter) velocity v rotation (i.e. orientation) R angular velocity a

140

3D video games

notes on the sand-box coding done in class

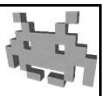




Marco Tarini

144

Objective of this sandbox



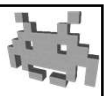
Implement a PBD system

(particle based, with Verlet integration) on Unity

- Plan:
  - do NOT enable default Unity physics system (physX)
  - instead, implement our ad-hoc physics "by hand", with scripts
  - *note*: in a normal project, there's no good reason to do this!
- How to NOT enable physics in Unity:
  - Just don't add, to any GameObject, (or remove them) any "RigidBody" component (implements *dynamics*) and any "Collider" component (implements *collision handling*)
- we will still use the normal Unity Graphics engine
  - scene-graph support: GameObjects, their Transforms

145

Background: "behaviors" in Unity



- In Unity, a behavior is a script associated to a Game Object
- It is a C# class, with predefined methods used by the rest of Unity engine:
  - Start() – called at start at before the first rendering
  - FixedUpdate() – called at fixed interval, just before the hard-wired physics step
  - Update() – called before rendering this object
- The value *dt* is exposed as Time.FixedDeltaTime

For details on methods used in this sandbox, refer to the implementation on the website!

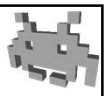
146

Our Particles and their behavior

- Our particle is a game object
  - rendered as a small sphere
- Its associated behavior class includes the fields:
  - pOld (point): for Verlet dynamics ("transform.position" is the current position)
  - mass (scalar): constants (public, so it is exposed in the GUI)
  - similarly, we could also add drag (another scalar)
- and the methods:
  - Start(): initializes Verlet
  - FixedUpdate(): performs a Verlet integration step

147

Implementation detail: pNow VS transform.position



- For each particle, the current position is already stored as its transform.position :
  - Technically, it's the translation/position component of the global transformation
  - It's not really a field, but pretends to be (C# property)
  - Remember: physics always works in world space
- That is used by the rendering engine, the GUI, etc.
- For clarity, we use a local var pNow at the beginning of the Verlet integration step:
  pNow ← transform.position
  at the end:
  transform.position ← pNow

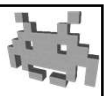
148

Fixed-update of particles

- Basic Verlet integration occurs here
- We may add velocity dumping
  - see dump computation in prev slides
- Includes addition of any force *that depends only on this one particle*
  - Such as gravity
- Includes enforcement of positional constraints *which depend only on this one particle*
  - ground collision ("please stay above ground")
  - box collision ("please stay inside this 10x10 box")

151

Adding "sticks"



- Sticks are GameObjects representing rigid rods connecting two particles
- Rendering (just for the looks):
  - A stick is rendered as a small cylinder
    (a cylinder mesh associated to the Game Object)
  - Before each rendering (so, in the Update() method)
    its transformation is computed anew,
    so that the cylinder is scaled, rotated, and translated
    to make it graphically connect the two particles
  - This new transformation substitutes the old
  - (therefore, it doesn't matter where we place them in the scene: they will teleport to the right location at each frame)

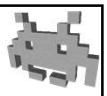
152

Stick effects on physics

- Fields:
  - References to connected particles A and B
    This is a public field: set them in the Unity GUI !
  - Rest length (scalar)
    This is automatically computed on Start
    as the initial distance between particles A and B
- Methods:
  - FixedUpdate: enforces the positional constraints, acting on the position (transform.position) of the two particles
  - See slides for how this is to be computed from their current positions and masses

153

Sand-box project: results.



- Combining multiple particles and sticks, we construct meta-objects such as…
  - Rigid objects
  - TODO: ropes, pendulums
- Rigid objects exhibit a plausible…
  - Angular velocity
  - Angular momentum
  - Corrent barycenter around which to rotate (try assigning a different mass to a particle)
  - Reaction of impacts with the ground / walls (bounces)

without having coded any of that

154

A problem in the current implementation

- We are relying on Unity hard-coded mechanism to run the FixedUpdates (and Start) methods for all scene objects
  - Therefore, we have no control on the order in which they are run
- In particular, the positional constraints of the sticks are run
  - only once per physics step
  - either before, or after the Verlet integration step
- In theory, we want to enforce them
  - just after swapping current and old positions
  - and multiple times, or until convergence
  - together with the collision of particles with ground etc
- Still, the simulation works with only small inconsistencies

155