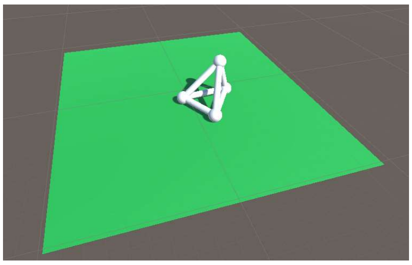
3D video games

notes on the sand-box coding done in class

Marco Tarini



152

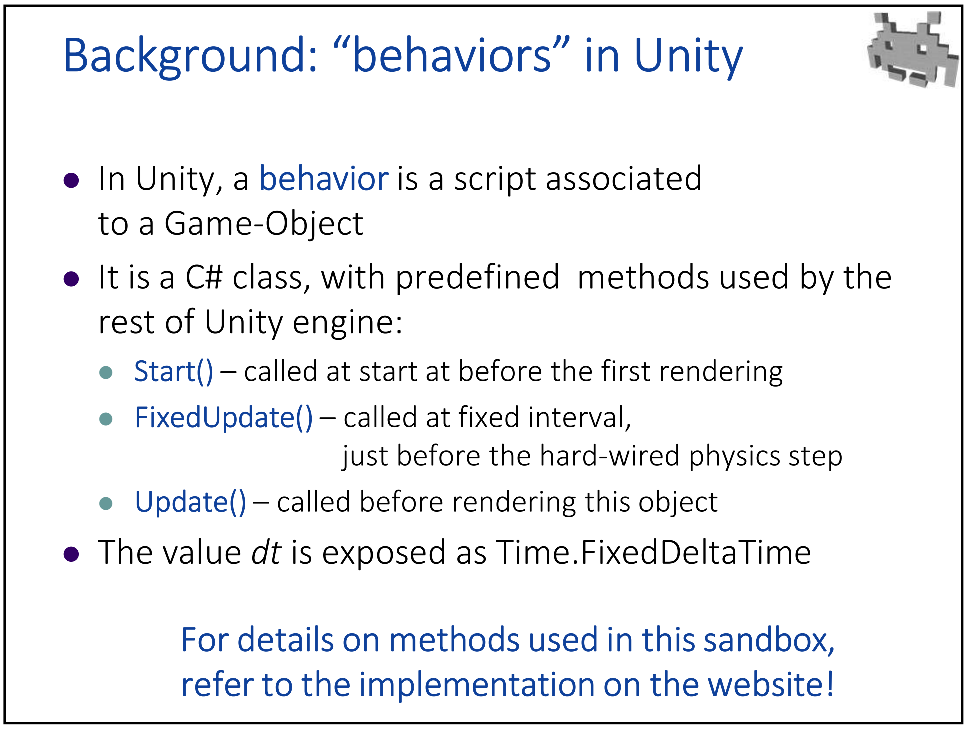
Objective of this sandbox

Implement a PBD system

(particle based, with Verlet integration) on Unity

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

153

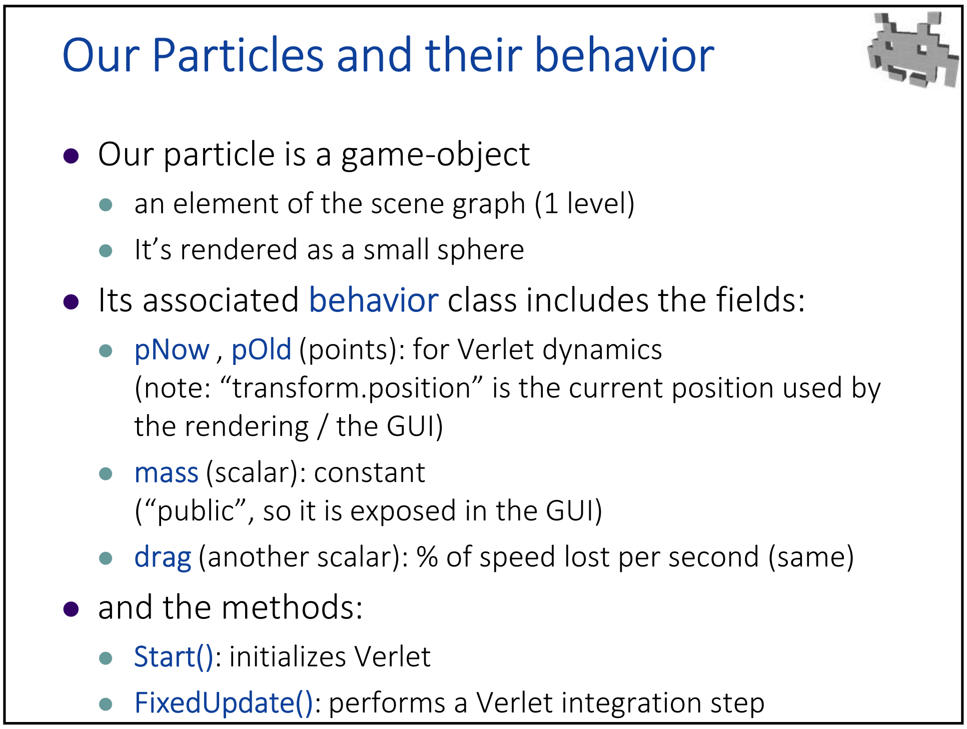


Background: “behaviors” in Unity

- 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!

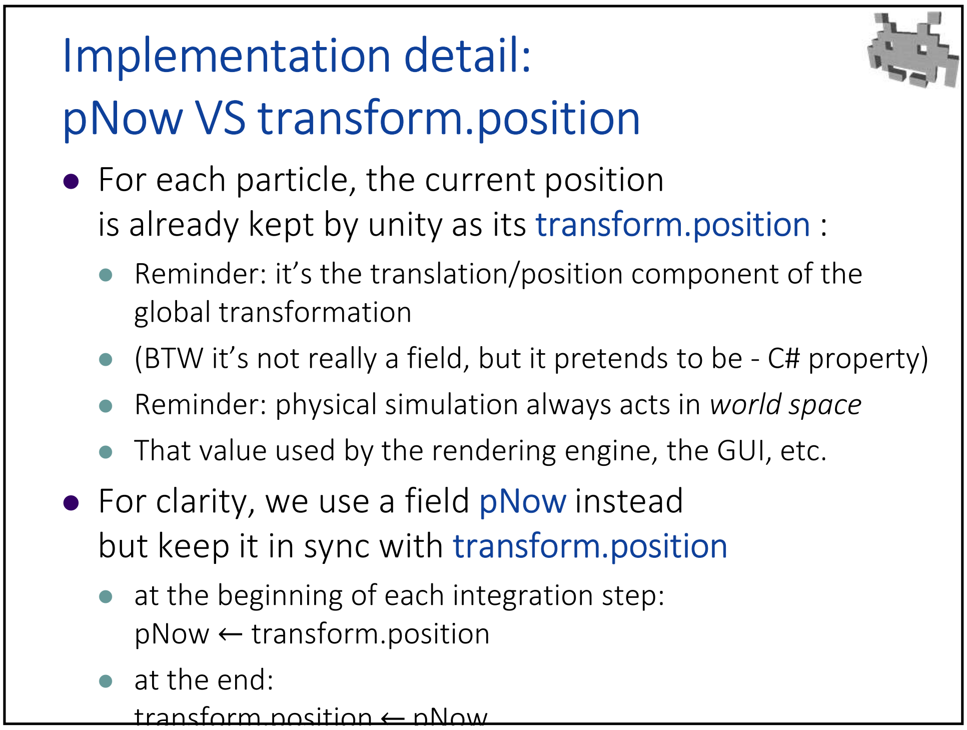
154



Our Particles and their behavior

- Our particle is a game-object
  - an element of the scene graph (1 level)
  - It’s rendered as a small sphere
- Its associated behavior class includes the fields:
  - pNow , pOld (points): for Verlet dynamics (note: “transform.position” is the current position used by the rendering / the GUI)
  - mass (scalar): constant (“public”, so it is exposed in the GUI)
  - drag (another scalar): % of speed lost per second (same)
- and the methods:
  - Start(): initializes Verlet
  - FixedUpdate(): performs a Verlet integration step

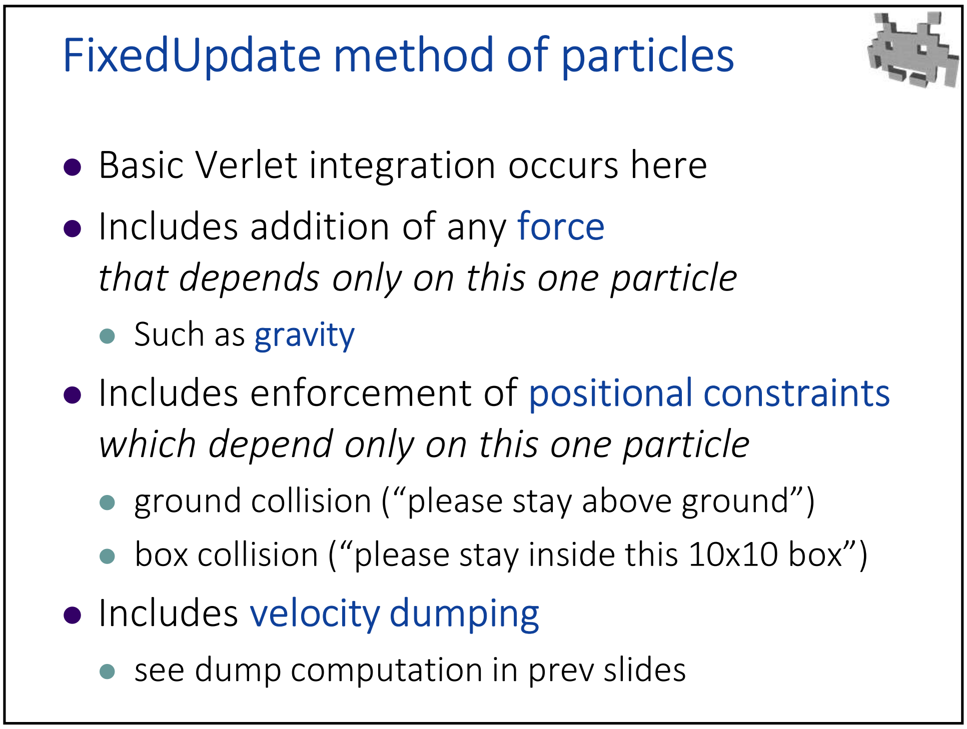
155



Implementation detail: pNow VS transform.position

- For each particle, the current position is already kept by unity as its transform.position :
  - Reminder: it's the translation/position component of the global transformation
  - (BTW it's not really a field, but it pretends to be - C# property)
  - Reminder: physical simulation always acts in *world space*
  - That value used by the rendering engine, the GUI, etc.
- For clarity, we use a field pNow instead but keep it in sync with transform.position
  - at the beginning of each integration step:
    pNow ← transform.position
  - at the end:
    transform.position ← pNow

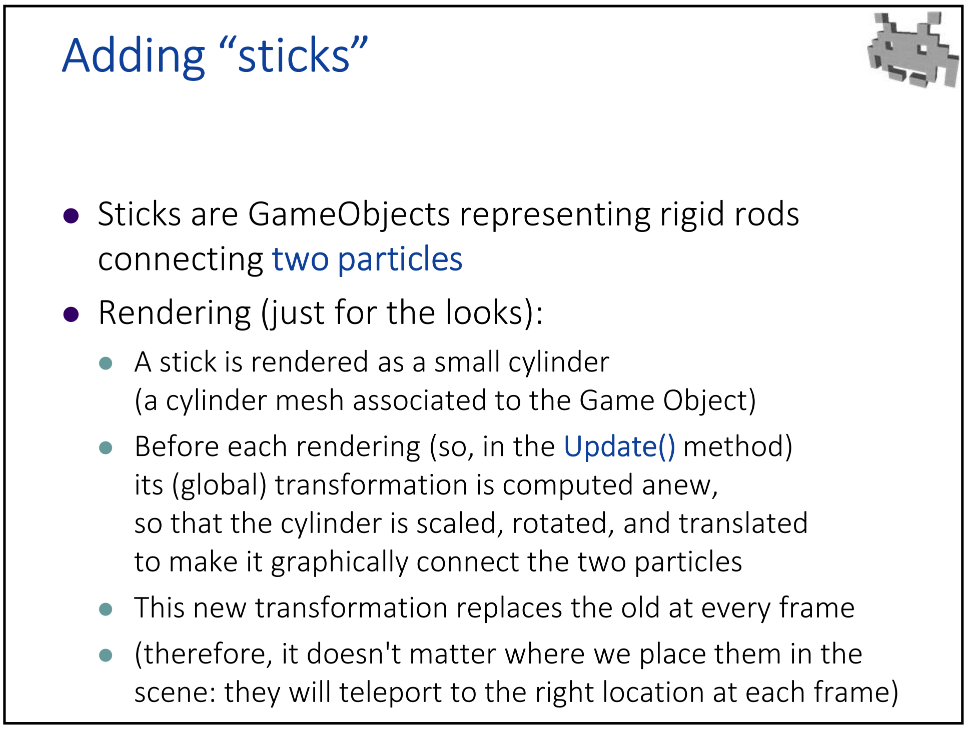
156



FixedUpdate method of particles

- Basic Verlet integration occurs here
- 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")
- Includes velocity dumping
  - see dump computation in prev slides

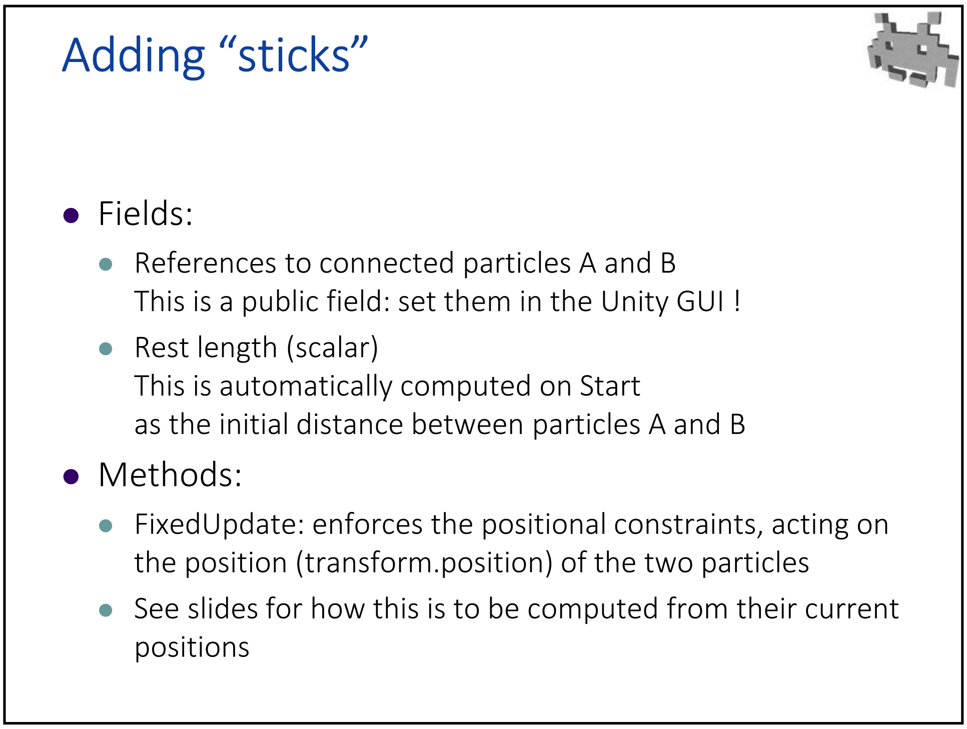
159



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 (global) transformation is computed anew,
    so that the cylinder is scaled, rotated, and translated
    to make it graphically connect the two particles
  - This new transformation replaces the old at every frame
  - (therefore, it doesn't matter where we place them in the scene: they will teleport to the right location at each frame)

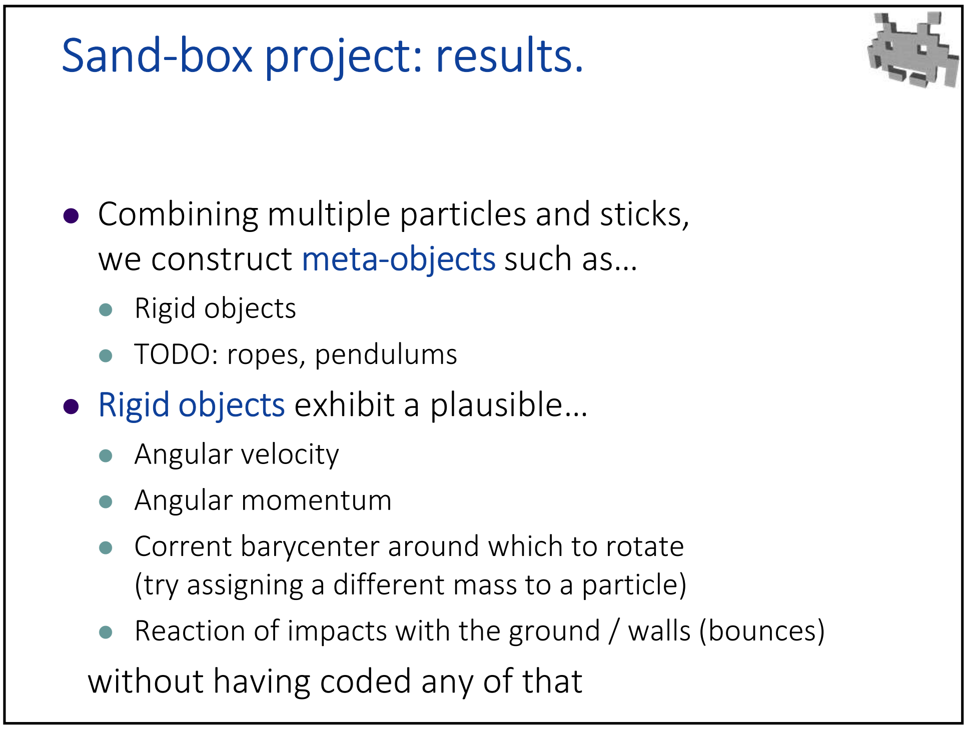
160



Adding “sticks”

- 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

161

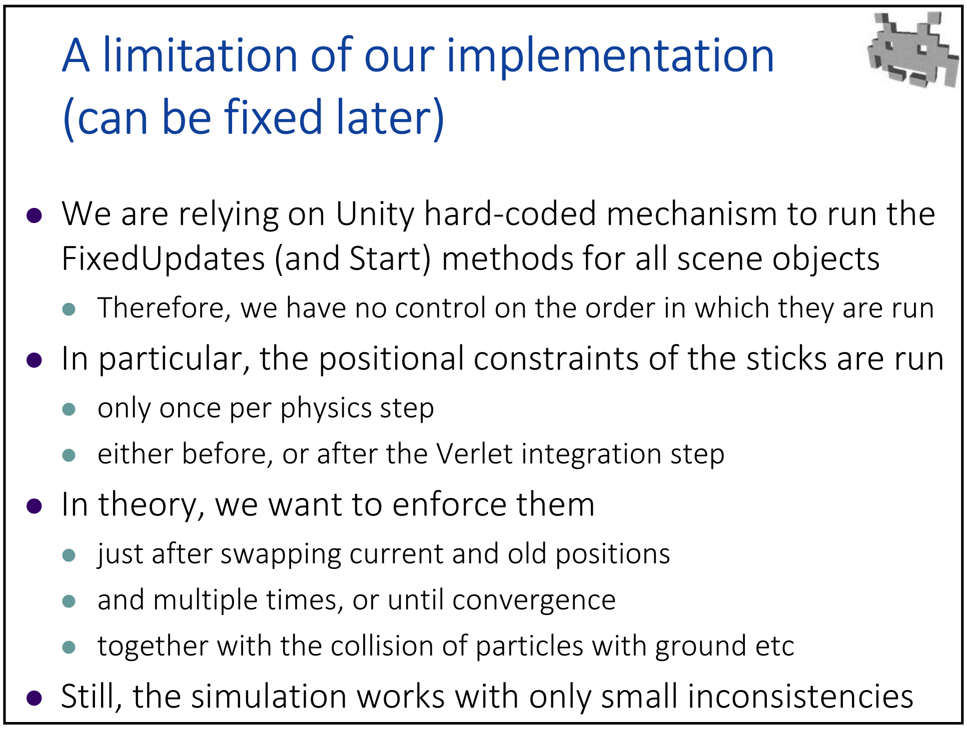


Sand-box project: results.

- 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

162



A limitation of our implementation (can be fixed later)

- 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

163