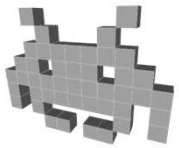
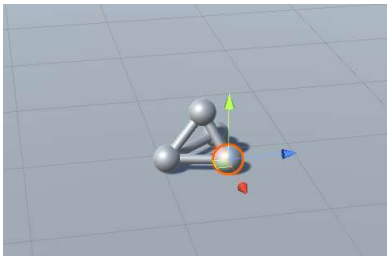
3D video games

notes on the coding done in class

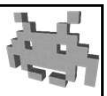


Marco Tarini



131

Objective of this sandbox

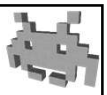


Implement a simple Verlet based, PBD physics system on Unity

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

132

Background: "behaviors" in Unity

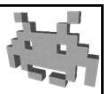


- In Unity, a behavior is a script associated to a Game Object
- It is a C# class, with predefined methods used by the resto of Unity engine:
  - Start() – called at start at before the first rendering
  - FixedUpdate() – called for each fixed step
  - Update() – called before rendering this object
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

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

133

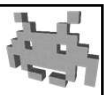
Particles and Particle behavior



- Our particle is a game object
  - rendered as a small sphere
- Its associated behavior includes the fields:
  - pNow, pOld (points): for Verlet dynamics (pNow is the current position)
  - mass, drag (scalars): constants (exposed to the interface)
- and the methods:
  - Start(): initializes Verlet
  - FixedUpdate(): performs a Verlet integration step

134

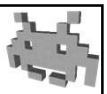
Implementation detail: pNow VS transform.position



- For each particle, the current positions is stored twice:
  - The position according to our custom physics engine: pNow – a custom field in the "behavior" of the particle
  - rendering position: the position used by the rendering engine transform.position, i.e. the position Unity uses for everything
- We keep them separated, just for code clarity
- At the beginning (start method)
  - physic position ← rendering position
    (so that the objects starts where we placed them in the GUI)
- Before each rendering (update method)
  - rendering position ← physic position
    (so that the object is rendered where the physics moved it)

135

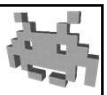
Implementation detail: pNow VS transform.position



- When to synchronize graphics (transform.position) and our physics (pNow)?
- Best solution we found in class:
  - In method `Start()`: pNow ← transform.position
    - This makes the physics be initialized with the position set from the Unity GUI
  - At end of `FixedUpdate()`: transform.position ← pNow
    - this makes the engine show the particle on screen at its' correct physics position
  - In `LateUpdate()`: pNow ← transform.position
    - this allows us to control the (physics) position from the GUI
    - Observe: because it's Verlet, by changing the position we control also the velocity. E.g. we can "toss" objects

136

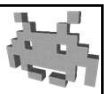
Fixed-update of particles



- Basic Verlet integration occurs here
- Includes velocity dumping
  - see dump computation
- Includes addition of forces
  which depend only on this one particle
  - Such as gravity
- Includes enforcement of positional constraints
  which depend only on this one particle
  - Such as ground collision (“please stay above ground”)

137

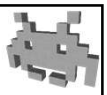
Adding sticks



- Sticks are GameObjects representing rigid rods connecting two particles
- Rendering:
  - A rod is rendered as a small cylinder
    (a cylinder mesh associated to the Game Object)
  - Before each rendering (update method)
    a transformation is computed so that the cylinder is scaled (on Y only), rotated, and translated
    to make it graphically connect the two particles
  - (therefore, it doesn't matter where we place them in the scene: they will teleport to the right location at each frame)

138

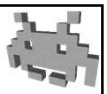
Stick behavior



- Fields:
  - Connected particles A and B
    It's a public field: set them in the Unity GUI !
  - Rest length (computed on Start as the initial distance between particles A and B)
- Methods:
  - FixedUpdate: enforce the positional constraints, acting on the position of the two particles
  - EnforcePositionalConstraint: self explainatory
    Note: this take in account correctly of their mass

139

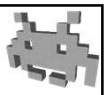
Sand box: results.



- Combining multiple particles and rods, we construct meta-objects such as…
  - Rigid objects
  - TODO: ropes, pendulums
- Observe: rigid objects behave correctly, with plausible…
  - Effect of impact with the ground
  - Angular velocity
  - Angular momentum
  - Barycenter (try assigning a different mass to a rigid)

140

Adding positional constraint: stay "fixed"

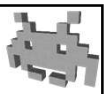


- A Particle can simply "be asked" to stay fixed
- How-to notes:
  - Add a public Boolean field isFixed
  - Add the Vector3 field fixedPos , the pos where this particle is fixed in the scene (initialize it on Start on the fixed position)
  - Trivially impose the constraint in the mothod()
- Small hack:
  - fixedPos is also updated at every frame, as the current rendering position
  - (so that we can move this particle from the GUI)

141

A problem and a fix

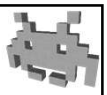
(fix was done in the minutes after the official end of the lecture)



- The problem:
  - In Position Based Dynamics, all positional constraints must be solved multiple times per frame, on cascade
  - In our code, the constraints are enforced by EnforceConstraints methods of to Particles and Sticks, and run in the FixedUpdate methods
  - Therefore: we enforce them only once (plus, we don't know in which order!)
  - Result: the simulation was a little unstable (object won't stop moving, etc)

142

A problem and a fix



(fix was done in the minutes after the official end of the lecture)

- The solution: (this is typical in Unity programming)
  - Create a global empty GameObject ("global script holder")
  - Associate a global script to it.
  - In its "fixedUpdate", this scripts enforces all constraints
    - multiple times
    - and in a order which we can control
  - To enforce all constraints:
    - Remove execution of "enforceConstraint" from the FixedUpdate particle and sticks scripts
    - Make these methods public!
    - In the global script, find all particles and all sticks in the scene
    - For each one, call its "enforceConstraint" method

143