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

171

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

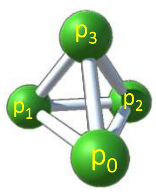
172

Rigid-body as particles + constraints: challenges

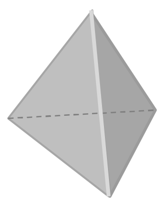
- Approximations are introduced
  - e.g.: mass is concentrated in a few locations
- Scalability issues
  - many constraints to enforce, many particles to track
- Some of the info which is kept *implicit* is needed by the rest of the game engine
  - and must therefore be extracted ☹
  - mainly: the transform (position + orientation) of the "rigid body" is needed to render the associated meshes
  - or: velocity, angular velocity may be needed for… gameplay reasons (e.g. damage), graphics (motion blur), etc

173

How to extract…



Particle Compound



Virutal Rigid Body

	Particle Compound		Virutal Rigid Body	
STATIC (initialization)	masses $m_0 \ldots m_N$ initial positions $r_0 \ldots r_N$	? →	mass m barycenter b (local space origin) moment of inertia I (matrix)	STATIC
DYNAMIC (during simulation)	positions $p_0 \ldots p_N$ velocities $v_0 \ldots v_N$	? →	position p (of barycenter) velocity v (linear) rotation R (i.e. orientation) angular velocity a	DYNAMIC

175

For example:

	Particle Compound		Rigid Body
STATIC	masses $m_0 \dots m_N$	A	mass m
DYNAMIC	positions $p_0 \dots p_N$	B	position p (of barycenter)
DYNAMIC	velocities $v_0 \dots v_N$	C	velocity v (linear)

A: $m = \sum m_i$

B: $\mathbf{p} = \frac{1}{m} \sum m_i \mathbf{p}_i$

C: $\mathbf{v} = \frac{1}{m} \sum m_i \mathbf{v}_i$

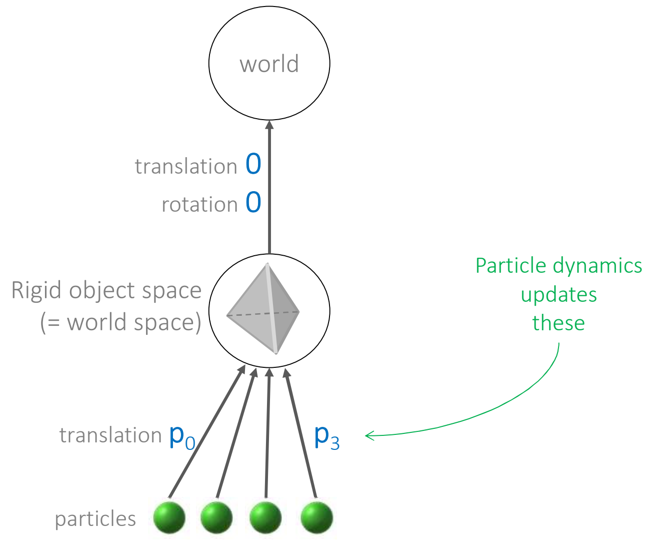
Questions (beyond this course):
How to find / update the...

- rotation
- angular velocity
- moment of inertia matrix

of the rigid body?

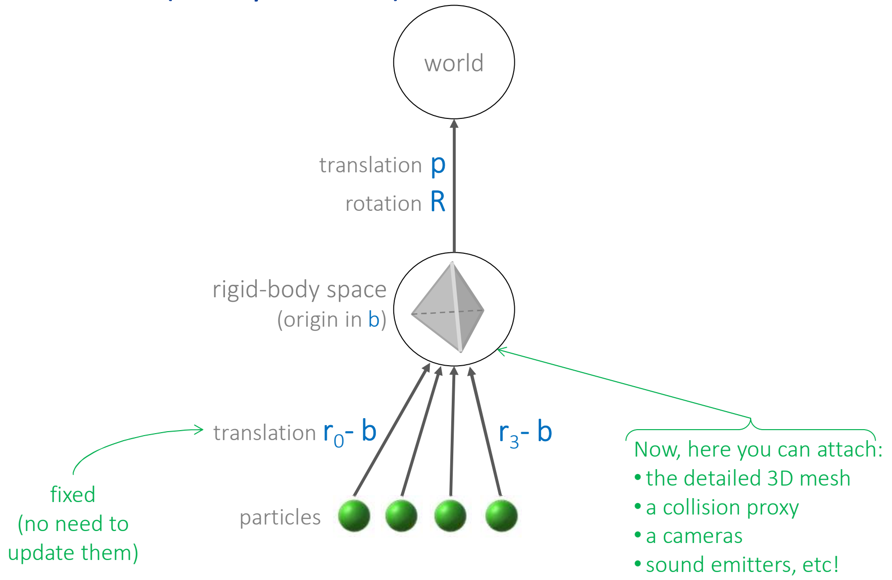
176

Scene-graph interpretation: from this



177

Scene-graph interpretation: to this (every frame!)



178

Summary: two ways to handle rigid-bodies

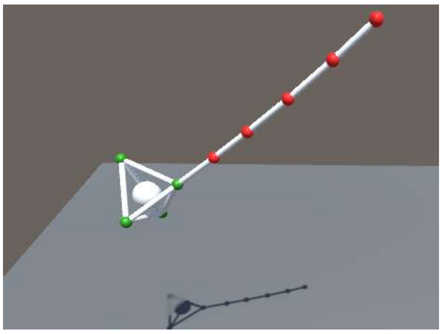
- As a compund of particles, with PBD
  - Bonus: you can also handle… deformable bodies, articulated bodies
  - Bonus: easy to define many useful constraints
  - Cost: need to extract implicit status of the rigid body
  - Cost: mass concentrated at particles
- With Rigid Bodies dynamics
  - With explicit rotation / angular velocity
- Or, mixed systems:
  - Convert (dynamically) between the two

179

3D video games

notes on the sand-box coding done in class

Marco Tarini



181

Objective of this sandbox

Implement a PBD system
(particle based, with Verlet integration) on Unity

- Plan:
  - we will 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

182

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!

183

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:
  - P_now , p_old (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

184

Implementation detail: p_now 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 p_now instead but keep it in sync with transform.position
  - at the beginning of each integration step: p_now ← transform.position
  - at the end: transform.position ← p_now

185

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

188

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)

189

Adding "sticks"

- Fields:
  - References to connected particles A and B
    This is a public field: so we will 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

190

Adding a visible barycenter to the virtual object

- BaricenterOf:
  A "behavior" that just teleports its object (a white sphere) in the barycenter of a given compound of particles.
- Fields:
  - References to all particles of the compoundnd B
- Methods:
  - Update: computes the current barycenter and teleport the white ball there
- note:
  - we can get away with the rotation because the sphere is rotationally symmetric
  - How would we compute the "rotation"

191

Sand-box project: results.

- Combining multiple particles and sticks, we construct meta-objects such as…
  - Rigid objects
  - Ropes, pendulums
- Rigid objects exhibit a plausible…
  - Angular velocity
  - Angular momentum
  - Correct barycenter around which to rotate (try assigning a different mass to a particle)
  - Stability (does the barycenter "fall inside the basis"?)
  - Reaction of impacts with the ground / walls (bounces)

…without having coded any of that

192

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

193

Future work: Idea for how to progress 1/3

- Current problem:
  - Each positional constraint is enforced only once per frame
- Fix it: make a global "behavior"
  - Associated to the root of the scene
  - instead of relying on Unity to execute fixed updates of every object, use only the fixed update of the global behavior, making a sequence of loops:
    - 1st loop: execute Verlet integration (loop over all particles)
    - 2nd loop: enforce all positional constraints (loop over all particle *and* over all rods in the scene)
    - Repeat 2nd loop multiple times

200

Future work: Idea for how to progress: 2/3

- Add springs
- How to: add spring object (similar to rods)
  - 1. Rest length: computed at start (like for rods)
  - 2. Particles at the extremes: a public field, just as for rods
  - 3. Elastic constant *k*: a (public) scalar parameters
  - 4. Write fixed update(): add to forces of the two particles
  - 5. Profit! Add spring to your compound meta-objects
- Caveats:
  - Unless you use a global script, you will need to set forces to 0 (InitForces method) at the *end* of the FixedUpdate (not the beginning) and at initialization (why?)

201

Future work: Idea for how to progress: 3/3

- Floor is lava (or water)
  - Instead of having a hard-granite floor, make it liquid
- How to:
  - 1. Remove the "stay above ground" constraint
  - 2. Add buoyancy (ita: forza di Archimede) to the particles
    - (as an approximation, you don't need it for the rods or the rigid objects: just the particles)
    - Reminder: buoyancy is an upward force with a magnitude = mass of the submerged volume if it was made of water
    - Math task: compute the volume of the part of sphere (of a given radius) which has $y > 0$
  - 3. Profit! See how object float, or sink
    - (and which parts stays up if they float) – depends on masses and size

202