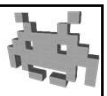
Course Plan



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: 3D Audio for 3D Games
lec. 11: Networking for 3D Games
lec. 12: Artificial Intelligence for 3D Games
lec. 13: Rendering Techniques for 3D Games

144

Positional constraints (in general terms, and more formally)

- A predicate defined on the position(s) of a number of particles (usually, a small number: 1 - 4)

$$\mathcal{C}: (\mathbf{p}_a, \mathbf{p}_b, \mathbf{p}_c, \dots) \rightarrow \{ true, false \}$$

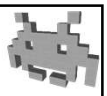
- For example, the equidistance constraints is

$$\mathcal{C}(\mathbf{p}_a, \mathbf{p}_b) \Leftrightarrow \|\mathbf{p}_a - \mathbf{p}_b\| = k_{CONST}$$

- They can be an equality ($=$) or an inequality ($\leq$ or $\geq$)

145

Equality positional constraints: examples



- Equidistance constraint (the one we have seen):
  «these N particles must stay at distance k»
  - E.g: because they are linked by a metal rod of length k
- Fixed positions:
  «this particle must stay in position $\mathbf{p}_a$ »
  - the particle is "pinned" in position
  - trivial to impose, but still useful!
- Coplanarity / collinearity:
  «these N particles must stay on a line / on a plane»

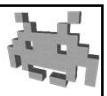
146

Equality positional constraints: other examples

- Volume preservation:
  "The volume delimited by the squishy ballon defined by these particles is a constant k_{CONST} "
  (e.g., because it's filled with water)
- How to impose it (approximation):
  1. Estimate current total volume v
  2. uniform scale the entire object by factor $\sqrt[3]{f_{\text{CONST}}/v}$ around its barycenter

147

Inequality positional constraints: example



- "please don't sink below the ground"
  assuming the ground is the plane Y = 0

$$C(\mathbf{p}_a) \Leftrightarrow \mathbf{p}_a.y \geq 0$$

- Trivial to impose:
  just set the y to 0, if it is < 0

148

Inequality positional constraints: example

- "this particle must stay above
  this fixed (and arbitrary) plane"
  - For example, because the plane is a solid unmovable slab
  - The plane is given by a point on it $\mathbf{p}_q$ and its normal $\hat{n}_q$

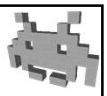
$$C(\mathbf{p}_a) \Leftrightarrow (\mathbf{p}_a - \mathbf{p}_q) \cdot \hat{n}_q \geq 0$$

- To enforce it (when it's not already true)

$$(\mathbf{p}_a - \mathbf{p}_q) \cdot \hat{n}_q = 0$$

149

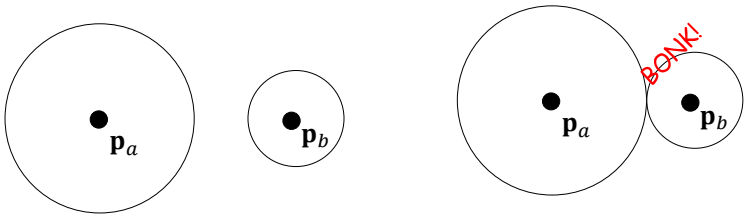
Inequality positional constraints: example



- These two particles must be at least k_{CONST} apart

$$\mathcal{C}(\mathbf{p}_a, \mathbf{p}_b) \Leftrightarrow \|\mathbf{p}_a - \mathbf{p}_b\| \geq k_{CONST}$$

  - For example, because they are the centers of two rigid spheres and k_{CONST} is the sum of their radii
  - part of "collision handling" (a topic for later)



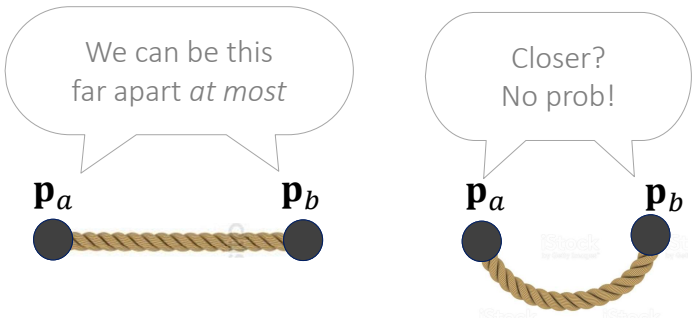
150

Inequality positional constraints: example

- These two particles must be at most k_{CONST} apart

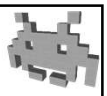
$$\mathcal{C}(\mathbf{p}_a, \mathbf{p}_b) \Leftrightarrow \|\mathbf{p}_a - \mathbf{p}_b\| \leq k_{CONST}$$

  - For example, because they are tied by an inextensible rope that has length k_{CONST} (but can fold)

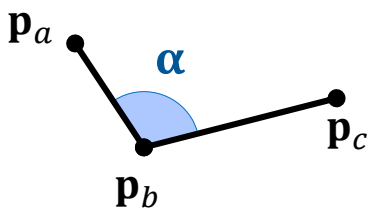


151

Inequality positional constraints: example



- Angle constraints, e.g. $\alpha < \alpha_{max}$
  with α the angle between $\mathbf{p}_a, \mathbf{p}_b$ and $\mathbf{p}_b, \mathbf{p}_c$
  - e.g., on joints: *«elbows cannot bend backward»*
  - (a constraint between three particles!)



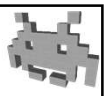
152

Enforcing one positional constraint (in general terms)

- Inequality constraint:
  1. *Test:* does the inequality already hold?
  2. If so: do nothing
  3. If not: enforce it as an equality (=) instead (see below)
- Equality constraint:
  - All involved particles must be displaced from that current position, so that it now holds
  - There can be infinite ways to achieve this!
    Question: Which one to pick?

153

Enforcing one equality constraint:
(assuming for now all particles have same mass)



- Answer:
  minimize the sum of *squared* displacements
  (with respect to current position)
- For each kind of constraint, we need to find the minimizer analytically
  ("analytically" = in closed form = "with formulas"
  = "solving a simple math problem on paper")
  - That's what we did for the equality constraint

154

Enforcing one equality constraint
(assuming for now all particles have same mass)

- We want to enforce a constraint $\mathcal{C}$ on particles $\mathrm{a}, \mathrm{b}, \mathrm{c}, \ldots$
  currently in positions $\mathbf{p}_\mathrm{a}, \mathbf{p}_\mathrm{b}, \mathbf{p}_\mathrm{c}$ …

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

- We must apply the displacements $\vec{d_\mathrm{a}}$, $\vec{d_\mathrm{b}}$, $\vec{d_\mathrm{c}}$ that are the

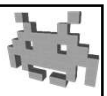
$$\underset{\vec{d_\mathrm{a}}, \vec{d_\mathrm{b}}, \vec{d_\mathrm{c}}, \ldots}{\operatorname{argmin}} \left(\|\vec{d_\mathrm{a}}\|^2 + \|\vec{d_\mathrm{b}}\|^2 + \|\vec{d_\mathrm{c}}\|^2 + \cdots \right)$$

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

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

155

Enforcing one equality constraint
(in general, for particles with different mass)



- 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 $m_\mathrm{a}, m_\mathrm{b}, m_\mathrm{c}, \ldots$

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

- We must apply the displacements $\overrightarrow{d_\mathrm{a}}$, $\overrightarrow{d_\mathrm{b}}$, $\overrightarrow{d_\mathrm{c}}$ which are the:

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

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

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

Example: solve the "please don't sink under this plane"

$$\mathrm{C}(\mathbf{p}_\mathrm{a}) \Leftrightarrow (\mathbf{p}_\mathrm{a} - \mathbf{p}_\mathrm{Q}) \cdot \hat{n}_\mathrm{Q} \geq 0$$

a point on plane (const)
plane normal (const)

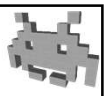
- We need to find displacement $\overrightarrow{d_\mathrm{a}}$ as:

$$\underset{\overrightarrow{d_\mathrm{a}}}{\operatorname{argmin}} \left(m_\mathrm{a} \left\| \overrightarrow{d_\mathrm{a}} \right\|^2 \right)$$

$$\text{such that } \left(\mathbf{p}_\mathrm{a} + \overrightarrow{d_\mathrm{a}} - \mathbf{p}_\mathrm{Q}\right) \cdot \hat{n}_\mathrm{Q} \geq 0$$

- And the solution (in closed form) is…

Enforcing it (in pseudocode)



```
Vector3 pA; // curr positions of a
float mA;   // its mass (no effect in this case)
Vector3 pQ; // point on the plane
Vector3 nQ; // normal of the plane (unitary)

Vector3 v = pA - pQ;
float currDist = Vector3.dot( v , nQ );

if (currDist < 0.0) {
   pA -= currDist * nQ; // project it out!
} else {} // constrain already ok: nothing to do
```

158

Example: the equidistance constraint (for unequal masses)

$$\mathcal{C}(\mathbf{p}_a, \mathbf{p}_b) \Leftrightarrow \|\mathbf{p}_a - \mathbf{p}_b\| = k_{CONST}$$

- With particle masses m_a , m_b
- We need to the displacements $\overrightarrow{\mathrm{d}_a}$, $\overrightarrow{\mathrm{d}_b}$ found by minimizing:

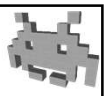
$$\underset{\overrightarrow{d_a}, \overrightarrow{d_b}}{\operatorname{argmin}} \left(m_a \left\|\overrightarrow{\mathrm{d}_a}\right\|^2 + m_b \left\|\overrightarrow{\mathrm{d}_b}\right\|^2 \right)$$

$$\text{such that } \left\|\left(\mathbf{p}_a + \overrightarrow{\mathrm{d}_a}\right) - \left(\mathbf{p}_b + \overrightarrow{\mathrm{d}_b}\right)\right\| = k_{CONST}$$

- And the solution (in closed form) is…

159

Example: the equidistance constraint (for unequal 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;
```

160

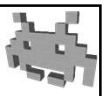
Observe and verify

- The way we have seen last time to impose…
  - The “fixed position” constraint
  - The “equidistance” constraint
  - The “stay above ground” constraint
  - Etc.

  are the ones that minimize the (mass-weighted) squared displacements of the particles
  - (assuming equal mass)
  - (the mass is not always relevant)

161

Position Based Dynamics (PBD) summary



- A general approach for computing dynamics
- Ingredients:
  1. Use Verlet integration **on particles**
     - their velocities are implicit
     - changes in positions induce changes in velocities
  2. Implement positional constraints **on particles** (e.g., equidistance constraint) to model things like:
     - Rigid bodies (their angular speed is an emerging feature!)
     - Articulated / non rigid bodies
     - Basic collision response

162

Not forces: a summary

$$\dots$$
$$\vec{f} = \text{function}(\mathbf{p}_i, \dots)$$
$$\dots$$

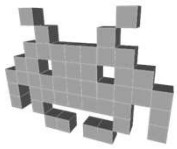
not in here

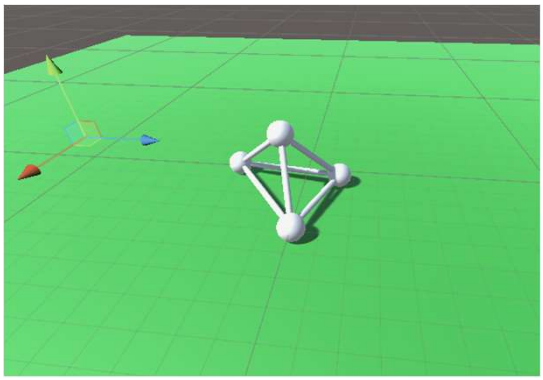
- We have seen many types of real-world forces that are modelled by things that aren't "forces" in our simulation:
  - Frictions
    - *In reality:* a ("dissipative") force contrasting motion
    - Can be simulated by: drag / velocity damp
  - Violent and sudden events, such as impacts
    - *In reality:* a very strong force that is sustained for a very short time << *dt*
    - E.g.: hitting a ball with a baseball bat
    - Must be simulated by: impulses
  - Resistance forces
    - *In reality:* a force that contrast and nullifies an external force (e.g. gravity)
    - E.g.: what prevents your computer from falling through the table RN
    - Can be simulated by: positional constraints

163

3D video games

notes on the sand-box coding done in class





Marco Tarini

174

Objective of this sandbox

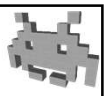
Implement a PBD system

(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 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, etc

175

Background: "behaviors" in Unity



- In Unity, a behavior is a script associated to a Game-Object
- It's a C# class, with predefined methods employed by the rest of the 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 (*if* it is rendered)
- The value *dt* is exposed as Time.FixedDeltaTime

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

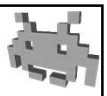
176

Our Particles and their behavior

- Our particle is a game-object
  - an element of the scene graph (1st level)
  - rendered as a small sphere
- Its associated behavior class includes the fields:
  - pOld (a point): for Verlet dynamics
  - mass (scalar): a constant
  - drag (another scalar): % of speed lost per second
- and the methods:
  - Start(): sets pOld
  - FixedUpdate(): is called by unity every physical step, and calls oneStep()
  - oneStep(): performs a Verlet integration step

177

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 is used by the rendering engine, the GUI, etc.
- For clarity, we use a variable pNow instead but we will keep it in sync with transform.position
  - at the beginning of each integration step: pNow ← transform.position
  - at the end: transform.position ← pNow

178

OneStep() method of particles (Verlet integration step)

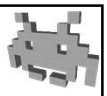
Called by fixedUpdate() (that is, once per frame)

Basic Verlet integration is called here. In it:

- We add forces *that depends only on this one particle*
  - Such as gravity
- We include enforcement of positional constraints *which depend only on this one particle*
  - ground collision ("please stay above ground")
- We include velocity dumping
  - see dump computation in prev slides

181

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)

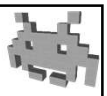
183

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

184

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

185

A limitation of our 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 multiple times, or until convergence
  - together with the collision of particles with ground etc
- Still, the simulation works with only small inconsistencies

186