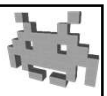
Course Plan

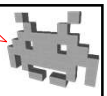


lec. 1: Introduction
lec. 2: Mathematics for 3D Games
lec. 3: Scene Graph
lec. 4: Game 3D Physics + 
lec. 5: Game Particle Systems
lec. 6: Game 3D Models
lec. 7: Game Textures
lec. 8: Game Materials
lec. 9: Game 3D Animations
lec. 10: 3D Audio for 3D Games
lec. 11: Networking for 3D Games
lec. 12: Interactive Agents for 3D Games
lec. 13: Rendering Techniques for 3D Games

156

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

refer to last slide of previous lecture



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

$\mathbf{p}_a$: a point on plane (const); $\hat{n}_Q$: plane normal (const)

- We need to find displacement $\overrightarrow{d_a}$ as:

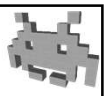
$$\underset{\overrightarrow{d_a}}{\operatorname{argmin}} \left(m_a \left\| \overrightarrow{d_a} \right\|^2 \right)$$
$$\text{such that} \quad \left(\mathbf{p}_a + \overrightarrow{d_a} - \mathbf{p}_Q \right) \cdot \hat{n}_Q \geq 0$$

- And the solution (in closed form) is…

The trivial one we know (see "project a point on a plane")
The point here is that we can see that solution as the minimizer of the function

157

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:

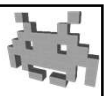
$$\underset{\overrightarrow{d_a}, \overrightarrow{d_b}}{\operatorname{argmin}} \left(m_a \left\| \overrightarrow{\mathrm{d}_a} \right\|^2 + m_b \left\| \overrightarrow{\mathrm{d}_b} \right\|^2 \right)$$

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

- 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

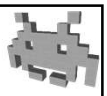
Observe and verify

- The way we have seen last time to impose…
  - The “fixed position” constraint
  - The “equidistance” constraint
  - The “stay above ground” constraint
  - Inequality constraints
    (“don’t do anything if state already valid”)
  - Etc.

  all minimize the (mass-weighted)
  squared displacements of the particles
  - assuming equal mass, when relevant

161

Changing the value of *dt* in Verlet
(whenever it's not constant, for any reason)



Problem:

if dt now changes to a new dt'

then, all $\mathbf{p}_{old}$ must be updated to some $\mathbf{p}'_{old}$

Find $\mathbf{p}'_{old}$:

$$\vec{v} = (\mathbf{p}_{now} - \mathbf{p}_{old})/dt$$
$$\vec{v} = (\mathbf{p}_{now} - \mathbf{p}'_{old})/dt'$$

$$\Longrightarrow$$

current velocity $\vec{v}$ and position $\mathbf{p}_{now}$ must *not* change, so we can only change $\mathbf{p}_{old}$

$$\mathbf{p}'_{old} = \mathbf{p}_{now} \cdot \frac{dt - dt'}{dt} + \mathbf{p}_{old} \cdot \frac{dt'}{dt} = \mathrm{mix}\left(\mathbf{p}_{now}, \mathbf{p}_{old}, \frac{dt'}{dt}\right)$$

162

Simple velocity damping in Verlet

- Velocity at next frame: $\vec{v} = (\mathbf{p}_{next} - \mathbf{p}_{now})/dt$ (implicit)
- We want to multiply $\vec{v}$ a factor c_{DAMP}
  - before applying accelerations

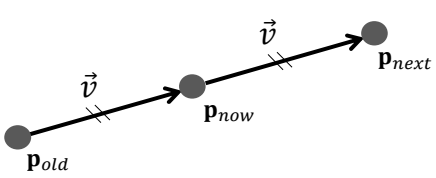
  e.g. 0.98 obtained as $(1\text{-}dt \cdot c_{\mathrm{DRAG}})$
- We can do that using a more general formula for $\mathbf{p}_{next}$

$$\mathbf{p}_{next} = 2 \cdot \mathbf{p}_{now} - 1 \cdot \mathbf{p}_{old} + dt^2 \cdot \vec{a}$$

$$\mathbf{p}_{next} = (1 + c_{\mathrm{DAMP}}) \cdot \mathbf{p}_{now} - c_{\mathrm{damp}} \cdot \mathbf{p}_{old} + dt^2 \cdot \vec{a}$$

164

Simple velocity damping in Verlet (a geometric interpretation)

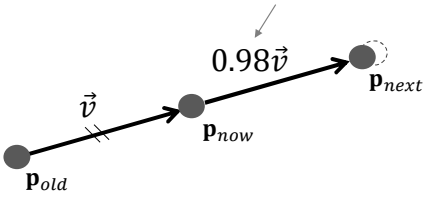


$\mathbf{p}_{next} = 2 \cdot \mathbf{p}_{now} - 1 \cdot \mathbf{p}_{old}$

Equivalently,
$\mathbf{p}_{next}$ is an extrapolation of $\mathbf{p}_{now}$, $\mathbf{p}_{old}$:

$\mathbf{p}_{next} = mix(\mathbf{p}_{old}, \mathbf{p}_{now}, 2)$

a bit shorter



$\mathbf{p}_{next} = 1.98 \cdot \mathbf{p}_{now} - 0.98 \cdot \mathbf{p}_{old}$

Equivalently,
$\mathbf{p}_{next}$ is a *different* extrapolation of $\mathbf{p}_{now}$, $\mathbf{p}_{old}$:

$\mathbf{p}_{next} = mix(\mathbf{p}_{old}, \mathbf{p}_{now}, 1.98)$

165

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

166

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 right now
    - Can be simulated by: positional constraints

167

Rigid-bodies as compounds of particles + constraints

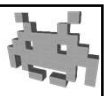
- Interesting/rich/useful set of "emerging behaviors" (they just… automatically happen) :
  - rigid, deformable, jointed objects
    - made of particles + hard constraints
  - their angular velocities
    - rotation around proper axis
  - their barycenter
  - their momentum of inertia
    - angular velocity is maintained
  - somewhat believable bounces on "impacts" (with ground?)
    - 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

168

Rigid-body as particles + constraints: issues



- Approximations are introduced
  - e.g.: mass is concentrated in a few locations only
- 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 virtual "rigid body" is needed to render the associated meshes
  - or: its velocity, angular velocity, total mass may be needed for… gameplay reasons (e.g. damage), graphics (motion blur), etc

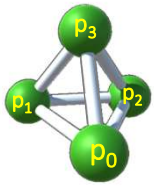
169

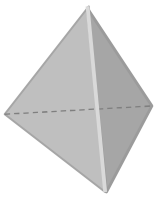
Particles + constraint, or rigid bodies?

- 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?

170

How to extract...





	Particle Compound		(Virutal) Rigid Body
STATIC (initialization)	masses $m_0 \dots m_N$ initial positions $\mathbf{r}_0 \dots \mathbf{r}_N$	?	mass m barycenter $\bar{\mathbf{r}}$ (local sp. origin?) moment of inertia $\mathbf{M}$ (matrix)
DYNAMIC (during simulation)	positions $\mathbf{p}_0 \dots \mathbf{p}_N$ velocities $\mathbf{v}_0 \dots \mathbf{v}_N$	?	position $\mathbf{p}$ (of barycenter) velocity $\mathbf{v}$ (linear) rotation $\mathbf{R}$ (i.e. orientation) angular velocity $\mathbf{a}$

171

For example:

	Particle Compound		Rigid Body
STATIC	masses $m_0 \dots m_N$	A	(total) mass m
DYNAMIC	positions $\mathbf{p}_0 \dots \mathbf{p}_N$	B	(current) position $\bar{\mathbf{p}}$ (of barycenter)
	velocities $\vec{v}_0 \dots \vec{v}_N$	C	(linear) velocity $\vec{v}$

A: $m = \sum m_i$

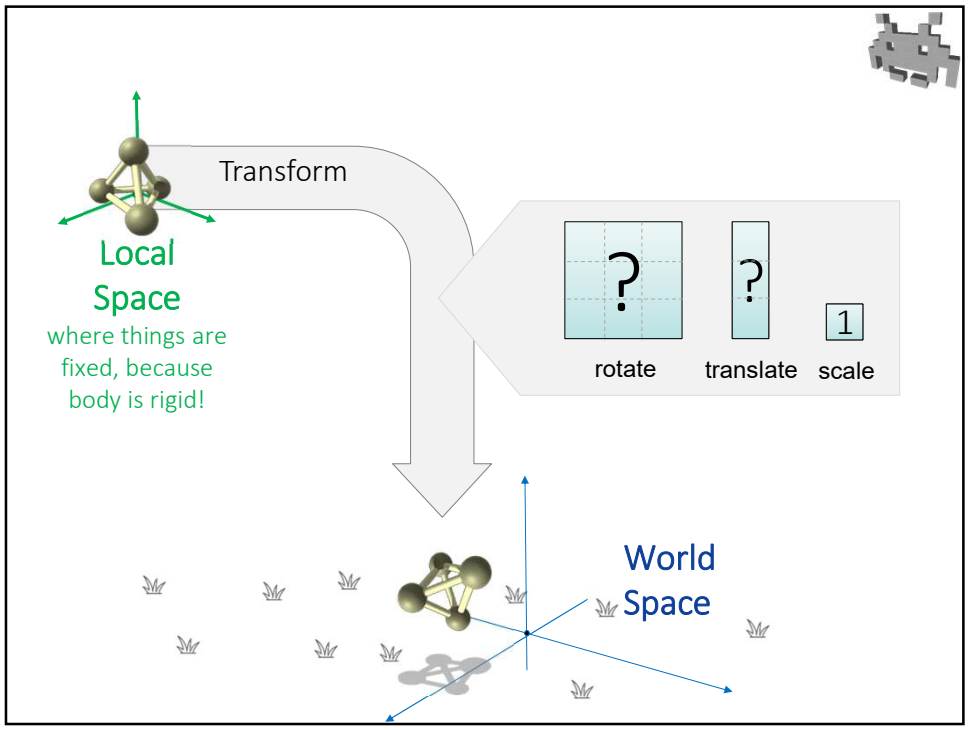
B: $\bar{\mathbf{p}} = \frac{1}{m} \sum m_i \mathbf{p}_i$

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

Questions:
How to find / update the...
- rotation
- angular velocity
- moment of inertia matrix (const)

...of the rigid body?

174



175

Extract the (global) transform of the "virtual" rigid-body 1/2

- Input:
  - particle "rest" positions $\mathbf{r}_0 \dots \mathbf{r}_n$ in local space (fixed)
  - their masses $m_0 \dots m_n$ (fixed)
  - their current positions $\mathbf{p}_0 \dots \mathbf{p}_n$ in global space (this frame)
- Output: current transform ← a roto-translation, by definition
  - Scaling: 1 ← it's rigid, duh
  - Translation: difference between the barycenters: $\bar{\mathbf{p}} - \bar{\mathbf{r}}$
  - Rotation: see the next slide
    (just for completeness)
    (the algorithm is not part of the exam)

176

Extract the (global) transform of the "virtual" rigid-body 2/2

this part is just for completeness, **not** part of the exam. But maybe useful in life...

- Find the rotation (as a 3x3 matrix):
  - Step 1 : remove the respective barycenters from pts
    $\vec{r}_i \leftarrow \mathbf{r}_i - \bar{\mathbf{r}}$ and $\vec{p}_i \leftarrow \mathbf{p}_i - \bar{\mathbf{p}}$
  - Step 2: find 3x3 rot matrix R such that $\forall i \,.\, \mathrm{R}(\vec{r}_i) \cong \vec{p}_i$ as closely as possible (weighted by the mass);

    that is, such that $\Sigma\, m_i \, \|\mathrm{R}(\vec{r}_i) - \vec{p}_i\|^2 \rightarrow \min$

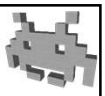
as objects rotate around their barycenter!

External product: $\vec{v} \otimes \vec{w} = \vec{v}\ \vec{w}^{\mathrm{T}}$

The theory gives us this:
first, compute $\mathrm{M} = \Sigma\, m_i\, (\vec{r}_i \otimes \vec{p}_i)$
then, find R as the closest *rotation* matrix to M
using Principal Values Decomposition (PVD) – look it up

177

Summary: two ways to handle rigid-bodies



- As a compund of particles, with PBD
  - Bonus: we can also handle… deformable bodies, articulated bodies
  - Bonus: mixes well with many other useful constraints
  - Cost: need to extract status of the «virtual» rigid body
  - Cost: mass concentrated at particles (approximation)
- With Rigid Bodies dynamics
  - With explicit rotation / angular velocity
  - The integration methods (Euler, Leapfrog, Verlet) can be adapted
- Or, mixed systems:
  - Requires to convert (dynamically) between the two

179