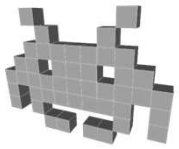
3D video games

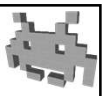
3D Game Physics



Marco Tarini

2

Course Plan



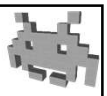
lec. 1: **Introduction**
lec. 2: **Mathematics** for 3D Games
lec. 3: **Scene Graph**
lec. 4: Game **3D Physics**
lec. 5: Game **Particle Systems**
lec. 6: Game **3D Models**
lec. 7: Game **Textures**
lec. 8: Game **Materials**
lec. 9: Game **3D Animations**

computer animation

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

3

Animation in games



but, a note on terminology:
in some contexts, procedural means
"produced by a ***simple*** procedure"
as opposed to "physically simulated"

Non procedural

- Assets!
- Fully controlled by artist/designer (dramatic effects!)
- Realism: depends on artist's skill
- Does not adapt to context
- Repetition artefacts

Procedural

- Physics engine
- Less control
- Physics-driven realism
- Auto adaptation to context
- Naturally repretition free

4

Physics simulation in videogames

- 3D, or 2D
- "soft" real-time
- efficiency
  - 1 frame = 33 msec (at 30 FpS)
  - physics = 5% - 30% max of computation time
- plausibility
  - but not necessarily *accuracy*
- robustness
  - should almost never "explode"
  - it's tolerable to have inconsistencies over a few frames, as long as it recovers in subsequent frames

6

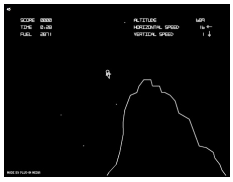
Physics in games: cosmetics or gameplay?

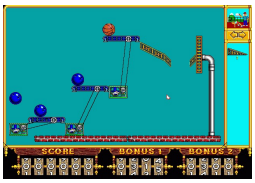
- Just a graphic accessory? (for realism!)
  - e.g.:
    - particle effects (w/o feedback)
    - secondary animations
    - Ragdolling
- Or a gameplay component?
  - e.g. physics based puzzles
  - Popular approach in 2D (since always!)

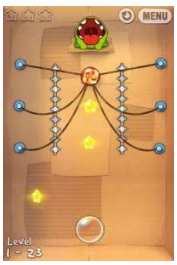
7

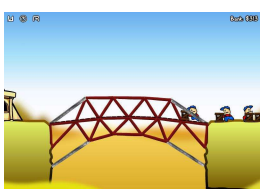
Physics in games: cosmetics or gameplay?

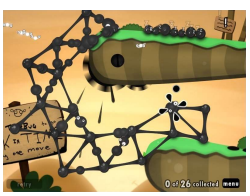
- Just a graphic accessory? (for realism!)
  - e.g.:
    - particle effects (w/o feedback)
    - secondary animations
    - Ragdolling
- Or a gameplay component?
  - e.g. physics based puzzles
  - Popular approach in 2D (since always!)





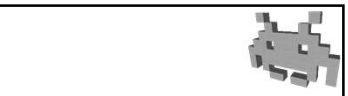




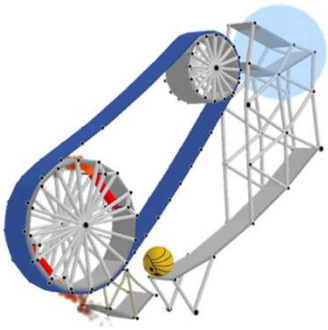


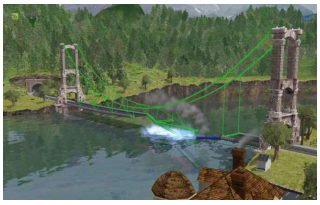
8

Physics in games: cosmetics or gameplay?



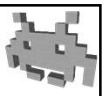
- Just a graphic accessory? (for realism!)
  - e.g.:
    - particle effects (w/o feedback)
    - secondary animations
    - Ragdolling
- Or a gameplay component?
  - e.g. physics based puzzles
  - Rising trend in 3D





9

Physics engine: intro



- Game engine module
  - executed in real time at game run-time
- A high-demanding computation
  - on a very limited time budget!
- …but highly parallelizable
  - potentially, highly parallel

==> good fit for hardware support

(just like the Rendering Engine)

10

Hardware for Physics engine

To exploit a **strong parallelism**,
you need a **strongly parallel** hardware!

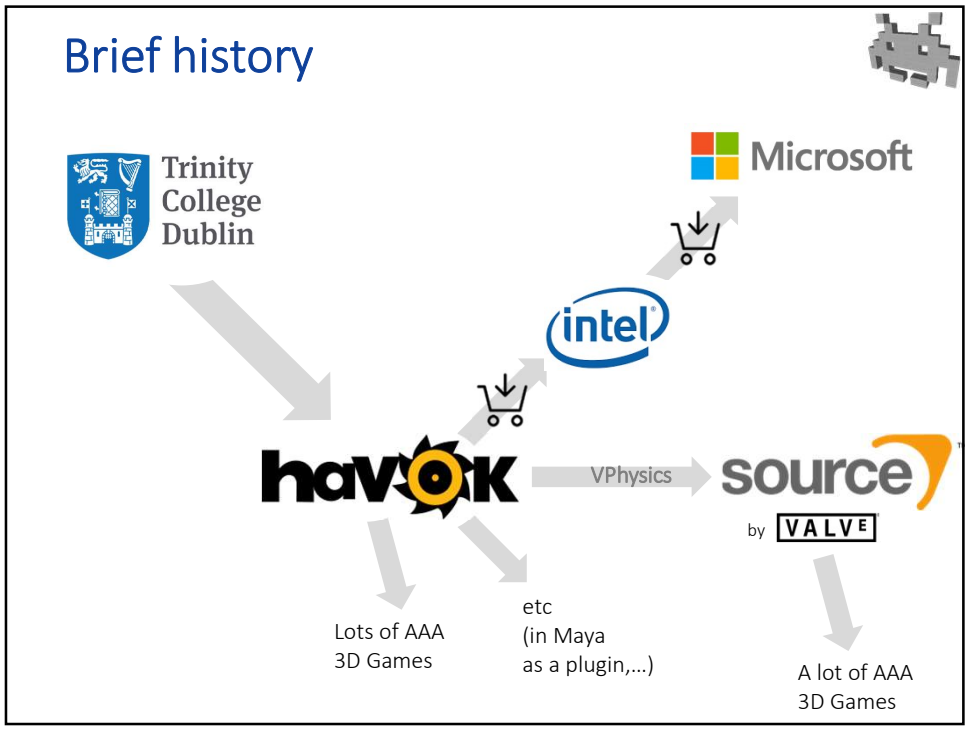
- For a brief moment ~2006: PPU
  - "Physics Processing Unit"
  - HW unit specialized for physics
- After that: GP-GPU
  - "General Purpose Graphics Processing Unit"
    = Use of the graphics card for generic tasks
    (not related with 3D computer graphics)
  - or, Cuda (nVidia), OpenCL (openSource)

11

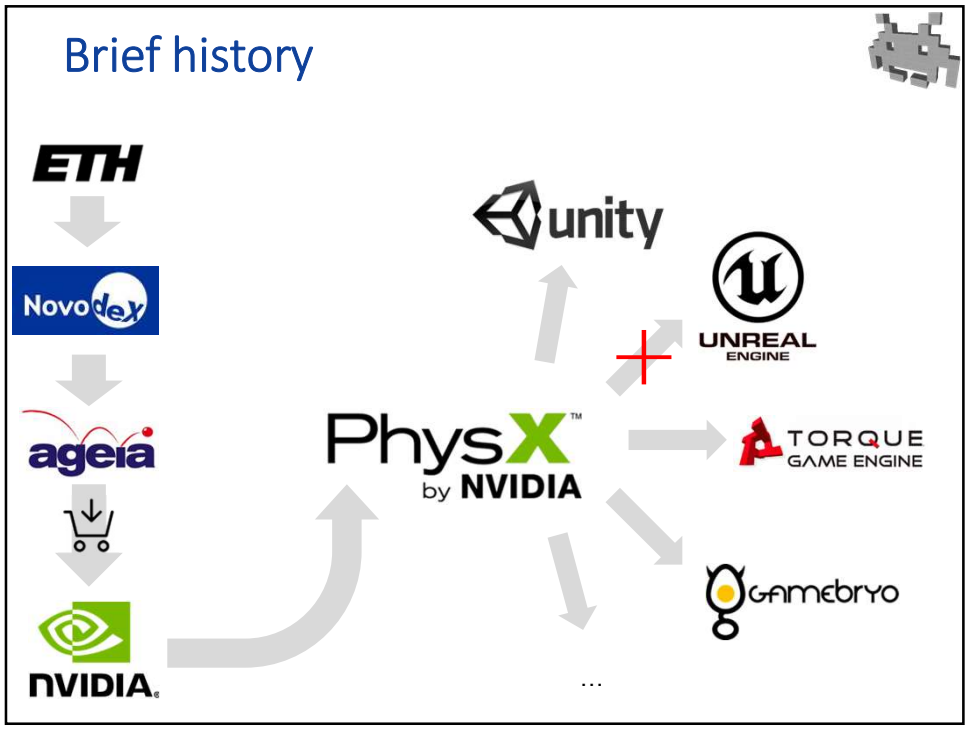
Main Software (libraries, SDK)

havok	mostly CPU (Microsoft)
PhysX	CPU+GPU (CUDA) NVidia
BULLET PHYSICS LIBRARY	open source, free, HW accelerated (OpenCL) + CPU
OPEN DYNAMICS ENGINE	open source, free
Box2D	2D, open source, free (adopted by Unity 2D)
Jolt Physics	open source, free

12


Brief history
Trinity College Dublin
Microsoft
intel
havok
VPhysics
source
by VALVE
Lots of AAA 3D Games
etc (in Maya as a plugin,...)
A lot of AAA 3D Games

13


Brief history
ETH
Novodex
ageia
NVIDIA
unity
UNREAL ENGINE
PhysX by NVIDIA
TORQUE GAME ENGINE
Gamebryo
...

14

Fields of study

Dynamics

The motion, as a result of **forces**

Example:
"Subject to gravity, how will this pendulum swing?"

Statics

Equilibrium states, minimal **energy** states

Example:
"In which state(s) can this pendulum be still?"

Kinematics

The **motion** itself, no matter why it moves

Example:
"If the angular speed of the pendulum is currently X, how fast is the ball moving?" (or vice versa)

16

The 2 tasks of the Physics engine

1. Dynamics (Newtonian)

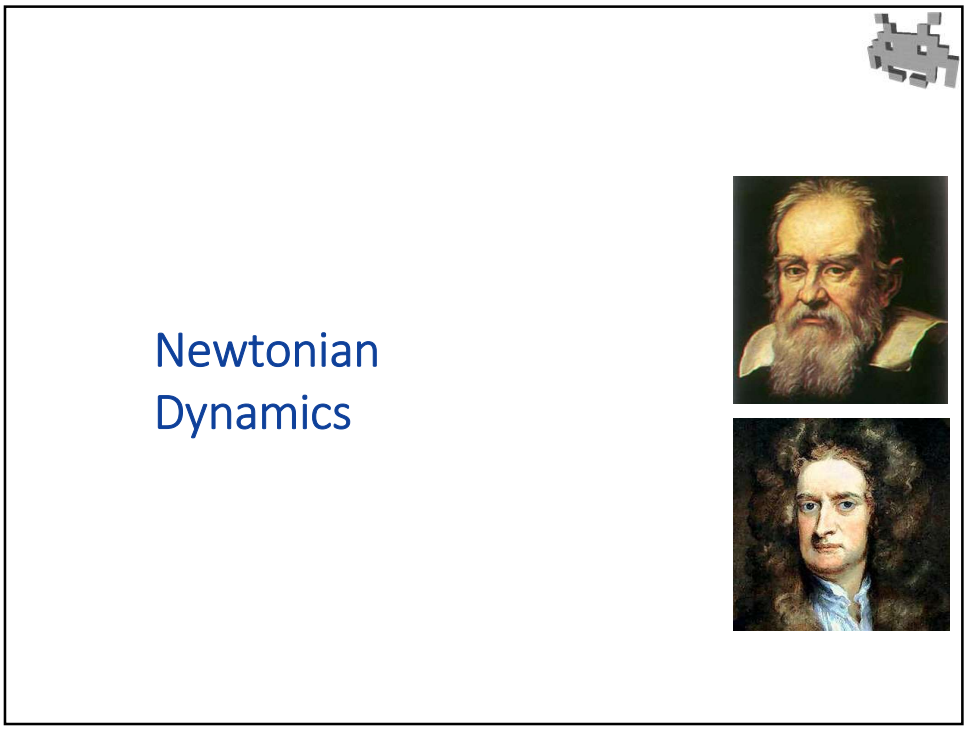
for objects such as:

- Particles
- Rigid bodies
- Articulated bodies
  - e.g. "ragdolling"
- Soft bodies
  - Ropes (specific solutions)
  - Cloth (specific solutions)
  - Hair (specific solutions)
  - Free-form deformation bodies (general)
- Fluids
  - Expensive!

2. Collision handling

- Collision detection
- Collision response

17



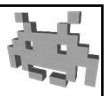
18

Physics and spaces (observation)

- The scene hierarchy (the scene graph), and the entire distinction between local and global space, its's entirely "in our mind"
  - It's a useful abstraction to control or code *scripted* animations
  - E.g., kinematics animations, skeletal animations…
- But physics *doesn't care* about any of it
  - Physics happens entirely in global (world) space
  - Persistent spatial relationships (e.g., between a car and its wheels) either exists due to physical constraints, or they are irrelevant
  - Even if they physically exists, they are still enforced in global space, like all the rest of the physics simulation
  - Physics simulation computes changes to objects states (position, orientation…) in global space
  - But, as we know, these updates can be converted/stored in local space

19

Spatial placement of a (rigid) object



2D Physics

- Position:
  (x,y)
- Orientation:
  (α) – angle (scalar)

3D Physics

- Position:
  (x,y,z)
- Orientation:
  quaternion or
  axis,angle or
  axis * angle or
  3x3 matrix or
  Euler angles

20

Newtonian dynamics: summary

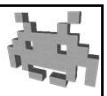
Current object placement	Rate of change of ← (d / dt)	← "with mass" (momentum)	What changes the rate of change (d^2 / dt^2)	← "with mass"
Position p p = (x,y,z)	Velocity $\vec{v}$ $\vec{v} = \dot{p}$ ($\|\vec{v}\|$ = "speed")	Momentum $m\,\vec{v}$	Acceleration $\vec{a} = \dot{\vec{v}} = \ddot{p}$	Force $\vec{f}$ $\vec{f} = m\,\vec{a}$
Orientation (e.g. quaternion)	Angular velocity $\vec{\omega}$	Angular momentum $I\,\vec{\omega}$ I = moment of inertia	Angular acc. $\vec{\alpha}$	Torque $\vec{\tau}$ $\vec{\tau} = I\,\vec{a}$ · ("mechanic momentum")

state (is kept! inertia!)
(changes, but only continuously)

change the state
(no memory)

22

Per-object constant: mass & its distribution (for non point-shaped ones)



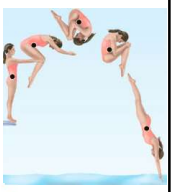
A few quantities associated to each rigid object

- constants: they don't (normally) change
- *input* of the physics dynamic simulation, not output

- Mass:
  - resistance to change of velocity

distribution of mass

- Moment of Inertia:
  - resistance to change of *angular* velocity



- Barycenter:
  - the center of mass

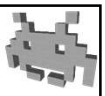
23

Mass: notes

- resistance to change of velocity
  - also called *inertial* mass
- also, incidentally:
  ability to attract every other object
  - also called *gravitational* mass
  - happens to be the same
- it's what you measure with a scale
- Unity of measure:
  kg, g, etc…

24

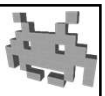
Barycenter (of a rigid body): notes



- Aka the center of mass of an object
  - constant: it's a fixed point in *local* space for a rigid body
- Often (but not necessarily) is the origin of the local frame
  - if so, the *position* of a rigid body (the translation of its transform) = the position of its barycenter
- It's the *weighted average* of the positions of the subparts composing an object
  - literally "weighted": with their masses
- In absence of external forces, the object rotates (orbits, spins) around this position.

25

Moment of inertia: notes 1/3



- Resistance to change of angular velocity

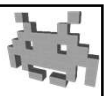
high

low

- (an object rotates around its barycenter)

26

Moment of inertia: notes 2/3



- Scalar moment of inertia
  - Resistance to change of angular velocity
  - Depends on the total mass, and also on its *distribution*
    - the farthest one sub-mass from the axis, the > the resistance
- In 2D: it's a fixed value
  (for a given rigid object)
  - The object always spins around its barycenter

27

Moment of inertia: notes 3/3

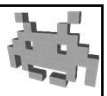
- In 3D: the rigid objects spins around an axis passing through the barycenter
  - for any possible axis of rotation,
    you have a different *scalar* moment of inertia
  - for a given axis $\hat{a}$ the scalar moment is given by

    $$\hat{a}^{\mathrm{T}}\ \mathrm{M}\ \hat{a}$$

    where 3×3 matrix M is the «(moment of) inertia *matrix*»
    aka the «(moment of) inertia *tensor*»)
  - M can be computed for a given rigid object
    - how: that's beyond this course
    - in practice: use given formulas for common shapes
    - or, sum the contributions for each sub-mass
  - M describes the scalar moment of inertia for any possible axis or rotation

28

State of a rigid object in a physical simulation



```
Point position
Rotation orientation
```
current

```
Vector velocity
Vector angular_velocity
```
current rates of change

updated by physics (dynamics)

```
Scalar mass
Matrix moment_of_inertia
Point barycenter
Scalar drag
...
```
frictions; see later

constants

setup at initialization, rarely changes (e.g. by scripts)

*Note: acceleration/forces/torques are **not** part of the state. They will be ... local variable, used and*

29

Unity

Adding a "RigidBody" component to a Game Object is to say: *"please let the Physics engine take care of the dynamics of this object"*

note: they are the components of the **global** transformation!

```
Vector3 position
Quaternion rotation
```
part of **Transform** component

note: speed = velocity.magnitude

```
Vector3 velocity
Vector3 angular_velocity
float mass
Vector3 inertiaTensor
Quaternion inertiaTensorRotation
Vector3 centerOfMass
float drag
bool isKinematic
```
the **RigidBody** component

per second (not per frame!)

moment of inertia matrix

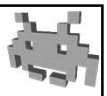
the Vector3 = a diagonal matrix D by rotating it $R_I^T D R$ → the final matrix

the barycenter (in local space)

if true: disable dynamics (but keeps e.g. collisions)

30

The case of particles



- For now, we will study a simpler case:
  the dynamics of particles (and its simulation)
- Particle = ideal object shaped like a point,
  with all the mass concentrated in that point
- Particles-only is easier because the following are irrelevant:
  - rotation (orientation): a point does not rotate
  - the center of mass (it's the position of the particle itself);
  - the distribution of mass, i.e. the moment of inertia (there's none);
  - the torques (only forces matter);
  - the angular velocity (there's only linear velocities)
- These things are only relevant for non-point sized (rigid) objects
- The algorithms we are bout to see can be extended to rigid bodies

31

State of a particle (point sized obj) in a physical simulation

```
Point position
Rotation orientation
Vector velocity
Vector angular_velocity
Scalar mass
Matrix moment_of_inertia
Point barycenter
Scalar drag
…
```

not used for point sized objects! ("particles")

One possibility in a game phys engine is to only simulate point-particles.

Simpler: no rotation needed!

We will see later how to still get rigid bodies back.

For now, we focus on this simpler case.

32