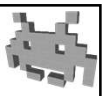
3D video games

Collision Handling

Marco Tarini

1

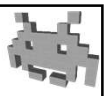
Course Plan



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

2

Collision Handling: the other half of the physic system



- Collision detection
  - find out when they occur
- Collision response
  - compute their effects

4

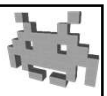
Collision Handling: a preliminary consideration

- Two types of objects in a game:
  - static
    - Never moves (speed = 0)
    - Part of the setting, background
    - Affects other objects, not affected by other objects
  - non-static
    - Can move around (for any reason)
- Two types of collisions:
  - one-way : a non-static object with a static object
  - two-ways : a non-static object with a non-static object

	Static	Movable
Static	⊘	One Way
Movable	One Way	Two Ways

6

Collision Handling: a preliminary consideration



By labelling every object as either static or movable, we reduce the needed computation considerably!

E.g., if 50% static, 50% movable then…

- 1/4 of the potential collisions cease to exists (*).

Of the rest:

- 2/3 are one ways (easier to handle)
- only 1/3 are two-ways

(*) No collision handling for Static VS static:
That's not just an "optimization", but a feature:

- Wall models can compenetrate, to build a house (no collision!)
- Buildings can sink into the terrain (no collision!)
- Etc.

7

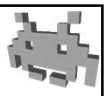
Collision Handling: two tasks

- Collision detection
  - find out when they occur
- Collision response
  - compute their effects

next topic

8

Collision response

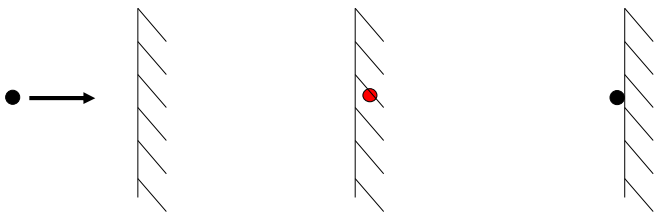


- Enforce non-penetration
  - objects must be placed in valid positions
  - (*when to:* **always**)
- Impacts
  - with impulses (bounces)
  - (*when to:* collision occurred now, but not in the pref frame)
- Frictions between the two objects
  - energy dissipation
  - (*when to*: from 2° consecutive step of collision)
- Ad-hoc effects
  - breaking objects, gameplay effects (HP loss?), etc (by scripts)
  - (*when to - if at all:* entirely gameplay dependent)

9

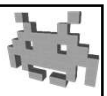
Enforcing non-penetration

- Invalid position?
  - strategy 1: revert to last valid pos (easy to do, not ideal)
  - strategy 2: project to closest valid pos (necessary, in PBD)



10

Enforcing non-penetration



- In PBD:
  just another positional constraint
  - bonus: velocity updates
    (similar to inelastic impacts)
  - but we will need to explicitly compute
    impacts if we want a better control
    of the behavior (see later)

Note: asymmetrical constraint (≥ not =)

A practical problem: the existence of the constraint it is **not** known a-priori.

- How to enforce this constraint:
  - *two-ways* :
    displace both of them,
    minimizing the summed squared displacements × the mass
  - *one-way* :
    only displace the one movable objects by the minimal amount
    (equivalent to the above, when fixed object mass → ∞)

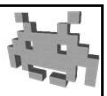
11

Contact frictions

- To be applied on prolonged contact
  - collision with an object which was colliding last frame too
- Affects the component of velocity *parallel* to the contact plane
- Can be implemented with:
  (1) explicit forces,
  - Verse: to current velocity, *projected on contact plane*
    (note: I need its normal)
  - Magnitude: proportional to the speed
- (2) or, velocity damping (but, isotropic!)
  - "Tax" on velocity, applied only to the component of speed parallel to contact plain

12

Resolving the impacts



so, it's the effect of an **impulse**

And, for rigid body dynamics: also new angular velocity

- **Sudden** velocity change
  - resolve the impact = determine the new velocities $\vec{v}_{new}$
  - ***equivalently***, determine the impulses $\vec{\imath} = m\,(\vec{v}_{new} - \vec{v}_{old})$

we'll write formulas for whichever is easier to write

- *All* impacts preserve total **momentum** $= m\,\vec{v}$
  - *Always*, no matter what

a vector (italian: *«quantità di moto»*)

- To resolve the impact, we need further assumptions, different for each type of the impact:
  - **elastic** ("preserves energy")
  - **inelastic** (doesn't)

13

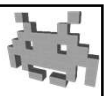
Different type of impacts

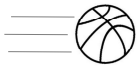
(fully) **elastic** impact

(fully) **inelastic** impact

14

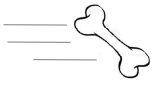
"Bounciness" (or impact elasticity)





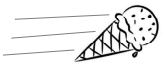
"Bounciness" = 1.0

…



"Bounciness" = 0.5

…



"Bounciness" = 0.0

15

"Bounciness" (or impact elasticity)

[notes]

- Elastic impact: no energy lost
- How is energy lost, in reality? (examples)
  - objects get damaged, heat is produced, sounds are emitted…
- "Bounciness": (or restitution coefficient)
  a simulated property of physical objects in games
  - it models the behavior of the object under impacts, as a mix between the two "pure" behaviors above
  - associated by designers to all virtual objects in the game
- Note: that's not how real stuff works!
  - not even at the two extremes (fully elastic / inelastic)
  - ours it's an approximation (especially for mixed bounciness)
  - remember: we are just aiming at *plausibility*

16

What about this impact?

"Bounciness" = ???

- Practical solution:
  adopt some formula between the bounciness values associated to the two objects
  - for example: avg, min, max
  - it's a choice of the game engine
  - (can be hard-wired in the physics engine, or left customizable for the dev to choose)

17

The assumptions for the two "pure" types of impact

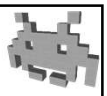
	assumptions		
elastic	after the impact, the total energy is the same as before	the impulse is in the direction of the impact normal	...and the total momentum is the same as before
inelastic	immediately after the impact, the two bodies share the same velocity		...and the total momentum is the same as before

imagine the two ice creams as temporarily glued together

remember that the impulse (force x time) is also the (instantaneous) change of momentum!
So, this is a way to say that the total impulse is zero
$i_A + i_B = 0$ that is, $i_A = -i_B$ aka the 3rd law of dynamics.

19

The assumptions for the two "pure" types of impact

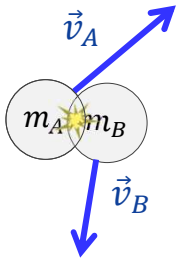


- (completely) elastic impacts:
  - preservation of total kinetic energy $\frac{1}{2} m \|\vec{v}\|^2$ (a scalar)
  - impulse direction = the normal of impact point
- (completely) inelastic impacts:
  - after the impact, the two bodies have the same velocity
  - (as if the impact momentarily glued them together) (they will still move apart in subsequent frames)
- mixed cases:
  - solve for both cases, interpolate resulting velocities
  - the weight of the interpolation = the "bounciness"

20

(completely) inelastic impact

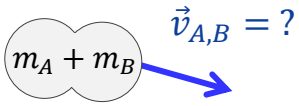
BEFORE:



Momentum:

$$m_A\,\vec{v}_A + m_B\,\vec{v}_B$$

AFTER:

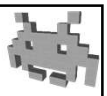


Momentum: (the only unknown, so …)

$$(m_A + m_B)\,\vec{v}_{A,B}$$

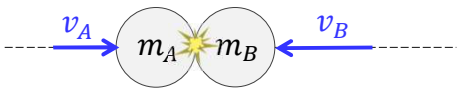
21

(completely) elastic impact: 1D case



BEFORE:

signed scalar



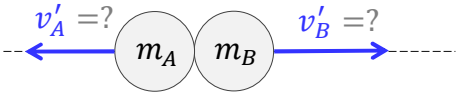
momentum:

$$m_A\, v_A + m_B\, v_B$$

energy:

$$\frac{1}{2} m_A\, {v_A}^2 + \frac{1}{2} m_B {v_B}^2$$

AFTER:



momentum:

$$m_A\, v'_A + m_B\, v'_B$$

energy:

$$\frac{1}{2} m_A\, {v'_A}^2 + \frac{1}{2} m_B {v'_B}^2$$

22

(completely) elastic impact: 1D case

new velocities are defined by the impulses:

signed scalars

$$v'_A = v_A + \frac{i_A}{m_A} \qquad v'_B = v_B + \frac{i_B}{m_B}$$

momentum conservation:

$$i_B = -i_A$$

(a.k.a. the 3rd law of dynamics)

energy conservation:

$$\frac{1}{2} m_A\, {v_A}^2 + \frac{1}{2} m_B {v_B}^2 = \frac{1}{2} m_A\, {v'_A}^2 + \frac{1}{2} m_B {v'_B}^2$$

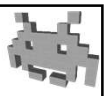
$$\Rightarrow \quad m_A\, {v_A}^2 + m_B {v_B}^2 = m_A \left(v_A + \frac{i_A}{m_A} \right)^2 + m_B \left(v_B + \frac{i_B}{m_B} \right)^2$$

$$\Rightarrow \quad m_A\, {v_A}^2 + m_B {v_B}^2 = m_A\, {v_A}^2 + \frac{{i_A}^2}{m_A} + 2\, v_A\, i_A + m_B {v_B}^2 + \frac{{i_B}^2}{m_B} + 2\, v_B\, i_B$$

$$\Rightarrow \quad 0 = \frac{{i_A}^2}{m_A} + 2\, v_A\, i_A + \frac{{i_B}^2}{m_B} + 2\, v_B\, i_B$$

26

(completely) elastic impact: 1D case



substituting:

$$\frac{i_A^{\,2}}{m_A} + 2\,v_A\,i_A + \frac{i_A^{\,2}}{m_B} - 2\,v_B\,i_A = 0$$

$$i_A^{\,2}\,\frac{m_A + m_B}{m_A\,m_B} + i_A\,2(v_A - v_B) = 0$$

$$i_A\left(i_A\,\frac{m_A + m_B}{m_A\,m_B} + 2(v_A - v_B)\right) = 0$$

solution 1

$$i_A = i_B = 0$$

before the impact

solution 2

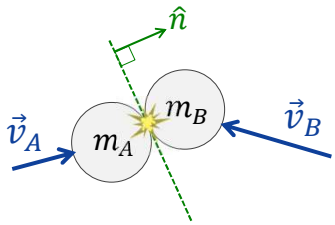
$$i_A = -i_B = \frac{2\,m_A\,m_B}{m_A + m_B}(v_B - v_A)$$

after the impact

27

(completely) elastic impact: 3D case

BEFORE:



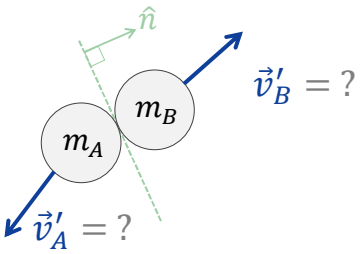
momentum:

$$m_A\vec{v}_A + m_B\vec{v}_B$$

energy:

$$\frac{1}{2}m_A\,\|\vec{v}_A\|^2 + \frac{1}{2}m_B\|\vec{v}_B\|^2$$

AFTER:



momentum:

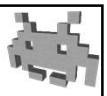
$$m_A\,\vec{v}'_A + m_B\,\vec{v}'_B$$

energy:

$$\frac{1}{2}m_A\|\vec{v}'_A\|^2 + \frac{1}{2}m_B\|\vec{v}'_B\|^2$$

28

(completely) elastic impact: 3D case



so, we need this data!

scalar impulses, pos. or neg. (the unkonwns)

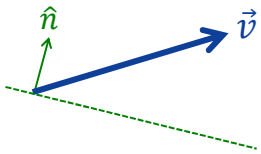
vector impulses

- Additional assumption:
  - ∃ impact plane, with normal $\hat{n}$
    - o, in 2D: impact line
  - impulses must be orthogonal to this plane $\vec{\imath}_{A,B} = i_{A,B}\hat{n}$
- To solve the impact
  - find scalar velocities $v_{A,B}$ as the component of vector velocities $\vec{v}_{A,B}$ along $\hat{n}$: $v_{A,B} = \vec{v}_{A,B} \cdot \hat{n}$
  - find scalar impulses $i_{A,B}$ (use the 1D case)
  - find vector impulses $\vec{\imath}_{A,B} = i_{A,B}\hat{n}$
  - apply them to vector velocities

29

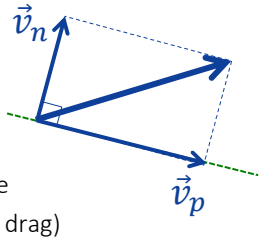
Remember this geometric subproblem?

- Given: velocity vector $\vec{v}$
  impact plane normal $\hat{n}$,
  split $\vec{v}$ in the vector sum
  $\vec{v} = \vec{v}_n + \vec{v}_p$ with
  - $\vec{v}_n$ orthogonal to the plane (= parallel to $\hat{n}$)
  - $\vec{v}_p$ parallel to the plane (= orthogonal to $\hat{n}$)



- Solution in 3 steps:
  (1) $s_n \leftarrow \vec{v} \cdot \hat{n}$
  (2) $\vec{v}_n \leftarrow s_n \hat{n}$
  (3) $\vec{v}_p \leftarrow \vec{v} - \vec{v}_n$ (or: $\vec{v}_p \leftarrow \vec{v}_n \times \vec{v} \times \vec{v}_n$)

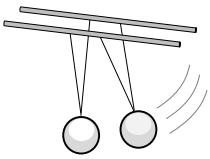
  with s_n a scalar (the signed "speed")



- Needed because:
  - only $\vec{v}_n$ is affected by elastic impacts with the plane
  - only $\vec{v}_p$ is affected by frictions with the plane (e.g.: drag)
  - s_n is used to *solve* elastic impacts (use 1D case)

30

Special case: (exercise: verify!) Equal masses

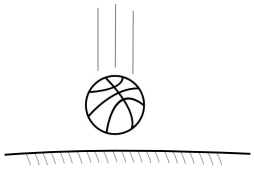


- Completely elastic case (1D):
  - the two velocities just *swap*
- Completely elastic case (3D):
  - The two velocity components orthogonal to the impact plane *swap*
- Completely inelastic case (3D):
  - the new velocity of both particles is the average of their pre-impact velocities

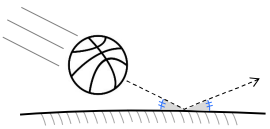
32

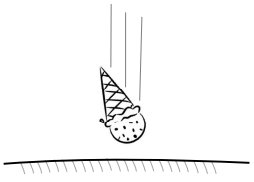
Special case: (exercise: verify!) one-way collision (A is static)

$m_A \longrightarrow \infty$ & $\vec{v}_A = 0$



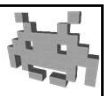
- Completely elastic case (1D):
  - v_b just *flips*
- Completely elastic case (3D):
  - The component of v_B orthogonal to impact plane just *flips*
- Completely inelastic case (3D):
  - B stops dead ($\vec{v'}_B = 0$)





33

Notes on
impacts between *rigid bodies*



that is, considering angular velocities too

- We only have seen impacts between *particles*
  - i.e., we disregarded angular velocities
  - when rigid bodies are implicitly implemented as particles + distance constraints, this is all we need to do!
  - Effect of elastic / inelastic impacts on angular velocities will be an (approximated) emerging behavior 👍
- Impacts between *explicit* rigid bodies require to *explicitly* compute the two post-impact angular velocities too
- Different math, stemming form the same principles:
  - Angular momentum: it is *always* preserved, no matter what
  - *Anelastic impact*: post-impact angular velocities must also match
  - *Elastic impact*: kinetic rotational energy must also be preserved
  - *Bounciness* $\in [0,1]$: interpolate angular velocities of the above

34

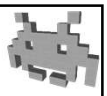
Collision Handling: two tasks

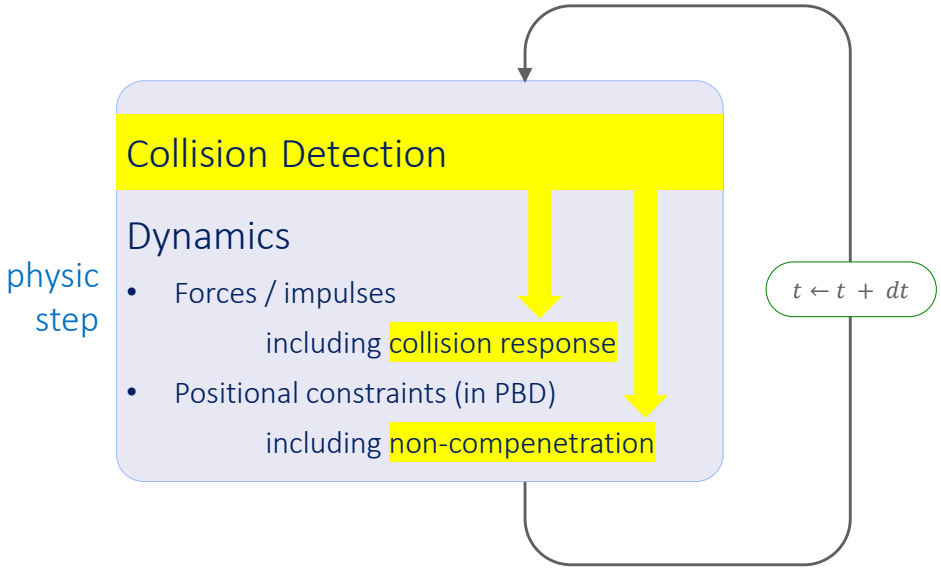
- Collision detection
  - find out when they occur
  - if so, produce collision data for the response
- Collision response
  - compute their effects

next topic

36

Collision detection: When?

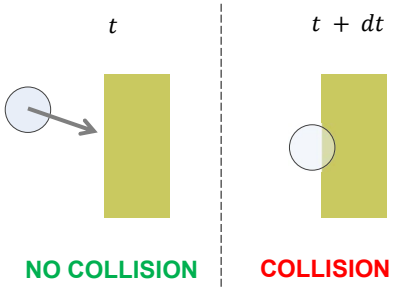




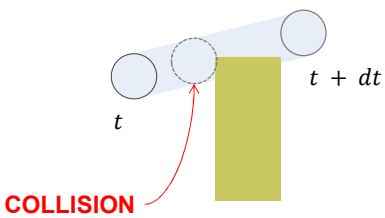
37

Collision detection: strategies

- Static Collision detection
  - ("a posteriori", "discrete")
  - approximated
  - simple + quick



- Dynamic Collision detection
  - ("a priori", "continuous")
  - accurate
  - resource consuming

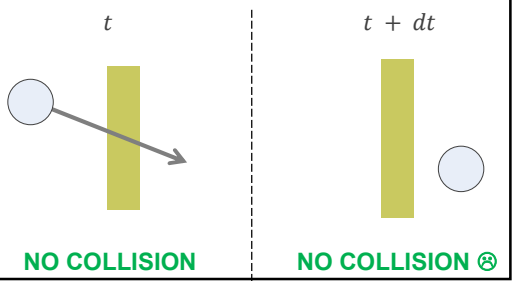


38

Collision detection: Static

aka
- «static» (because objects are tested as if they are still)
- «a posteriori» (because coll. are detected after they happen)
- «discrete» (because we check at discrete time intervals)

- Check for collision only after each step
- Problem: non-penetration is temporarily violated
  - patching it in collision response not always easy
- Problem: «tunneling»
  - Can happen if:
    - dt too large,
    - or, speed too large
    - or, objects too thin



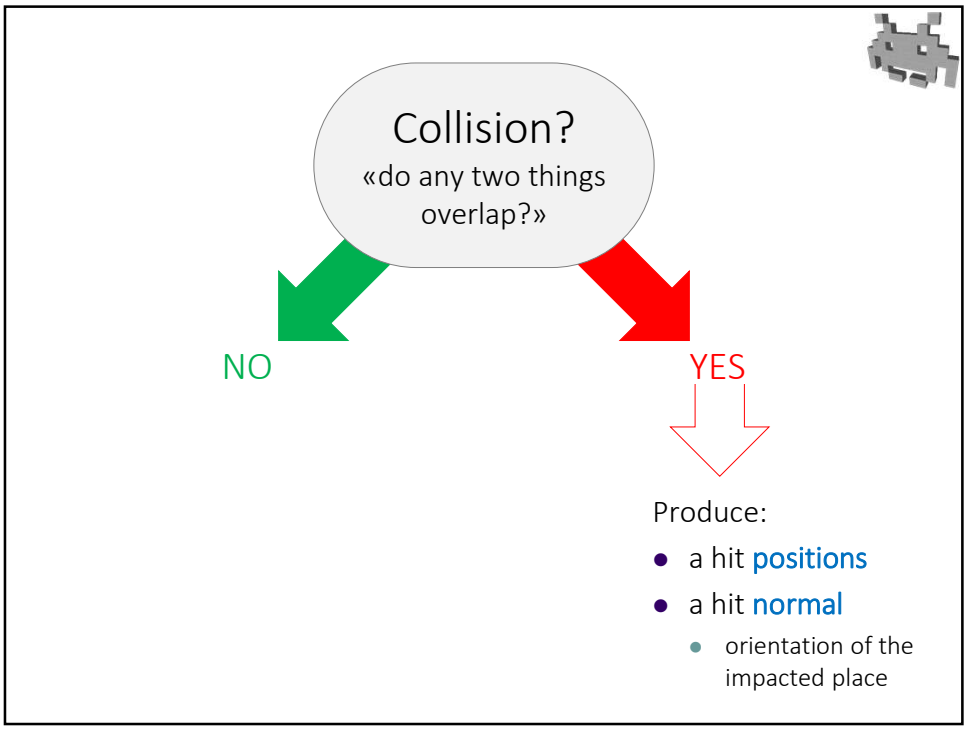
39

Collision detection: Dynamic

aka
- «dynamic» (because moving objects are tested)
- «a priori» (because coll. are detected before they happen)
- «continuous» (because it is checked over a temporal interval)

- Much more accurate detection
- Bonus:
  - no need to «*teleport the object in the safe position*».
  - it never left a safe position!
  - It can be easier to prevent penetrations than to heal them
- Much more difficult to do
  - for one-way collision: check the penetration between the static object and the volume swept (ita: *spazzato*) by the moving object *during the entire duration of the frame*
  - easy for: points (swept volume = segment)
  - easy for: spheres (swept volume = capsule – which one?)
- Basically, not practical to do in any other these
  - and even then, should only be used when required

40



41

From detection to response

The collision detection needs to tell us, for any pair of objects:

- Collision? Yes / No
  - «do these two things overlap?»

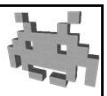
And, when it's a Yes…

- a hit positions
- normal of one collision plane
  - ~orientation of the impacted part
  - needed to: resolve the impact (except for purely inelastic)
  - needed to: apply frictions

«collision data»
output of detection, input of rensponse

42

Collision detection: a preliminary observation



- The usual concern: *efficiency*
- Key observation:
  - almost 100% of the object pairs, almost 100% of the times, do NOT collide.
  - for efficiency, the «no-collision» case needs to be optimized
  - «early reject» of the collision test

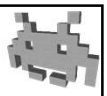
43

Example: this billiard shot

- A very "collidey" situation, right?
- Let's do the math
  - Balls: 16 (=15+1)
  - Ball pairs: 136 (=16 x 17 / 2)
  - Shot duration: ~10 seconds = ~600 physics frames
  - Assume ~2 collisions for each ball (a lot!) during the shot: ~16 collision events (each involving two balls)
  - Total: 16 collisions over 136 x 600 tests.
  - only < 0.02% of the potential collisions will collide!

45

Collision detection



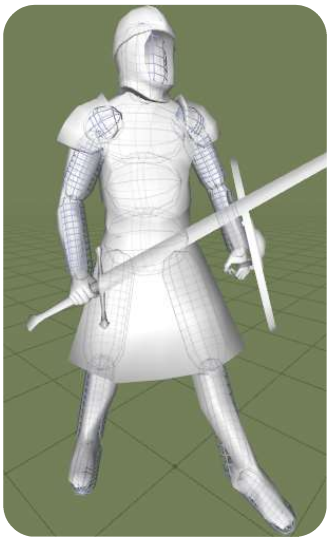
- Efficiency issues:

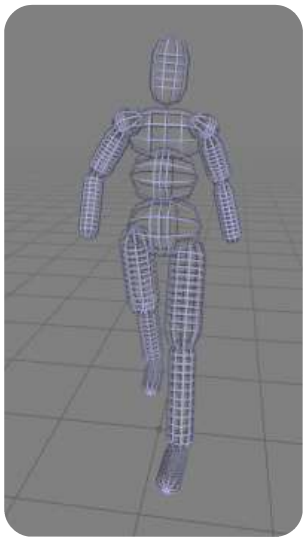
a) how to test between object pairs:
   - In an efficient way

b) how to avoid quadratic explosions of needed tests
   - n objects → n^2 tests ?

46

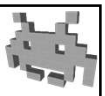
Geometric proxies





47

Geometric proxies



A simplified representation of the shape (the geometry) of the object, to be used in its place

- Note:it can be a *much* cruder approx. than the 3D model used for rendering

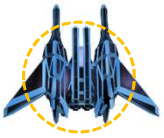
Two possible uses:

- as Bounding Volume
  - upper bound of the object spatial extension; object is *all inside* the proxy

  → for *conservative* tests

- as Collider (or hit-box, or collision proxy)
  - approximation of the object spatial extension

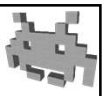
  → for *approximate* tests



("hit-box" is a misnomer: it's not necessarily a "box")

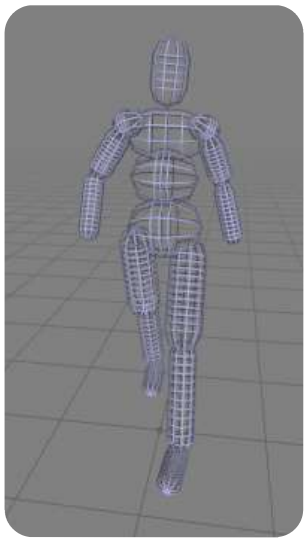
48

Geometric proxies: not only for collision detection, but also:



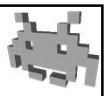
- physic engine
  - extract data for collision response
  - extract *barycenter* position & *moment-of-inertia* matrix of rigid bodies assuming uniform density (*Ita.: peso specifico*)
- rendering *optimizations*
  - "view frustum culling" *(bounding volumes)*
  - "occlusion culling" *(bounding volumes)*
- AI
  - visibility tests
  - in general, simulation of NPC senses
- GUI
  - picking (one of the ways to do that)
- 3D sounds
  - sound absorption in 3D sound propagation

Basically, for any other task except rendering: internally, objects *are* their proxies.



49

Semantic of a geometric proxy



Another proxy, a point, a ray…

`intersection( proxy_A , <something> ) ≠ ø` ?

- if `proxy_A` serves as Bounding Volume :
  - if NO: no collision
  - if YES: we don't know yet

An «early reject» optimization

- if `proxy_A` serves as Collider :
  - if NO: no collision
  - if YES: collision detected !
    - Must compute collision data from proxy_A

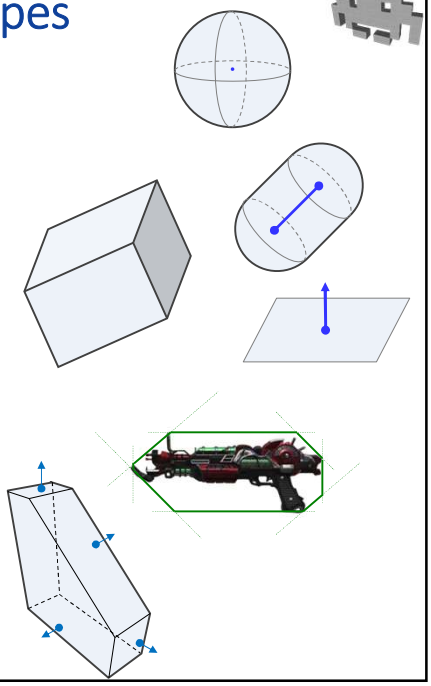
An approximation of the collision detection

Despite this semantic difference,
the data types used to represent proxies are the same.

50

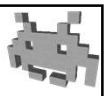
Geometric proxies: shapes

- Spheres
- Capsules
- Half-spaces
- Axis Aligned (Bounding) Boxes
  - aka AABB
- Generic Boxes
- Discrete Oriented Polytopes
  - aka DOP
- Ellipsoids
  - axis aligned or not
- Cylinders
- Convex polyhedrons
- Non-convex polyhedrons
  - Meshes
- …



51

🤔 choosing Geometric Proxies: things to consider



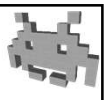
assets!

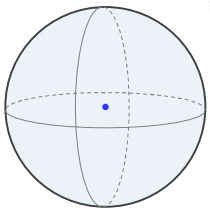
by algorithms — by artists

- Workload needed to compute / create them
- RAM space needed to store them
- Behavior under transformations
  - the ones we plan to use, e.g., roto-translations
- How good is the geometric approximation
  - for the objects we will use in the game
  - for bounding volumes ==> how *small / tight* is it?
  - for colliders ==> how *accurate* is the approximation?
- Workload for an intersection test
  - with other proxies, points, rays
  - also, how { easy to compute | good } is the collision data?

52

Geometric proxies: A sphere

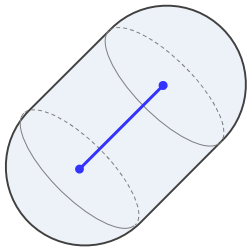




- ☺ easy to compute as a boundary
  - only the approximatively optimal one
- ☺ tiny to store
  - center (a point) + radius (a scalar) – or, a vec4 (c_x, c_y, c_z, r)
- ☺ collision test: trivial (against spheres or other things)
  - how? exercise – including collision data computation
- ☺ can easily undergo translation/rotation/scaling
  - how? exercise – note: scaling must be uniform
- ☹ approximation quality:
  - it depends on the object (as usual)
  - often, quite poor:
  - e.g.: a head? A character? A house? A sword?

54

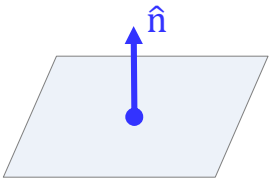
Geometry proxies: «Capsule»



- Generalizes the sphere:
  - Sphere ≙ the set of points having dist. from a point ≤ radius
  - Capsule ≙ the set of points having dist. from a segment ≤ radius
    - i.e. 1 cylinder ended with 2 half-spheres (all 3 with same radius)
- Stored as:
  - a segment (its two end-points)
  - a radius (a scalar)
- Exercise :
  - Q: how does it «score» w.r.t. the above measures?
  - (A: quite well → a very popular proxy in games!)

56

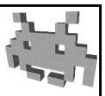
Geometry proxies: a half-space



- Trivial, but useful!
  - e.g. for a flat terrain,
  - or a wall
  - or an invisible “force field” to limit the game level (hated by players :-)
- Storage:
  - a point on the plane + its normal
  - better: a normal + a distance from the origin
  - which is a vec4 (n_x, n_y, n_z, k)
- how to test , transform, etc:
  - easy and efficient algorithms (check me)

58

Mini-exercise: Plane VS Point test

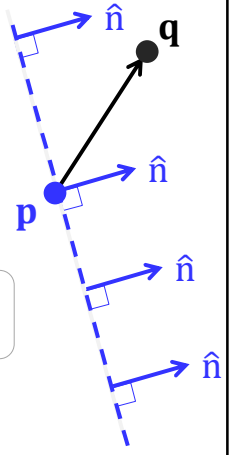


- Input: a point $\mathbf{q}$ and a plane given by:
  - its normal: $\hat{\mathrm{n}}$
  - a point on it at random: $\mathbf{p}$
- *Q*: on which side of the plane is $\mathbf{q}$?
- *A*: it's the sign of

$$\hat{\mathrm{n}} \cdot (\mathbf{q} - \mathbf{p}) =$$
$$\hat{\mathrm{n}} \cdot \mathbf{q} - \hat{\mathrm{n}} \cdot \mathbf{p} =$$
$$\hat{\mathrm{n}} \cdot \mathbf{q} + k =$$
$$(n_x, n_y, n_z, k) \cdot (q_x, q_y, q_z, 1)$$

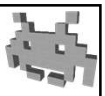
$k = -\hat{\mathrm{n}} \cdot \mathbf{p}$ (minus distance of plane from origin)

a 4D vector representing the plane



59

Which geometric proxy types to support in a game (-engine)?



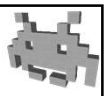
- an implementation choice of the Physics Engine
- # of intersection-test algorithms to be *implemented* : quadratic with # of supported types

VS	Type A	Type B	Type C	a Point	a Ray
Type A	*algorithm 1*	*algorithm 2*	*algorithm 3*	*algorithm 4*	*algorithm 5*
Type B		*algorithm 6*	*algorithm 7*	*algorithm 8*	*algorithm 9*
Type C			*algorithm 10*	*algorithm 11*	*algorithm 12*

useful, e.g. for visibility

60

Example: algorithm to testing capsule VS capsule



Input:

- Capsule 0: point $\mathbf{a}_0$ $\mathbf{b}_0$ radius r_0
- Capsule 1: point $\mathbf{a}_1$ $\mathbf{b}_1$ radius r_1

Output:

- Do they intersect?
- If so, intersection point $\mathbf{p}$ and normal $\hat{\mathbf{n}}$?

Solution (trace):

1. Find $\mathbf{c}_0$ and $\mathbf{c}_1$, the two points on the two segments closest to each other (see exercises on points and vectors)
2. Test: $\|\mathbf{c}_0 - \mathbf{c}_1\| < r_0 + r_1$?
3. Is so, collision detected with $\hat{\mathbf{n}} = \frac{\mathbf{c}_0 - \mathbf{c}_1}{\|\mathbf{c}_0 - \mathbf{c}_1\|}$

61

Exercise…

Imagine the algorithms for the proxies

- Sphere VS Sphere
- Sphere VS Capsule
- Sphere VS Plane

In each instance, find collision (Y/N) and collision data

Next time:

- More proxies
- Global phase: how to reduce the quadratic explosion of collision

62