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

2

Collision handling!

- The other half of physical engine

3

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

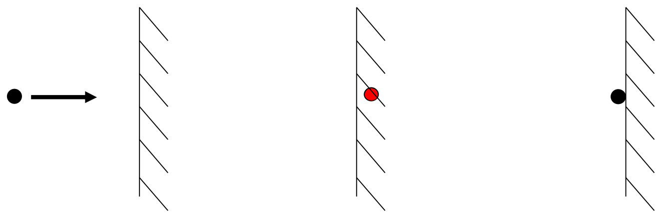
Collision response

- 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)
- 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 → ∞)

Note: asymmetrical constraint (> not =)

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

11

Contact friction

- Apply it on prolonged contact
  - collision with an object that was colliding last frame too
- Affects component of velocity parallel to contact plane
- Can be implemented with: (1) forces, or (2) velocity damping
- Forces:
  - Opposite to current velocity, *projected on contact plane* (note: I need its normal)
  - Magnitude: proportional to the speed

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 $\vec{\imath} = (\vec{v}_{new} - \vec{v}_{old}) \cdot m$

    we'll write formulas for whichever is easier to write
- *All* impacts preserve total momentum $m \cdot \vec{v}$
  - *Always*, no matter what

  a vector (ita: «*quantità di moto*»)
- To resolve the impact, we need further assumptions, different for each type of the impact:
  - elastic
  - inelastic

13

Different type of impacts

(completely)
elastic
impact

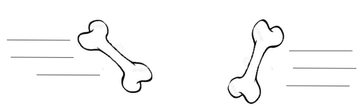
(completely)
inelastic
impact

14

"Bounciness" (or impact elasticity)

"Bounciness" = 1.0

...



"Bounciness" = 0.5

...

"Bounciness" = 0.0

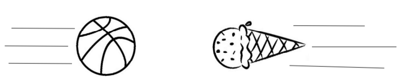
15

"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":
  a (made-up) 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 for the two extremes
  - 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 exposed to the users)

18

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	After the impact, the two bodies share the same velocity		...and the total momentum is the same as before

Remember that the impulse (force x time) is 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

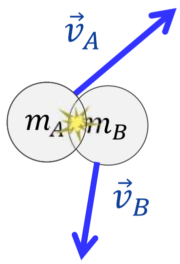
The assumptions for the two "pure" types of impact

- (completely) elastic impacts:
  - preservation of total kinetic energy $\frac{1}{2}m \cdot \|\vec{v}\|^2$ (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

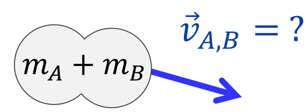
BEFORE:



Momentum:

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

AFTER:



the only unknown, so ...

Momentum:

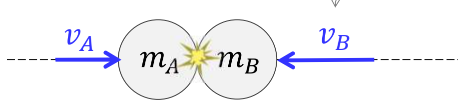
$$(\ m_A + \ m_B)\ \vec{v}_{A,B}$$

21

(completely) elastic impact: 1D case

BEFORE:

signed scalar



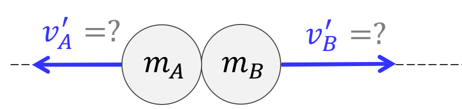
momentum:

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

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:

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

signed scalars

momentum conservation: $i_B = -i_A$ (it's just the 3rd law of dynamics)

energy conservation:

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

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

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

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

26

(completely) elastic impact: 1D case

substituting:

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

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

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

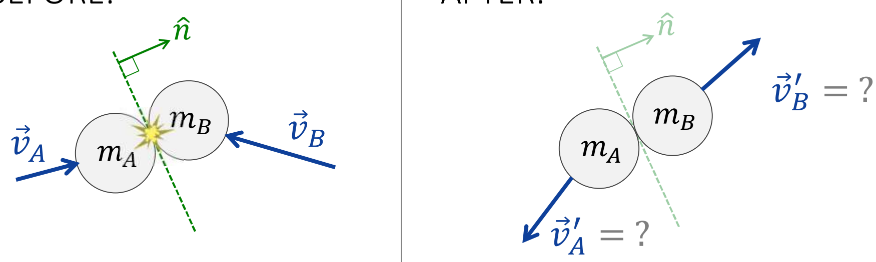
solution 1: $i_A = i_B = 0$ — *before the impact*

solution 2: $i_A = \dfrac{2\, m_A\, m_B}{m_A + m_B} (v_B - v_A)$ — *after the impact*

27

(completely) elastic impact: 3D case

BEFORE: AFTER:



momentum (before):

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

energy (before):

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

momentum (after):

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

energy (after):

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

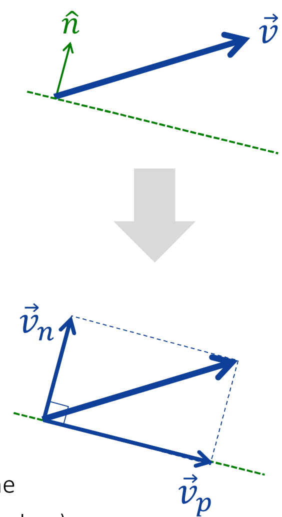
28

(completely) elastic impact: 3D case

- Additional assumption: *(we need this info!)*
  - ∃ impact plane, with normal $\hat{n}$
    - o, in 2D: impact line
  - impulses must be orthogonal to this plane $\vec{i}_{A,B} = i_{A,B} \hat{n}$ *(vector impulses; scalar impulses, pos. or neg. (the unkonwns))*
- 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 $\vec{i}_{A,B} = i_{A,B} \hat{n}$
  - apply them to vector velocities

29

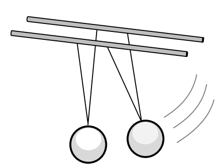
Remember this geometric subproblem?



- 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) $s_n \leftarrow \vec{v} \cdot \hat{n}$ with s_n a scalar (the signed "speed")
  (2) $\vec{v}_n \leftarrow s_n \, \hat{n}$
  (3) $\vec{v}_p \leftarrow \vec{v} - \vec{v}_n$ (or: $\vec{v}_p \leftarrow \vec{v}_n \times \vec{v} \times \vec{v}_n$)
- Useful 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

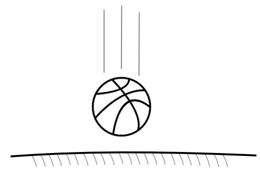


- Completely elastic case (1D):
  - the two velocities just *swap*
- Completely elastic case (3D):
  - The two velocity components orthogonal to the impact plane *swap*
- Completely inelastic case (3D):
  - the new velocity of both particles is the (vector) average of their pre-impact velocities

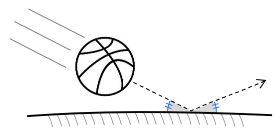
32

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

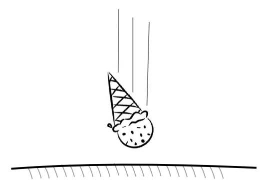
$m_A \longrightarrow \infty$ & $\vec{v}_A = 0$



- 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

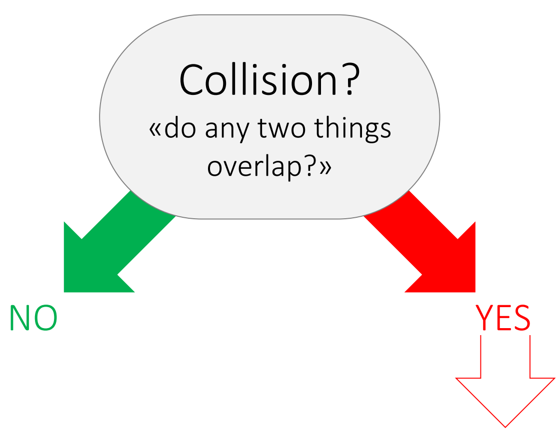
- We only have seen impacts between *particles*
  - i.e., we disregarded angular velocities
  - when rigid bodies are implicitly implemented as particles + distance constraints, this is all we need to do!
  - Effect of elastic / inelastic impacts on angular velocities will be an (approximated) emerging 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* ∈ $[0,1]$: interpolate angular velocities of the above

34

Collision Handling: two tasks

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

35



Produce:
- a hit positions
- a hit normal
  - orientation of the impacted place

36

From detection to response

The collision detection needs to tell us:

- Collision? Yes / No
  - «do any 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)

«collision data»
output of detection,
input of rensponse

37

Collision detection

- The usual concern: *efficiency*
- 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

38

Collision detection

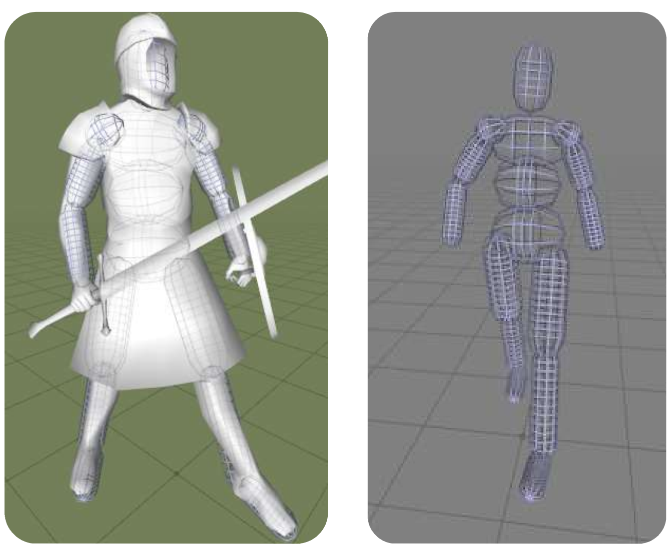
- Efficiency issues:

  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 ?

39

Geometric proxies



40

Geometric proxies

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

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

Two 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

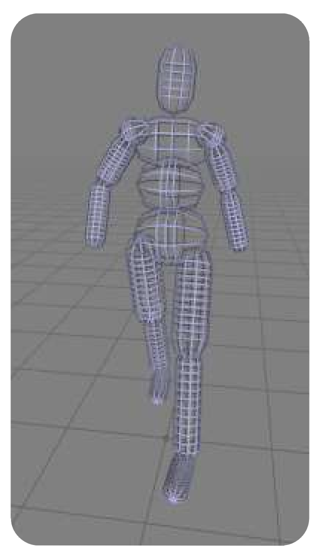
("hit-box" is a misnomer: it's not necessarily a "box")

41

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.



42

Semantic of a geometric proxy

Another proxy,
a point,
a ray…

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

- 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

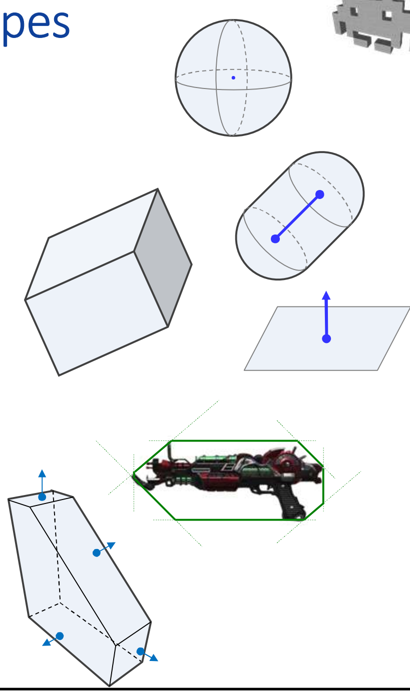
An approximation of the collision detection

Despite the semantic difference,
the same data type can be used for all proxies.

43

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
- …



44

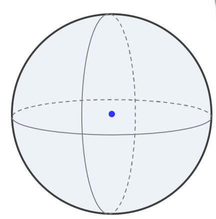
choosing Geometric Proxies: things to consider

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., isometries
- 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 *close* the approximation is it?
- Workload for an intersection test
  - with other proxies, points, rays…
  - how { easy to compute | good } is the collision data?

45

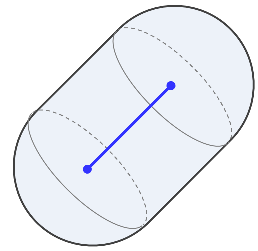
Geometric proxies: A sphere



- ☺ 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?

46

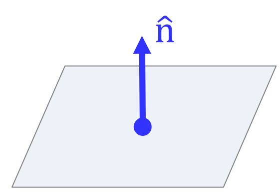
Geometry proxies: «Capsule»



- 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!)

47

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)

48

Mini-exercise: Plane VS Point test

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

$\hat{n} \cdot (\mathbf{q} - \mathbf{p}) =$

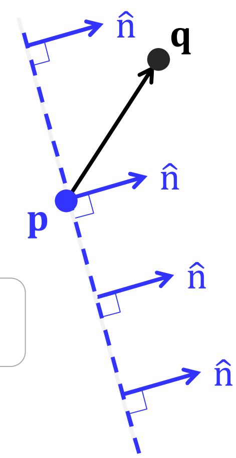
$\hat{n} \cdot \mathbf{q} - \hat{n} \cdot \mathbf{p} =$

$\hat{n} \cdot \mathbf{q} + k =$

$(n_x, n_y, n_z, k) \cdot (q_x, q_y, q_z, 1)$

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

a 4D vector
representing the plane



49

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

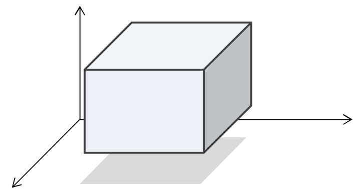
- an implementation choice of the Physics Engine
- # of intersection-test algorithms to be *implemented* :
  quadratic with # of supported types

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

50

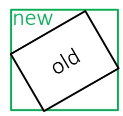
Geometry proxies: «AABB»



As the name implies, typically used as BOUNDING volume, not a collider

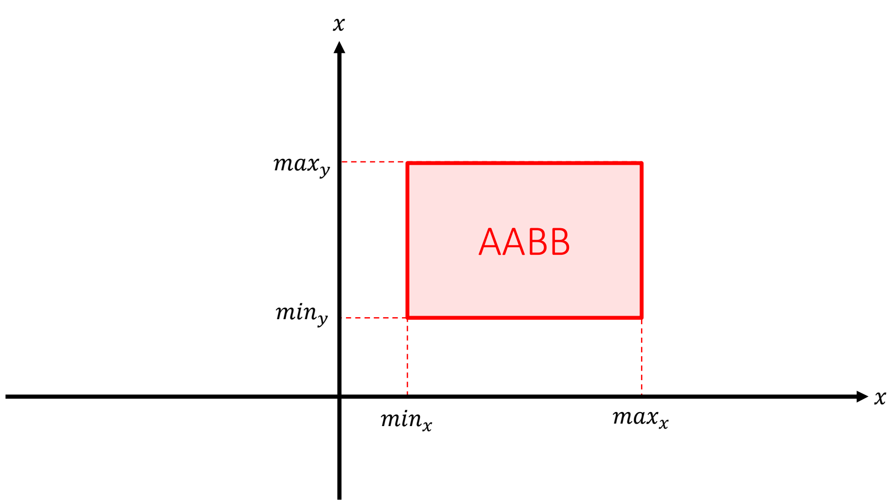
Axis Aligned Bounding Box

- Consists of three interval
  $[min_x, max_x] \times [min_y, max_y] \times [min_z, max_z]$ (Cartesian product)
- Concise to store
  - Two 3D points: (min_x, min_y, min_z) & (max_x, max_y, max_z)
- Easy to find the minimal AABB encapsulating a given set of points
- Easy to test for collision VS a point, or another AABB
  - Exercise: how?
- Under transforms:
  - ☹ ☹ ☹ if rotated, an AABB expands
  - (but can be easily scaled / translated)



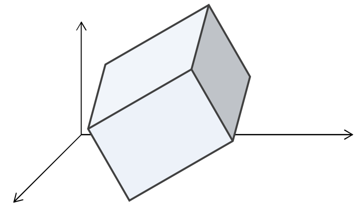
51

«AABB» : 2D example
(Axis Aligned Bounding… Rectangle)



52

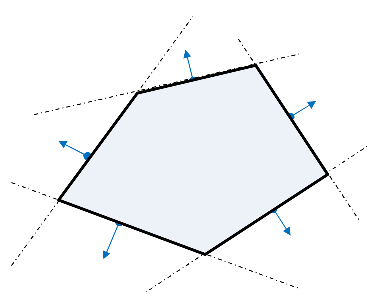
Geometry proxies:
Oriented Bounding Box (OBB)



- A “parallelepiped”
  - generalized version of AABB: it’s not axis-aligned
  - storage:
    - a rotation +
    - an AABB
  - Can be freely transformed
    - note: but only if scaling is uniform
  - Tests: still relatively easy (exercise: how to test points?)

53

Geometry proxies (in 2D):
a Convex Polygon

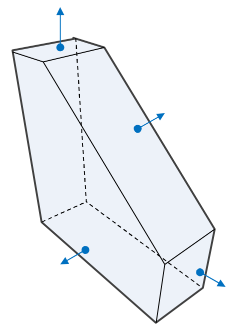


- Intersection of half-planes
  - each delimited by a line
- Stored as:
  - a collection of (oriented) lines
- Test:
  - a point is inside the proxy iff it is in each half-plane
- Flexible (good approximations)… and still moderate complexity

54

Geometry proxies (in 3D): a Convex Polyhedron

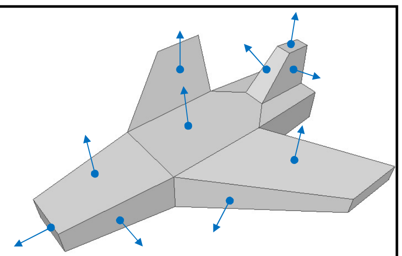
- Intersection of half-spaces
- Same as prev, put in but in 3D
  - stored as a collection of planes
  - each plane = a vec4 (normal, distance from origin)
  - tests: inside the proxy iff inside each half-space



55

Geometry proxies a (general) Polyhedron

potentially concave



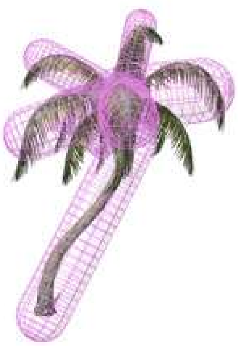
not worth it fora Bounding Volume !

- A… luxury Collider
  - The most accurate approximations
  - But, the most expensive tests / storage
- Specific algorithms to test for collisions
  - requiring some preprocessing
  - and data structures (BSP-trees, *see next lecture*)
- Creation (treat them as meshes):
  - sometimes, with automatic simplification
  - often, hand-designed by artists (low poly modelling)
- Similar to a 3D mesh used for rendering?
  - Many differences (compare with mesh, lecture 6)

56

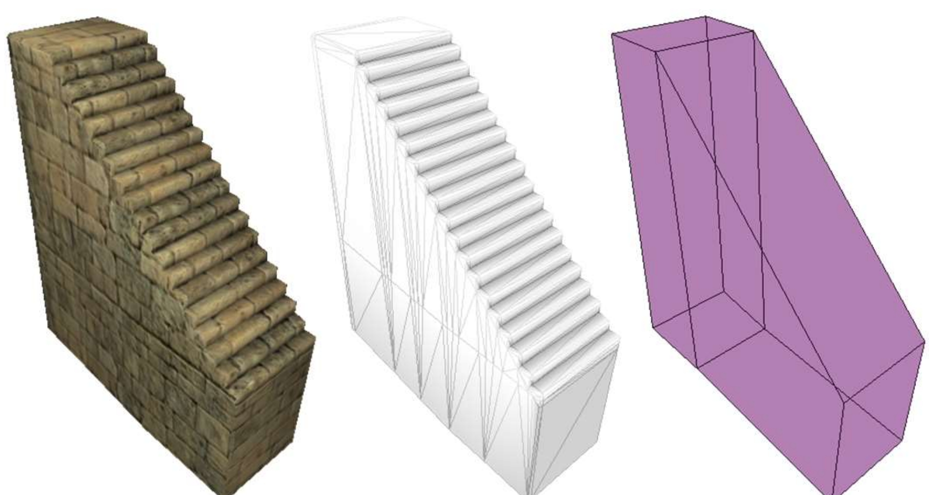
Composite Geometry Proxies

- A proxy can be a union of sub-proxies
  - inside the proxy *iff* inside of *any* sub proxy
- Very expressive
  - better approximation for many objects, even with few proxies
  - note: union of convex proxies can be concave !
- Still quite efficient to store / test
- Difficult to construct automatically
  - Open problem



61

Collision Proxy examples



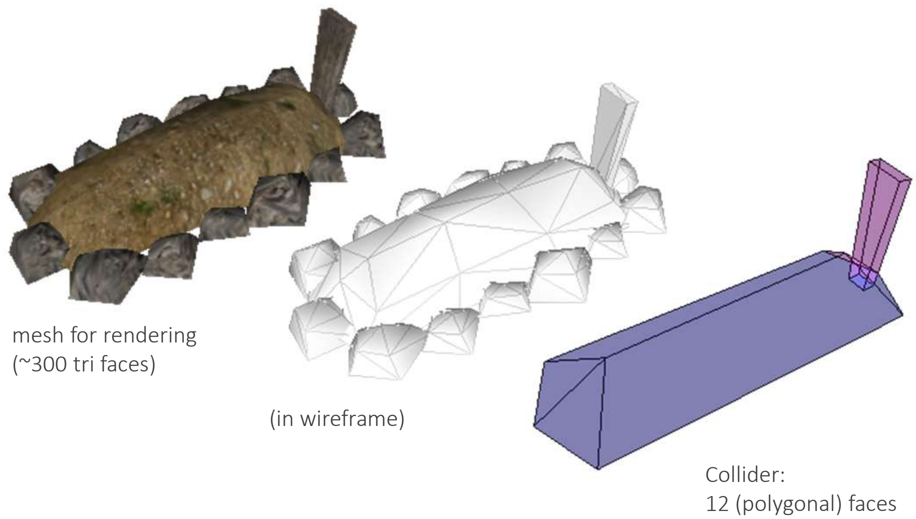
mesh for rendering (~600 tri faces)

(in wireframe)

Collider: 10 (polygonal) faces

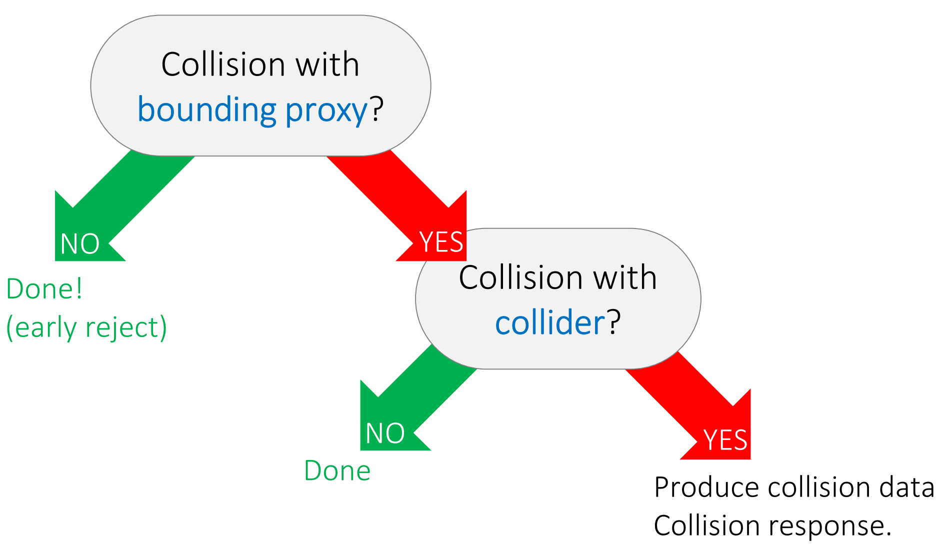
62

Collision Proxy examples



63

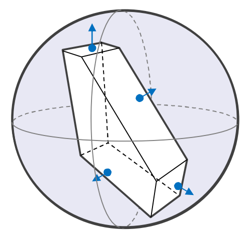
Bounding Volume + Collision Proxy



64

Bounding Volume + Collision Proxy

```
if (!intersect( boundingVol, X ) )
{
   // nothing to do: early reject!
}
else {
  CollisionData d;
  if (collide( hitBox, X , &d ))
  {
     collision_rensponse( d );
  }
}
```



a simpler
Bounding Volume
with, inside,
a more complex
Collision Object
approximating
the object

note: **intersect** and **collide** aren't the same function here

65

How to construct a geometry proxy to be used as a collider?

- *"Given an object representation M, build a good collision proxy for it"*
  - a *M* = 3D model of e.g. a dragon, a castle, a character…
- It's a difficult task to automatize
  - especially if we want to pick simpler (more efficient) proxies
    - such as compound of a few spheres, capsules, boxes
  - especially if we want good approximations
- It's often done manually by digital artists

Geometry proxies for colliders are assets !

66

How to construct a geometry proxy to be used as a bounding volume?

- *"Given an object representation M, build a thigh bounding volume for it"*
  - a *M* = 3D model of e.g. a dragon, a castle, a character…
- It's difficult to find the optimal (smallest possible) bounding volume automatically
- A lot easier to find a "good enough" bounding volume.
- For example, think about an algorithm to find bounding volumes of type…
  - AABB (trivial)
  - Sphere – i.e. a "bounding sphere" (less trivial)
  - Capsule (difficult!)

67

Dirgression: collision detection in traditional 2D sprite-based games

- An easier problem
- We can leverage collision detection for 2D sprites ← in screen space
  - *it's accurate:* «pixel perfect»
  - *it's efficient:* HW supported (hard-wired support, as part of sprite rendering)
  - little need for proxy approximations for colliders (same structure – the sprite – both for collision and for rendering)
  - easy bounding "volume": bounding-rectangle of the sprite

68