



Course Plan



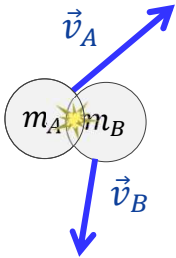
- lec. 1: **Introduction** ●
- lec. 2: **Mathematics** for 3D Games ●●●●●
- lec. 3: **Scene Graph** ●
- lec. 4: **Game 3D Physics** ●●●● + ●●●
- lec. 5: **Game Particle Systems** ●
- lec. 6: **Game 3D Models** ●
- lec. 7: **Game Textures** ●●
- lec. 8: **Game 3D Animations** ●●●
- lec. 9: **Game 3D Audio** ●
- lec. 10: **Networking** for 3D Games ●
- lec. 11: **Artificial Intelligence** for 3D Games ●
- lec. 12: **Game 3D Rendering Techniques** ●●

16

(completely) inelastic impact

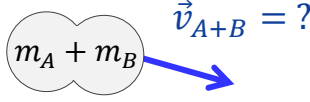


BEFORE:



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

AFTER:



the only unknown, so ...

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

17

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

18

(completely) elastic impact: 1D case

new velocities are defined by the impulses: $v'_A = v_A + \frac{i_A}{m_A}$ $v'_B = v_B + \frac{i_B}{m_B}$ signed scalars

momentum conservation:

$$m_A v_A + m_B v_B = m_A v'_A + m_B v'_B$$

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

$$\Rightarrow i_B = -i_A$$

energy conservation:

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

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

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

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

21

(completely) elastic impact: 1D case

momentum conservation:

momentum & energy conservation:

$$i_B = -i_A \quad (\text{it's just the 3rd law of dynamics})$$

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

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

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

solution 1

$i_A = i_B = 0$

before the impact

solution 2

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

after the impact

22

Some special cases (exercise: verify them)

- Completely **elastic** case (1D):
 - equal masses?
the two velocities just swap
 - **one-way** impact, with A is static?
 v_b just flips
- Completely **inelastic** case:
 - equal masses?
new velocity is the average
 - **one-way** impact, with A static?
B also stops dead

$m_A \rightarrow \infty$
&
 $v_A = 0$

23

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

24

(completely) elastic impact: 3D case

- Additional assumption:
 - \exists **impact plane**, with normal \hat{n}
 - o, in 2D: impact line
 - impulses must be orthogonal to this plane $\vec{i}_{A,B} = i_{A,B} \hat{n}$
- 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**

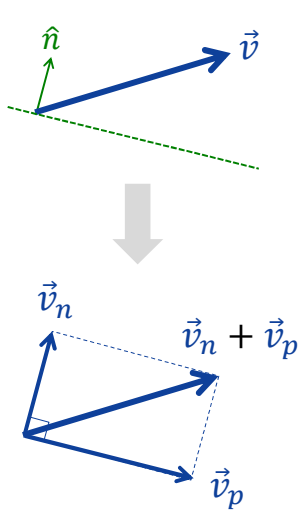
we need this data!

vector impulses scalar impulses, pos. or neg. (the unknowns)

25

A simple geometric subproblem

- Given velocity \vec{v} and the 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:
 - $v_n \leftarrow \vec{v} \cdot \hat{n}$ note: v_n is a signed scalar
 - $\vec{v}_n \leftarrow v_n \hat{n}$ \vec{v}_n is a vector
 - $\vec{v}_p \leftarrow \vec{v} - \vec{v}_n$
- Useful because:
 - only \vec{v}_n is affected by **elastic impacts** with plane
 - only \vec{v}_p is affected by **frictions** with plane (dump it!)
 - v_n is used to *solve* elastic impacts (use 1D case)



26

Impact between rigid bodies(*) - notes

(*) i.e. with *angular velocities* too

- We only have seen impacts between *particles*
 - i.e. we disregarded angular velocities
 - if **rigid bodies** are approximated with particles + distance constraints, then this is all we need to do
 - Effect of elastic / inelastic impacts on angular velocities will be an (approximated) **emerging behavior** 👍
- Impacts between *real rigid bodies* require to *explicitly* compute the two post-impact angular velocities too
- Same principles apply:
 - Angular** momentum: it is *always* preserved too
 - Anelastic impact: post-impact **angular velocities** must also match
 - Elastic impact: kinetic **rotational energy** must also be preserved
 - Mixes (bounciness): interpolate **angular velocities** too

27

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**...

- «safe» **positions**
 - overlap-free position for both objects
 - needed to teleport things there
- **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 response

28

Collision Handling



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



29

Collision detection



- The usual problem: efficiency
- Observation:
 - most of objects (by far),
most of time (by far),
do NOT collide.
 - for efficiency,
the case to optimize is the «no-collision» case
 - «early reject» of the text

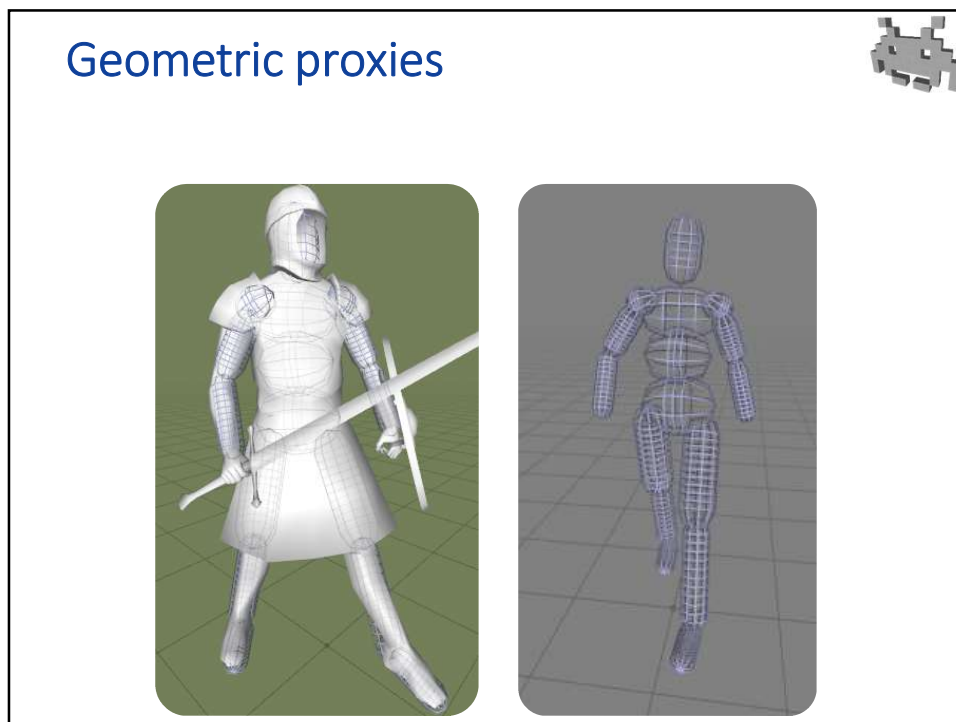
30

Collision detection



- Efficiency issues:
 - a) test between object pairs:
 - Must be efficient
 - b) avoid quadratic explosions
of needed tests
 - N objects $\rightarrow N^2$ tests ?

31



32

Geometric proxies

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

- usually, a *much* cruder approx. than the 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 object**)
 - approximation** of the object spatial extension
 - for *approximate* tests

(“hit-box” is a misnomer: it’s not necessarily a “box”)

33

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 yetAn «early reject» optimization
- if `proxy_A` serves as **Collider** :
 - if NO: no collision
 - if YES: **collision detected** !
 - Must compute **collision data** from `proxy_A`A (lossy) approximation of the collision detection

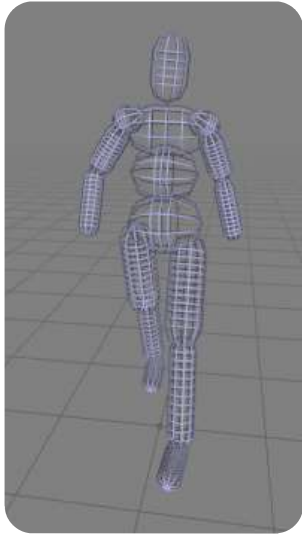
Despite the semantic difference, the same data type can be used for all proxies.

34

Geometric proxies - used not only for collision detection but also:

- **physic engine**
 - collision response
 - computation of barycenter / rotational inertia (assumes uniform specific weight)
- **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)
- **3D sounds**
 - sound absorption (rarely done)

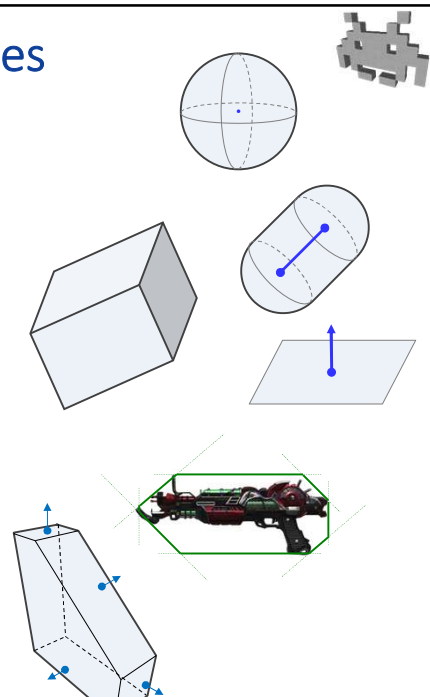
Basically, for any other task except rendering, for a game engine, objects *are* their proxies.



35

Geometric proxies: types

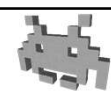
- Sphere
- Capsules
- Half-spaces
- Axis Aligned (Bounding) Box
 - aka AABB
- Generic (Bounding) Box
- Discrete Oriented Polytope
 - aka DOP
- Ellipsoid
 - axis aligned or not
- Cylinders
- Convex polyhedron
- Non-convex polyhedron
 - Meshes
- ...



36

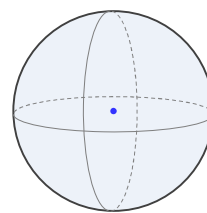
🧠 choosing Geometric Proxies: things to consider

- Workload needed to **compute** / **create** them
- RAM space needed to **store**
- Behavior under **transformations**
 - the ones we plan to use, e.g. isometries
- How good is the geometric **approximation**
 - for the objects we 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 ...
 - also, easy to compute / good is the collision data?



37

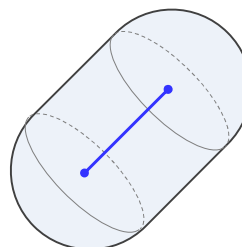
Geometry proxies: Sphere



- 😊 easy to compute automatically
 - 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 are trivial (against anything)
 - 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), but often, quite poor.
 - what about, e.g.: a head? A character? A house? A sword?

38

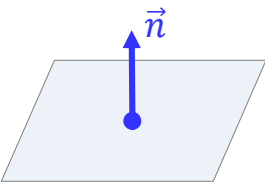
Geometry proxies: «Capsule»



- Generalizes the sphere:
 - Sphere \triangleq the set of points having dist. from a **point** \leq radius
 - Capsule \triangleq the set of points having dist. from a **segment** \leq radius
 - i.e. 1 cylinder ended with 2 half-spheres (all 3 with same radius)
- Stored with:
 - 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 \rightarrow a very popular proxy in games!)

39

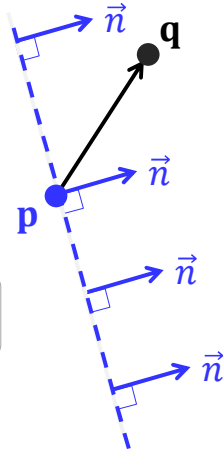
Geometry proxies: a half space



- Trivial, but useful!
 - e.g. for a flat terrain, or a wall...
- 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)

40

Reminder: Plane VS Point test



- Input: a point \mathbf{q} and a plane given by:
 - its normal: \vec{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

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

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

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

$k = -\vec{n} \cdot \mathbf{p}$
 (minus distance of plane from orig.)

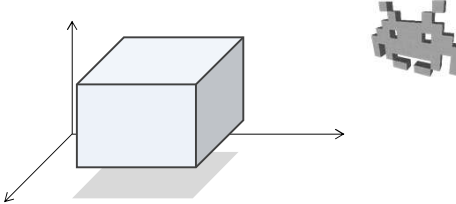
$$(n_x, n_y, n_z, k) \cdot (q_x, q_y, q_z, 1)$$

the vec4 representing the plane

41

Geometry proxies: «AABB»

Misnomer: not necessarily a “bounding” volume:
can be used as a collider too

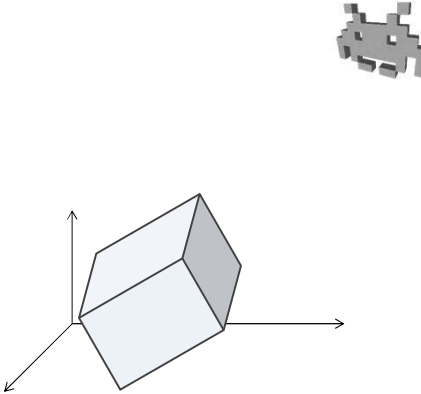


Axis Aligned (Bounding) Box

- Easy to compute / update
- Concise to store
 - Hint: it's three interval: on X, on Y, on Z
- Easy to do collision test...
- Transforms:
 - ☹️☹️☹️ cannot be rotated
 - can be easily scaled / translated

42

Geometry proxies: Box

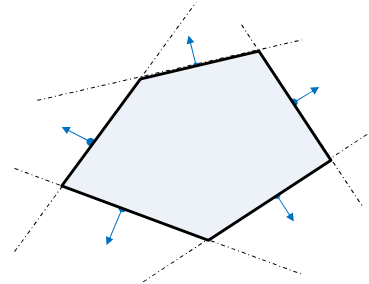


- “Parallelepiped”
 - non axis aligned
 - generalized version of AABB
 - storage:
 - a rotation +
 - an AABB
 - Can be freely transformed
 - note: only if scaling is uniform
 - Tests: a more computations needed

43

Geometry proxies (in 2D): a convex polygon

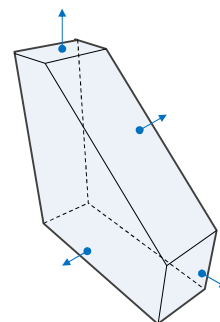
- Intersection of half-planes
 - each delimited by a line
- Stored as:
 - a collection of (oriented) lines
- Test:
 - a point is inside the proxy iff it is in each half-plane
- Flexible (good approximations)... and still moderate complexity



44

Geometry proxies (in 3D): a Convex Polyhedron

- Intersection of half-space
- Similar as previous, 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



45

Geometry proxies (in 3D): a (general) Polyhedron



potentially **concave**

they would be wasted,
as **Bounding Volumes** !

- Luxury **Colliders** :)
 - The most **accurate** approximations
 - The most **expensive** tests / storage
- Specific algorithms to test for collisions
 - requiring some preprocessing
 - and data structures (**BSP-trees**, see later)
- Creation (as meshes):
 - sometimes, with automatic simplification
 - often, hand-designed by artists (low poly modelling)
 - collision proxies are **assets**!
- Similar to a 3D mesh used for rendering?
 - Many differences (compare with mesh, lecture 6)

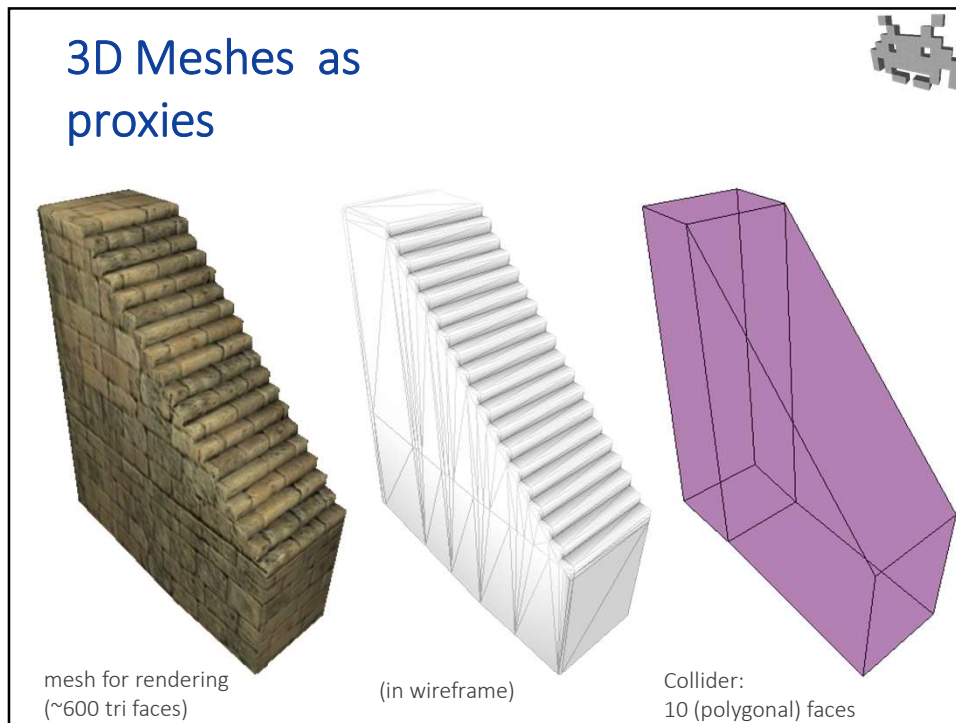
46

Geometry proxies: composite proxies

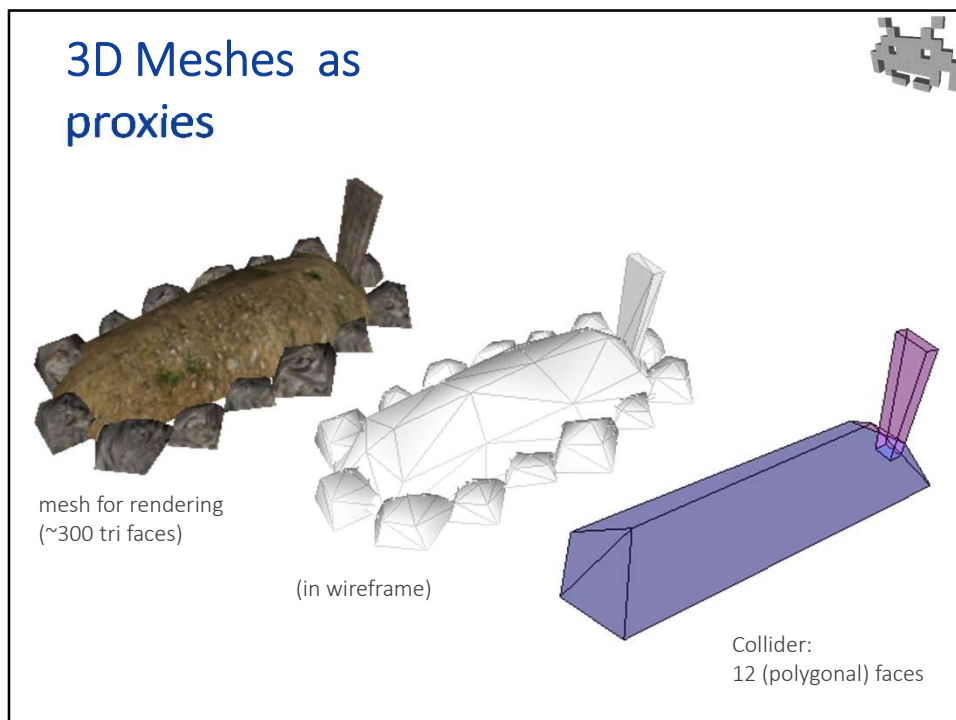


- A proxy can be a union of sub-proxies
 - inside the proxy *iff* inside of *any* sub proxy
- Very expressive
 - better approximation for many objects, even with very few proxies
 - note: union of **convex** proxies can be **concave** !
- Still quite efficient to store / test
- More difficult to construct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

48



49



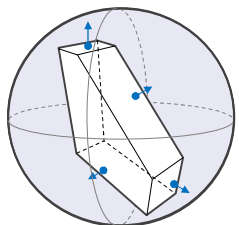
50

Bounding Volume + Collision Object

```

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

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



a simpler **Bounding Volume** around a more complex **Collision Object** approximating the same object

51

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

- an implementation choice of the **Physics Engine**
- # of intersection tests algorithm to be *implemented quadratic with* # of types supported
- supported proxy types can be used as either **Bounding Volumes** or **Hit-Boxes**

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

52

How to construct geometry proxies?



- “Given an object representation M , build an appropriate proxy for it”
 - a M = 3D model of e.g. a dragon, a castle, a character...
- It’s a difficult task to automatize
 - especially for **colliders**
 - it’s a bit easier for **bounding volumes**
 - especially if we want to pick simpler (more efficient) proxies
 - such as collection of a few spheres, capsules, boxes
 - especially if we want good approximations
- It’s often done manually by digital artists

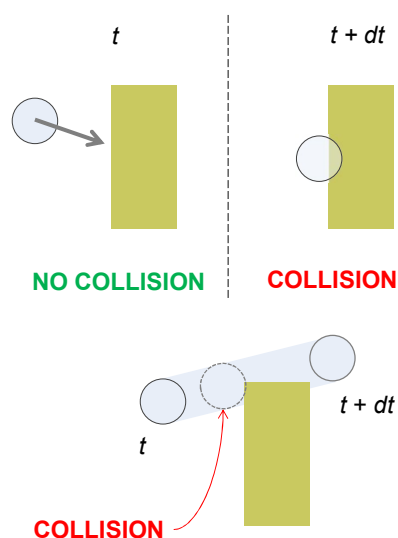
Geometry proxies are **assets** !

53

Collision detection: strategies



- **Static** Collision detection
 - (“a posteriori”, “discrete”)
 - approximated
 - simple + quick
- **Dynamic** Collision detection
 - (“a priori”, “continuous”)
 - accurate
 - demanding



54

Collision detection: Static

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

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)

NO COLLISION NO COLLISION ⊗

55

Collision detection: Dynamic

- Much more accurate detection
- Bonus:
 - no need to «teleport the object in the safe position».
 - it never left a safe position!
 - preventing penetrations easier than curing them.
- Much more difficult to do, too
 - 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, practical to apply only in these cases
 - and when required

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

56

Collision detection in traditional («real») 2D games



- Much easier problem
- We can leverage **collision detection for 2D sprites**
 - *it's accurate*: «**pixel perfect**»
 - *it's efficient*: **HW supported**
(hard-wired support like sprite rendering)
 - no need for **proxy** approximation,
and no much need for optimizations either

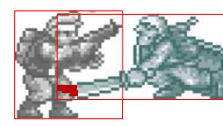
in screen space



NO COLLISION



NO COLLISION



COLLISION

57

Collision detection



- Efficiency issues:
 - a) test between object pairs:
 - Must be efficient
 - b) avoid quadratic explosions
of needed tests
 - N objects $\rightarrow N^2$ tests ?

58

How to avoid a quadratic explosions of needed tests



- Classes of solutions:
 - 1) **spatial indexing** structures
 - 2) **BVH** – Bounding Volume Hierarchies

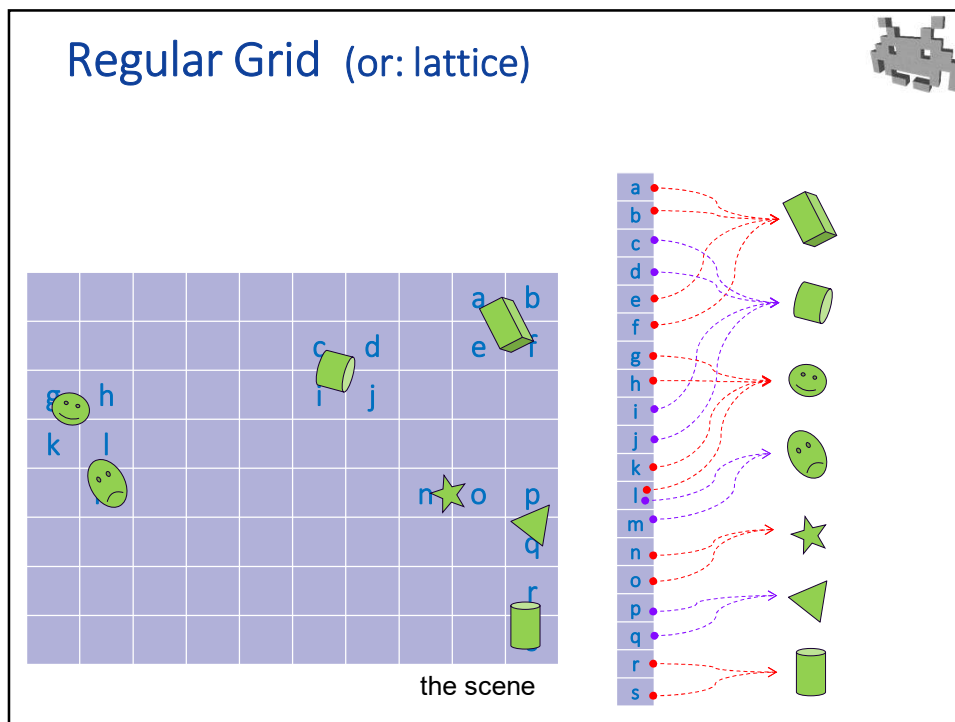
59

Spatial indexing structures



- Data structures to accelerate queries of the kind: “I’m here. Which object is around me?”
- Tasks:
 - (1) construction / update
 - for **static** parts of the scene, a preprocessing. Cheap! ☺
 - for **moving** parts of the scene, an update! Consuming! ☹
 - (another good reason to tag them)
 - (2) access / usage
 - as fast as possible
- Commonest structures (in games):
 - **Regular Grid**
 - **kD-Tree**
 - **Oct-Tree**
 - and it’s 2D equivalent: the **Quad-Tree**
 - **BSP Tree**

60



61

Regular Grid (or: lattice)

- Array 3D of cells (all the same size)
 - each cell = a list of pointers to collision objects
- Indexing function:
 - Point3D \rightarrow cell index, (constant time!)
- Construction: (“scatter” approach)
 - for each object B, find all the cells it touches, add a pointer to B to them
- Queries: (“gather” approach)
 - given query point p , return all object in corresponding cell and adjacent ones
- Difficult choice: cell size
 - too small: memory occupancy explodes
 - too big: too many objects in one cell (not efficient)
- Problem: RAM size
 - Cubic with resolution!
 - Most cells are empty: hash tables can be used to balance efficiency / storage-update cost

62