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

42

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

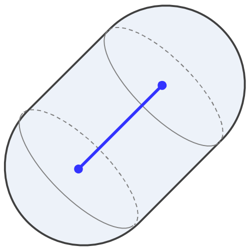
- an implementation choice of the Physics Engine
- # of intersection tests algorithm to be *implemented*
  quadratic with # of supported types
- note: any supported proxy types
  can be used for either Bounding Volumes or Colliders

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

43

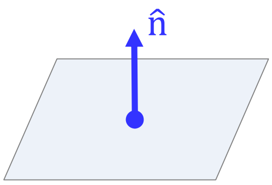
Geometry proxies: «Capsule»



- 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 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 → a very popular proxy in games!)

44

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)

45

Marco Tarini
Università degli studi di Milano

Mini-exercise: Plane VS Point test

No need to normalize it

- 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

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

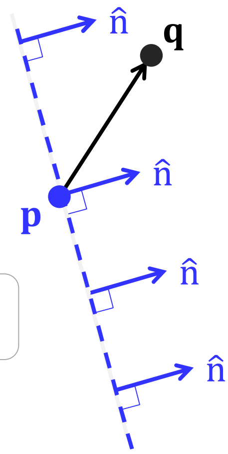
$\hat{\mathrm{n}} \cdot \mathbf{q} - \vec{n} \cdot \mathbf{p} =$

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

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

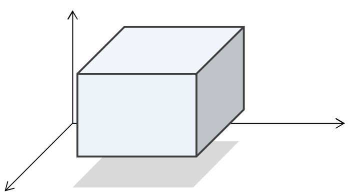
$(n_x, n_y, n_z, k) \cdot (q_x, q_y, q_z, 1)$

a 4D vector representing the plane



46

Geometry proxies: «AABB»



As the name implies, almost always used as BOUNDING volume

Axis Aligned Bounding Box

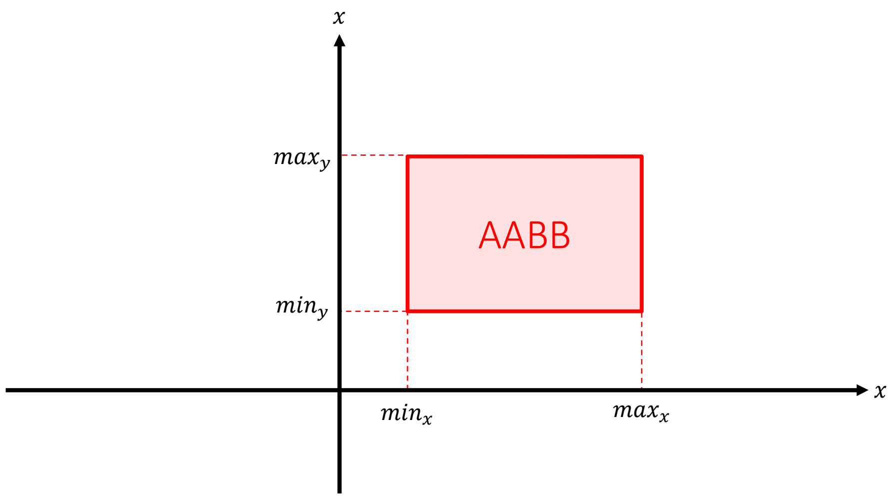
- Consists of three interval

$[min_x, max_x] \times [min_y, max_y] \times [min_z, max_z]$ (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, etc
  - (how?)
- Transforms:
  - ☹☹☹ cannot be rotated
  - But can be easily scaled / translated

47

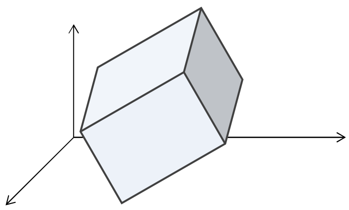
«AABB» : 2D example
(Axis Aligned Bounding... Rectangle)



48

Geometry proxies:
Box

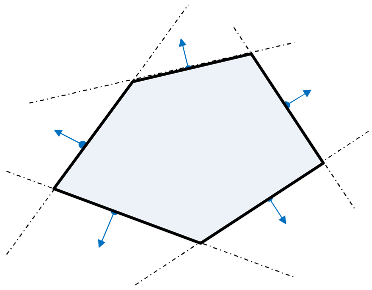
- “Parallelepiped”
  - non axis aligned
  - generalized version of AABB
  - storage:
    - a rotation +
    - an AABB
  - Can be freely transformed
    - note: only if scaling is uniform
  - Tests: a more computations needed



49

Geometry proxies (in 2D): a Convex Polygon

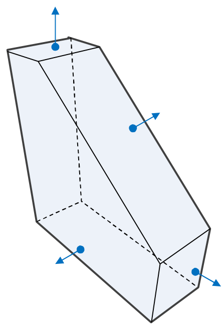
- Intersection of half-planes
  - each delimited by a line
- Stored as:
  - a collection of (oriented) lines
- Test:
  - a point is inside the proxy iff it is in each half-plane
- Flexible (good approximations)… and still moderate complexity



50

Geometry proxies (in 3D): a Convex Polyhedron

- Intersection of half-space
- 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



51

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

potentially concave

not worth it for a Bounding Volume !

- Luxury Colliders :)
  - 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 later)
- 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)

52

3D meshes for geometry proxies vs 3D meshes for rendering

see lecture on 3D models later

- Proxy meshes are
  - much lower res (e.g. < 10^2 faces)
  - no attributes (no uv-mapping, no color, etc)
  - based generic polygons, not just tris (as long as they are *flat*)
  - closed, water-tight (inside != outside)
  - sometimes: convex only
  - completely different internal data structures (e.g. set of bounding planes)

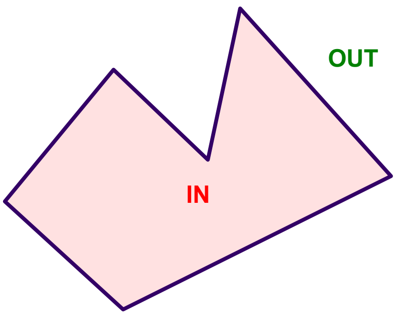
53

BSP-tree (Binary Spatial Partitioning tree)

- A way to store a (convex, or concave) polyhedorn
- A hierarchical structure
  - root = all space, child-nodes = partition of parent
  - a spatial query = traverse the tree from the top down (as usual)
  - a binary tree
  - each internal node is split by an *arbitrary* plane (in 2D: a line)
    - plane is stored at node, as (n_x, n_y, n_z, k)
  - each leaf: one bit: “inside” or “outside” the proxy
  - tree is precomputed (and optimized) for a given Proxy

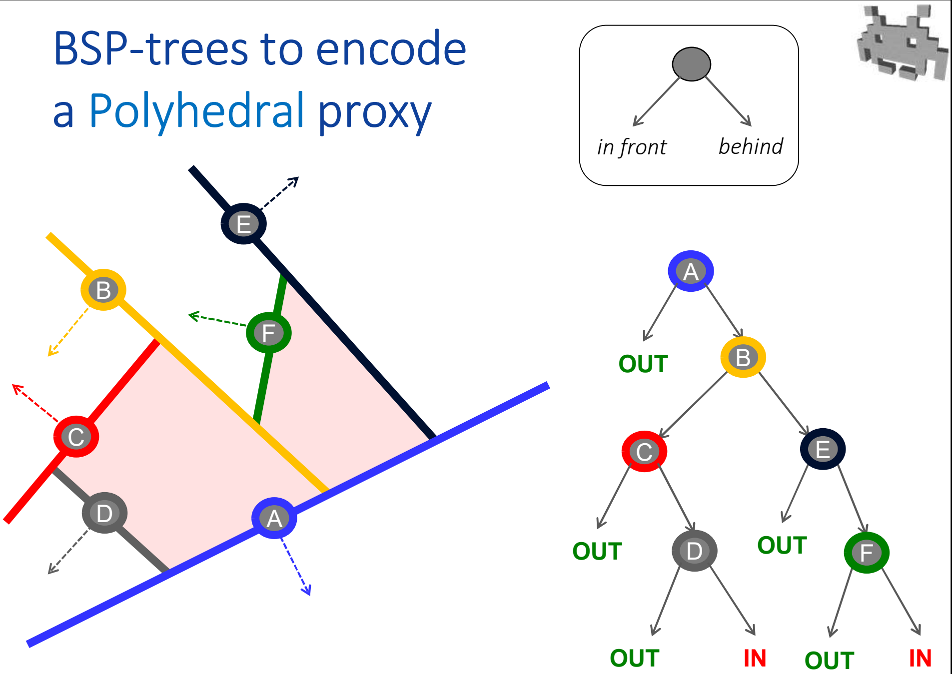
54

BSP-trees to encode a Polyhedral proxy (Concave too)



55

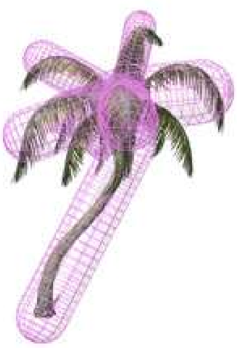
BSP-trees to encode a Polyhedral proxy



56

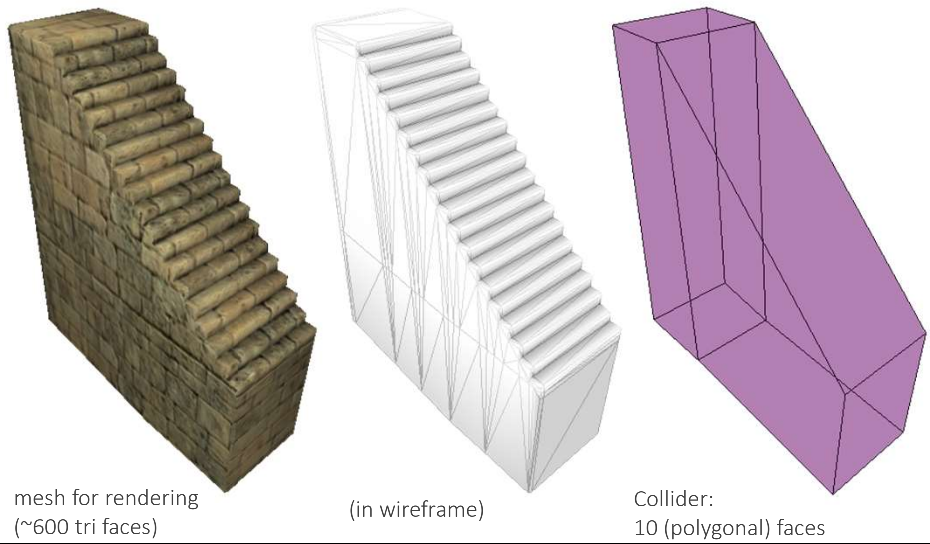
Composite Geometry Proxies

- 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 very few proxies
  - note: union of convex proxies can be concave !
- Still quite efficient to store / test
- Very difficult to construct automatically
  - Open problem!



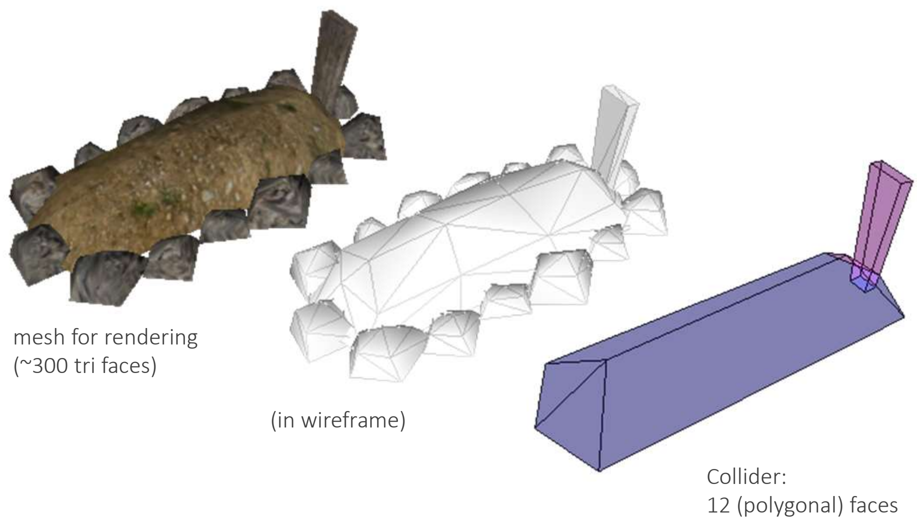
57

3D Meshes as proxies



58

3D Meshes as proxies

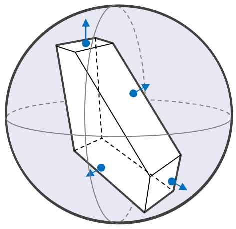


59

Bounding Volume + Collision Object

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

60

How to construct geometry proxies for colliders?

- *"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 !

61

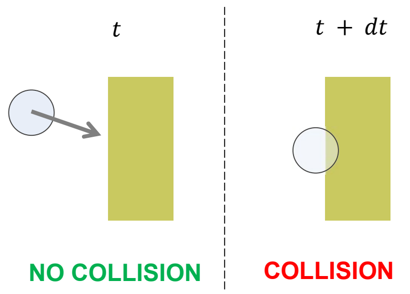
How to construct geometry proxies to be used as bounding volume?

- *"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!)

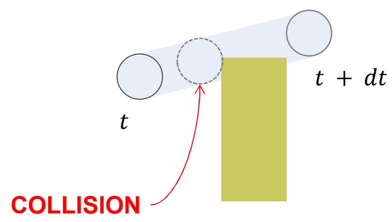
62

Collision detection: strategies

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



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

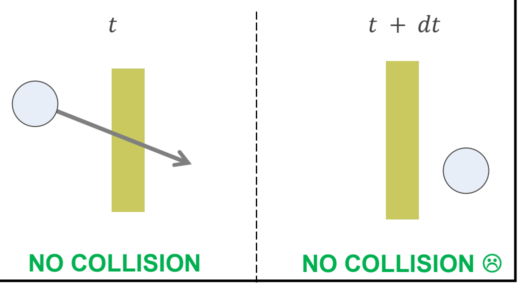


63

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



64

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's 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, only use when required

65

Dirgression: collision detection in traditional 2D games

- A much 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 like sprite rendering)
  - little need for proxy approximations for colliders
  - good proxy for bounding volumes: sprite rectangle

NO COLLISION NO COLLISION COLLISION

66

Collision detection: the broad phase

- Efficiency issues:

a) test between object pairs:
- Must be efficient

b) avoid quadratic explosions of needed tests
- N objects → N^2 tests ?

67

Collision detection: the broad phase

- So far, we have seen how to detect a collision between one given pair of objects
  - Problem: we don't want to test every pair of objects!
- Idea: in a «broad phase», we want to quickly identify pairs of objects that need testing
  - Objects that are safely far from each other are never even tested
  - Only objects that are… "suspiciously close" must be tested
- Note: the board phase must be *strictly* conservative
  - not ok: discard objects that actually collided,
  - ok: *not* discard objects that *didn't* actually collide
- Let's see strategies to do so

68

The «borad-phase» of coll. detection
(avoiding quadratic explosion of # of tests)

- Classes of solutions:

1) spatial indexing structures

2) BVH – Bounding Volume Hierarchies

3) Sorting-based algorithms

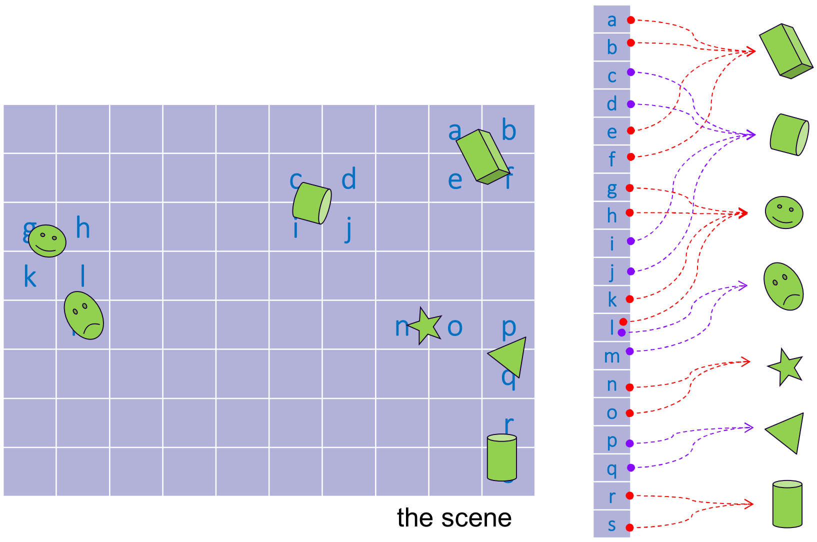
69

Spatial indexing structures

- Data structures to accelerate queries of the kind:
  "I'm in this 3D pos. Which objects are 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

70

Regular Grid (or: lattice)



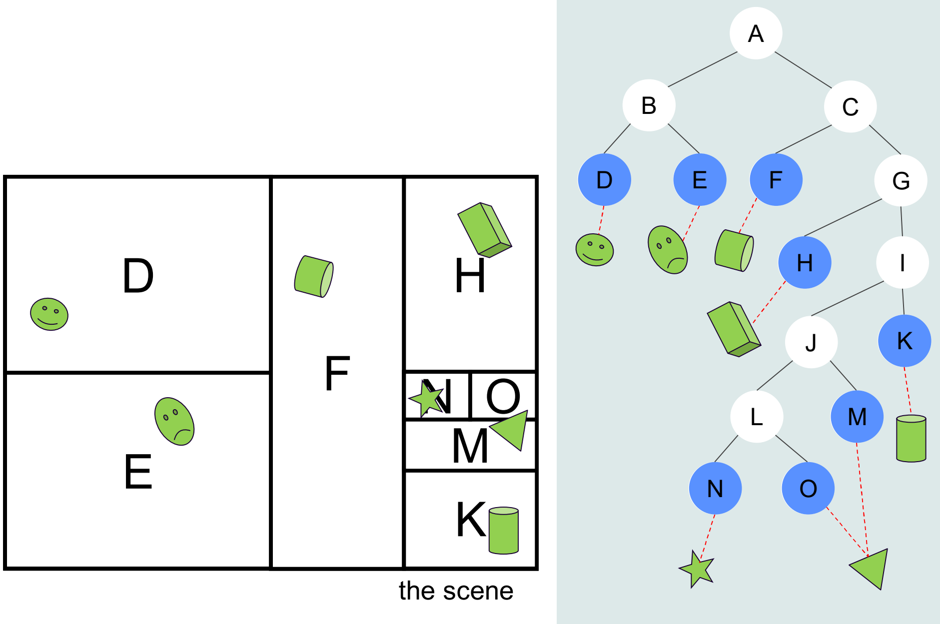
71

Regular Grid (or: lattice)

- Array 3D of cells (all the same size)
  - each cell = a list of pointers to collison objects
- Indexing function:
  - Point3D → 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

72

kD-trees



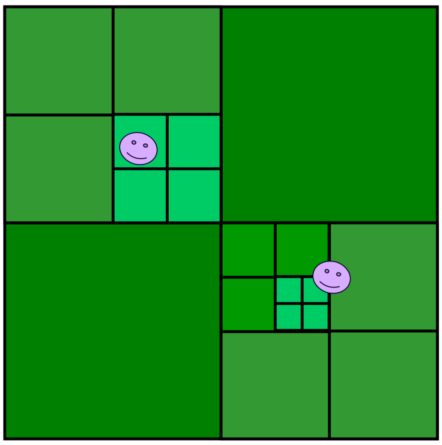
73

kD-trees

- Hierarchical structure: a tree
  - each node: a subpart of the 3D space
  - root: all the world
  - child nodes: partitions of the father
  - objects linked to leaves
- kD-tree:
  - binary tree
  - each node: split over one dimension (in 3D: X,Y,Z)
  - variant:
    - each node optimizes (and stores) which dimension, or
    - always same order: e.g. X then Y then Z
  - variant:
    - each node optimizes the split point, or
    - always in the middle

74

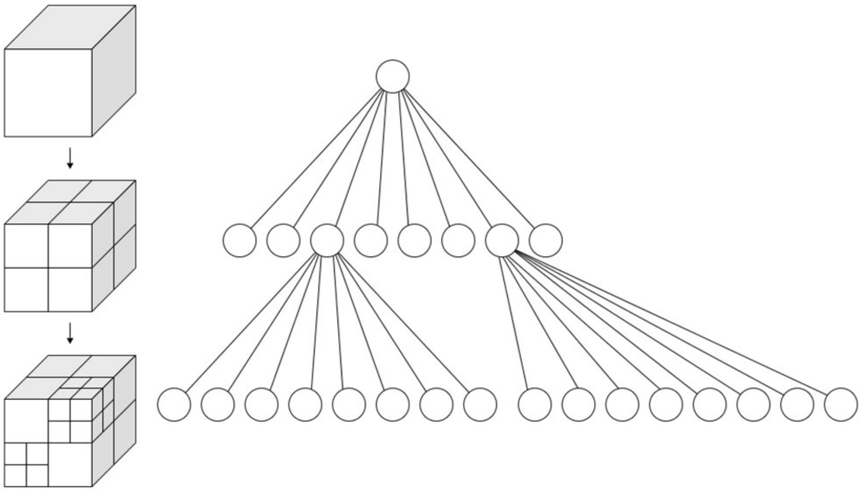
Quad-Tree (in 2D)



the (2D) world

75

Oct Tree
(same, for 3D)



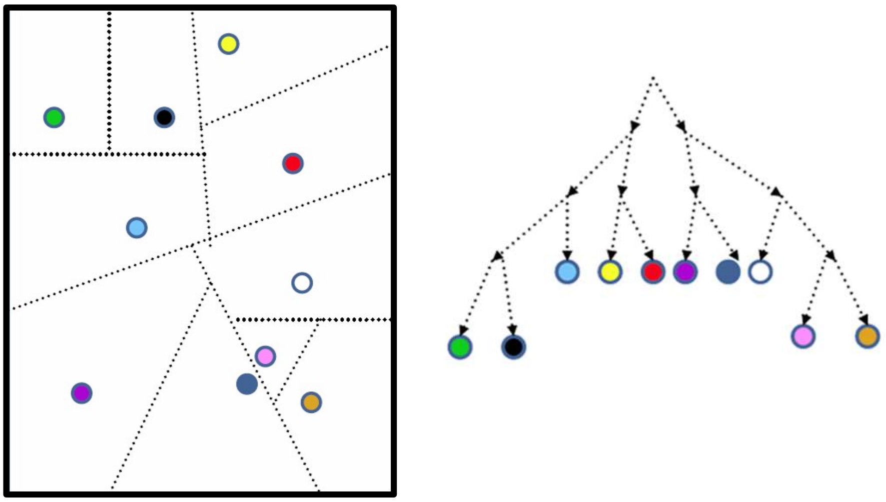
76

Quad trees (in 2D)
Oct trees (in 3D)

- Similar to kD-trees, but:
  - tree: branching factor: 4 (in 2D) or 8 (in 3D)
  - each node: splits into all dimensions at once
    X and Y in 2D
    X and Y and Z in 3D
    (in the middle)
- Construction (just as kD-trees):
  - continue splitting until a end nodes has few enough objects
    (or limit level reached)

77

BSP-tree
Binary Spatial Partition tree


the world

BSP-tree
Binary Spatial Partitioning tree

- Can be used as a hierarchical spatial indexing structure
  - root = all scene, child-nodes = partition of parent (as usual)
  - spatial query = traverse the tree from the top down (as usual)
  - a binary tree (same as as *k*D-trees)
  - each node is split by an *arbitrary* plane (in 2D: a line)
    - plane is stored at node, as (n_x, n_y, n_z, k)
  - planes can be optimized for a given scene
    - e.g., to go for a 50%-50% object split at each node
    - e.g., to leave exactly *one* object at leaves
    - Pro:
      they can be optimized for optimal queries: better query time!
    - Con:
      must be optimized during construction: worse construction time!

The «broad-phase» of coll. detection
(avoiding quadratic explosion of # of tests)

- Classes of solutions:

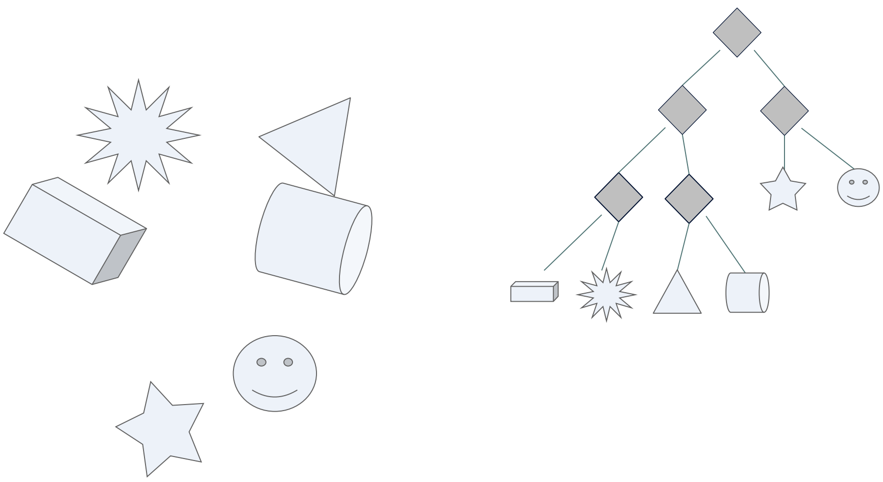
  1) spatial indexing structures

  2) BVH – Bounding Volume Hierarchies

  3) Sorting-based algorithms

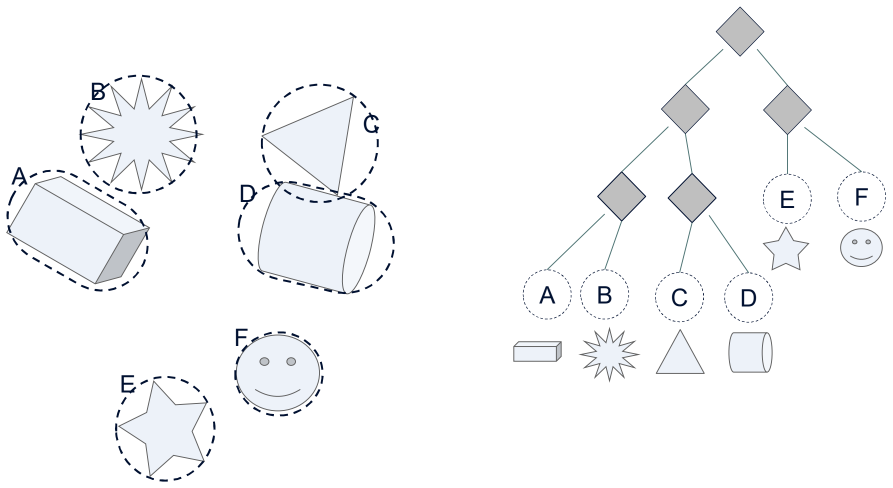
80

BVH
Bounding Volume Hierarchy



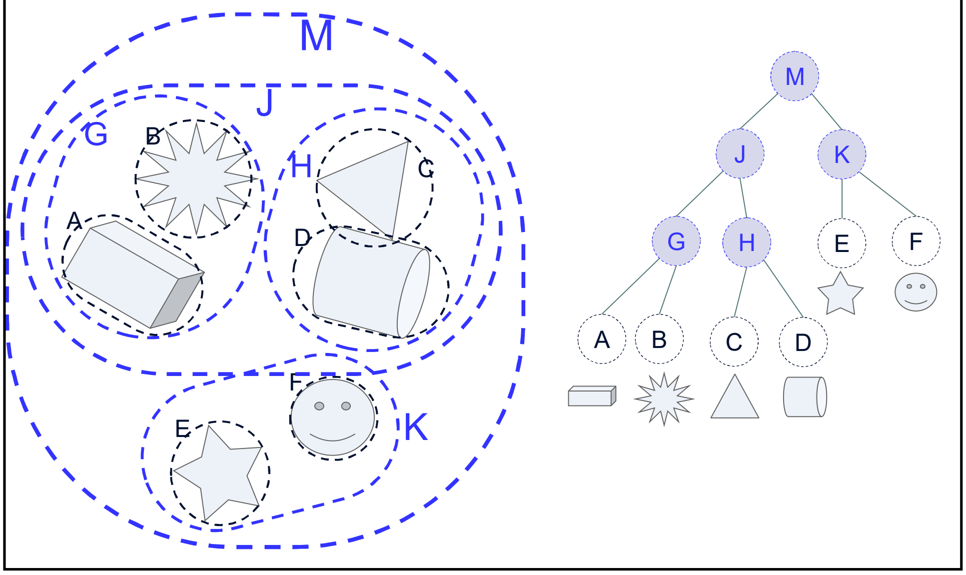
81

BVH – Bounding Volume Hierarchies



82

BVH – Bounding Volume Hierarchies



83

BVH
Bounding Volume Hierarchy

- Idea: use the scene hierarchy given by the scene graph
  - (instead of a spatial derived one)
- associate a Bounding Volumes to each node
  - rule: a BV of a node bounds all objects in the subtree
- construction / update: quick! ☺
  - bottom-up: recursive (how?)
- using it:
  - top-down: visit (how?)
  - *note:* ***not*** a single root to leaf path
    - may need to follow *multiple* children of a node (in a BSP-tree: only one)

84

The «borad-phase» of coll. detection
(avoiding quadratic explosion of # of tests)

- Classes of solutions:

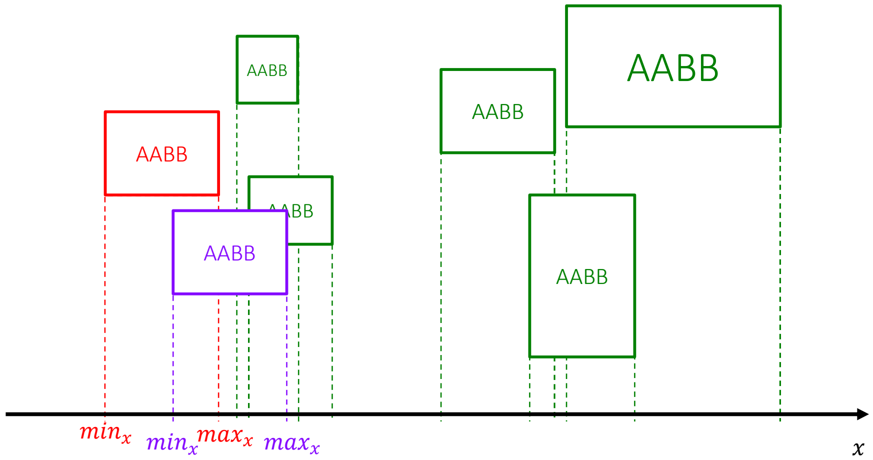
  1) spatial indexing structures

  2) BVH – Bounding Volume Hierarchies

  3) Sorting-based algorithms

85

Sorting based algorithms Sweep and Prune (SAP)



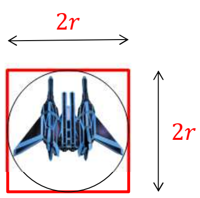
86

Sweep And Prune (SAP) strategy, for broad collision detection phase

- Preliminary:
  - Find the AABB for each object
  - To make it possible to rotate them, use (cubic) AABB encapsulating the Bounding Sphere
- Sort min_x and max_x of all AABB together
  - Just adjust the sorting used in the previous frame
  - It will be already *almost* sorted! To exploit this…
  - use an *incremental* sorting algorithm, such as quicksort
- Sweep the sorted intersections, from smaller to larger
  - Quickly detect intersecting intervals in x (how?)
- Among AABB intervals, prune the ones that don't *also* intersect in y and z
  - Only these objects need further testing for collisions

Fast! $O(n \log n)$

Even faster! $O(n)$



87

Physics Engine: an implementation issue for GPU

- Task: Dynamics
  - (forces, speed and position updates…)
  - simple structures, fixed workflow
  - highly parallelizable: GPU possible
- Task: Constraints Enforcement
  - still moderately simple structures, fixed workflow
  - problem: collision constraints not know a-priori
  - still highly parallelizable: hopefully, GPU possible
- Task: Collisions Detection
  - non-trivial data structures, hierarchies, recursive algorithms, sorting…
  - hugely variable workflow
    - e.g.: quick on no-collision, more work to do when the rare collisions occur
  - difficult to parallelize: CPU
  - but the outcome affects the other two tasks (e.g., creates constraints)
    - ==> CPU-GPU communication, and ==> GPU structures updates (problematic on many architectures)

89

End of Game Physics part. To gather more info…

- Erwin Coumans
  SIGGRAPH 2015 course
  http://bulletphysics.org/wordpress/?p=432
- Müller-Fischer et al.
  Real-time physics
  (Siggraph course notes, 2008)
  http://www.matthiasmueller.info/realtimephysics/

90