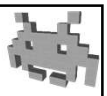
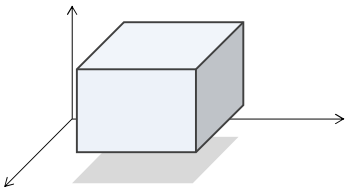
Course Plan



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

63

Geometry proxies: «AABB»



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

Axis Aligned Bounding Box

- Consists of three interval

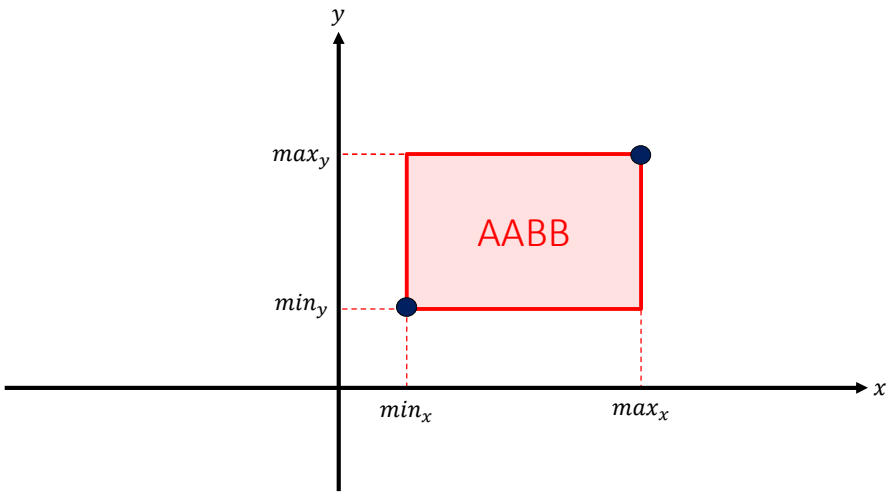
Cartesian product

$$[min_x, max_x] \times [min_y, max_y] \times [min_z, max_z]$$

- 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)

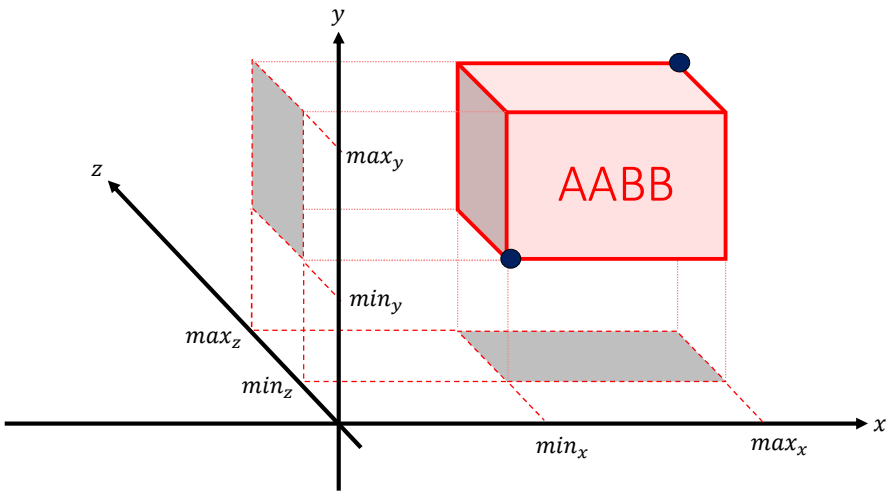
65

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



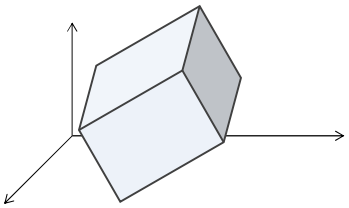
66

Axis Aligned Bounding Box



67

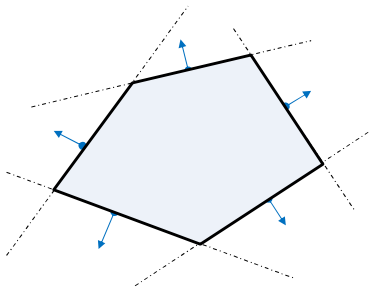
Geometry proxies: Oriented Bounding Box (OBB)



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

69

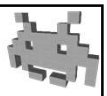
Geometry proxies (in 2D): a Convex Polygon



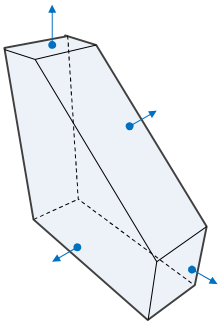
- Intersection of half-planes
  - each delimited by a line
- Stored as:
  - a collection of (oriented) lines
- Test:
  - a point is inside the proxy iff it is in each half-plane
- Flexible (good approximations)… and still moderate complexity

71

Geometry proxies (in 3D): a Convex Polyhedron



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

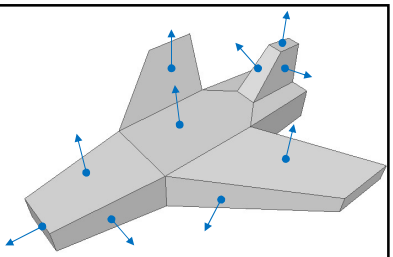


73

Geometry proxies a (general) Polyhedron

potentially concave

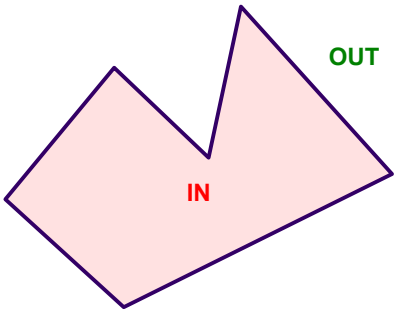
not worth it for a Bounding Volume !



- 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*)
- Creation (treat them as meshes):
  - sometimes, with automatic simplification
  - often, hand-designed by artists (low poly modelling)
- Wait, is this the same as a 3D mesh used for rendering?
  - Many differences (compare with mesh later, lecture 6)

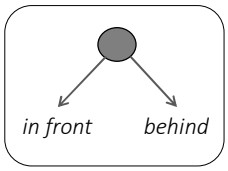
75

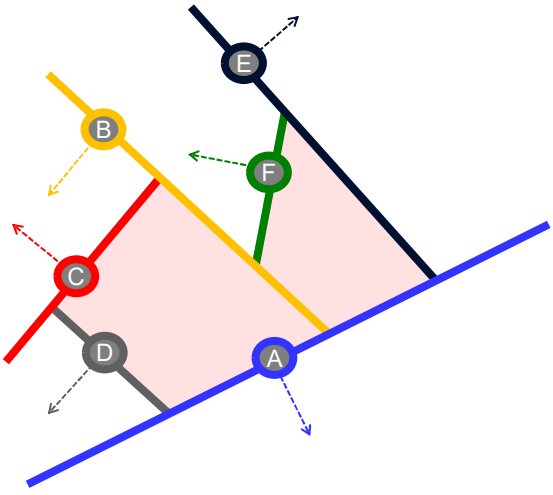
BSP-trees to encode a Polyhedral proxy (Concave too)

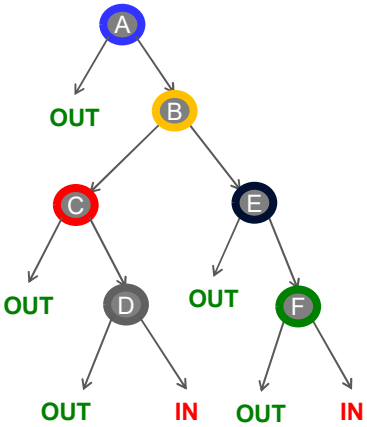


76

BSP-trees to encode a Polyhedral proxy

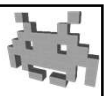






77

BSP-tree (Binary Spatial Partitioning tree)



- A way to store a (convex, or concave) polyhedron
- A hierarchical structure
  - a binary tree
  - root = all space, child-nodes = partition of parent
  - each internal node is split by an *arbitrary* plane (in 2D: a line)
  - plane stored as (n_x, n_y, n_z, k)
  - each leaf stores one bit: "inside" or "outside" the proxy
  - tree is precomputed (and optimized) for a given polyhedron
  - to test a point = traverse the tree root-to-leaf

78

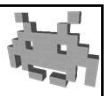
3D meshes for geometry proxies vs 3D meshes for rendering (notes)

(see lecture on 3D models later)

- Proxy-meshes are
  - much lower res (e.g. < 10^2 faces)
  - no attributes of course (no uv-mapping, no color, etc)
  - made of generic polygons, not just tris (as long as they are *flat*)
  - always closed, water-tight (inside != outside)
  - very different internal representation:
    a set of bounding planes (in a BSP tree probably)
    in addition to collection of vertices (3D points)

79

Collision detection on Polyhedral proxies: examples



- Point VS Polyhedron:
  just follow the tree, end in an IN or OUT leaf
- Sphere VS Polyhedron: more complex (think about it)
- Segment / Ray VS Polyhedron: also complex (think about it)
- Polyhedron VS Polyhedron: much more complex.
  A trace of an algorithm is:
  - Preprocessing: find and store all edges (segments) of all Polyhedra (each edge: two endpoints)
  - At testing time: test all edges of polyhedron A vs polyhedron B (segment VS polyhedron), and viceversa

80

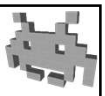
3D meshes for geometry proxies vs 3D meshes for rendering (notes)

see lecture on 3D models later

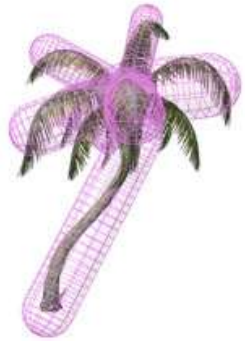
- Proxy meshes are
  - much lower res (e.g. < 10^2 faces)
  - no attributes (no uv-mapping, no color, etc)
  - based generic polygons, not just tris (as long as they are *flat*)
  - closed, water-tight (inside != outside)
  - different internal representation:
    if convex : a set of bounding planes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

81

Geometry Proxies: Composite

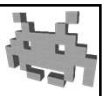


- A proxy can be a union of sub-proxies
  - inside the proxy *iff* inside of *any* sub proxy
- Very expressive
  - better approximation for many objects, even with few proxies
  - note: union of convex proxies can be concave !
- Efficient to test / store
  - Compared to alternratives
- Difficult to construct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

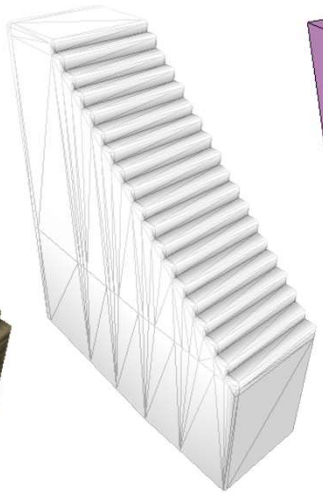


83

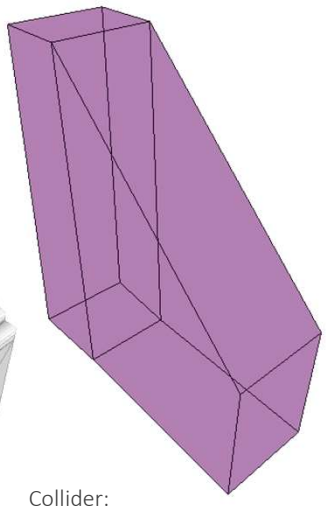
Collision Proxy examples



mesh for rendering
(~600 tri faces)



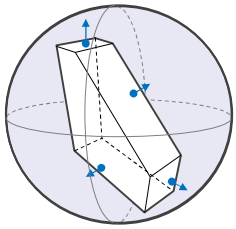
(in wireframe)



Collider:
10 (polygonal) faces

84

Bounding Volume + Collision Proxy
a very good combo



Collision with bounding proxy?

NO → Done! (early reject)

YES → Collision with collider?

NO → Done

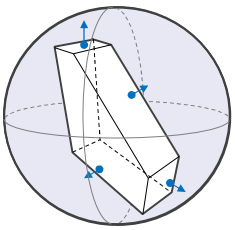
YES → Produce collision data. Trigger response.

86

Bounding Volume + Collision Proxy
a very good combo

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

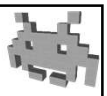
note: **intersect** and **collide** aren't the same function here



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

87

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



"Given an object representation M, build a good collision proxy for it"

- *M* = 3D model of e.g. a dragon, a castle, a character…

difficult task to automatize (by algorithms)

- especially if we want to pick simpler (more efficient) proxies (such as compound of a few spheres, capsules, boxes)
- especially if we need good approximations

often done manually (by digital artists)

Geometry proxies for colliders are assets !

88

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…

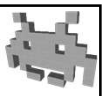
This task can be (and is) automatized

- note: finding the optimal (smallest possible) bounding volume: computationally difficult (can be NP complete)
- find a "good enough" bounding volume: a lot easier (heuristics)
- can be done on the fly during game execution

- For example, think about algorithms to find a bounding volumes of type…
  - AABB (trivial!)
  - Sphere – i.e. a "bounding sphere" (less trivial)
  - Capsule, OBB (more difficult!)

89

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

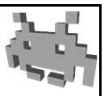


- An easier problem
- We can leverage collision detection for 2D sprites
  - *it's accurate:* «pixel perfect»
  - *it's efficient:* HW supported
    (hard-wired support, as part of sprite rendering)
  - little need for proxy approximations for colliders
    (same structure – the sprite – both for collision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

in screen space

NO COLLISION NO COLLISION COLLISION

Collision detection

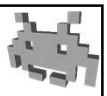


- Efficiency issues:

a) test between object pairs:
  - Must be efficient

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

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!
  - Even excluding static-static pairs: still way too many (quadratic)
- Idea: in a «broad phase», we 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 to discard object pairs that actually collided,
  - ok to test objects that *didn't* actually collide
- Let's see strategies to do so

92

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

- Classes of solutions:

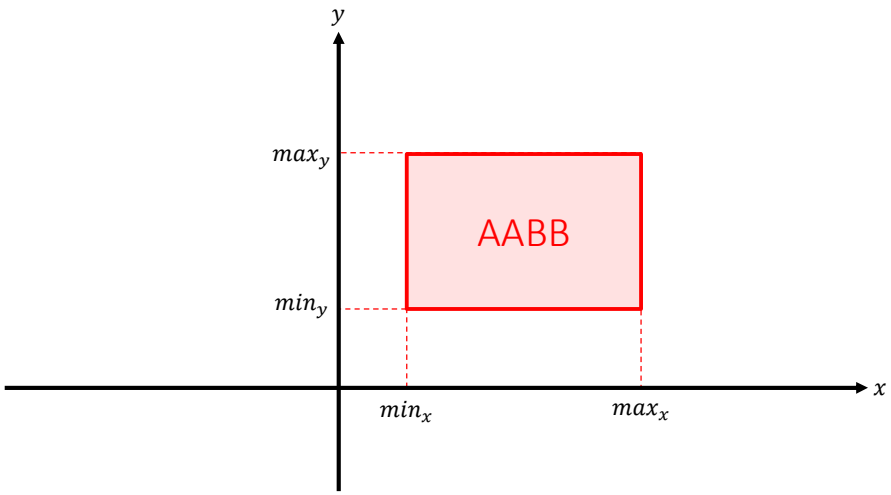
1) Sorting-based algorithms

2) spatial indexing structures

3) BVH – Bounding Volume Hierarchies

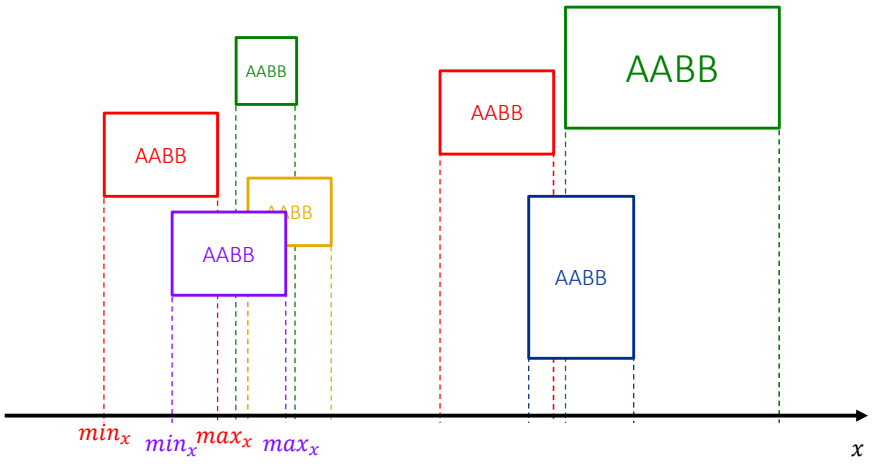
93

One AABB (here, in 2D)



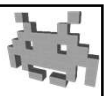
94

Sorting based algorithms
Sweep and Prune (SAP)

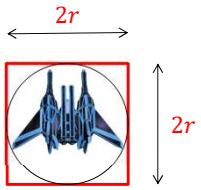


95

Sweep And Prune (SAP) strategy
(aka "Sort and Sweep")



1. Bound:
   - Quickly find the AABB for each collider (in its current rotation + translation)
   - E.g.: use the AABB encapsulating the transformed Bounding Sphere



2. 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, like bubblesort

only $O(n \log n)$

Even faster on almost sorted lists! $O(n)$

3. Sweep the sorted intersections, from smaller to larger
   - Quickly detect intersecting intervals in x (how?)
4. Prune: among AABB intervals, ignore the ones that don't *also* intersect in both y and z
   - Test the pairs that do

96

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

- Classes of solutions:

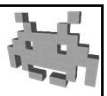
  1) Sorting-based algorithms

  2) spatial indexing structures — Not covered here

  3) BVH – Bounding Volume Hierarchies

97

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



- Classes of solutions:

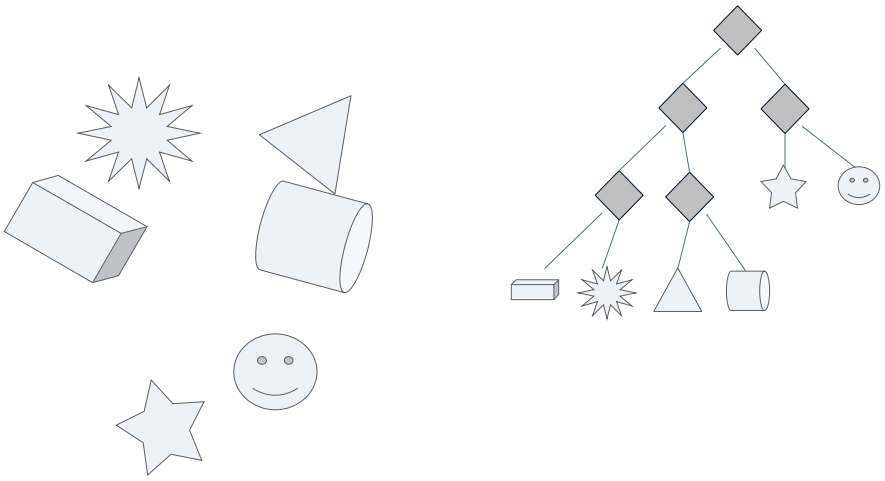
  1) Sorting-based algorithms

  2) spatial indexing structures

  3) BVH – Bounding Volume Hierarchies

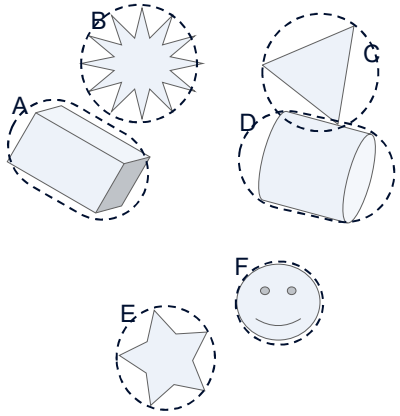
108

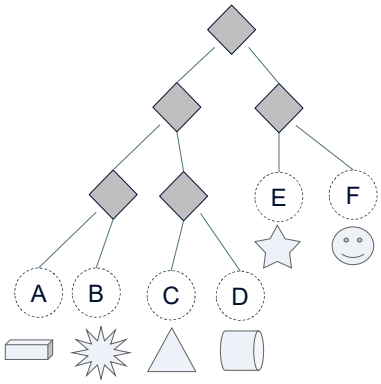
BVH
Bounding Volume Hierarchy



109

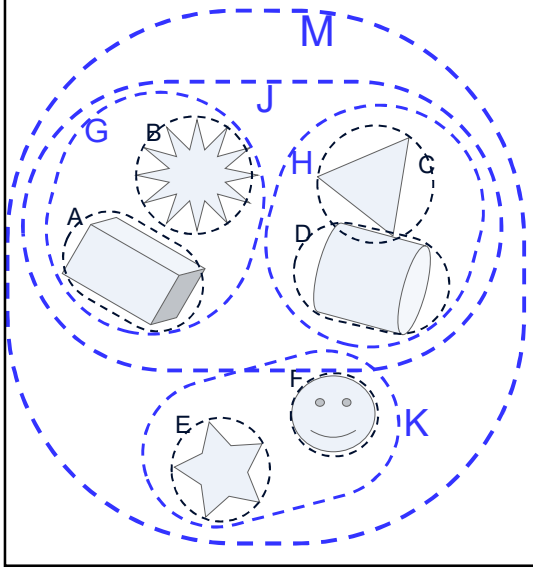
BVH – Bounding Volume Hierarchies

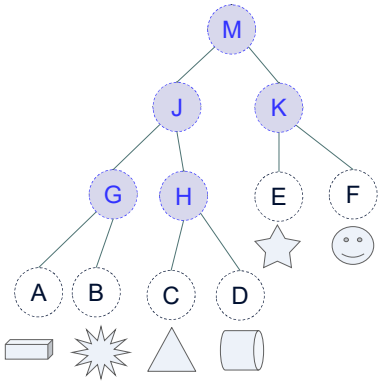




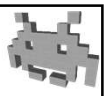
110

BVH – Bounding Volume Hierarchies





111



BVH
Bounding Volume Hierarchy

- We can use the hierarchy already defined by the scene graph
  - instead of a spatially 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
- using it:
  - top-down: visit (how?)
  - *note:* it's ***not*** a single root to leaf path
    - may need to follow *multiple* children of a node (in a BSP-tree: only one)

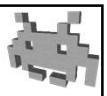
112

Collision Detection:
to learn more…

Christer Ericson (ACTIVISION):
Real-Time Collision Detection
The Morgan Kaufmann Series in Interactive 3-D Technology
HAR/CDR Edition
Elsevier

114

Physics Engine: implementation issues for GPU



- Task: Dynamics
  - (forces, speed and position updates…)
  - simple structures, fixed workflow
  - highly parallelizable: **GPU** implementation easy
  - but: GPU can be non-fully deterministic (different cards, different vendors: different results)
- Task: Constraints Enforcement
  - still moderately simple structures, fixed workflow
  - problem: collision constraints not know a-priori: **GPU** more difficult
- 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)

115

End of Game Physics. 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/
- David H. Eberly:
  Game Physics (2nd Edition)
  MK Press
- Ian Millington:
  Game Physics Engine Development (2nd Edition)
  MK Press

116