ROYAL INSTITUTE
OF TECHNOLOGY

DH2323 DGI18

INTRODUCTION TO
COMPUTER GRAPHICS AND INTERACTION

SCENE MANAGEMENT

Guest Lecturer :

Himangshu Saikia (saikia@kth.se)

CST, KTH Royal Institute of Technology, Sweden

Course responsible :

Christopher Peters (chpeters@kth.se)

Introduction

What is scene management?

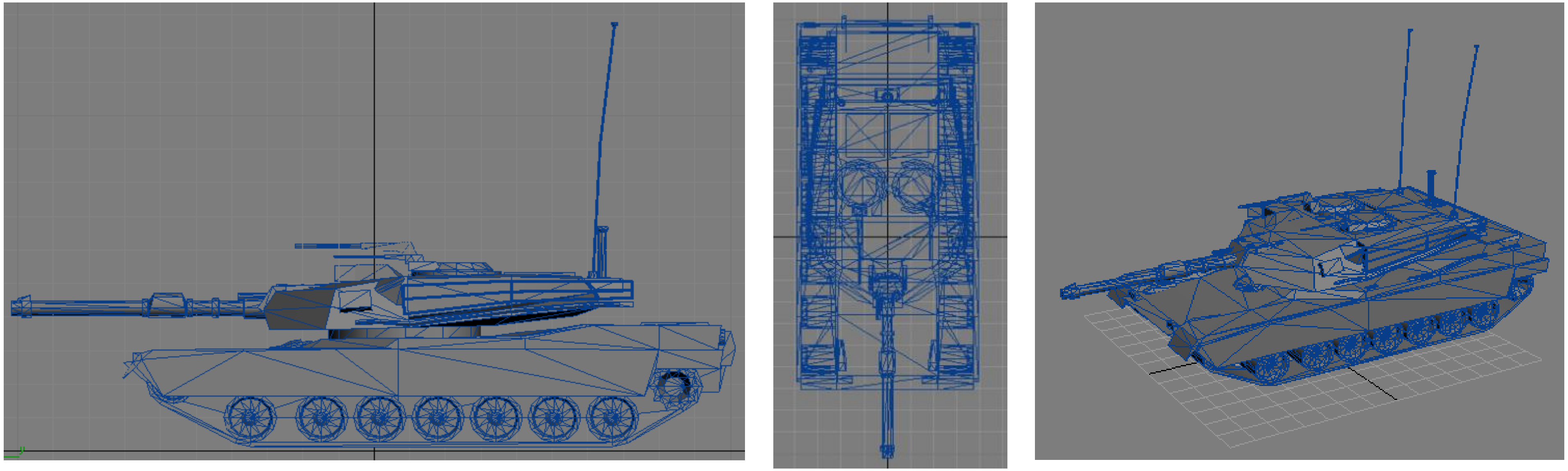
- The reduction of all scene data to a subset of only the data that could possibly be visible from the position of the viewer (*i.e. anything out of sight is not considered for rendering*)

Why is scene management necessary?

- Most graphics (*rendering*) calculations are complex and can be very time-consuming
- Even visibility calculations (*clipping against viewing volume & back-face culling*) can take a long time

Problem

A model contains lots of polygons:



Let’s say we want to figure out if a ray is intersecting with the tank

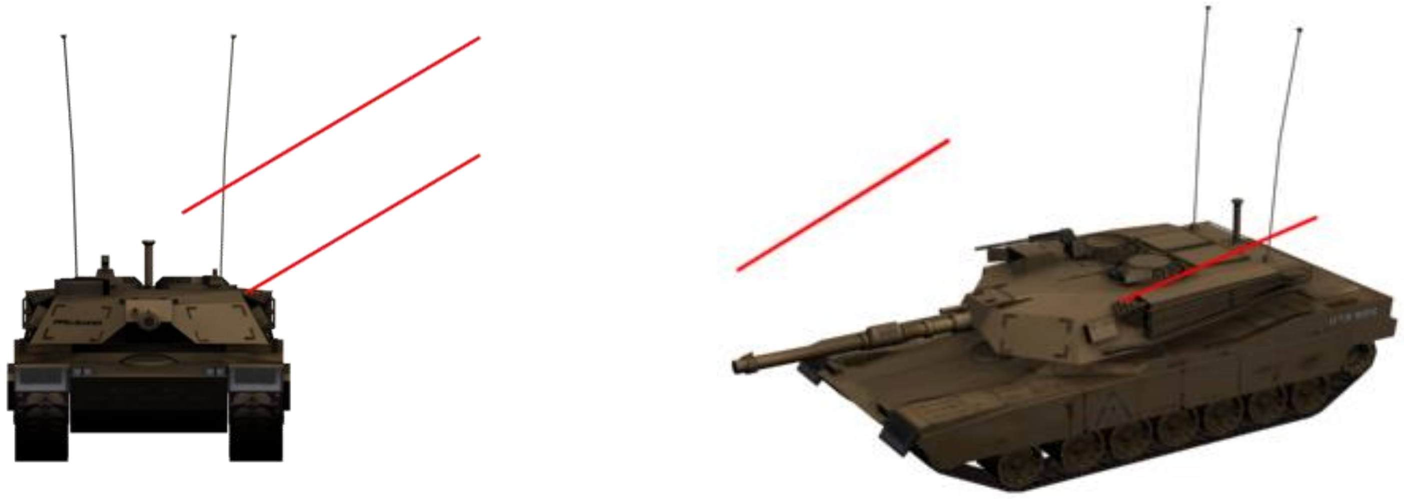
There are many reasons we may want to do this

E.g. the movement of a projectile fired at the tank

Problem

One option: test intersection of the ray with every polygon on the tank --

```
for (every polygon in the tank)
{
    Intersect the line with the polygon
    If the line intersects then collision = true
}
```



Problem

- Speed of the algorithm is **dependent on the number of polygons** in the object
- If a projectile is nowhere near the tank (e.g. is on the other side of the map), the algorithm will still have to check every single polygon in the tank to see if there is an intersection
- Need a quick way to decide whether something is near the tank (*And to be able to reject it very quickly if it is not*)

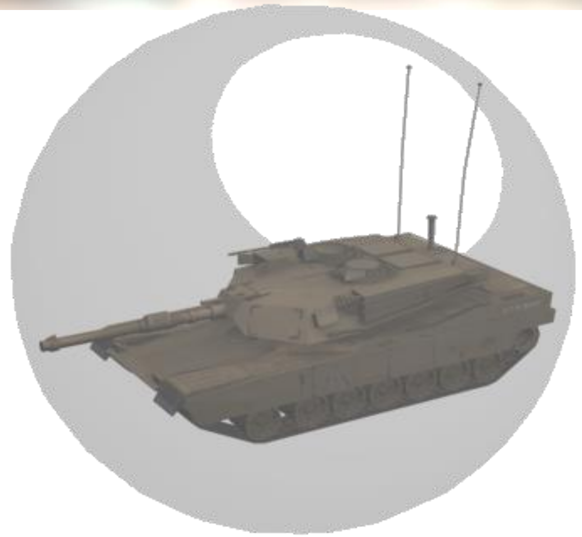
Bounding Volumes

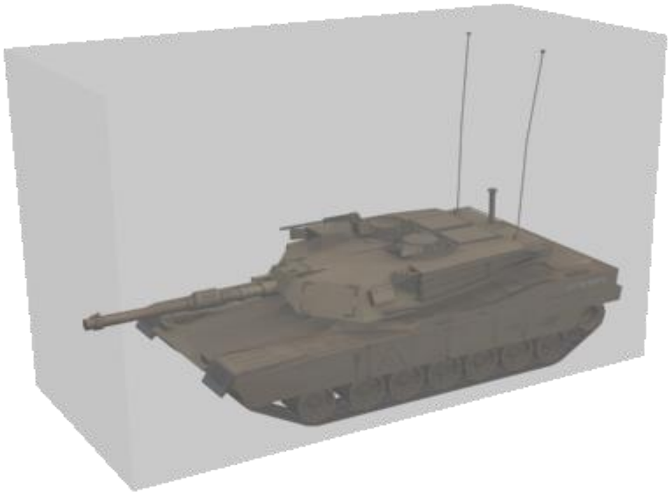
- Bounding volumes(BVs) offer (one part of) the solution
- BVs chosen to enclose all the vertices in a mesh
  - Ensure every triangle (polygon) is contained
- The bounding volume should be made as small as possible
- Different bounding volumes may be more appropriate depending on the layout of the object being fitted

Spheres and Boxes

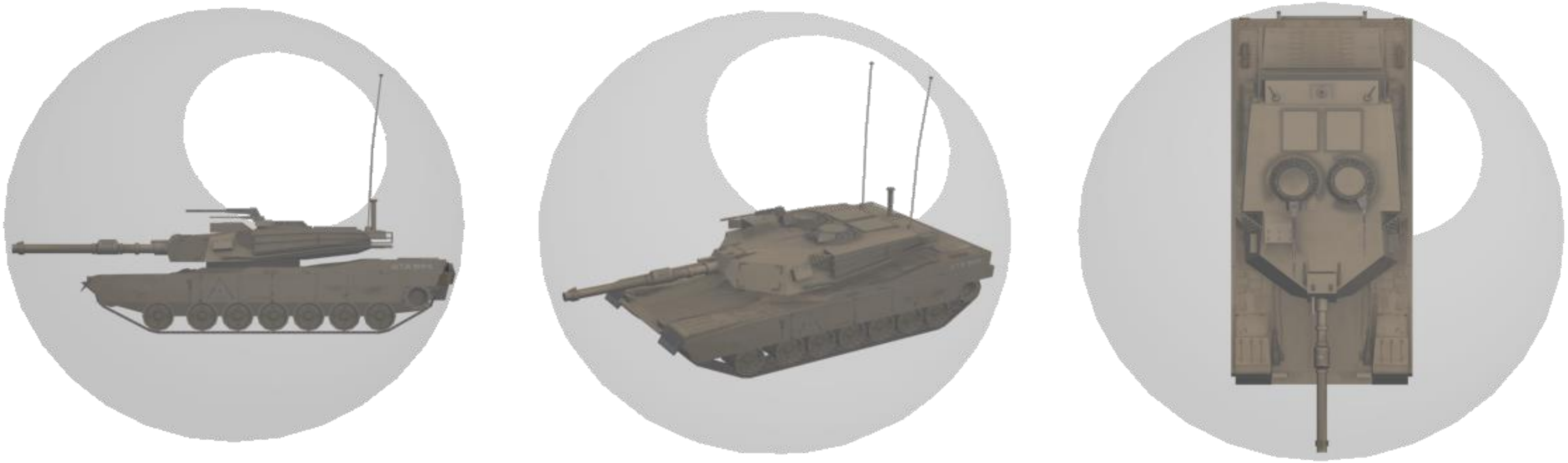
Common bounding volumes – Spheres and Boxes

- Volume definition very simple
- Ability to perform quick calculations - e.g. finding out whether a ray intersects a sphere or a box
- Tests using these bounding volumes are also popular for determining if something is within the view volume when doing visibility testing





Bounding Spheres



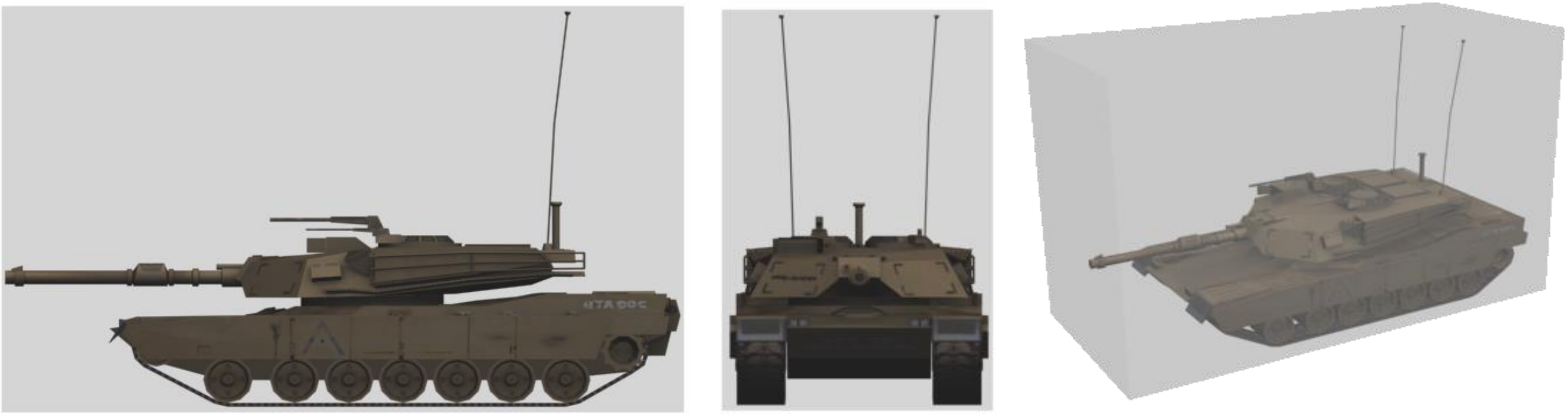
Spheres are a common volume type chosen for bounding objects

Simple representation = extremely fast calculations

Object rotates in the game world = with proper positioning, it is usually not necessary to update the sphere to match the objects new orientation

Bounding Boxes

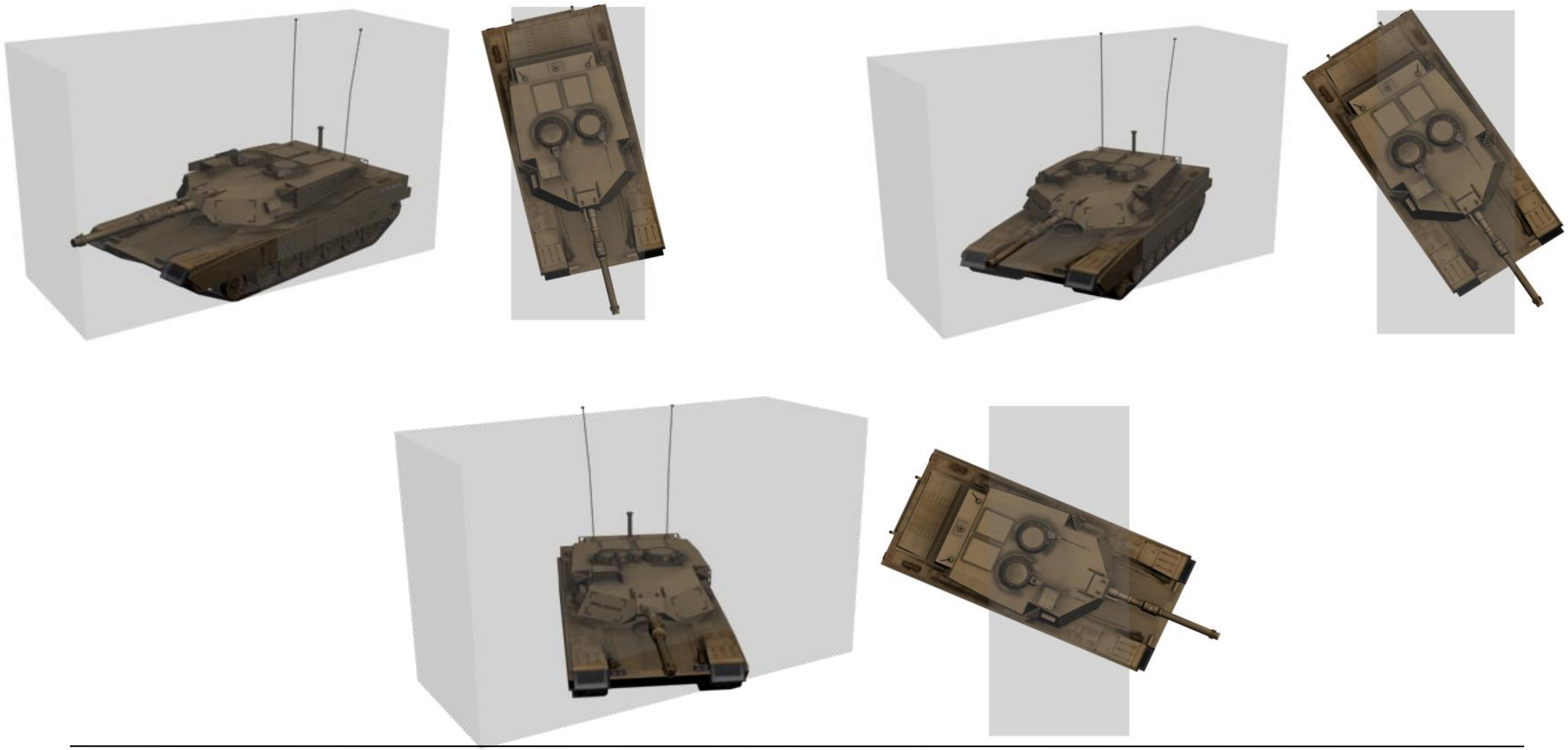
Another popular type of bounding volume unlike spheres, depending on the type of object, they may provide a better fit:



However, it may be slower to do tests against bounding boxes than spheres

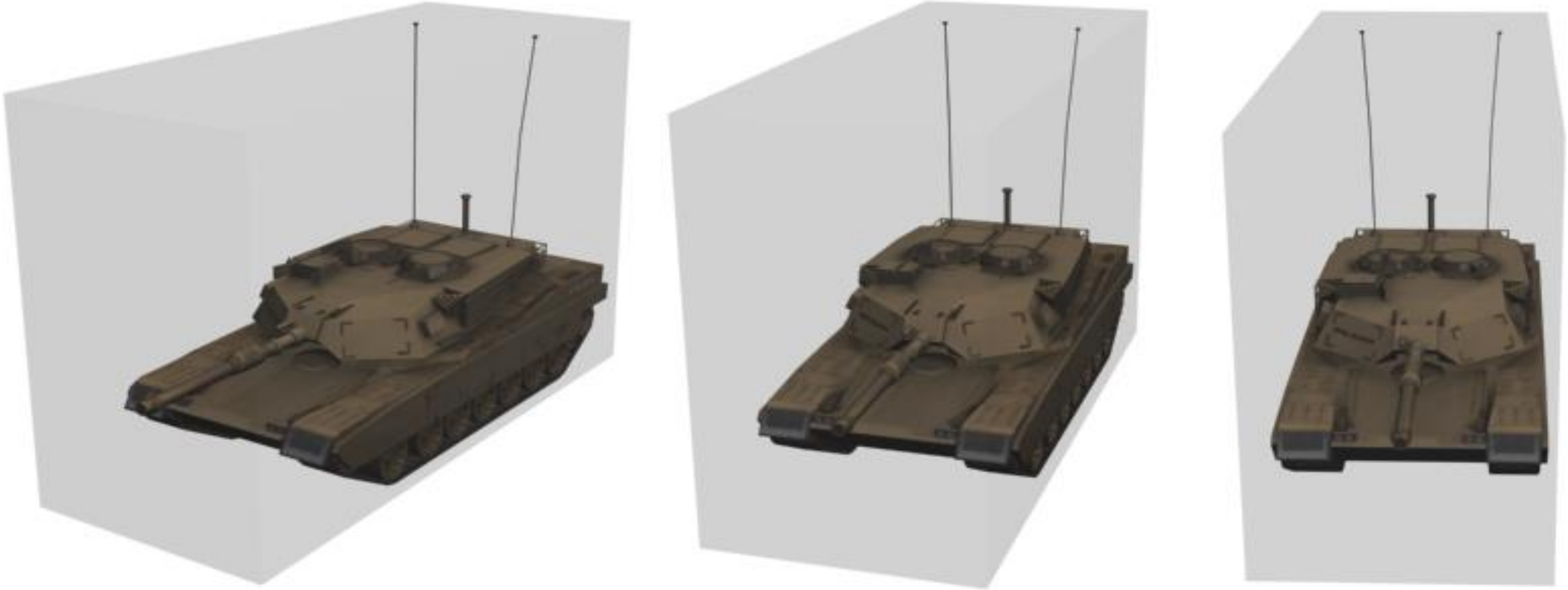
Updating Bounding Boxes

As the tank changes orientation, update the bounding box to ensure it still encapsulates object



OBB’s

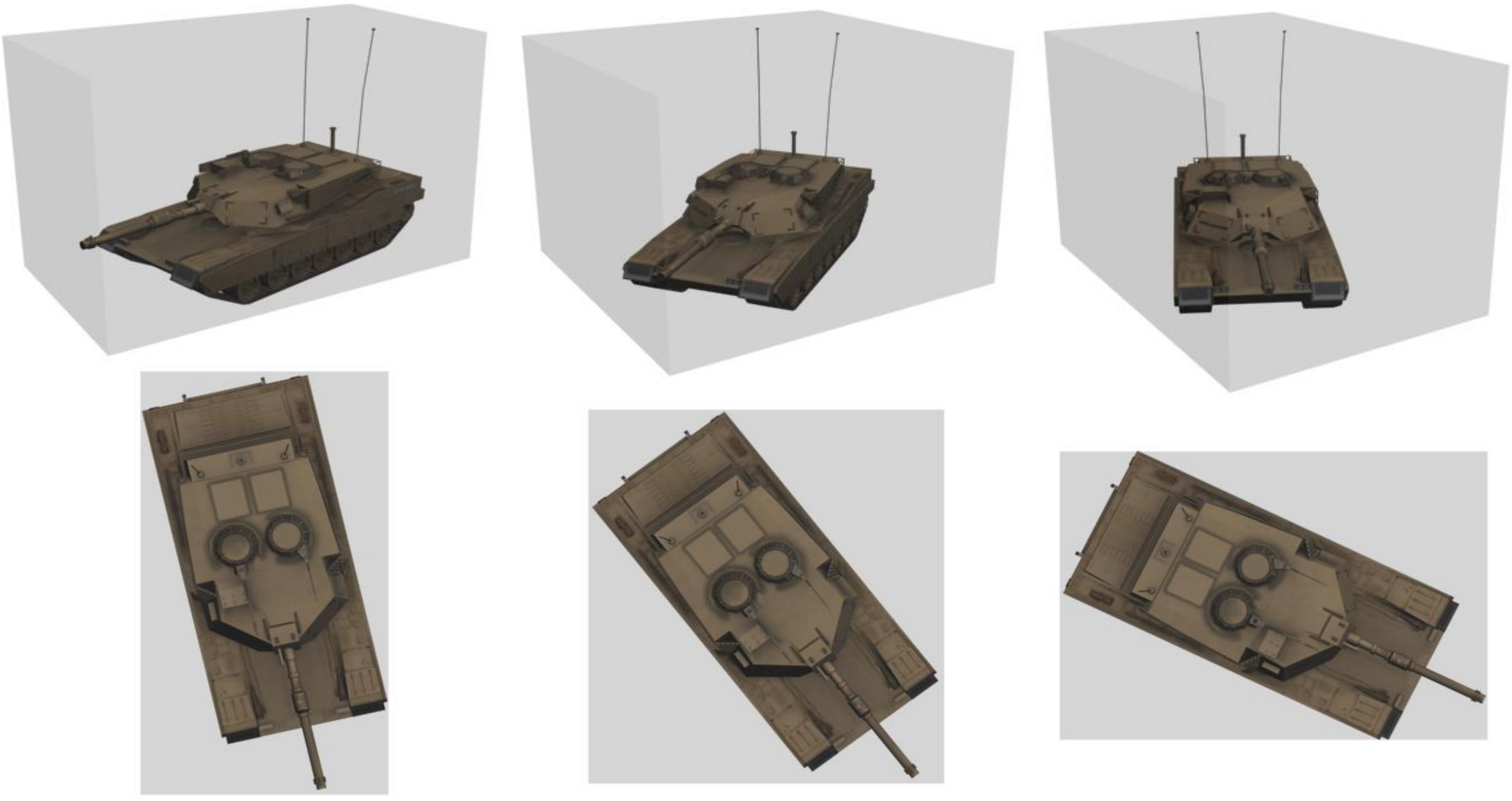
Here, the box is oriented with respect to the tank



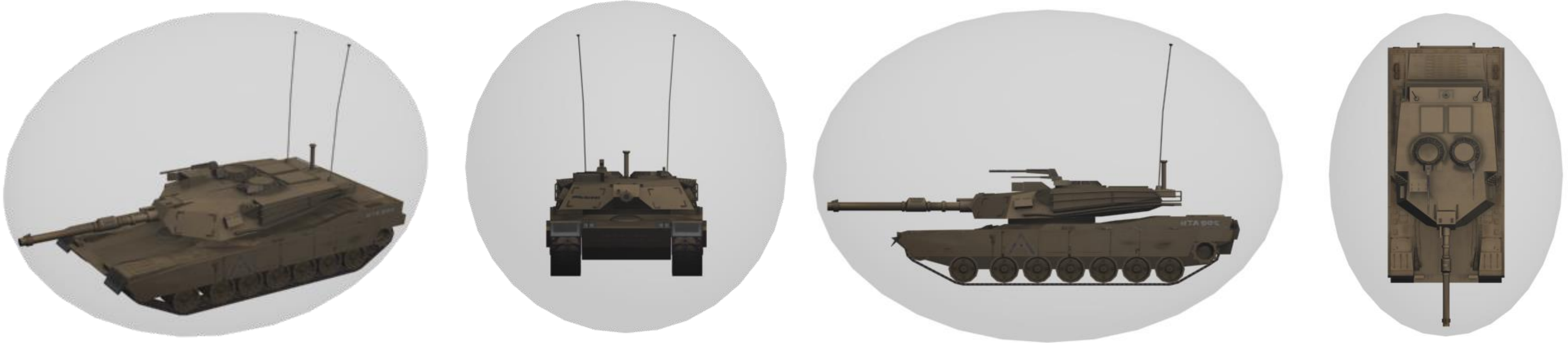
Extra calculations are needed when doing tests against the volume

Axis Aligned Bounding Boxes (AABBs)

Here, the box remains oriented with respect to the main axes



Bounding Ellipsoid



Useful for meshes of some other shapes

Other bounding volumes also possible

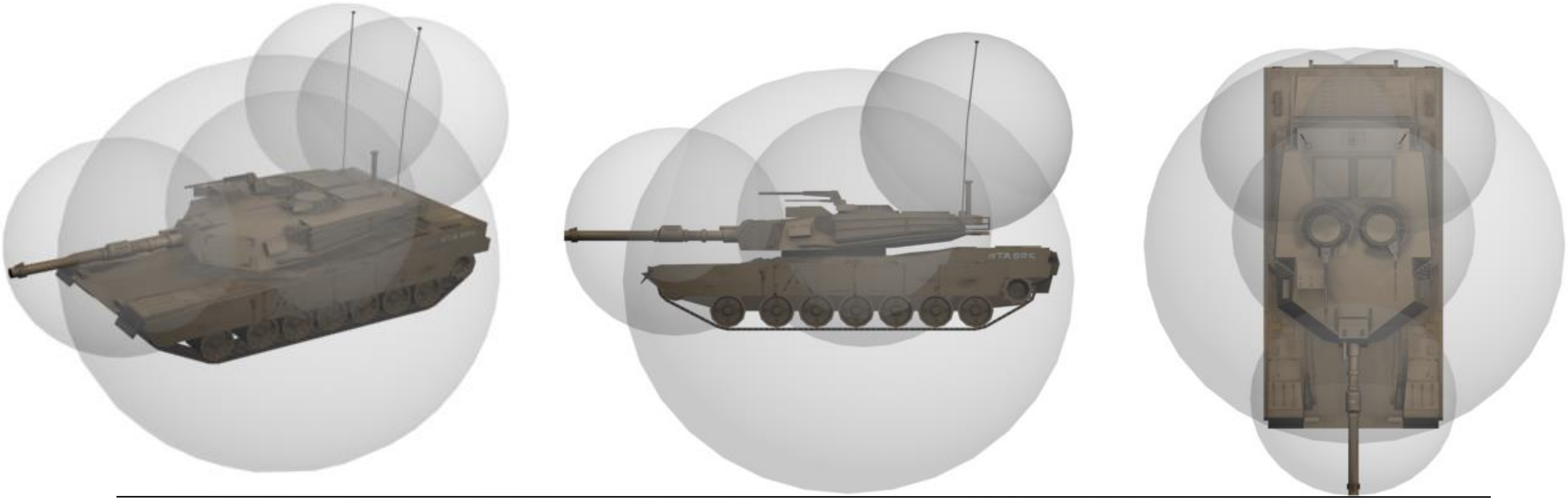
E.g. cylinder

In practice, bounding boxes and spheres are the most commonly used

Bounding Volume Hierarchies

We can also calculate bounding spheres that encapsulate other bounding spheres and bounding volumes for subparts of objects

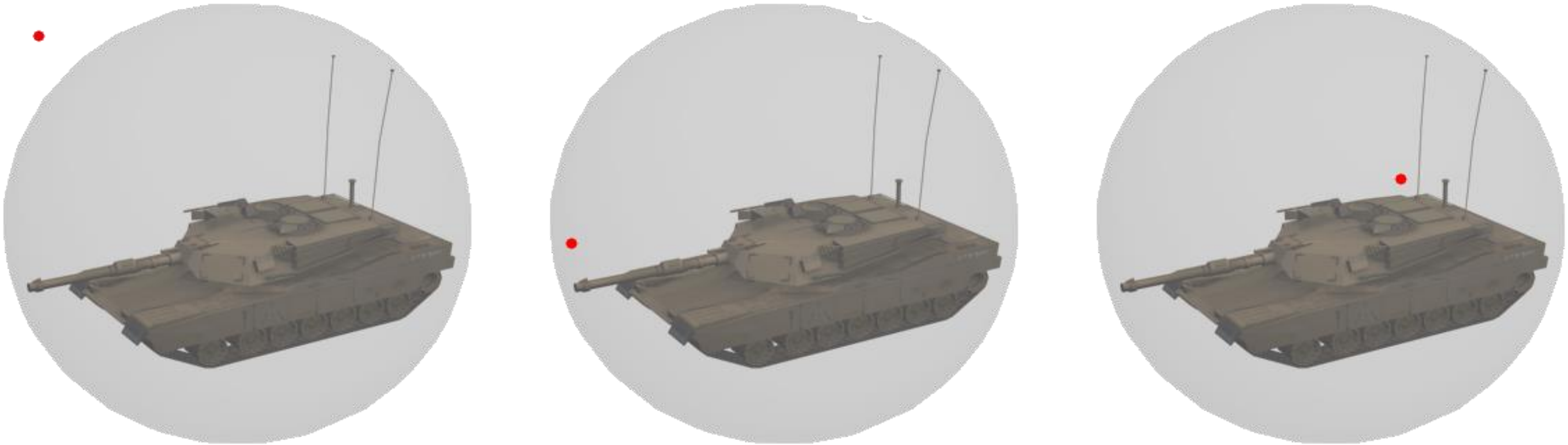
In this example, a separate bounding sphere is created for the turret, gun, body and antennae of the tank:



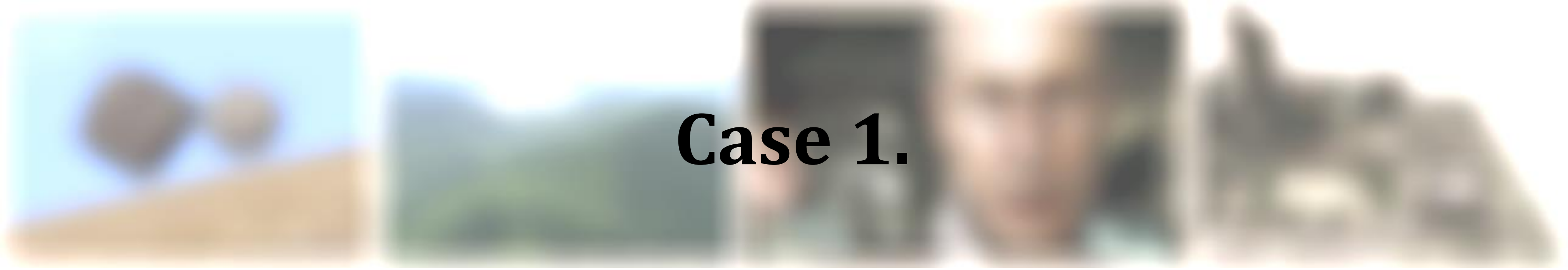
BVH Example Usage

Say we want to quickly test a point to see if it is in a danger zone for our tank

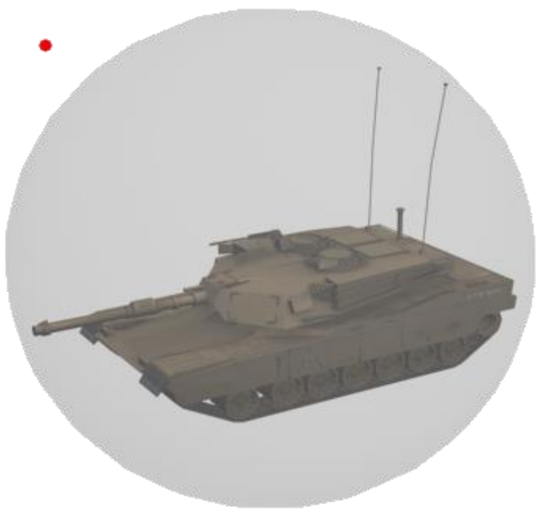
Take three scenarios:



Let’s see what happens in each case…



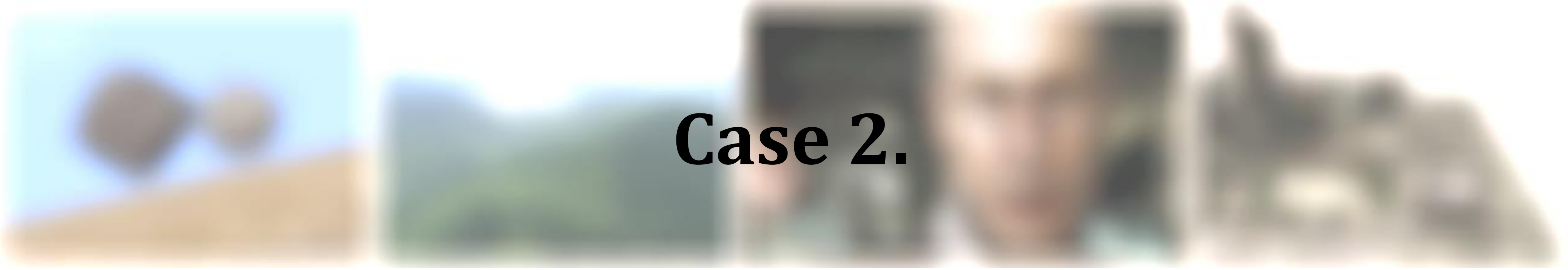
Case 1.



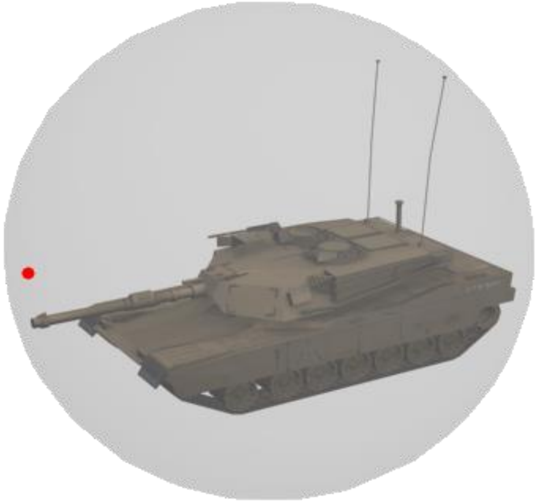
We do a quick test to see if the point is inside of the outer sphere

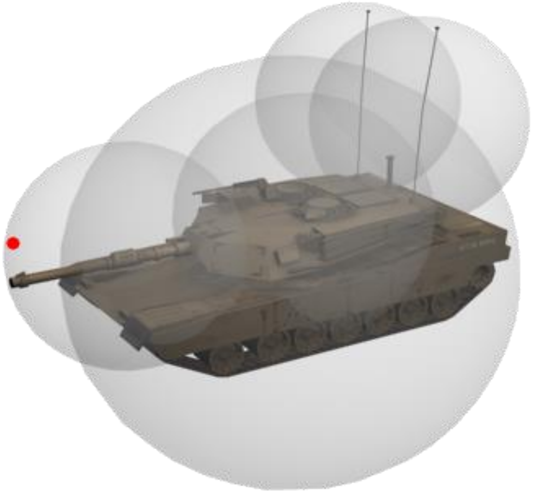
In this case, the point is outside

We can therefore reject it very quickly and do not need to do any more calculations



Case 2.



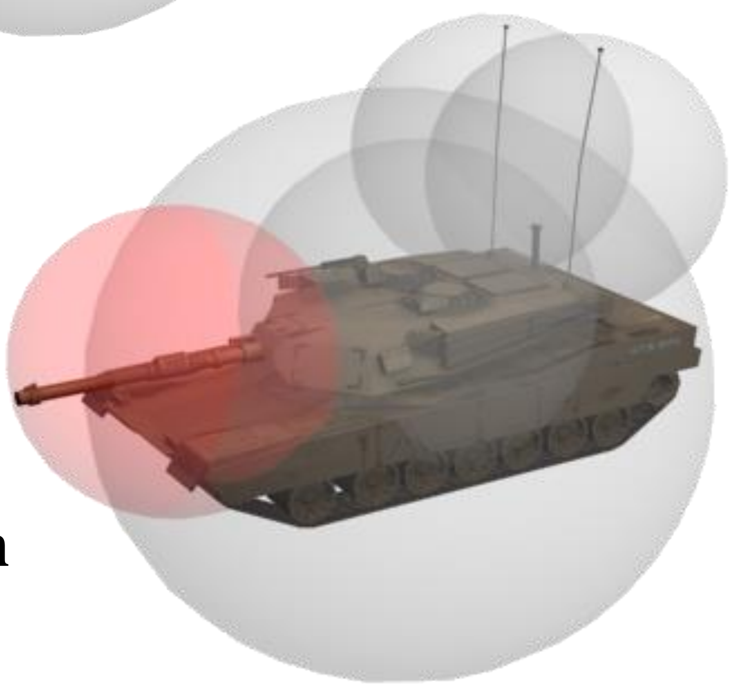


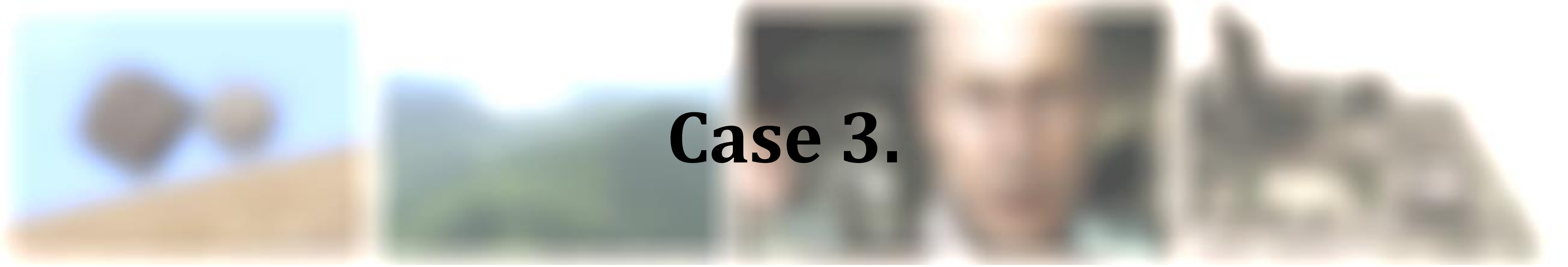
We do the same test as before with the outer sphere

This time we find the point is inside the outer sphere

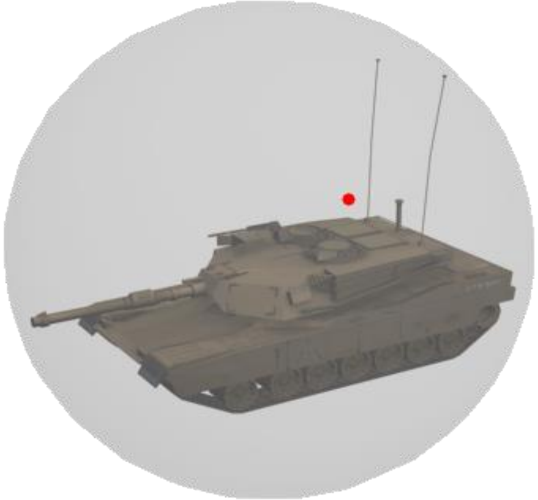
We therefore compare it with the lower level bounding spheres

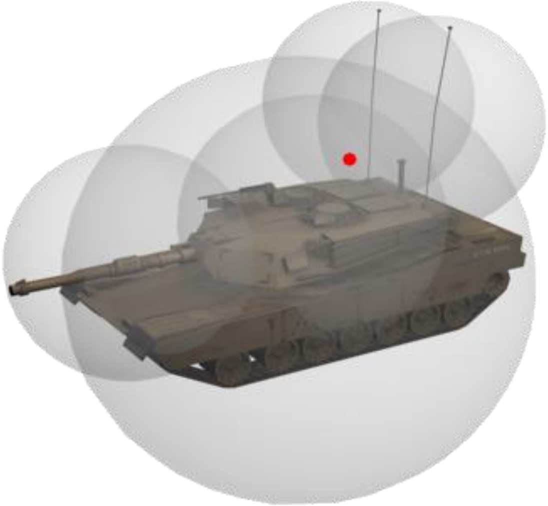
And end up testing the point with the main gun mesh





Case 3.



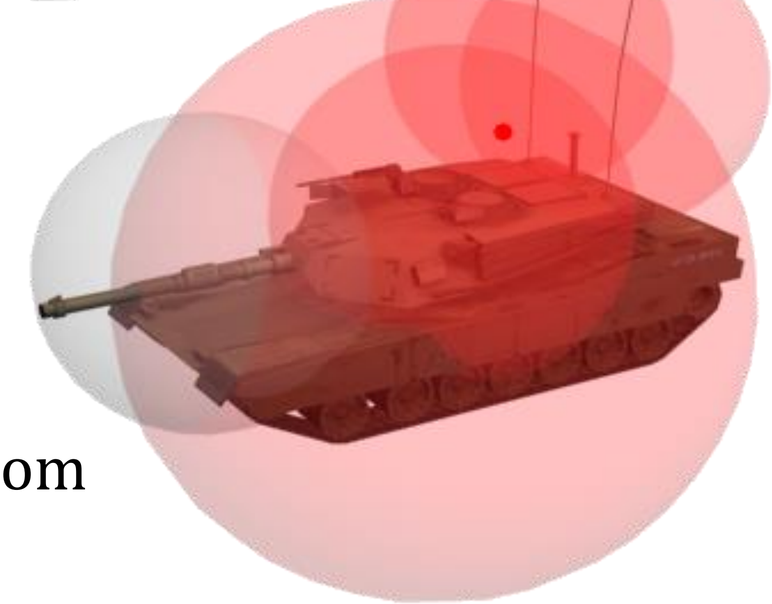


We do the same test as before with the outer sphere

Again, we find the point is inside the outer sphere

We compare it with the lower level bounding spheres…

Test the point with the all the meshes apart from the main gun mesh



Broad Phase Vs. Narrow Phase

These tests, which do quick high-level tests to see if objects are *potentially* intersecting, form what is called the **Broad Phase** collision detection

Quickly find the sets of objects that may be colliding with each other

What happens if the objects are found to be potentially colliding?

Then we need to do further tests to see if it is the case and, if so, find out where the objects are colliding

This is referred to as the **Narrow Phase** of collision detection

GJK Algorithm

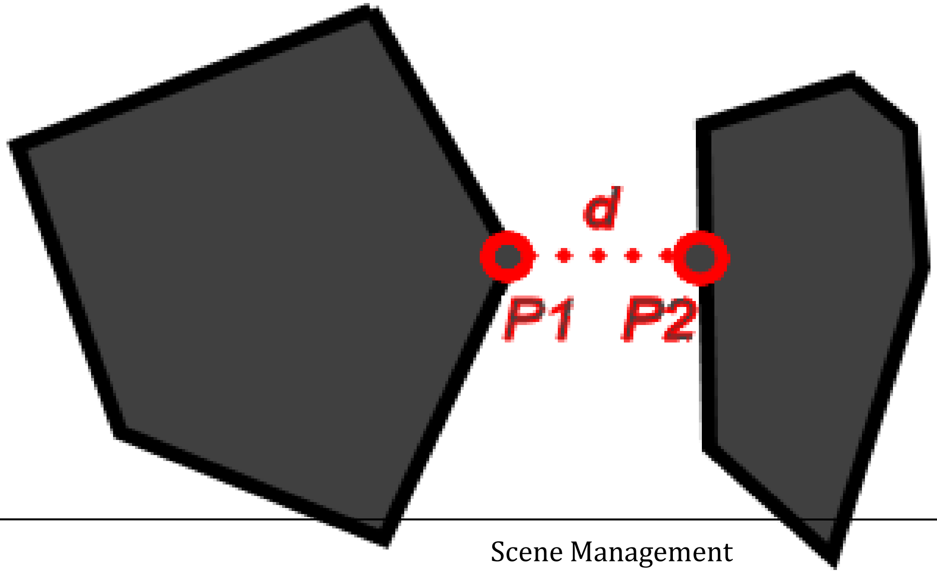
Gilbert-Johnson-Keerthi algorithm

Solves proximity queries between two complex **convex** polyhedra

Given the two polyhedra:

Computes the distance *d* between them

Can also return the closest pair of points on each polygons



Scene Management Techniques

Organizing the objects in a scene in a hierarchy helps.

Scene management methods

- BSP (*Binary Space Partitioning*) trees
- Quadtrees & Octrees
- Portals (*rendering the PVS – Potential Visibility Set*)

Scene management methods can be combined

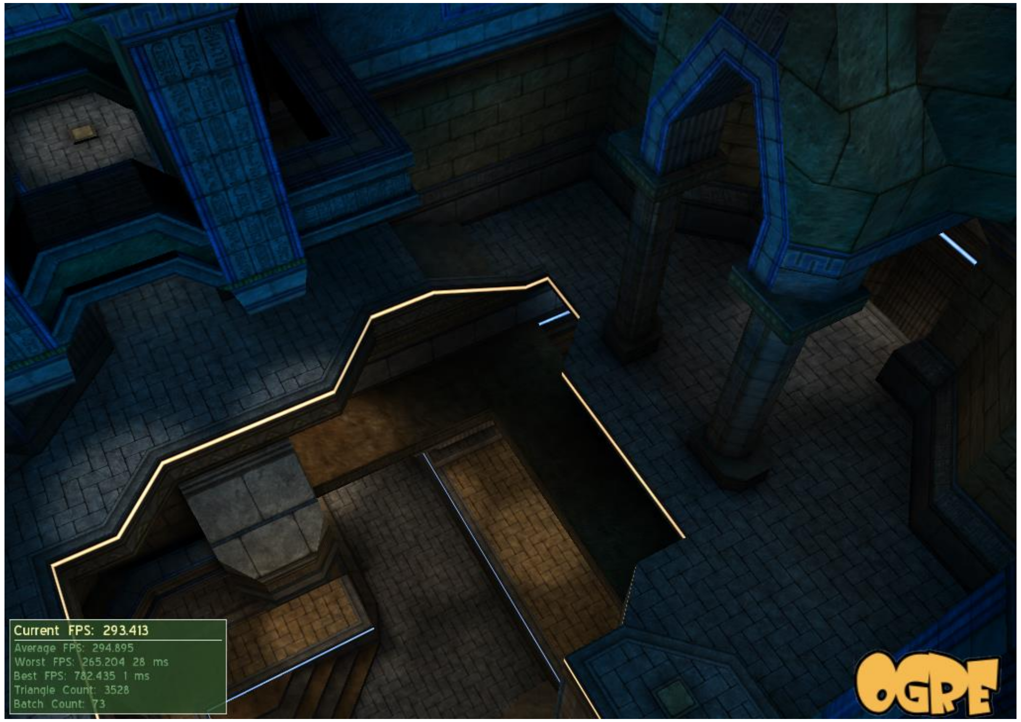
(*example: portals with BSP trees*)

Binary Space Partitioning Trees

- BSP trees use a binary tree structure to store the geometry of a scene [Fuchs et al. 1980]
- BSP trees are a very efficient scene management method that allows for very fast rendering of complex scenes
- BSP partitions the space successively in two parts in each step, using the hyperplanes induced by the polygons in the scene
- The construction of the BSP tree can take a long time and is done offline as a preprocessing step before any rendering takes place.

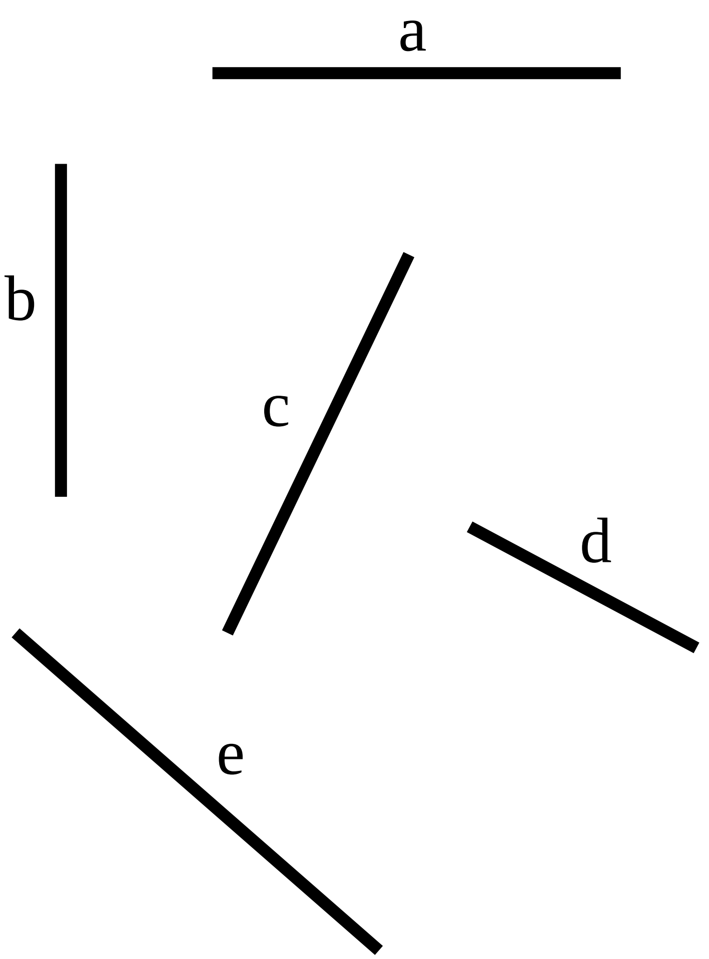
BSP trees in games

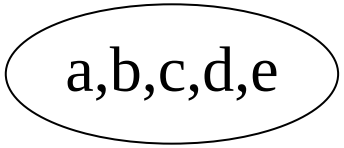
- BSP trees have been proven to be highly successful for real-time rendering in computer games
  - the rise of the FPS games genre would not have been possible without BSP trees
  - examples: Doom (*2D-space partitioning only*), Quake etc.



```
Current FPS: 293.413
Average FPS: 294.895
Worst FPS: 265.204 28 ms
Best FPS: 782.435 1 ms
Triangle Count: 3528
Batch Count: 73
```

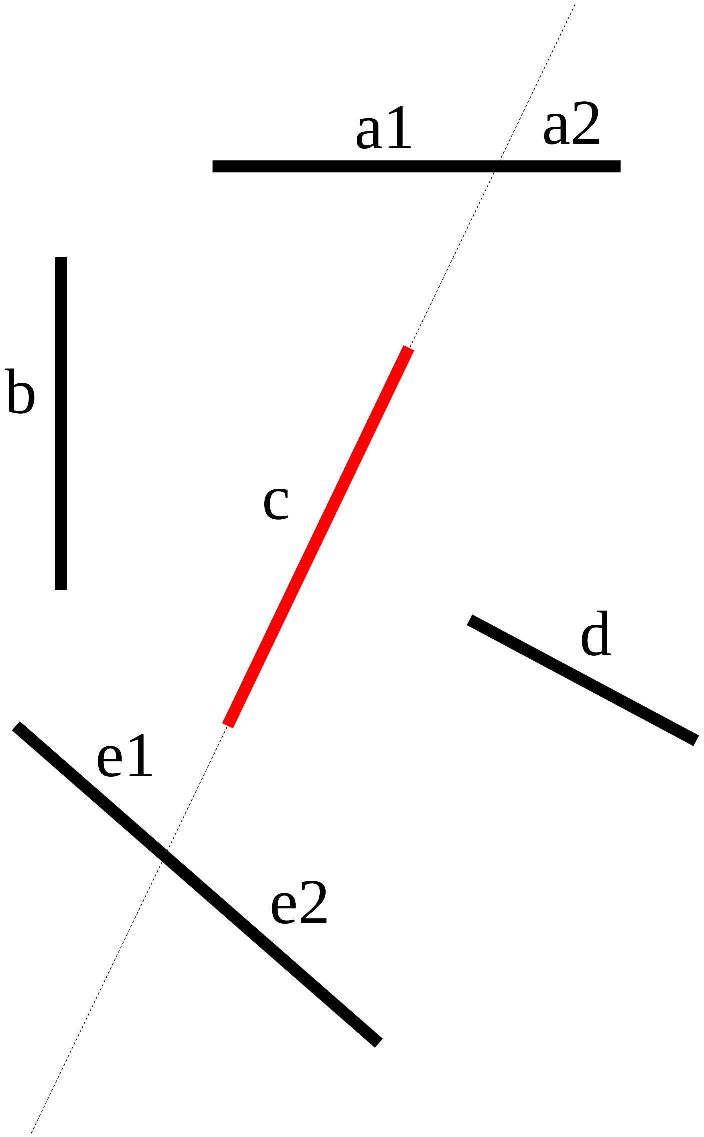
Construction of a BSP Tree



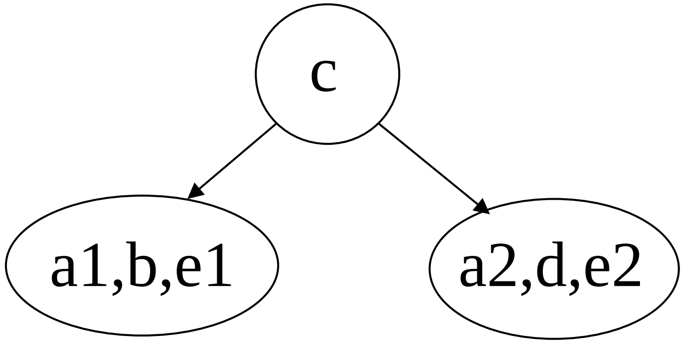


- 2D scene
- Hyperplanes are lines
- Position of label indicates forward facing side

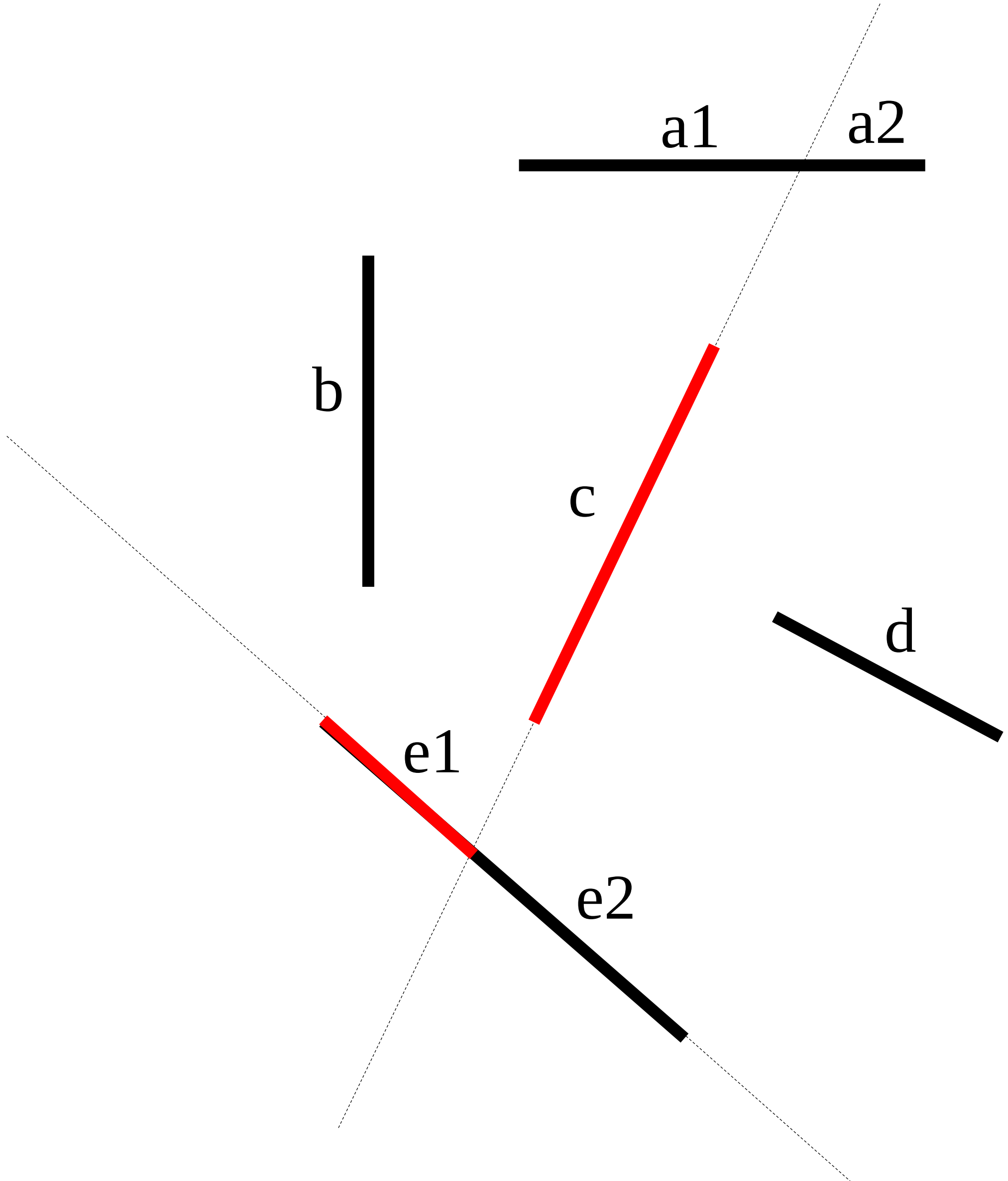
Construction of a BSP Tree



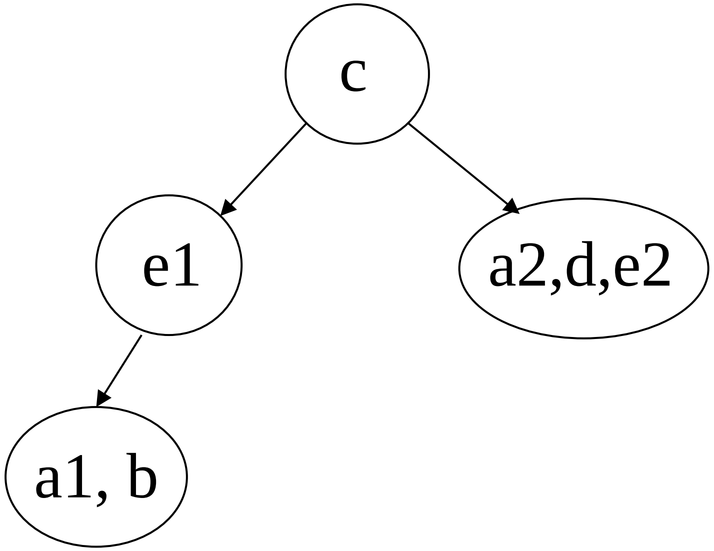
Nodes to the front -> left
Nodes to the back -> right



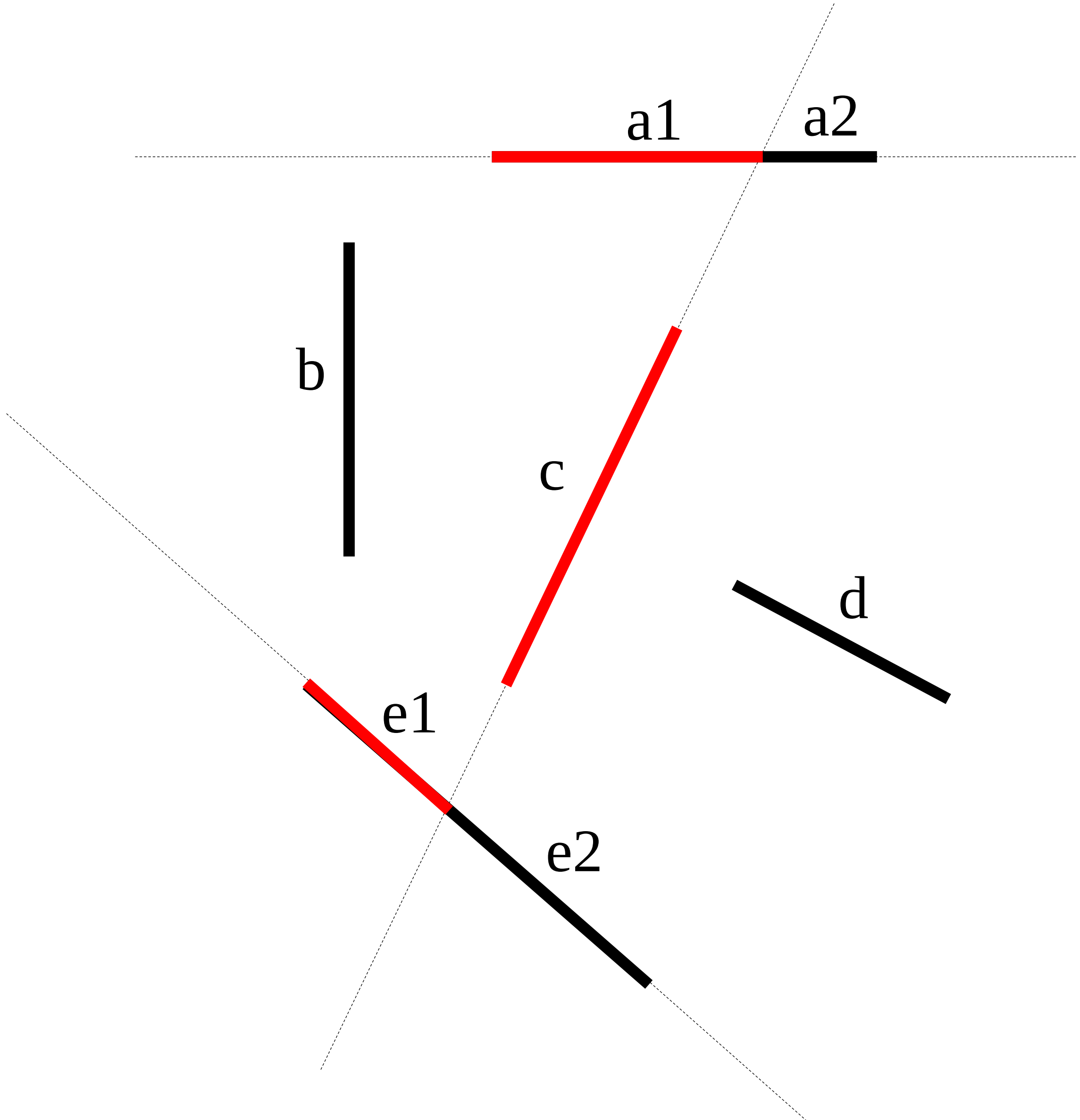
Construction of a BSP Tree



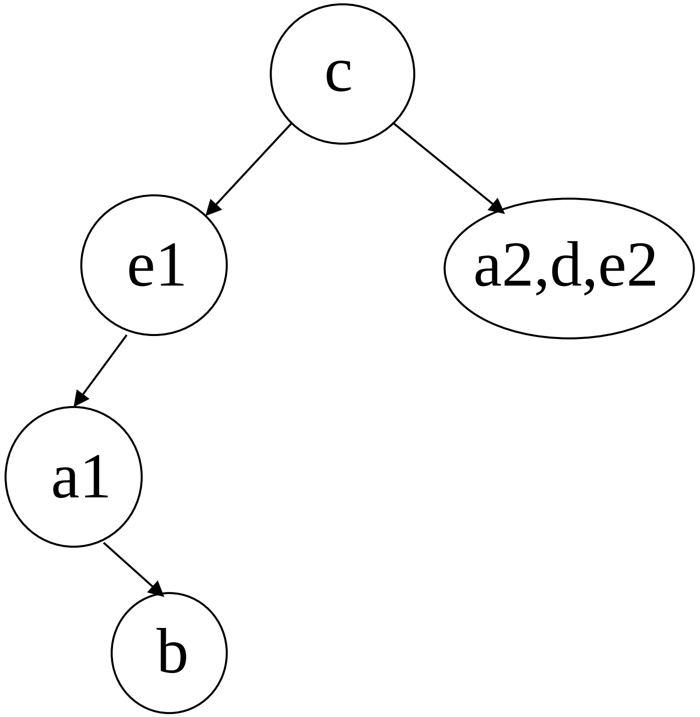
Nodes to the front -> left
Nodes to the back -> right



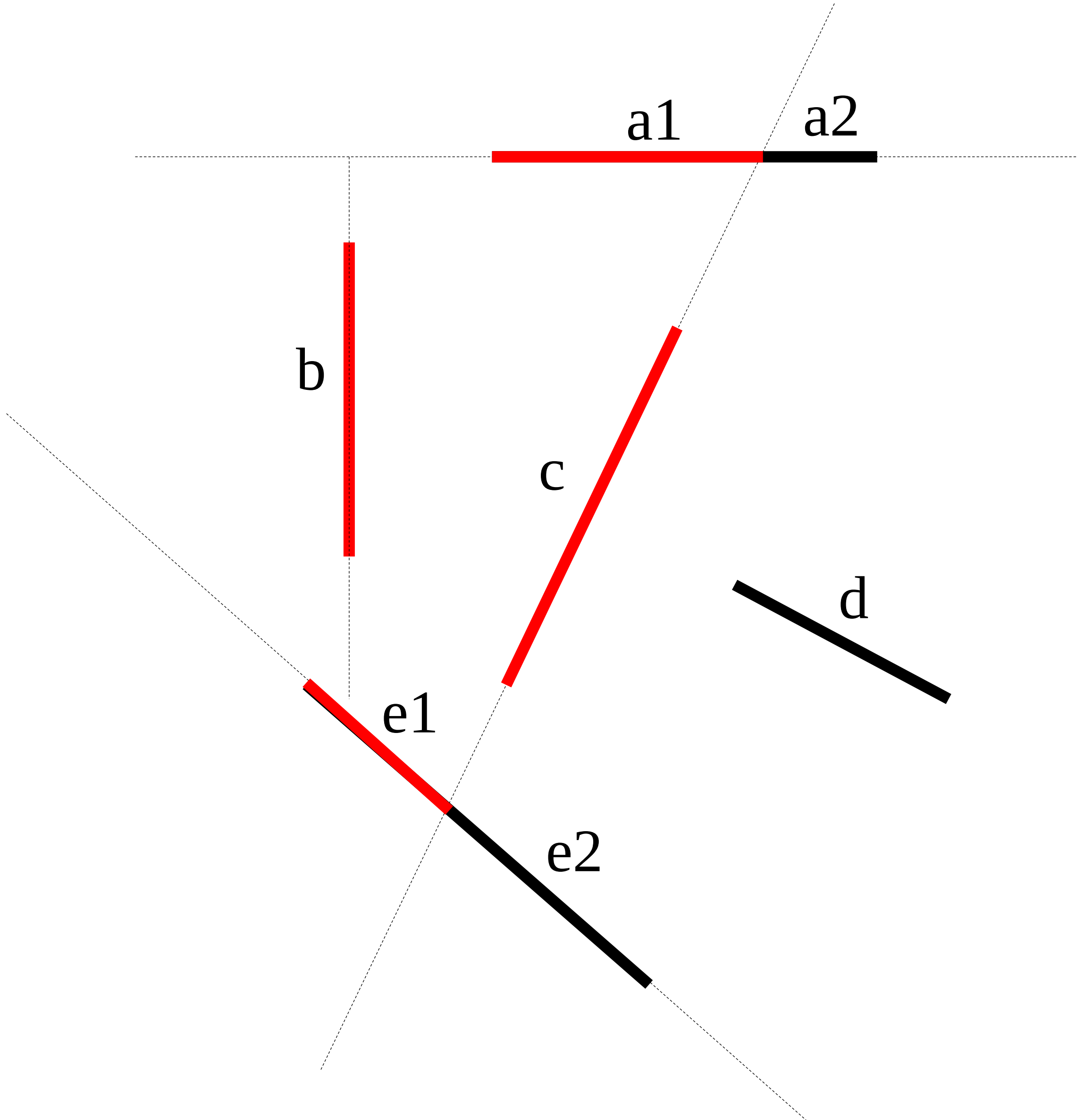
Construction of a BSP Tree



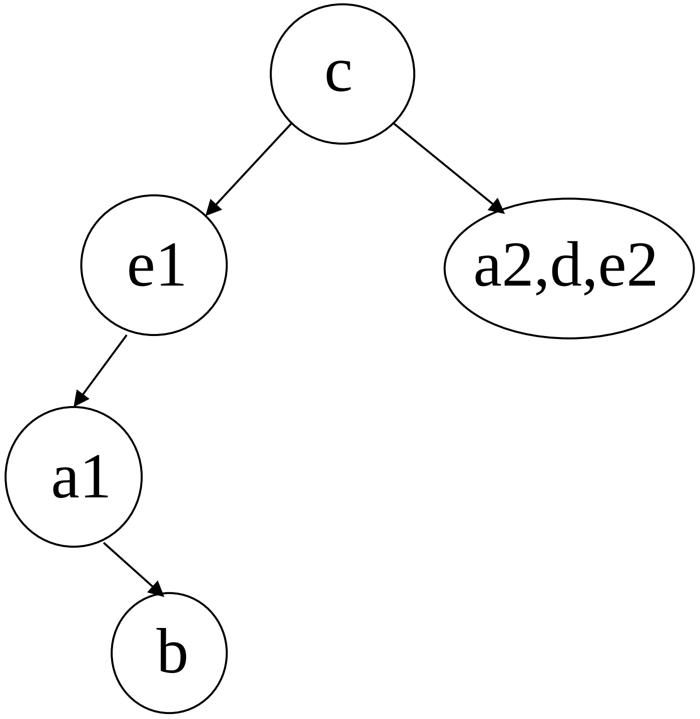
Nodes to the front -> left
Nodes to the back -> right



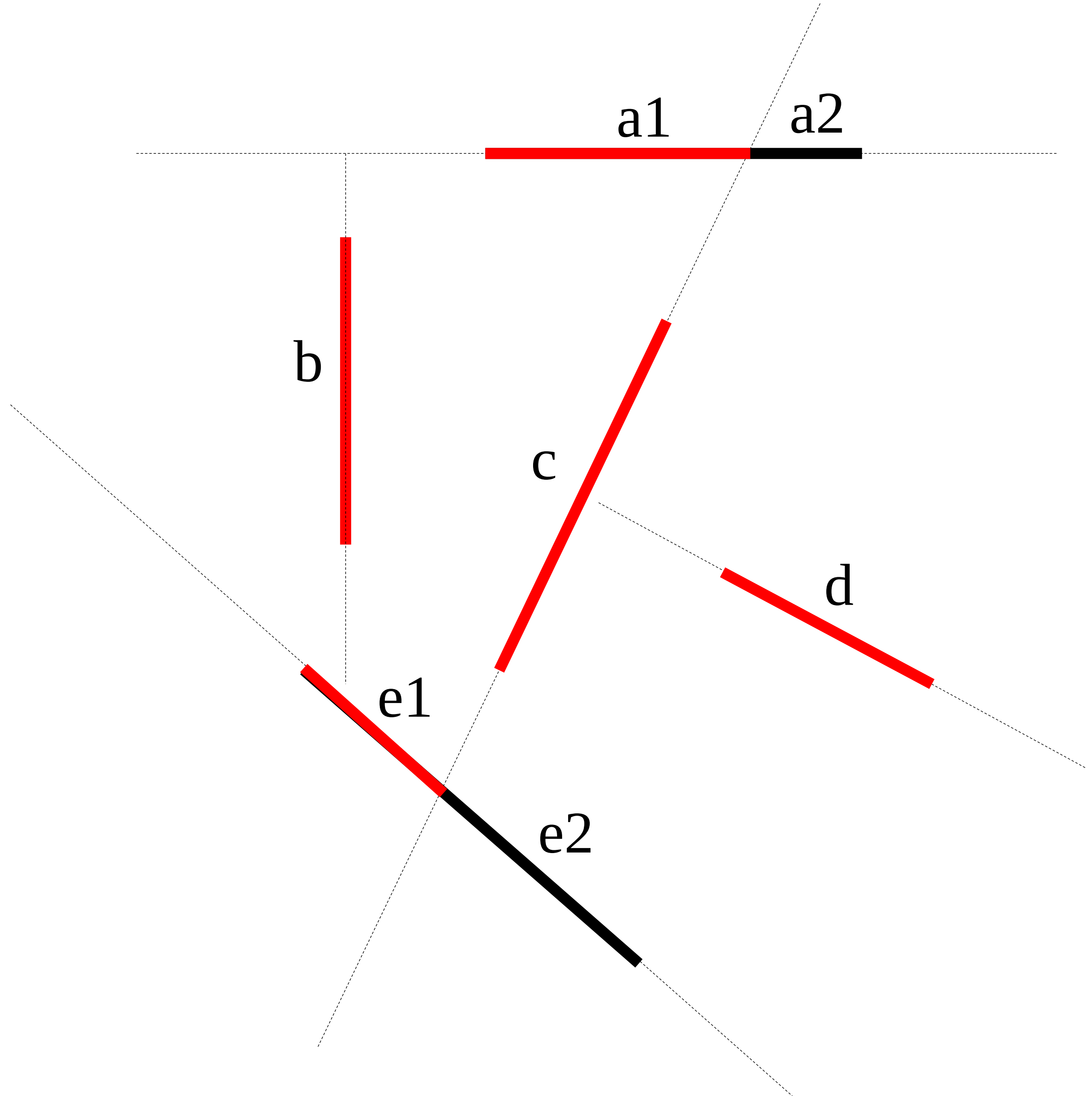
Construction of a BSP Tree



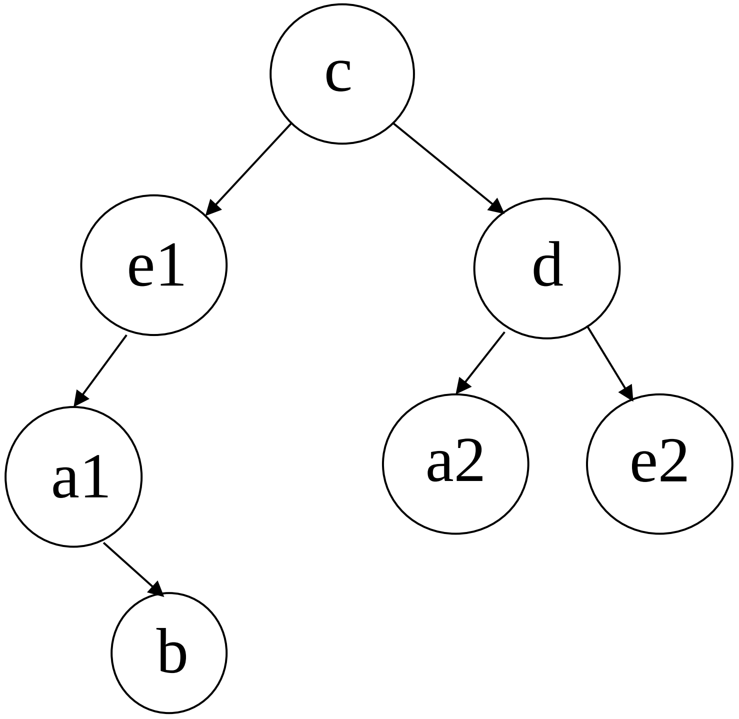
Nodes to the front -> left
Nodes to the back -> right



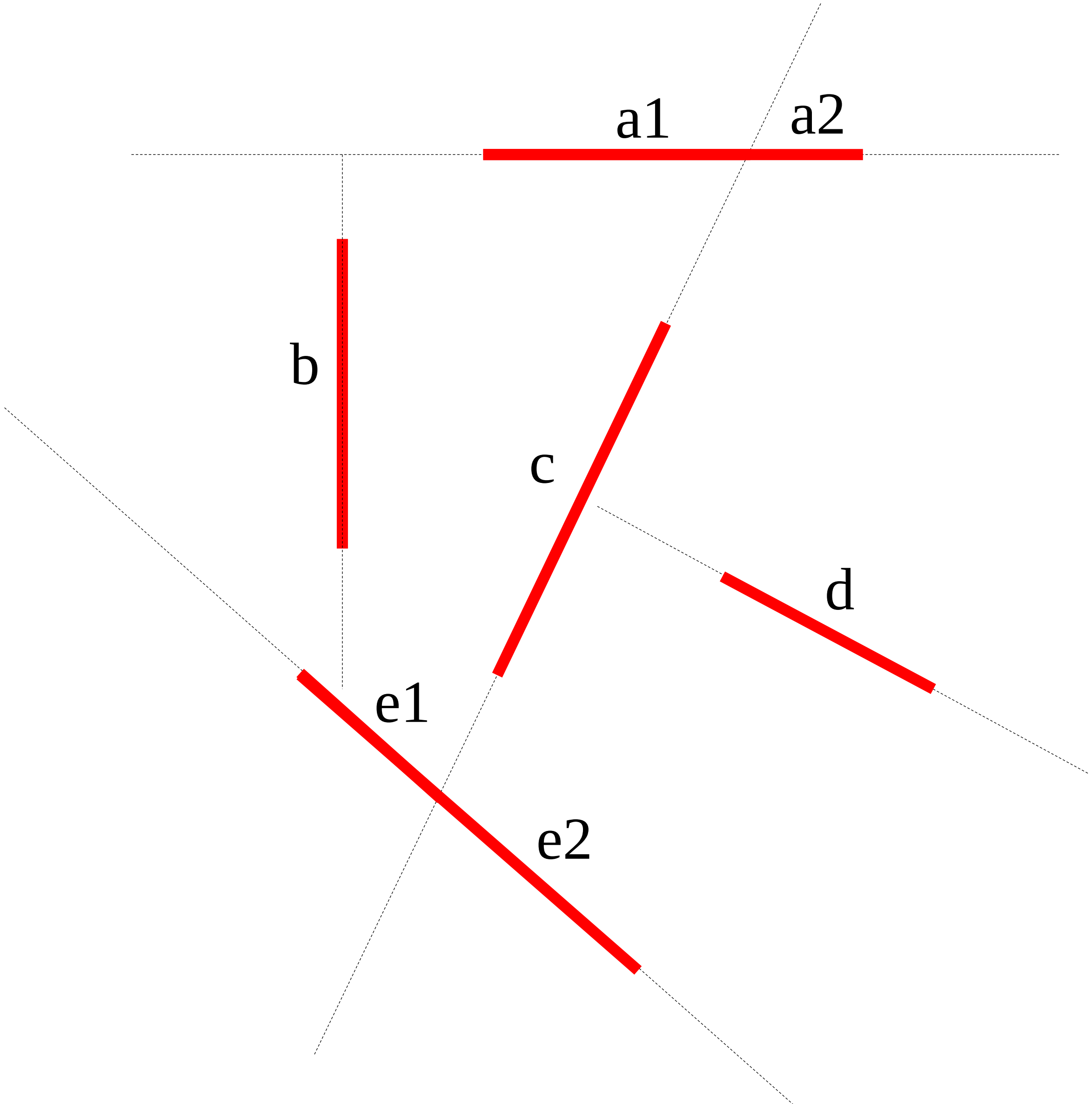
Construction of a BSP Tree



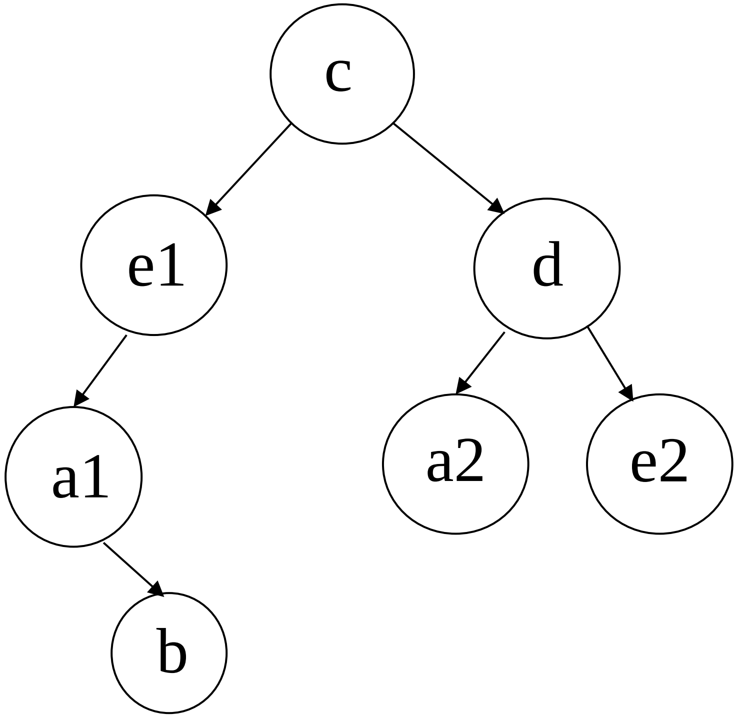
Nodes to the front -> left
Nodes to the back -> right



Construction of a BSP Tree

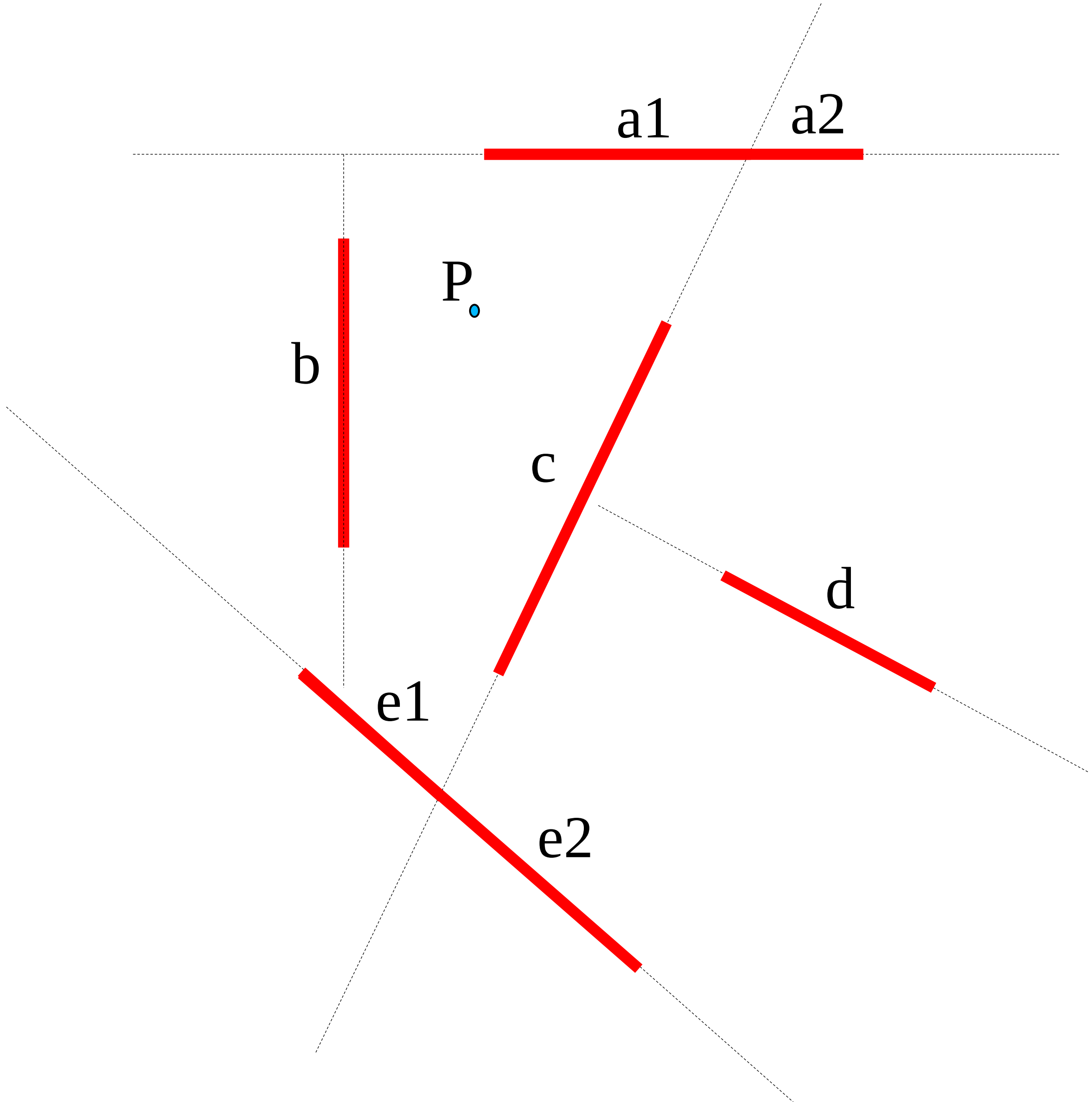


Nodes to the front -> left
Nodes to the back -> right



Done!

Rendering using a BSP



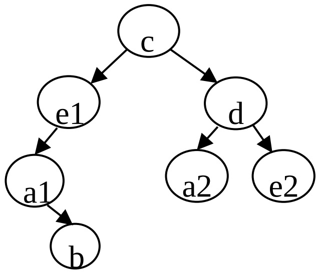
If viewpoint in front

- Render back (right) nodes
- Render node
- Render front (left) nodes

If viewpoint on back

- Render front (left) nodes
- Render node
- Render back (right) nodes

In the case of the viewpoint at **P**, the rendering order will be : e2, d, a2, c, e1, a1, b



Alternative BSP tree compilation

Polygon-Aligned BSP tree compilation [Akenine-Möller and Haines 2002]

- Starting from an arbitrarily selected polygon (*usually from the geometric centre of a scene*), all polygons of the scene are inserted into the tree
- The position of a polygon in relation to polygons that are already inside the tree decides into which branch of the binary tree (*left or right*) a polygon is entered
- If a polygon of the scene that has not yet been inserted into the BSP tree intersects with the plane defined by another polygon which is already inside of the tree, that polygon may have to be split into two polygons

This method can be simplified (*splitting of polygons can be disallowed*)

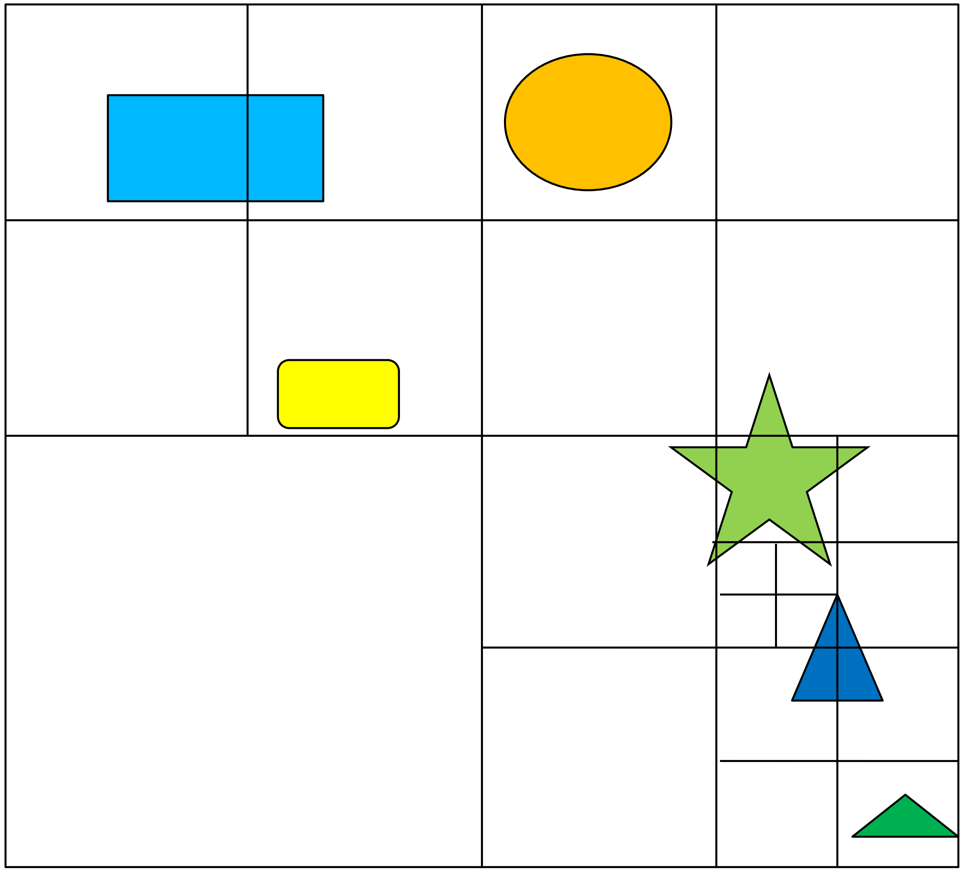
note: simplification of this method may lead to less accurate rendering

Quadtrees

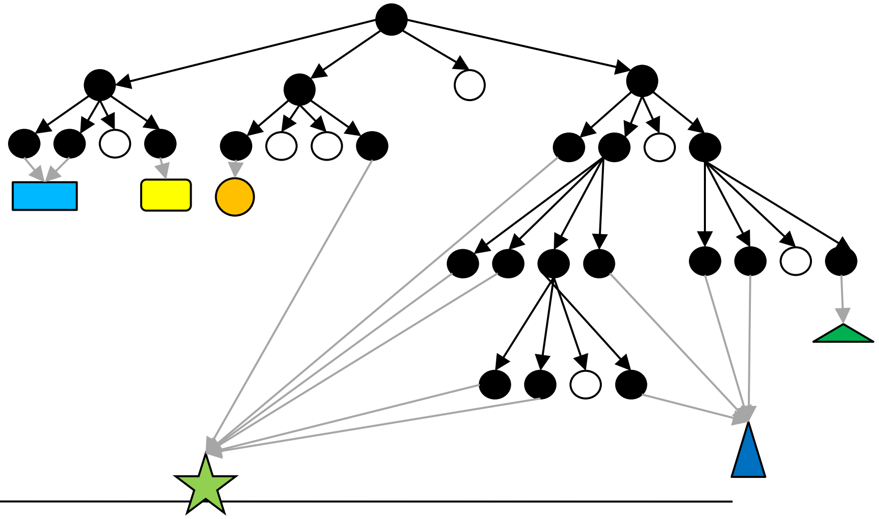
Tree structure in which every node holds ***four*** child nodes

- Special case of the BVH where splits are axis-aligned.
- Quadtrees divide a scene up into rectangular areas that contain objects (*or polygons*)
- Objects are usually stored at the deepest nodes of the tree (*exception: use of quadtree for CLOD rendering* [*Ulrich 2000*])
- If a quad (*or part of a quad*) is visible (*in front of the virtual camera*) then the child nodes of the quad need to be tested for visibility
- If a quad is not visible then none of its child nodes need to be traversed and consequently none of the objects (*or polygons*) contained within the quad need to be rendered

Quadtrees



- Recursively subdivide the bounding box of the entire scene into 4 boxes
- Stop dividing when a box is empty or has reached a minimum number of objects



Octrees

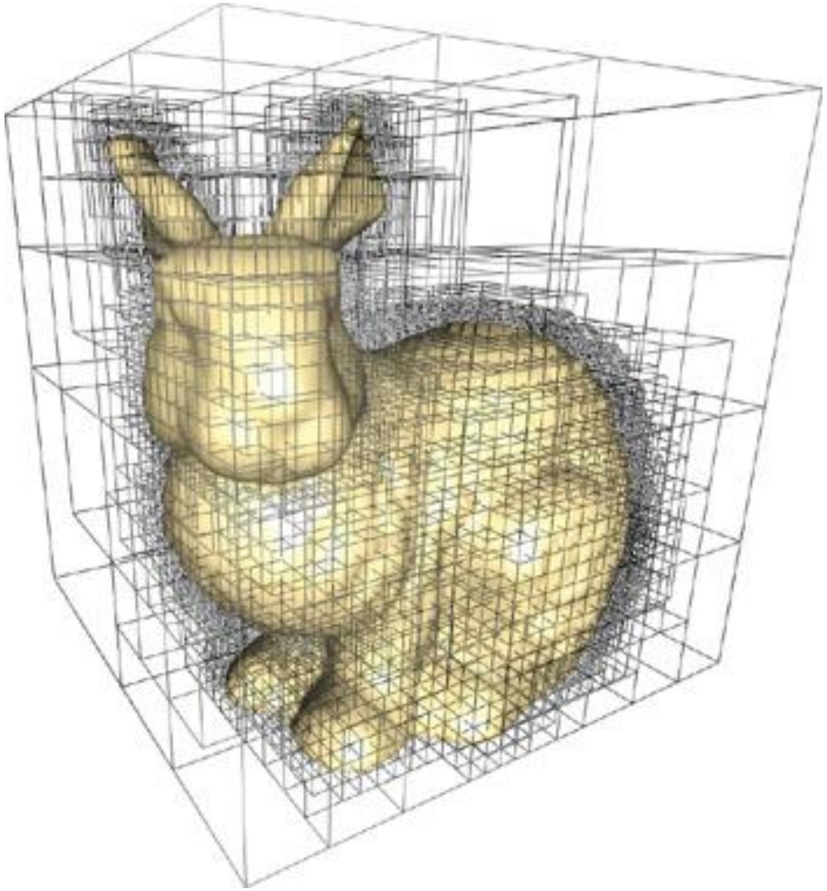


Image courtesy GPU Gems 2, Chapter 37

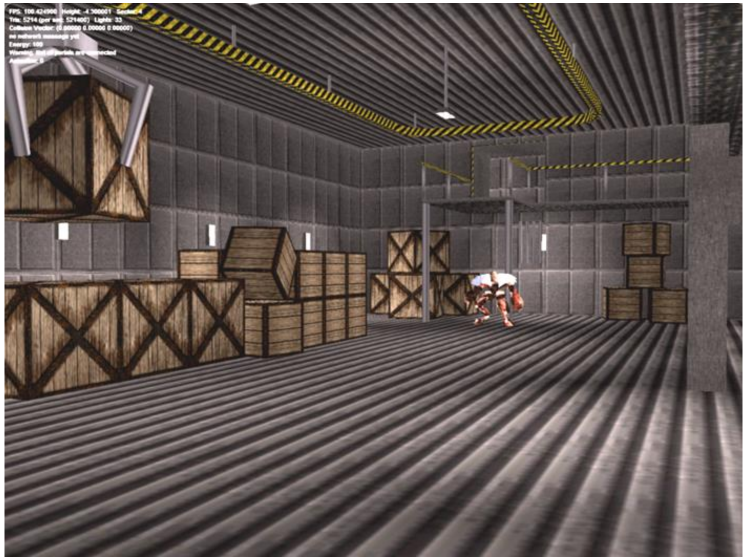
- 3D version of a quadtree.
- 2^3 = 8 divisions are made at each step

Portals

Portals provide a simple scene-management method

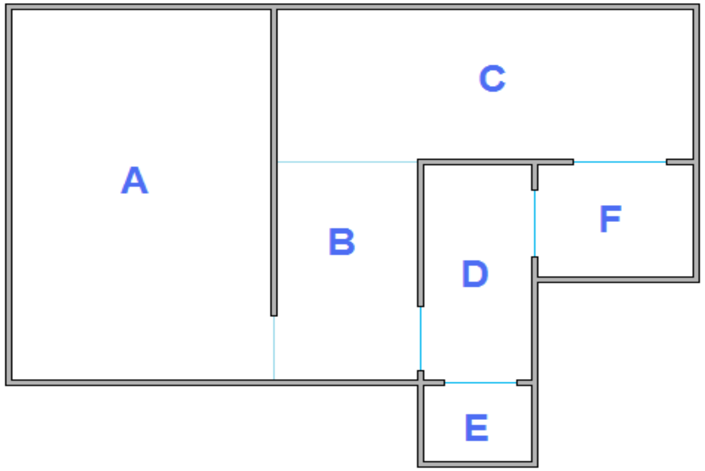
[Akenine-Möller and Haines 2002]

- environment is divided into cells that are connected through portals



Portal Rendering

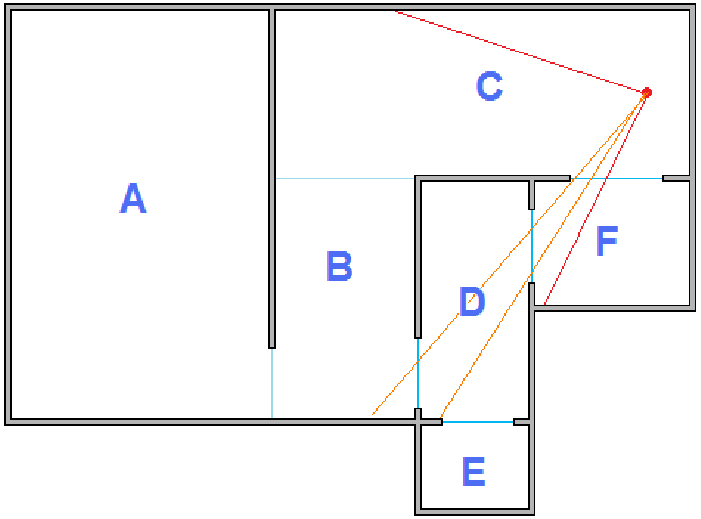
1. test which neighbouring cells are visible from camera position
2. recursively test visible cells for visibility of their neighbours
3. render (*draw*) all visible cells



Portal Rendering Examples

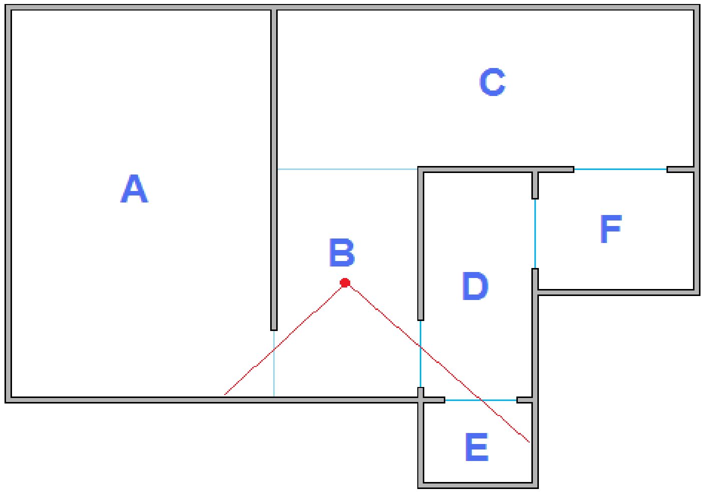
Visibility:

- cell C (*camera position*)
    - cell B (*from C*)
    - cell F (*from C*)
        - cell D (*from F*)
            - cell B (*from D*)



Visibility:

- cell B (*camera position*)
    - cell A (*from B*)
    - cell D (*from B*)
        - cell E (*from D*)



References

- Akenine-Möller, T. and Haines, E. (2002). Real-Time Rendering, 2nd Edition. AK Peters

- Fuchs, H., Kedem, Z. M. and Naylor, B. F. (1980). On visible surface generation by a priori tree structures. In SIGGRAPH ’80: Proceedings of the 7th annual conference on Computer graphics and interactive techniques, pp. 124–133

- Ulrich, T. (2000). Continuous LOD Terrain Meshing Using Adaptive Quadtrees. Gamasutra - http://www.gamasutra.com/view/feature/3434/continuous_lod_terrain_meshing_.php?print=1

Upcoming Lectures and Labs

- You should be working on Lab 3
- Project specifications! Canvas…

- Next lab sessions:

Tues May 8th 10:00-12:00, VIC Studio

Wed May 16th 10:00-12.00, VIC Studio

Fri May 25th 13:00-15:00, VIC Studio

- Animation and Image-based Rendering

9th May, 13:00–15:00, D2