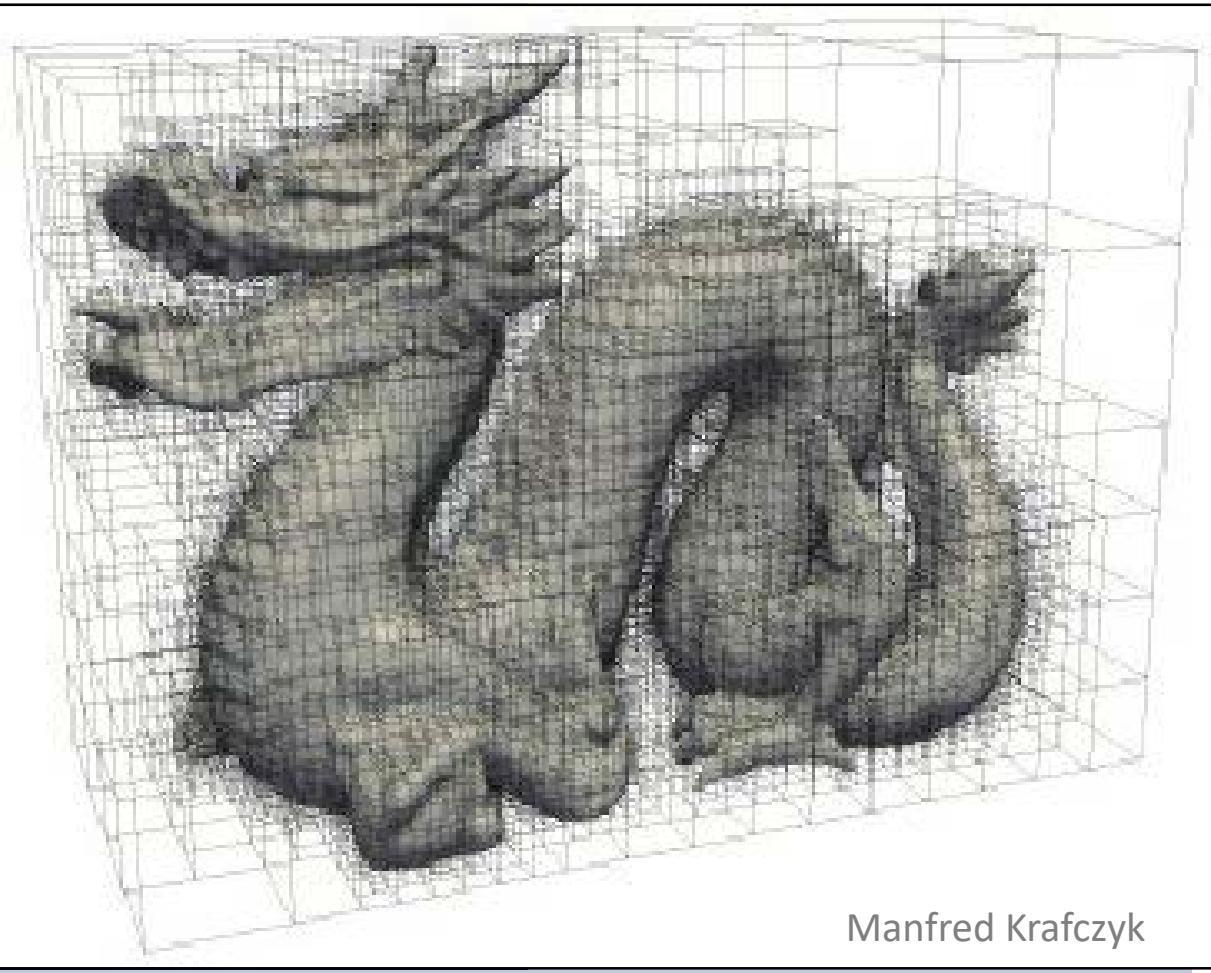
https://www.keithlantz.net/2013/04/kd-tree-construction-using-the-surface-area-heuristic-stack-based-traversal-and-the-hyperplane-separation-theorem/



Manfred Krafczyk

Γραφικά Υπολογιστών

Γεωμετρικές δομές δεδομένων και μέθοδοι επιτάχυνσης

Andreas Aristidou
andarist@ucy.ac.cy
http://www.andreasaristidou.com

Επιτάχυνση αλγορίθμων

- Οι μέθοδοι βελτιστοποίησης είναι αναγκαίοι σχεδόν παντού στα γραφικά:
  - Παρακολούθηση ακτίνας - πάνω από το 90% του κόστους έγκειται στην να βρούμε την τομή της κάθε ακτίνας με τα αντικείμενα
  - **Radiosity** – υπολογισμός των form factors*
  - Γραφικά πραγματικού χρόνου
    - Μεγάλα περιβάλλοντα (εκατομμύρια πολύγωνα) @ 30Hz
    - Καλύτερο φωτισμό
    - Σκιές
    - …

** Επόμενες Διαλέξεις*

Προσεγγίσεις για επιτάχυνση

- **Παράλληλη επεξεργασία**
  - parallel computers
  - multi-core
  - grid

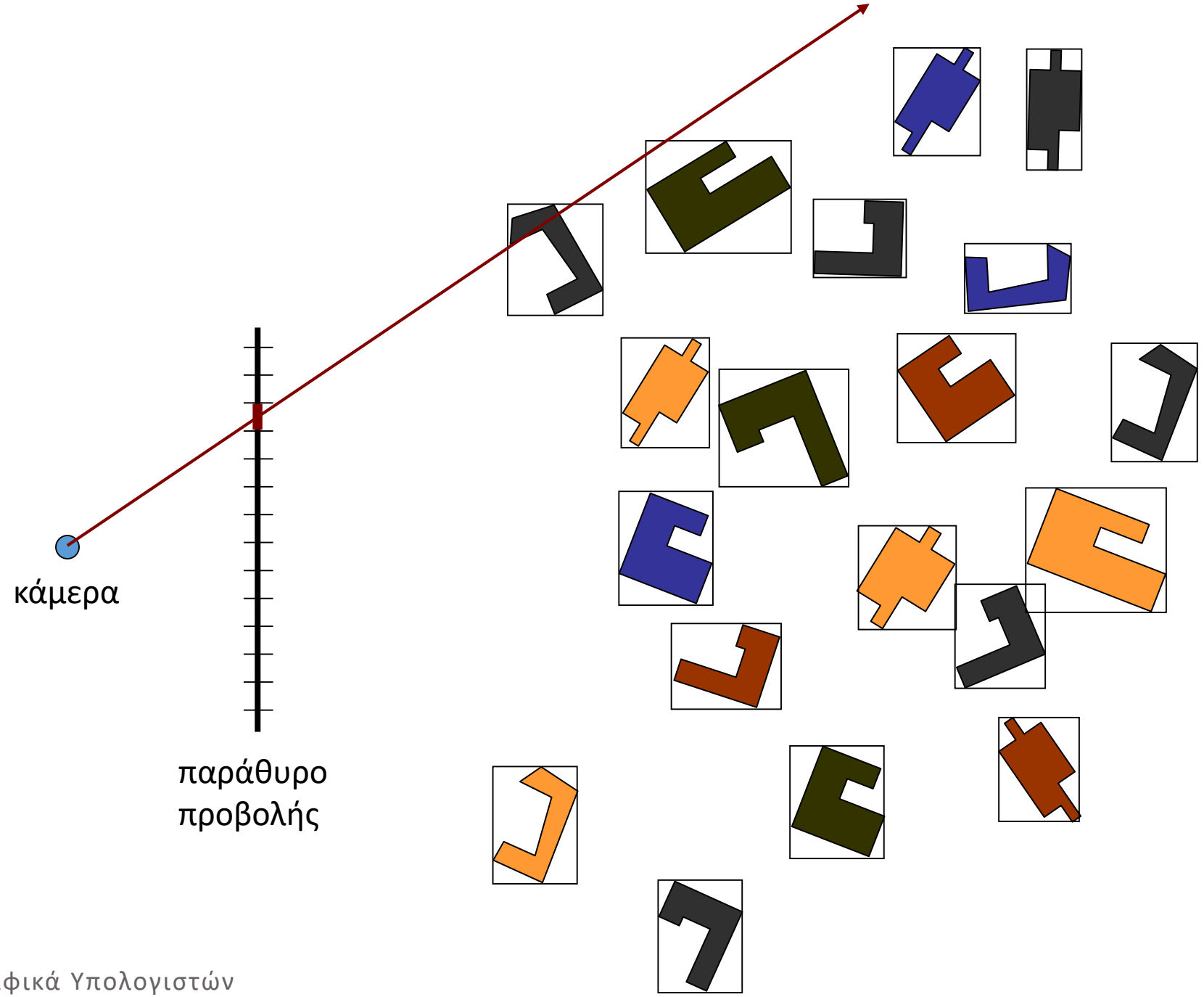
- Υλοποίηση κάποιων συναρτήσεων στην κάρτα γραφικών (**GPU** – shaders)
  - NVIDIA GeForce 1080 – μέχρι 3500 Stream Processors

- **Αλγοριθμικές μέθοδοι**
  - Λογαριθμική πρόσβαση, IBR, Culling, LOD
  - Βάση για τα περισσότερα: **γεωμετρικές δομές επιτάχυνσης**

Επιτάχυνση με χρήση γεωμετρικών δομών

- **Περιβάλλοντες όγκοι** (bounding volumes)
- **Ιεραρχικοί περιβάλλοντες όγκοι** (hierarchical bounding volumes)
- **Κατάτμηση χώρου** (space subdivision)

Bounding Volumes

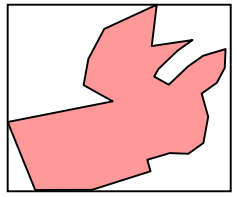

κάμερα
παράθυρο
προβολής

Bounding Volumes (BV)

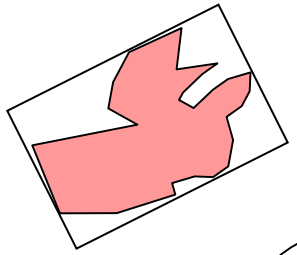
- Extents and bounding volumes
  - Enclose complex objects in simpler ones
    - Simpler = spheres, cuboids
  - If bounding volume isn't visible, neither is object inside it!
  - To increase efficiency, put multiple objects into one volume
- How does this speed up ray tracing?
  - Quick reject: check ray against bounding volume first
  - Quicker reject: check group of rays (frustum) against bounding volume of object, which is better because the clipping is done for containers instead of individual triangles
- Simple to implement and can offer noticeable speedups
  - The tighter the bounding box the better, less false positives

Bounding Volumes (BV)

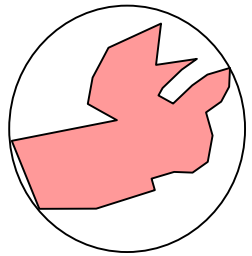
- Many different choices
  - Axis aligned bounding boxes
  - Oriented bounding boxes
  - Bounding spheres
  - Bounding ellipsoids
  - Discrete orientation polytopes (k-dop)
  - Other - convex hulls …



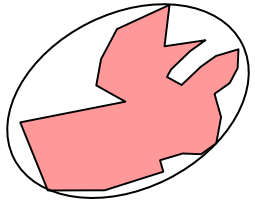
Axis aligned
Bounding Box



Oriented
Bounding Box

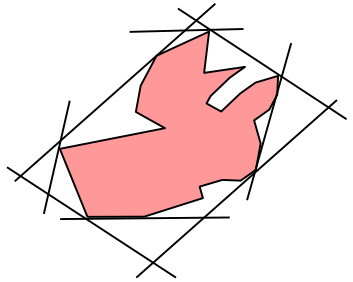


Bounding
Sphere



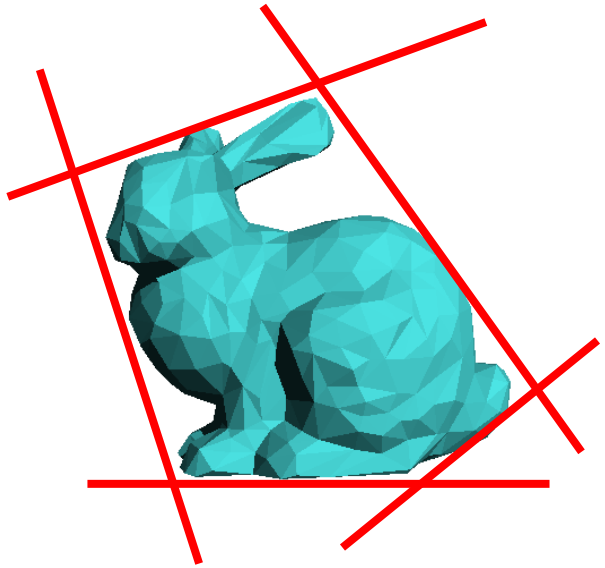
Bounding
Ellipsoid

Discrete orientation
polytopes (K-dop)

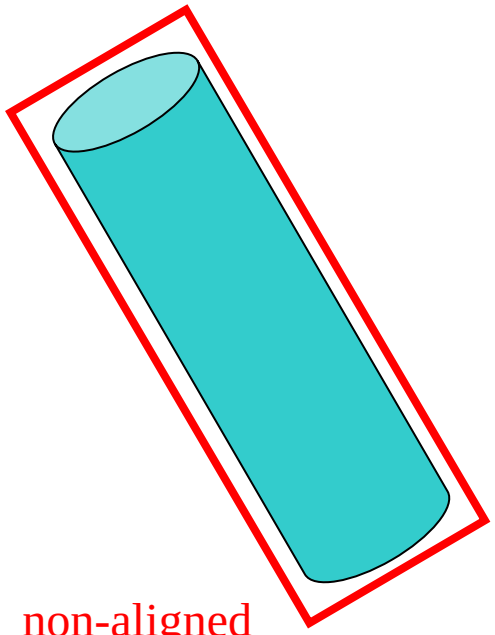


Conservative Bounding Regions

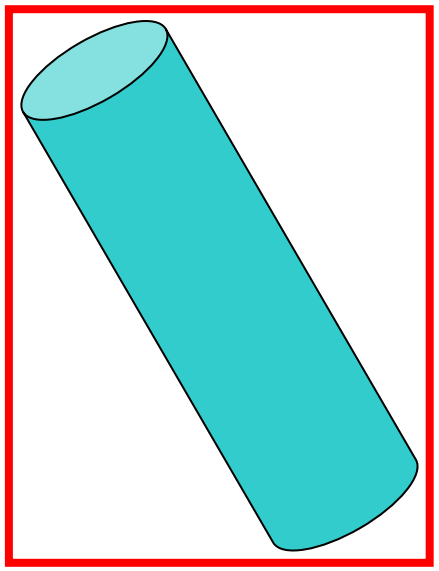
- tight → avoid false positives
- fast to intersect



arbitrary convex region (bounding half-spaces)



non-aligned bounding box

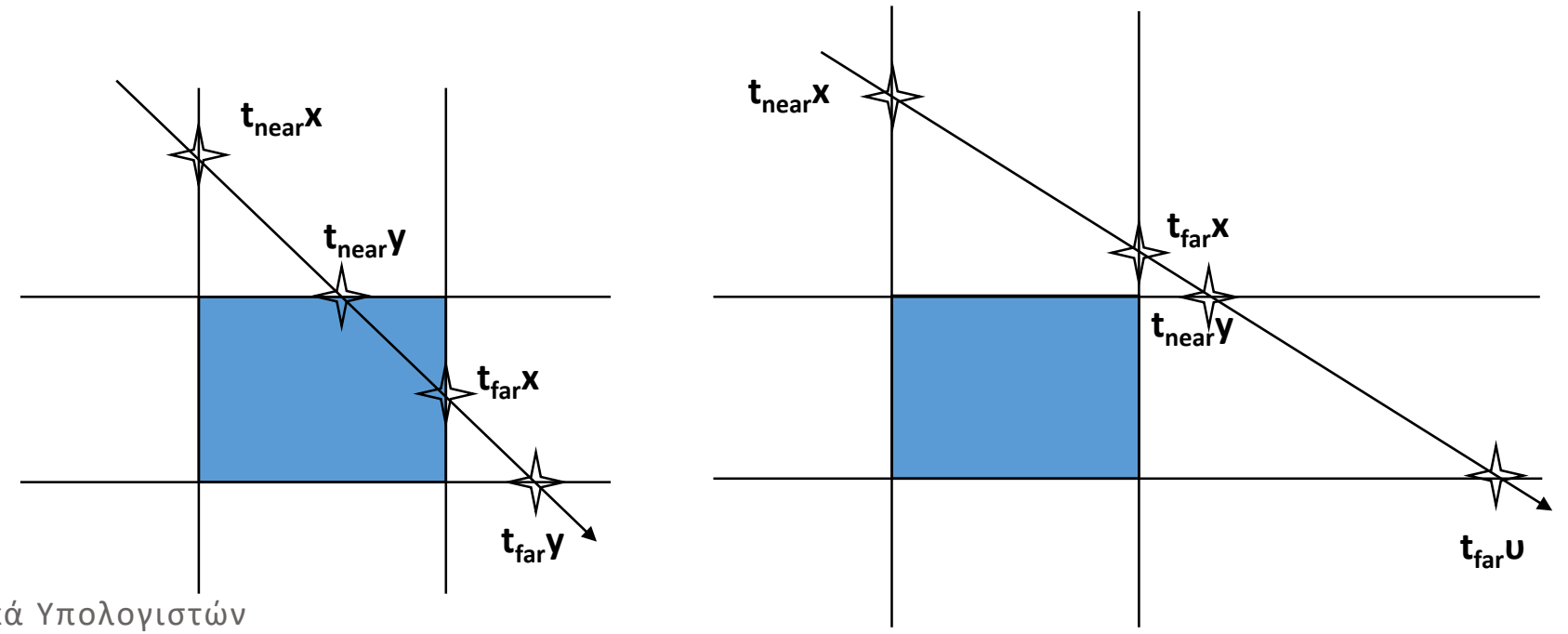


axis-aligned bounding box

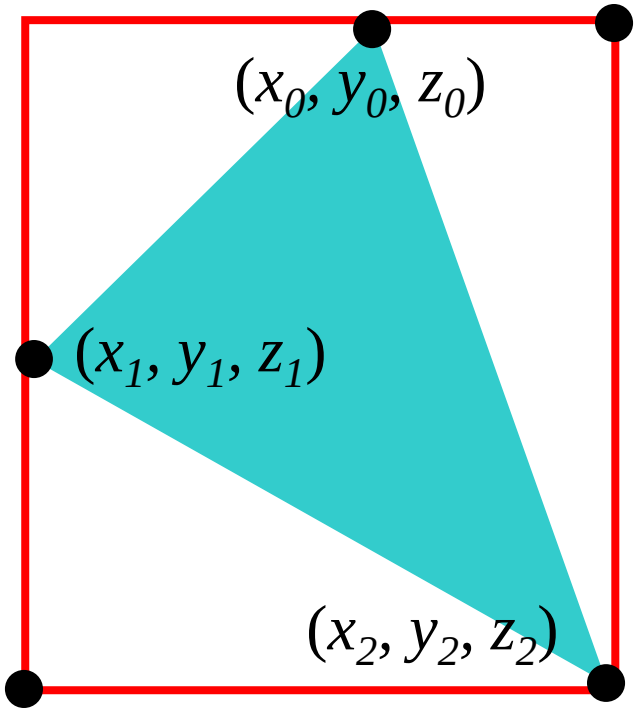
bounding sphere

Κανονικό περιβάλλον κουτί (Axis aligned Bounding Boxes)

- Ένα από το πιο συνηθισμένα και πιο εύκολα στην υλοποίηση
- Γρήγορο τεστ ακτίνας-κουτί
  - Το κουτί είναι 3 σετ από παράλληλα επίπεδα, το κάθε ένα κάθετο στα άλλα δύο
    - Υπολογίζουμε το t_{near} για κάθε ένα από τα 3 ζεύγη επιπέδων
    - Βρίσκουμε το μέγιστο από τα τρία t_{near}
    - Υπολογίζουμε το t_{far} κάθε ένα από τα 3 ζεύγη επιπέδων
    - Βρίσκουμε το μικρότερο από τα τρία 3 t_{far}
  - Αν το μέγιστο t_{near} είναι μεγαλύτερο από το μικρότερο t_{far}, τότε η ακτίνα δεν τέμνει το κουτί



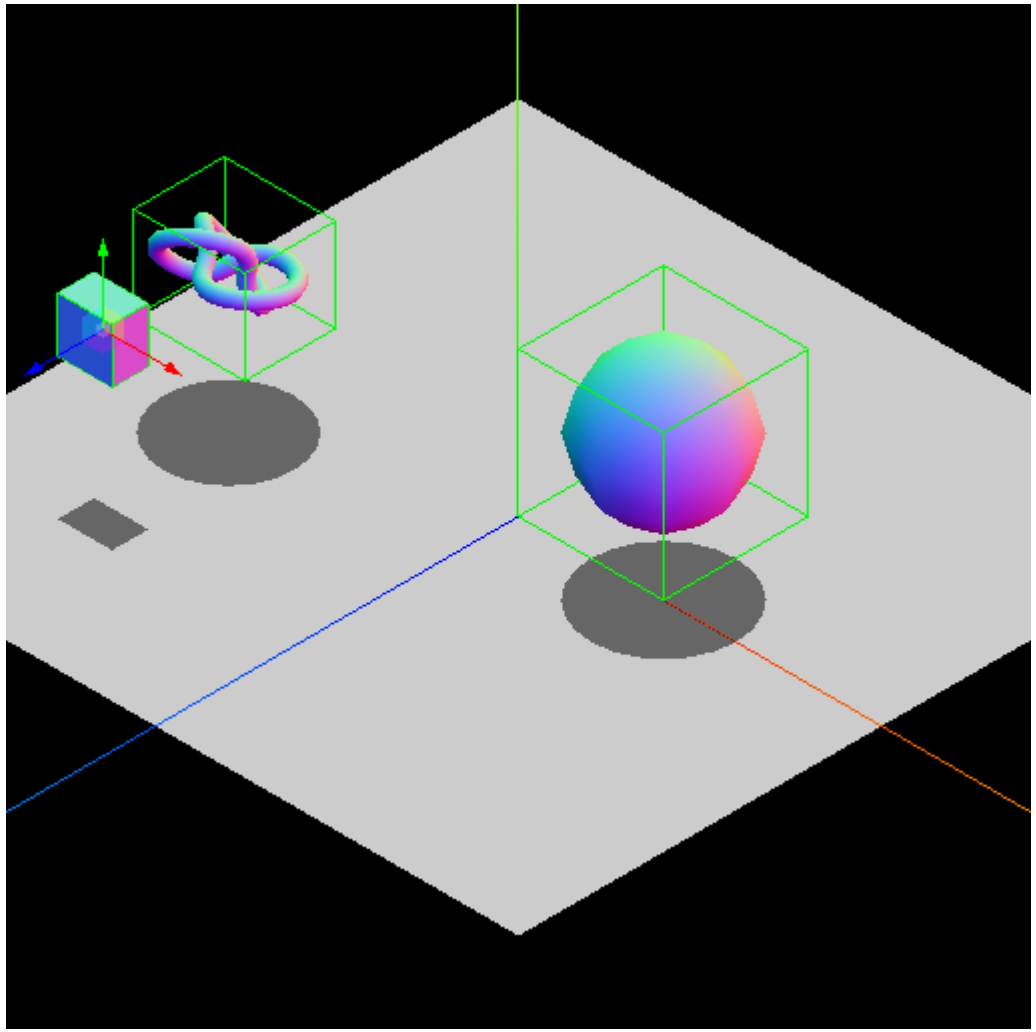
Bounding Box of a Triangle



$(x_{max}, y_{max}, z_{max})$
$= (\max(x_0,x_1,x_2),$
$\max(y_0,y_1,y_2),$
$\max(z_0,z_1,z_2))$

$(x_{min}, y_{min}, z_{min})$
$= (\min(x_0,x_1,x_2),$
$\min(y_0,y_1,y_2),$
$\min(z_0,z_1,z_2))$

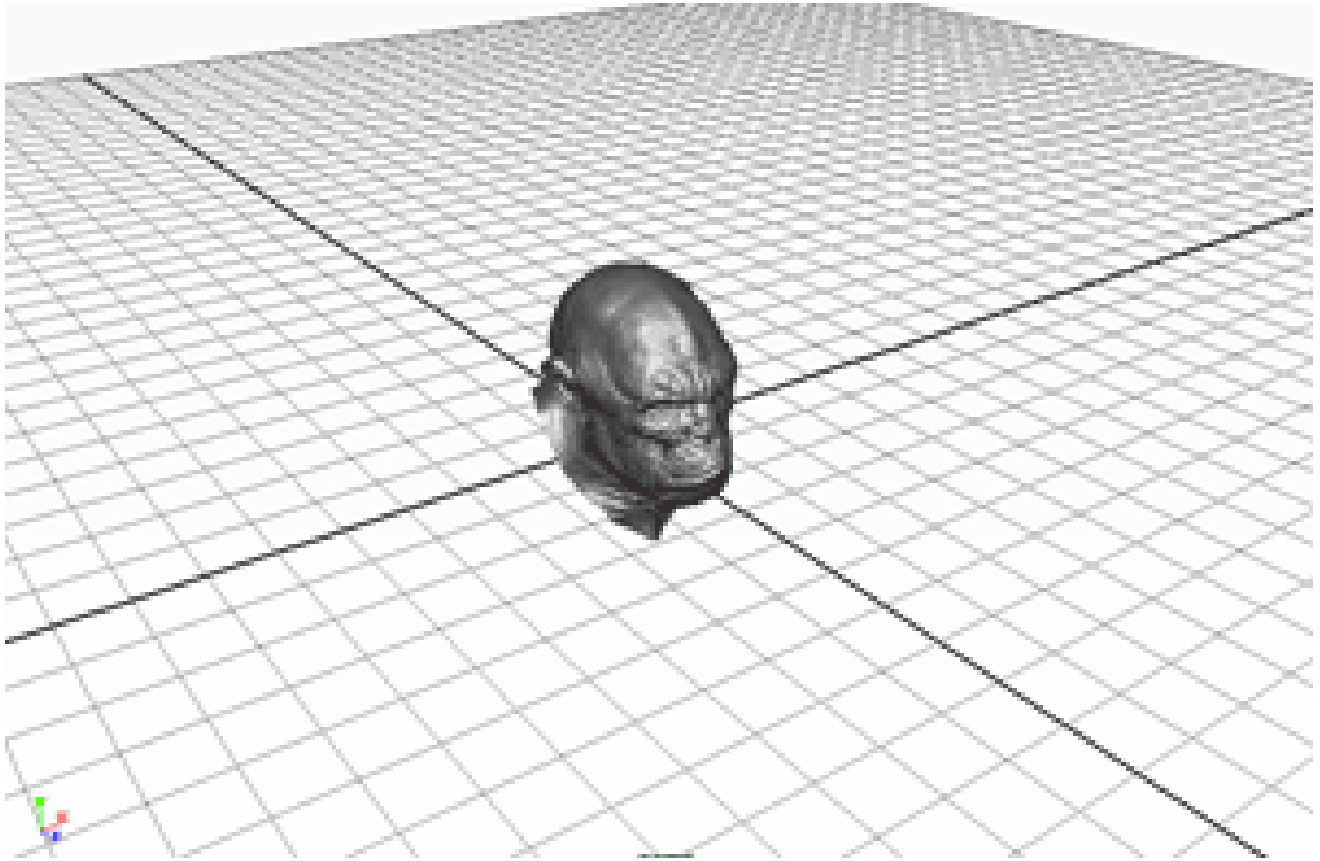
3D Axis-Aligned Bounding-Box (AABB)



Demo: http://mozdevs.github.io/gamedev-js-3d-aabb/api_box.html

More Details: https://developer.mozilla.org/en-US/docs/Games/Techniques/3D_collision_detection

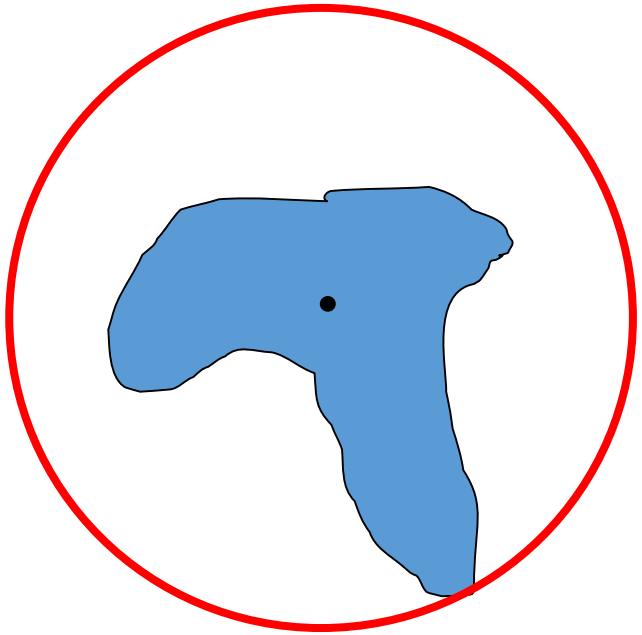
AABB's are calculated depending on object transformation



From: https://www.scratchapixel.com/lessons/advanced-rendering/introduction-acceleration-structure/bounding-volume-hierarchy-BVH-part1

Another Common Bounding Volume: Sphere

- Rotationally invariant
  - Usually
- Usually fast to compute with
- Store: center point and radius
  - Center point: object's center of mass
  - Radius: distance of farthest point on object from center of mass.
- Often not very tight fit



View volume culling

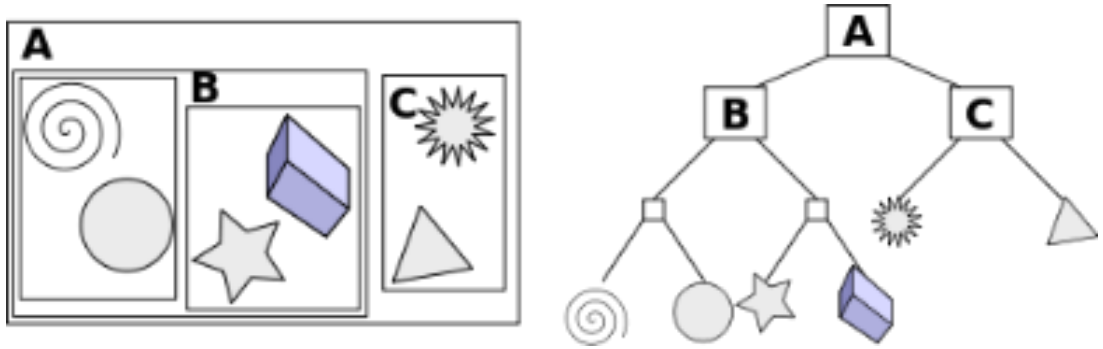
- Compare the scene hierarchically against the view volume
- When a region is found to be outside the view volume then all objects inside it can be safely discarded
- If a region is fully inside then render without clipping
- What is the difference with clipping?

Hierarchical Bounding Volumes

Collision Detection

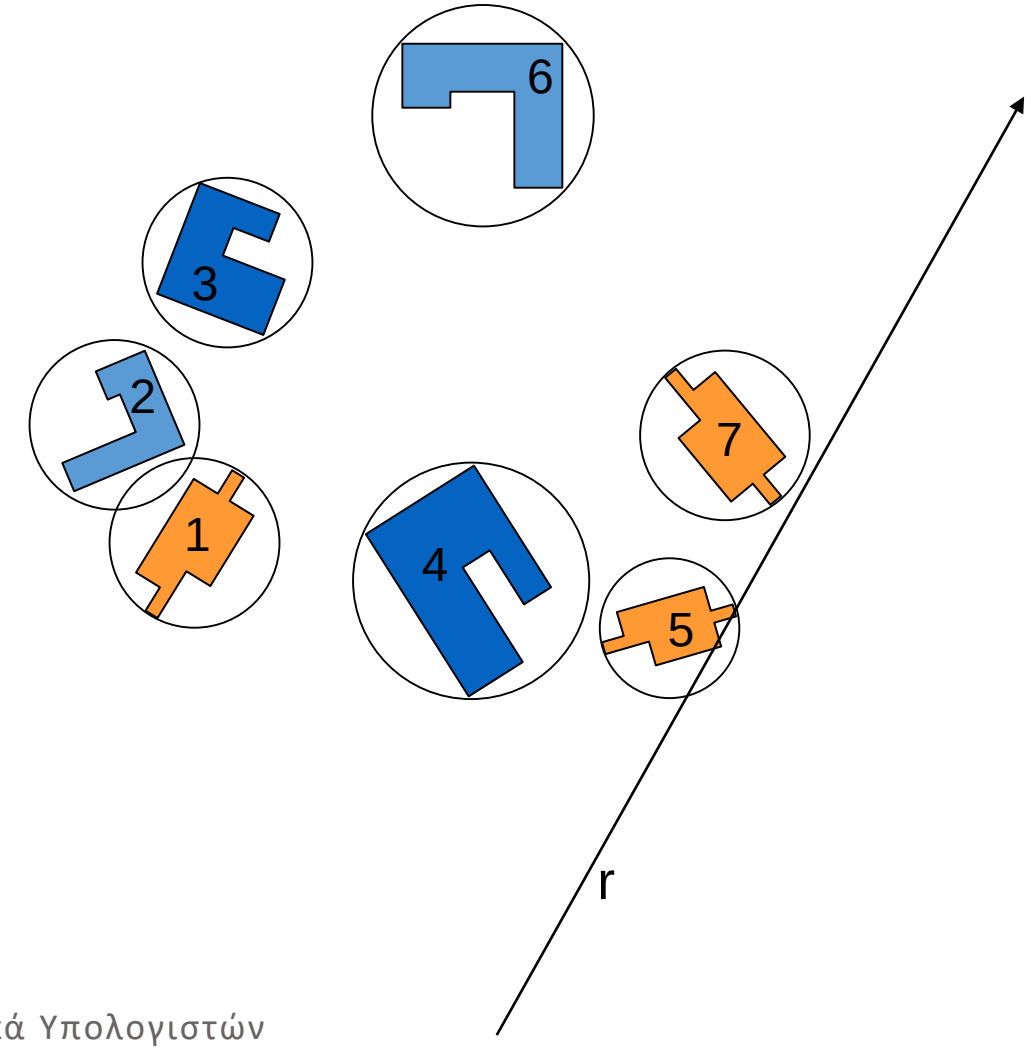
- What happens when the bounding volumes do intersect?
  - We must test whether the actual objects underneath intersect.
  - For an object made from lots of polygons, this is complicated.
  - So, we will use a bounding volume hierarchy (BVH)

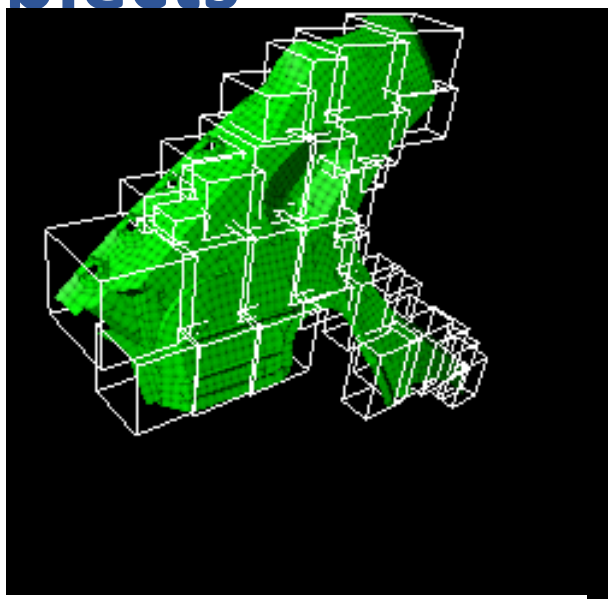
Hierarchical Bounding Volumes



- A technique for organizing the entire scene:
  - First introduced in a 1986 paper by Kay and Kajiya
  - Group bounding volumes in an encompassing bounding volume to create a tree hierarchy
- Repeatedly group bounding volumes of nearby objects together until entire scene is bounded
  - Bottom-up construction
  - To group two axis-aligned bounding boxes (AABBs) take the min and max of each AABBs
- However, finding objects that are close to each other can be difficult
  - Naïve solution involves comparing each object against each other and grouping (slow $O(n^2)$ run time)
- Could use original scene graph: bounding volumes at nodes = union of child bounding volumes
  - Easy to construct, yet…
  - Scene graph is usually logically organized but may not be spatially organized
- Commonly used spatial-partitioning algorithms include organizing objects with an octree or using top-down construction with a surface area heuristic (similar to kd-trees)

Large number of objects OR very complex objects

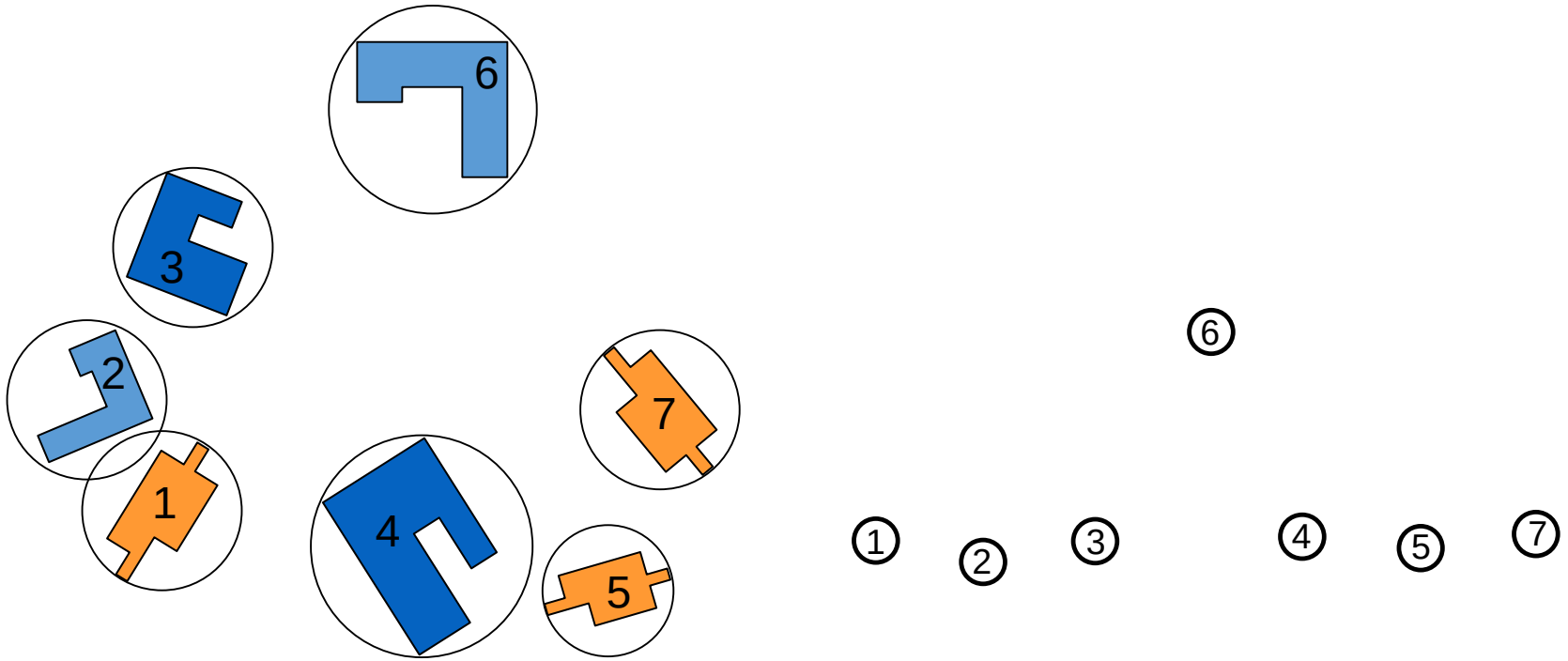




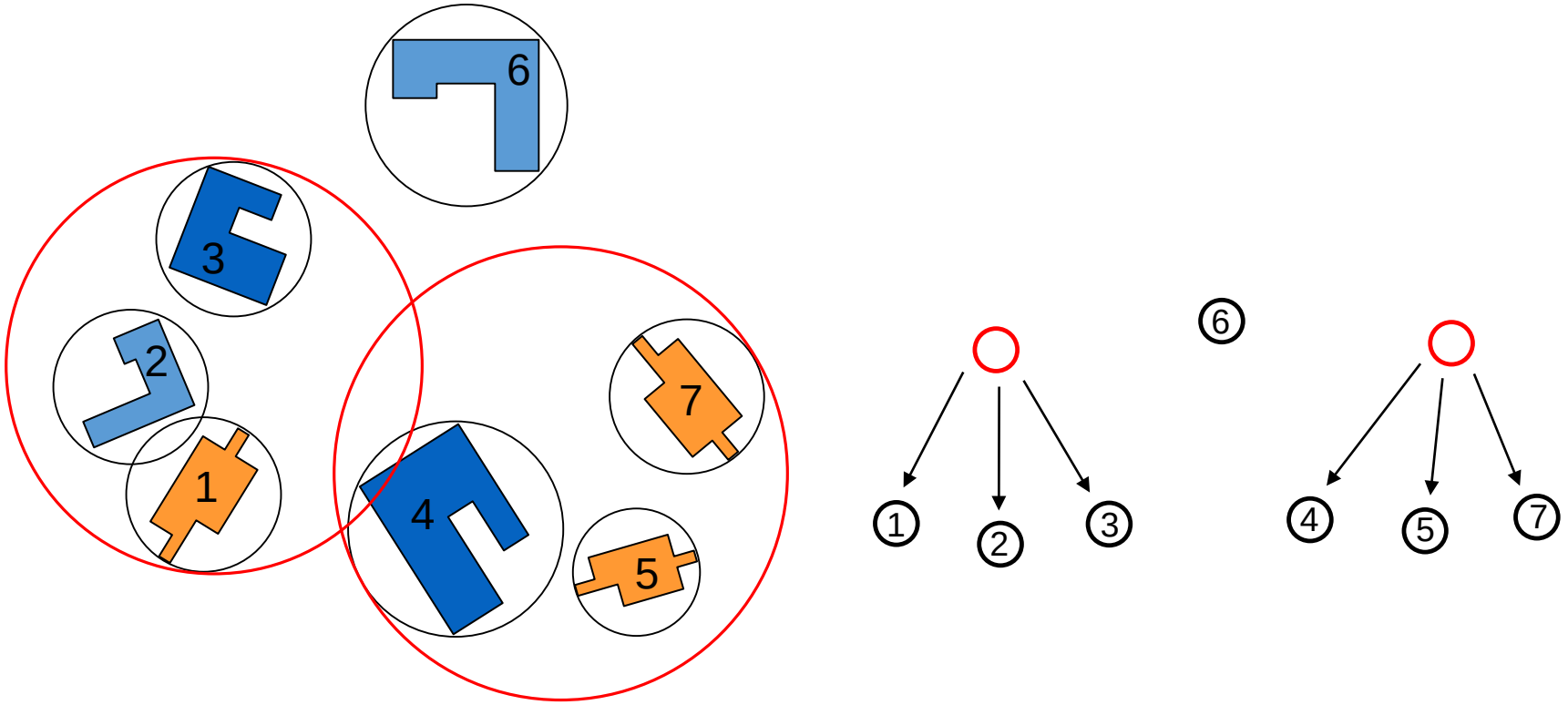
tyrannosaurus, 64 spheres

diplodocus, 60 spheres

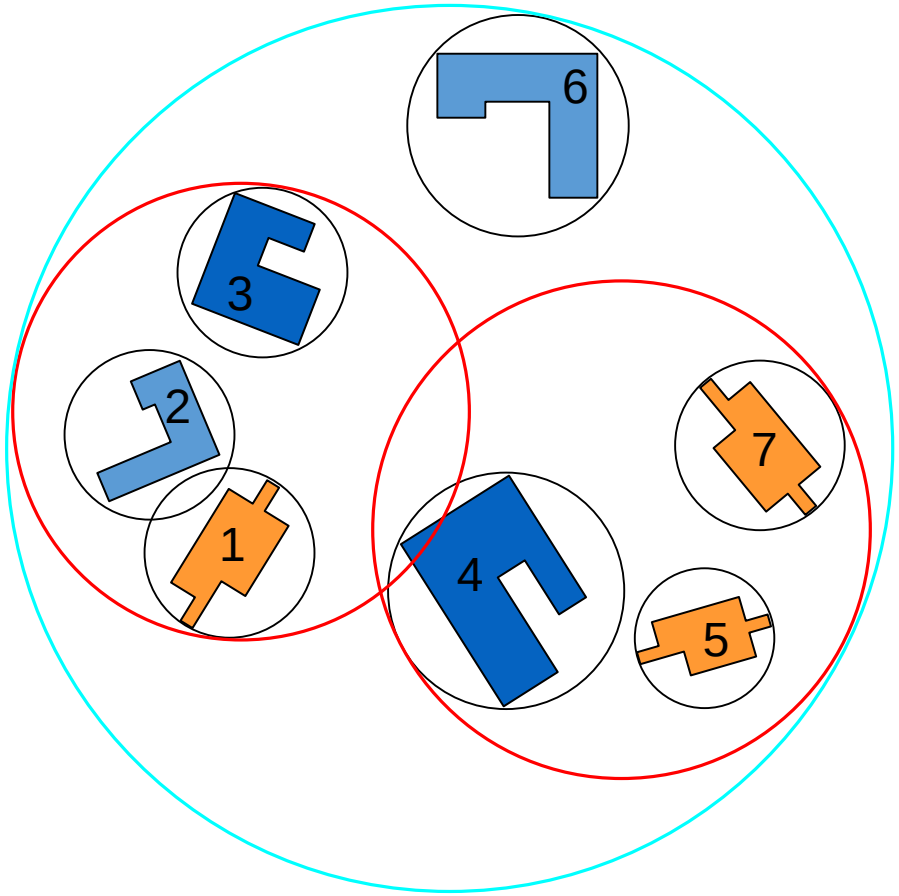
Hierarchical bounding volumes

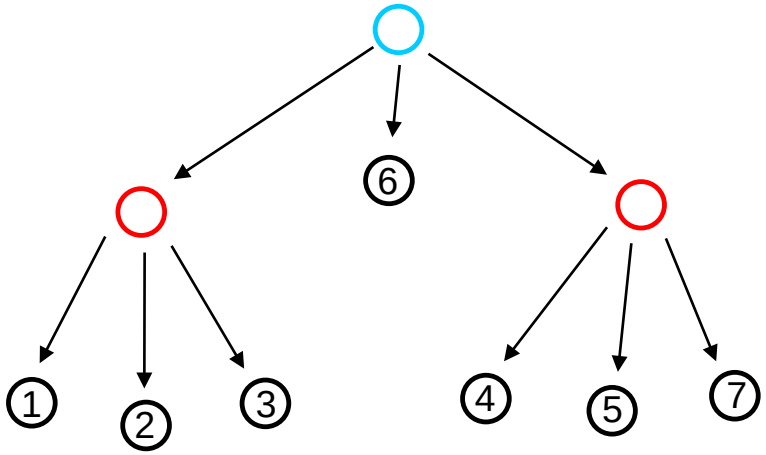


Hierarchical bounding volumes

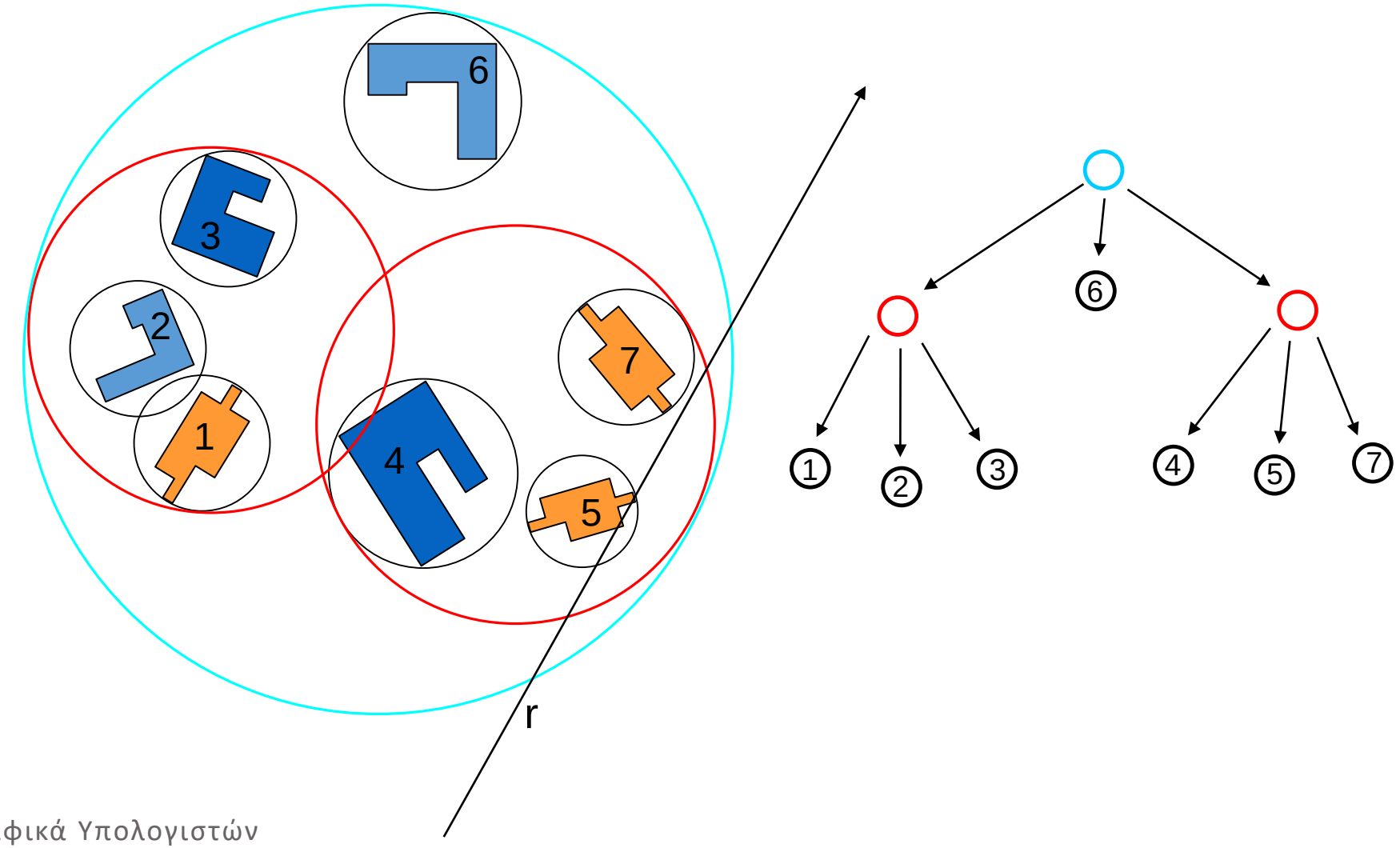


Hierarchical bounding volumes



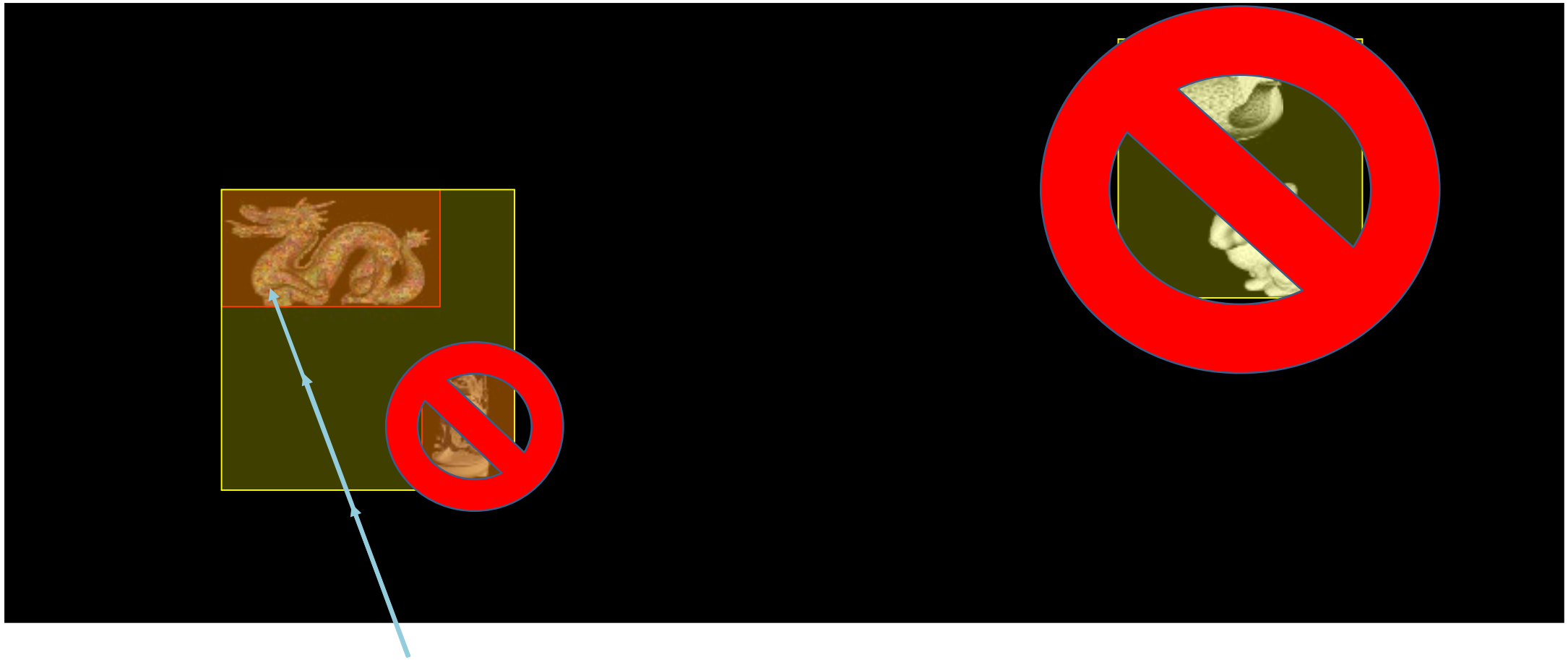


Hierarchical bounding volumes



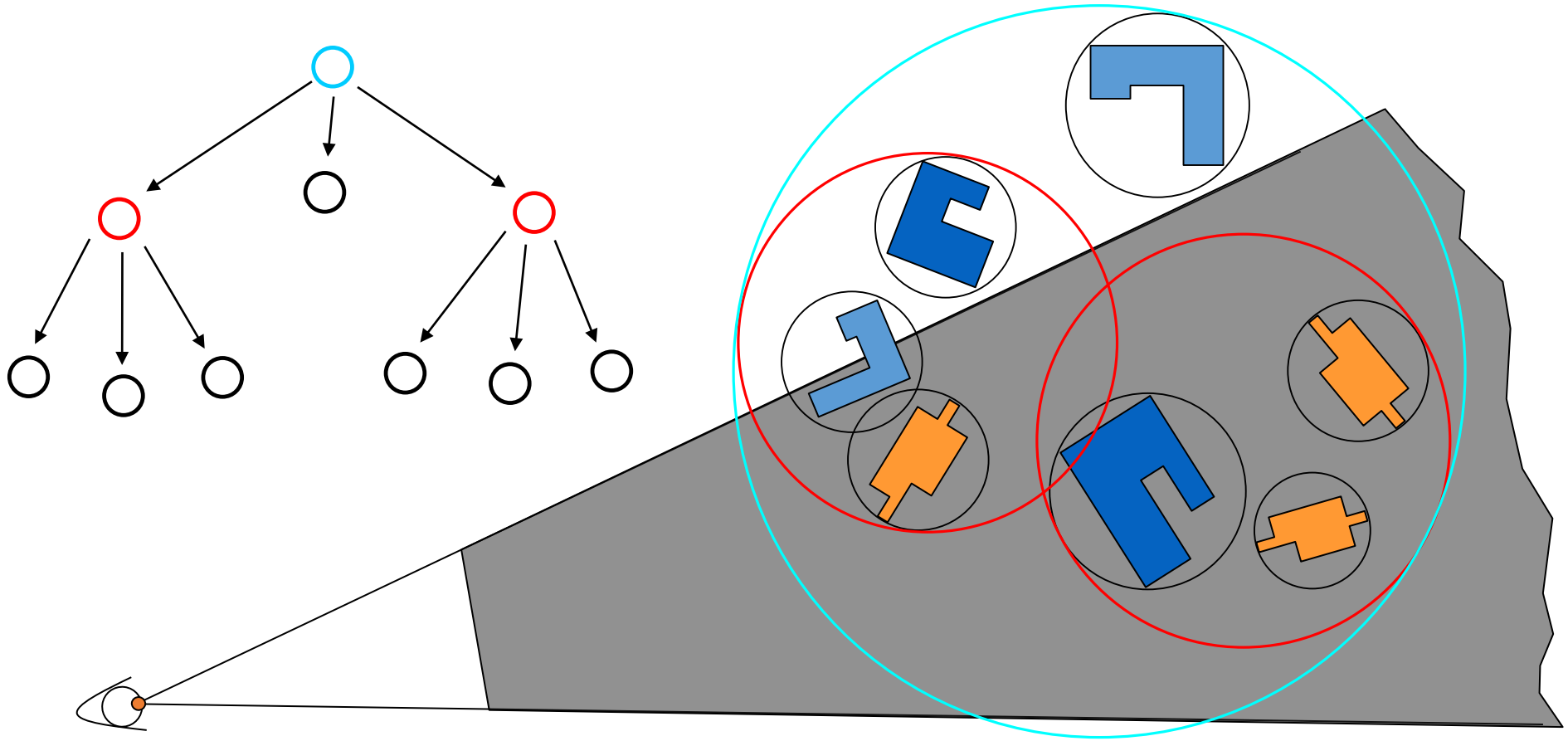
Bounding Volume Hierarchy (2/3)

- Ray Tracing: if ray intersects parent, check children, if not, discard parent[1]



1: Number of intersections needed will be O(log n) vs O(n) without BVH

View volume (or frustum) culling using a bounding volume hierarchy



Remember Clipping?

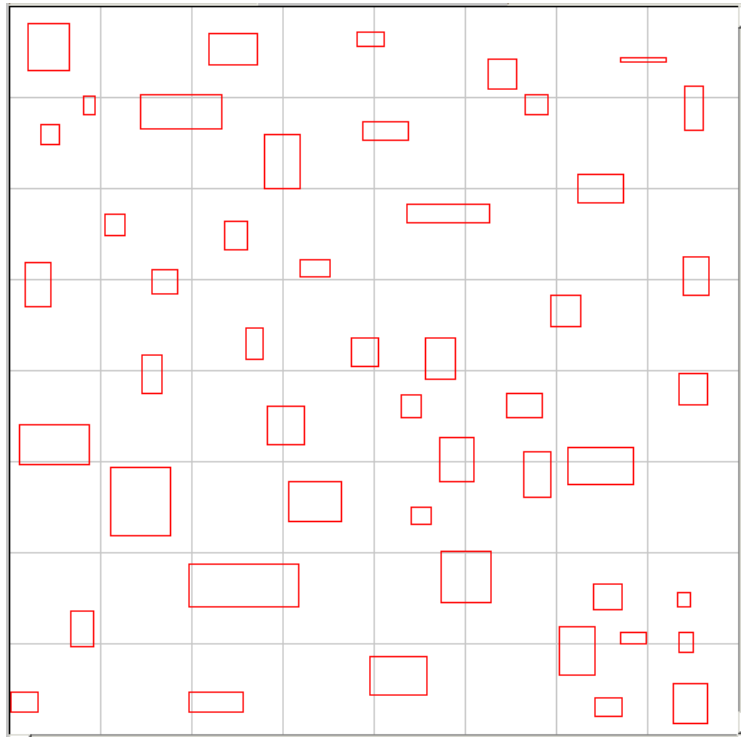
Πως κτίζουμε τις ΙΠΟ (Ιεραρχίες Περιβάλλοντων Όγκων)

Hierarchical bounding volumes

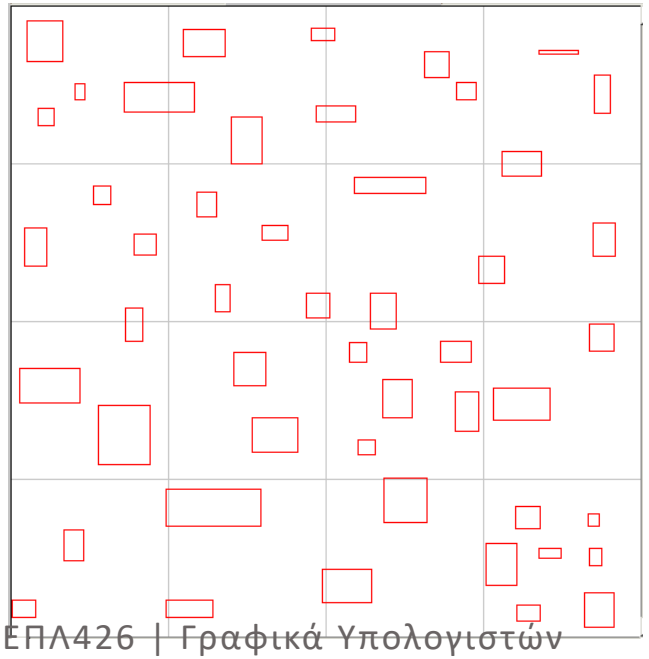
- Δύσκολο πρόβλημα!
- Η εύκολη λύση είναι να ακολουθήσουμε το γράφημα σκηνής
  - Το αποτέλεσμα εξαρτάται από την δουλειά του μοντελιστή
- Να ομαδοποιήσουμε τους ΠΟ με βάση κάποιο κριτήριο (π.χ. τις αποστάσεις)

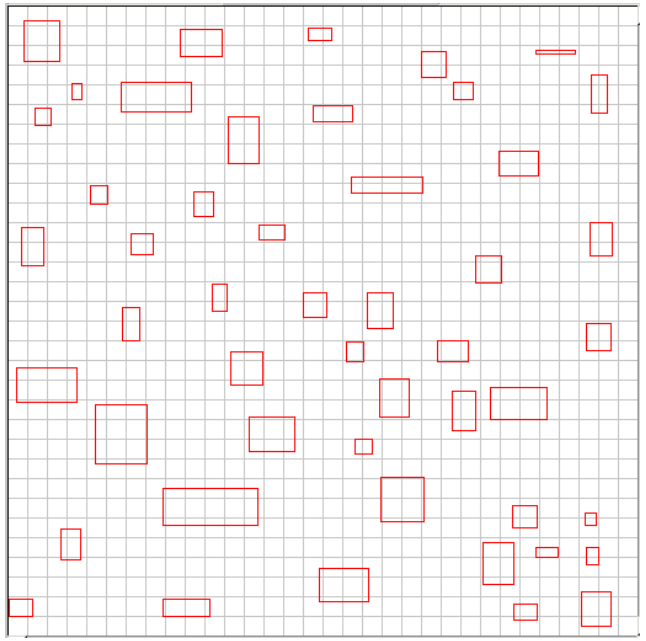
Regular Grid

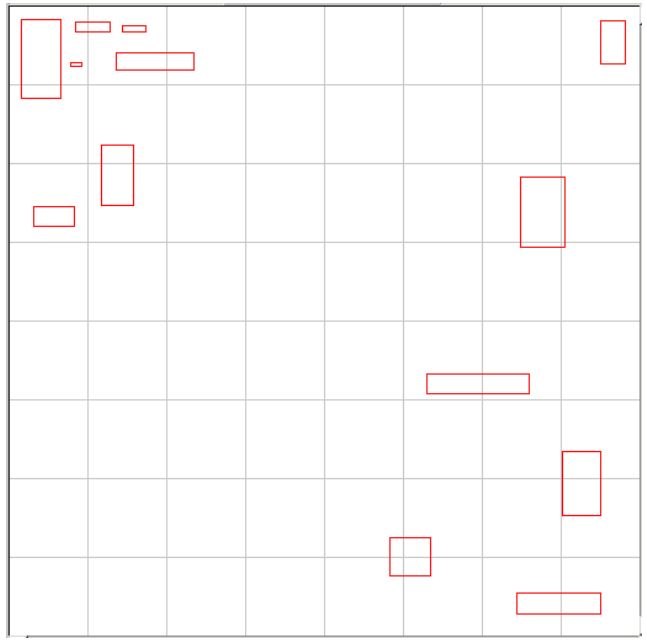
- Easy to implement
- Fast ray traversal
- Not a hierarchy
- Memory intensive, lots of empty cells
- Best suited for scenes that are evenly spread

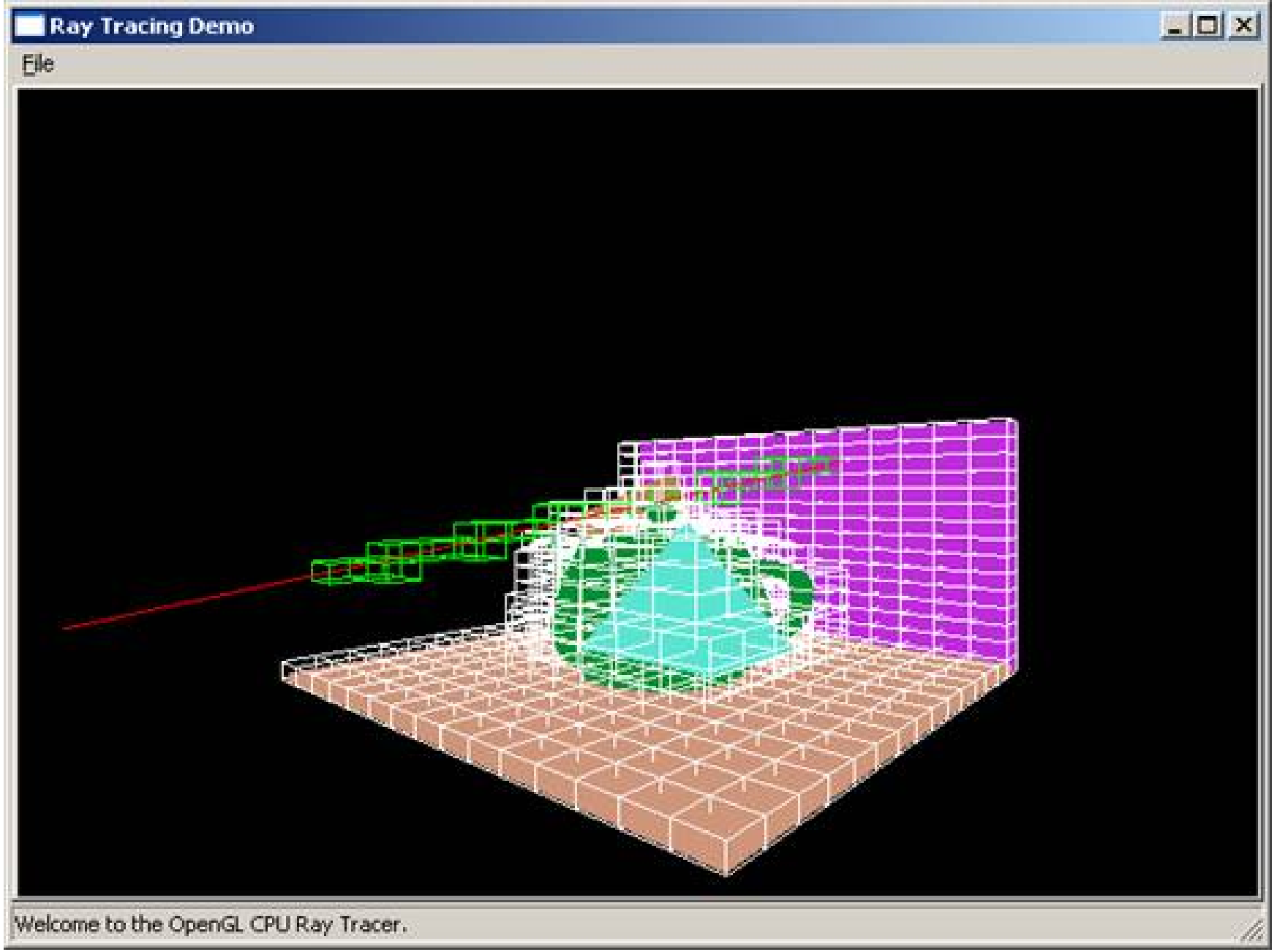


What resolution?







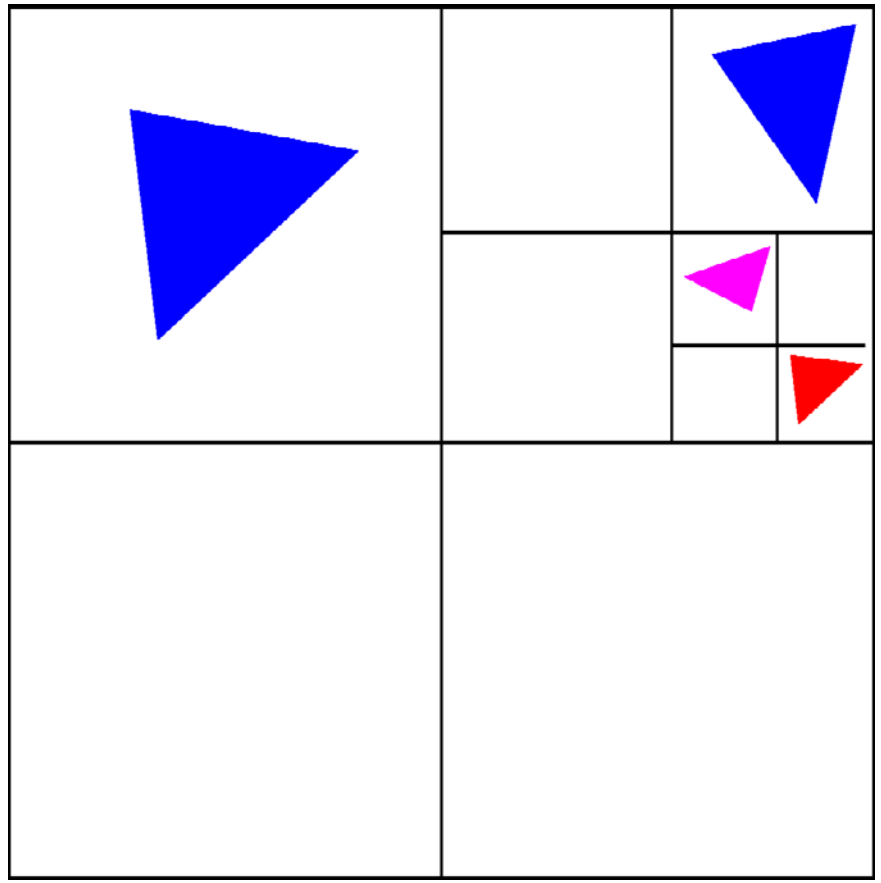

Ray Tracing Demo
File
Welcome to the OpenGL CPU Ray Tracer.

Grids – Pros and Cons

- Balance issues: some cells are clearly more important than others
  - How to determine the best cell size?
  - A lot of cells have nothing in them; it's a waste of resources to check them all
    - Use larger cell sizes?
  - But then some cells could also have too many objects
    - Use smaller cell sizes?
- Why don't we just use a finer grid?
  - Because of the expense of stepping through grid cells during traversal
  - Analogous to super-sampling from image processing unit: increasing resolution on monitors decreases visible effects of aliasing, but only at the expense of a significant amount of overhead (memory/processing)
- Not hierarchical: more traversal time, but less construction time
  - Useful for animated scenes: moving one object does not affect other objects in grid
- In general however, we'd like a smarter, more adaptive solution

Octrees – Adaptive Data Structures

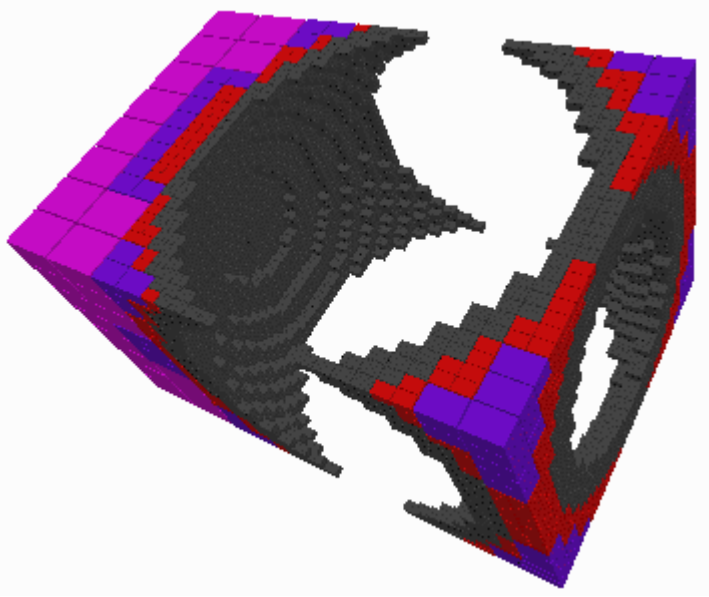
- Octrees are based on Warnock's algorithm for hidden surface removal
  - Project triangles on "screen" and recursively subdivide until the number in a given cell can be clearly solved for occlusion (typically 2)
- All nodes in the tree are AABBs (Axis-Aligned Bounding Boxes)
- Similar to grid except voxels (3D cells) do not have to be the same size.
  - Areas with greater density have more voxels



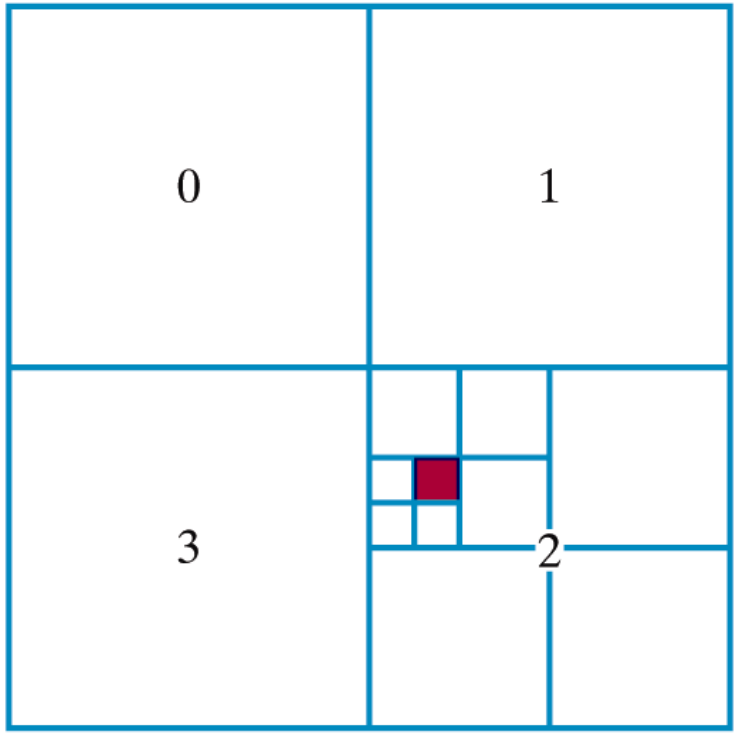
2-D example
called a quadtree

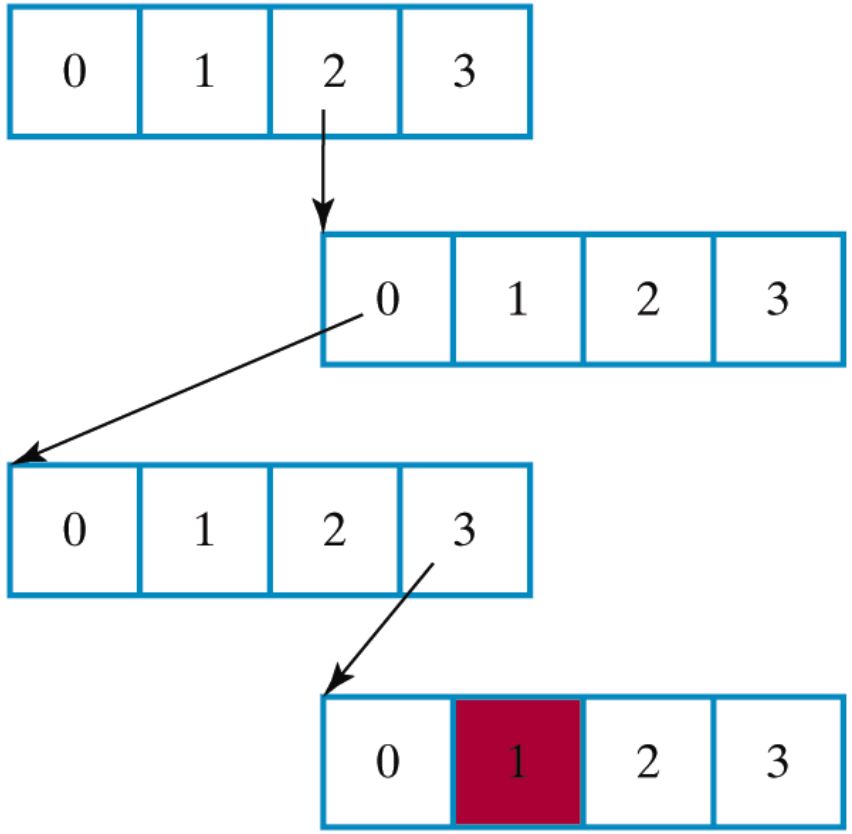
Octrees – Adaptive Data Structures

- Τα Octrees είναι ιεραρχικές δομές δέντρων που χρησιμοποιούνται για την αντιπροσώπευση στερεών αντικειμένων.
- Τα Octrees βασίζονται σε ένα δισδιάστατο σχήμα εκπροσώπησης που ονομάζεται κωδικοποίηση **quadtree**
- Η κωδικοποίηση Quadtree διαιρεί μια τετράγωνη περιοχή σε τέσσερις ίσες περιοχές έως ότου να γίνουν ομοιογενείς
- Αυτές οι περιοχές μπορούν έπειτα να τακτοποιηθούν σαν ένα δέντρο

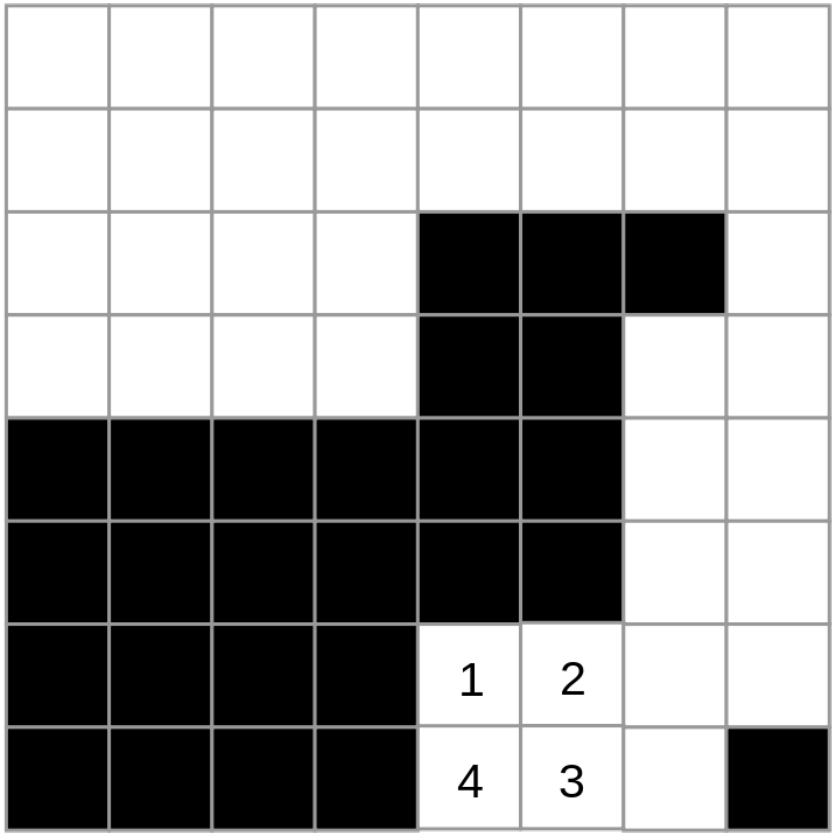


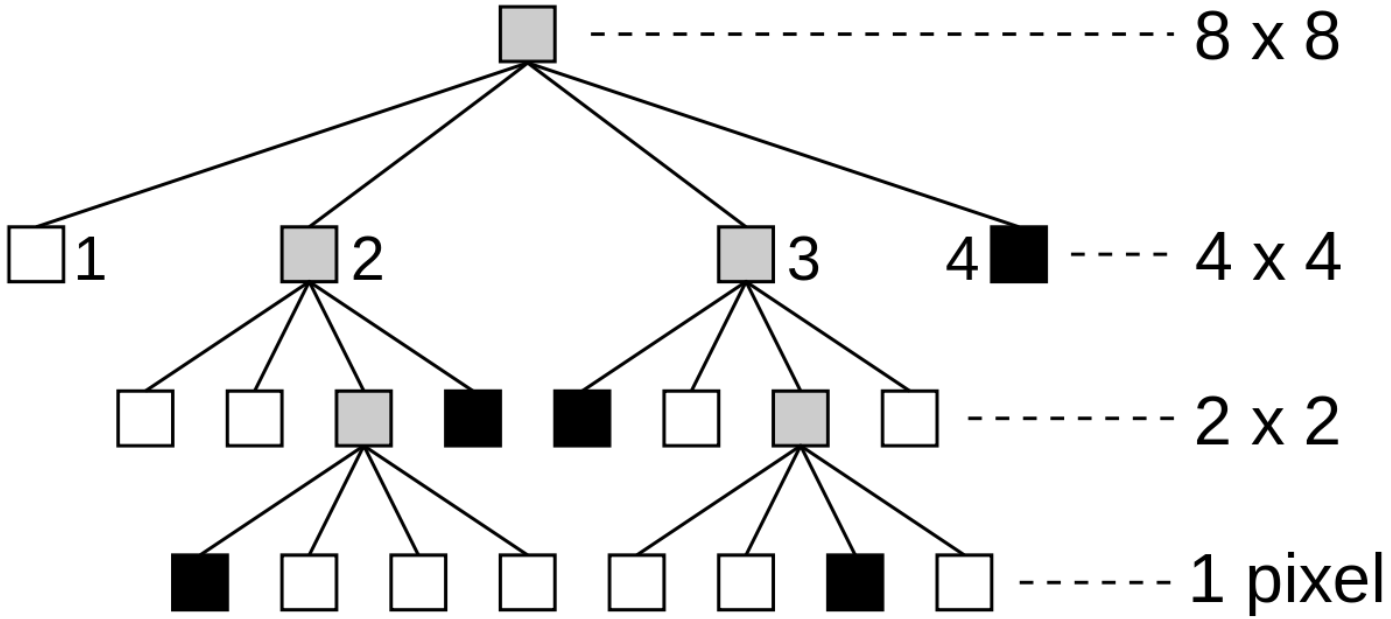
Παράδειγμα Quadtree





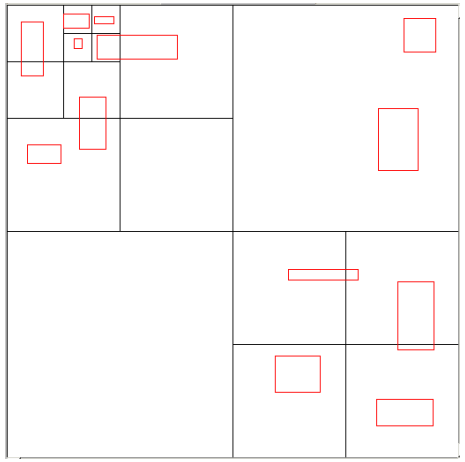
Παράδειγμα Quadtree

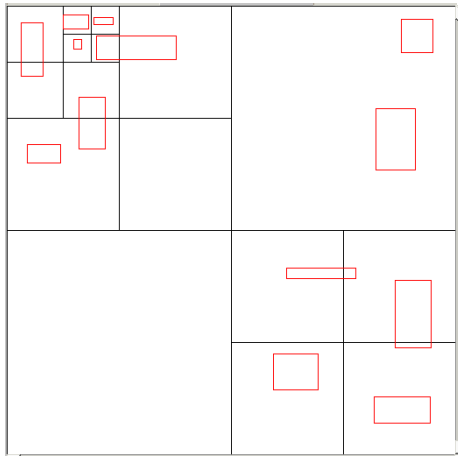


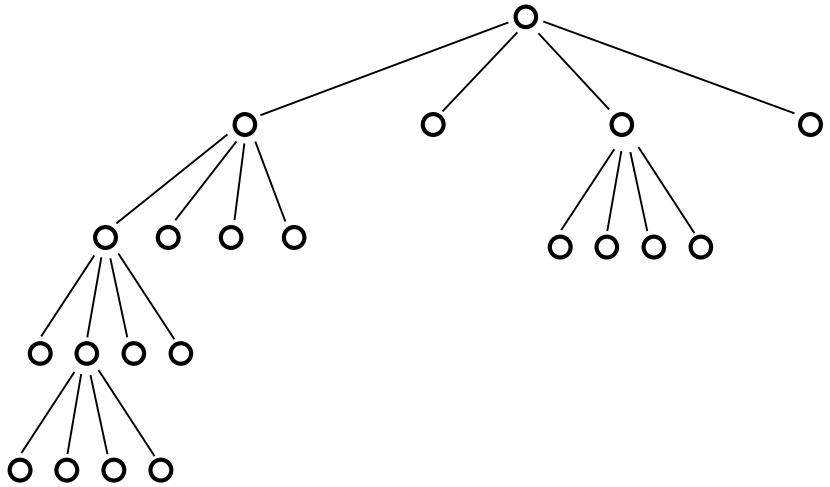


Octrees (quadtrees)

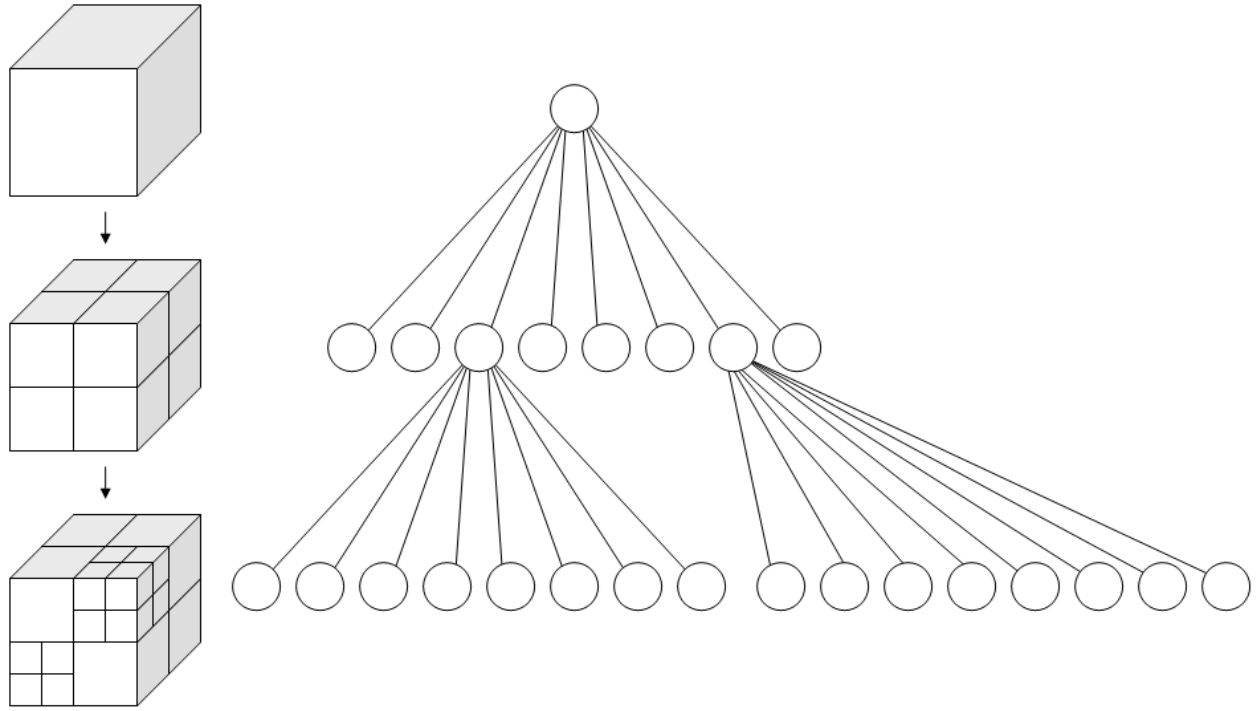
- Έχουμε τομές //ες με τους άξονες
- Κόβουμε στην μέση, με βάση κάποιο άξονα, σε κάθε βήμα
- Εύκολη υλοποίηση, χωρίς δύσκολες αποφάσεις
- Διασχίζεται πολύ «φτηνά»

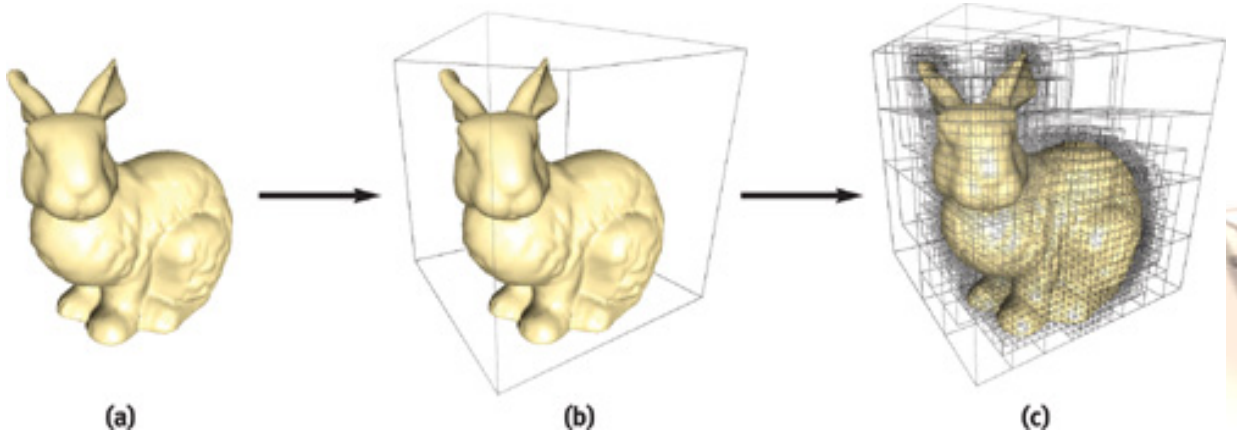


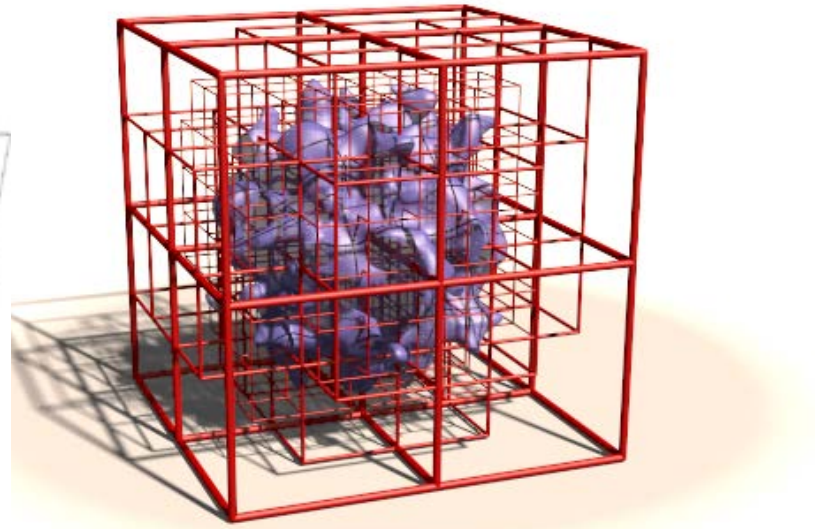




Octrees





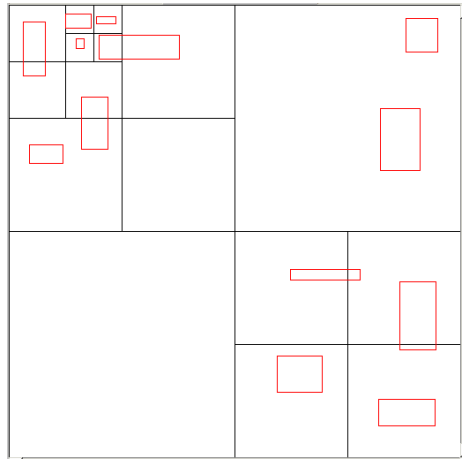


Octree Traversal

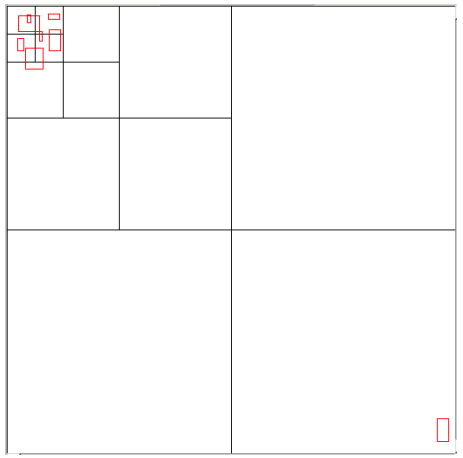
- the process of finding the subset of voxels in an octree pierced by a directed line
- Only ray trace the objects in the voxels

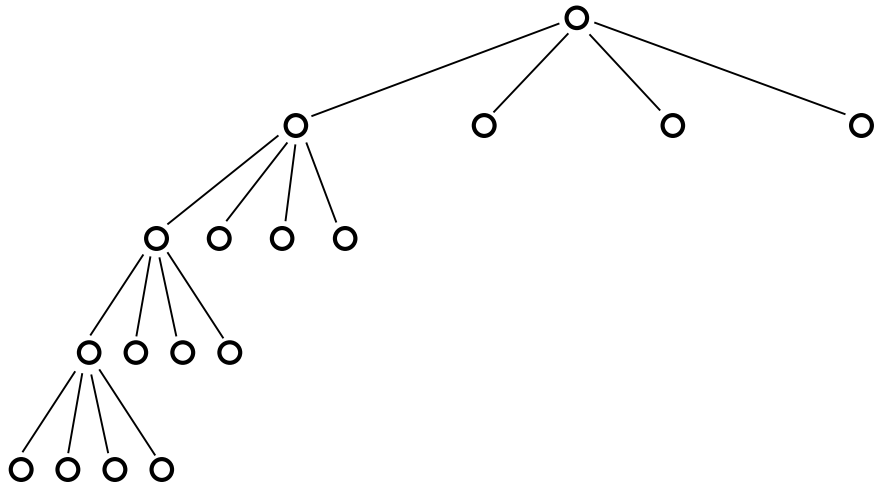
- Two groups of algorithms
  - Bottom-up: traversing starts at first terminal node intersected by ray. Use neighbor-finding to obtain the next node
    - κάπως πολύπλοκο στην υλοποίηση
  - Top-down: start from the root node; recursive down to the terminal voxel
    - πιο εύκολο στην υλοποίηση

Octrees (quadtrees)



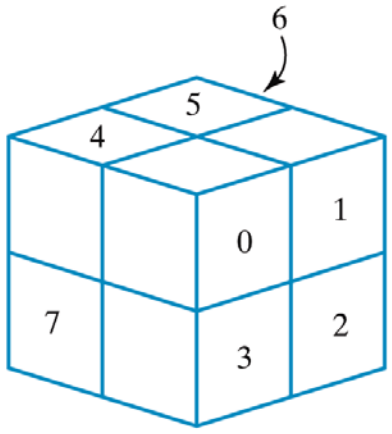
- Έχουμε τομές //ες με τους άξονες
- Κόβουμε στην μέση, με βάση κάποιο άξονα, σε κάθε βήμα
- Εύκολη υλοποίηση, χωρίς δύσκολες αποφάσεις
- Διασχίζεται πολύ «φτηνά»
- **Μπορεί όμως να μας δώσει υπερβολική κατάτμηση**



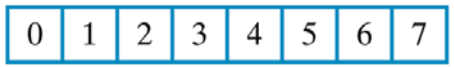


Octrees

- Η κωδικοποίηση Quadtree παρέχουν σημαντική εξοικονόμηση χώρου και μνήμης όταν υπάρχουν μεγάλες περιοχές με το ίδιο χρώμα σε μια περιοχή
- Τα octrees έχουν την ίδια λογική με τα quadtrees, αλλά διαιρούν μια περιοχή π.χ. κύβο σε octants
- Κάθε περιοχή σε ένα octree αναφέρεται σε **volume element** ή **voxel**
- Η διαίρεση συνεχίζεται μέχρι όλα να έχουν ομοιογενείς περιοχές

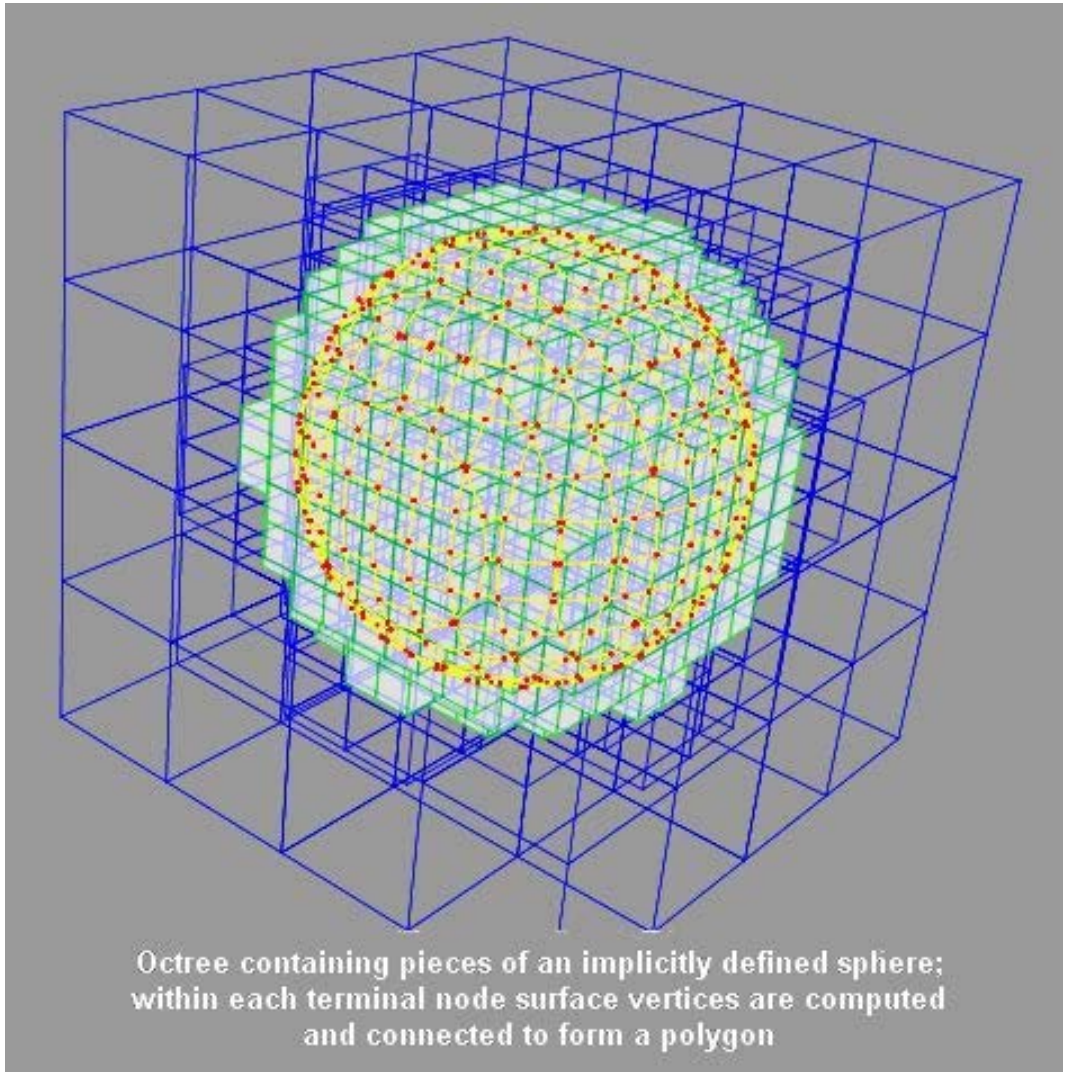


Region of a Three-Dimensional Space

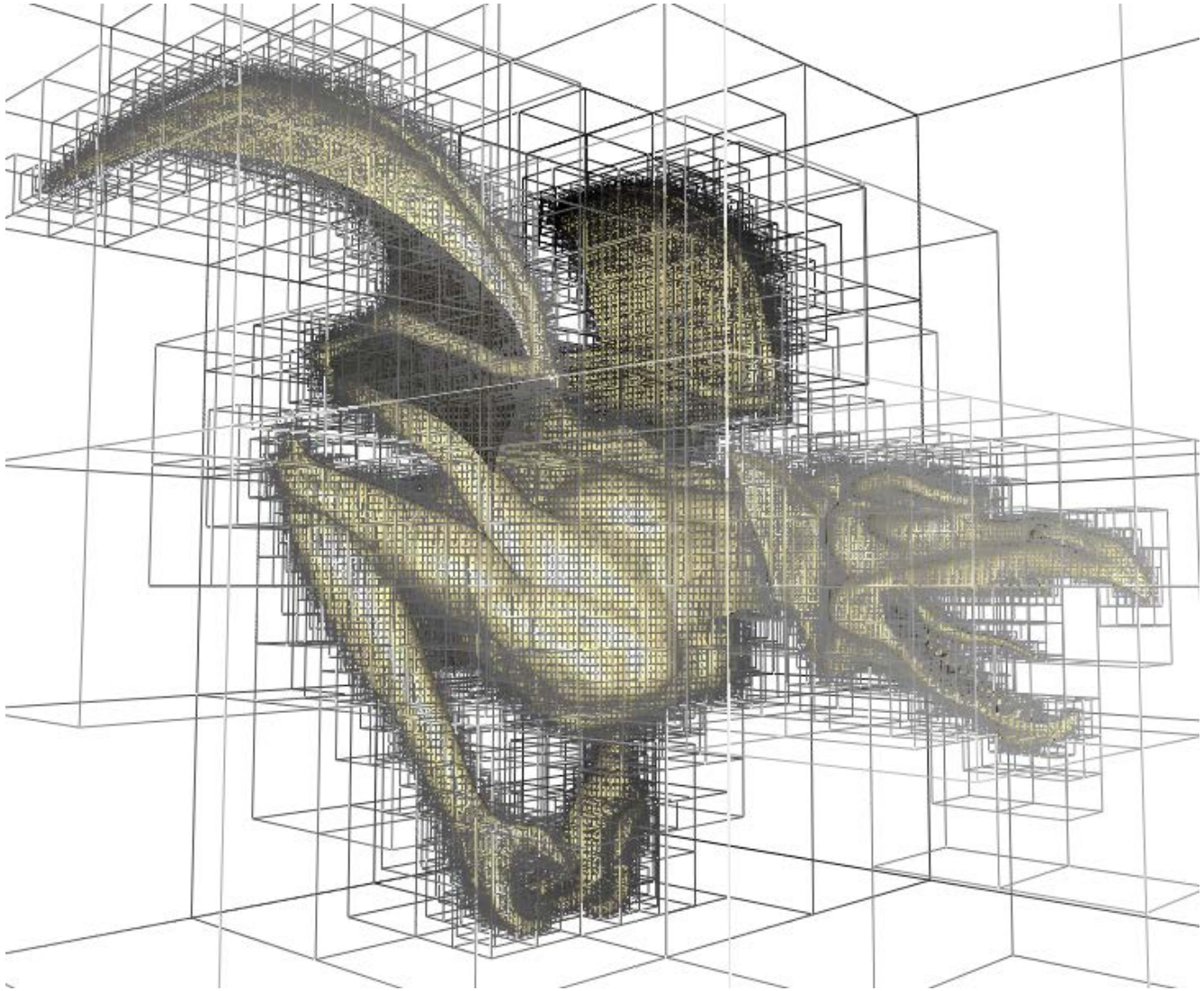


Data Elements in the Representative Octree Node

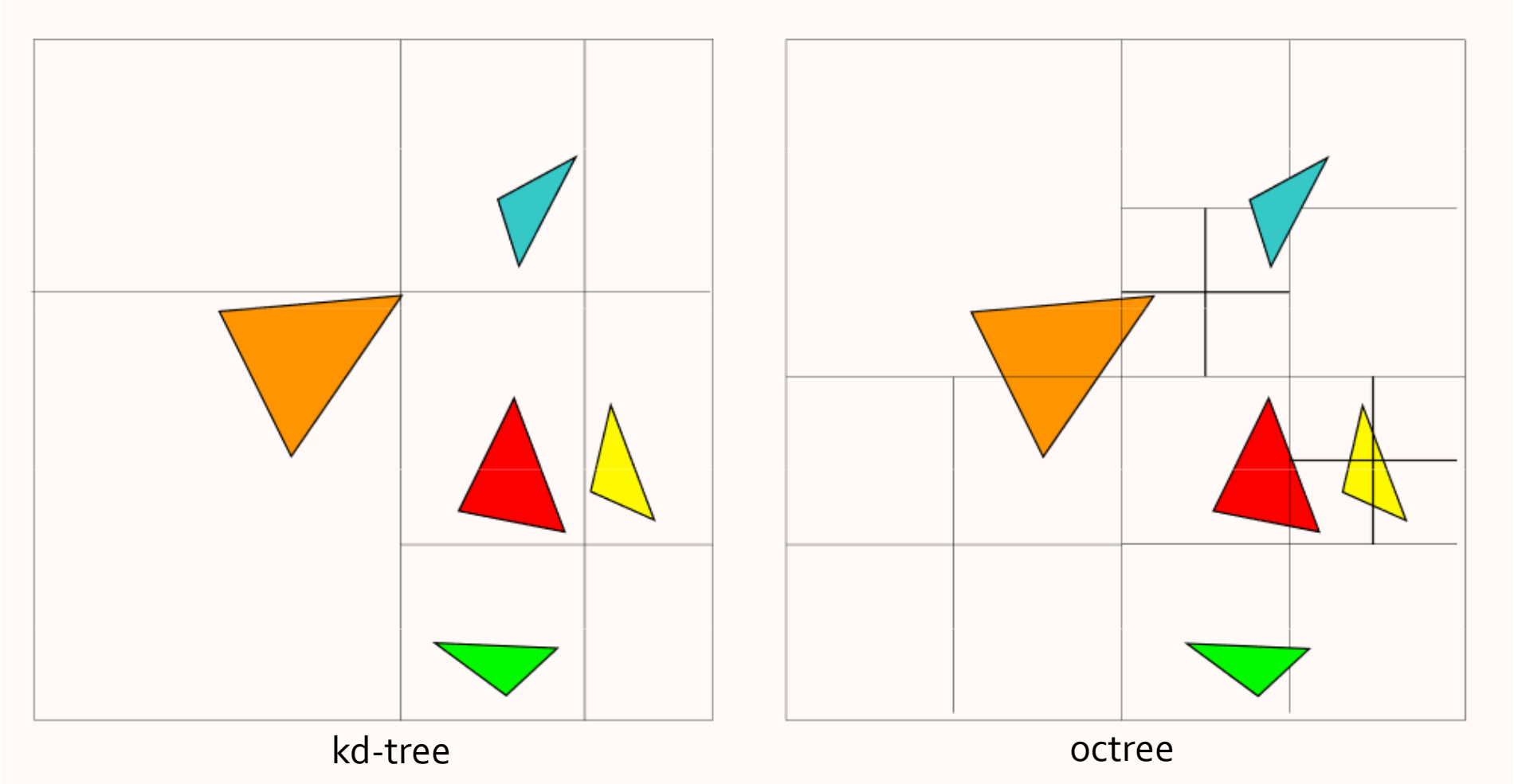
Παράδειγμα Octree



Παράδειγμα Octree



KD-tree Vs Octree

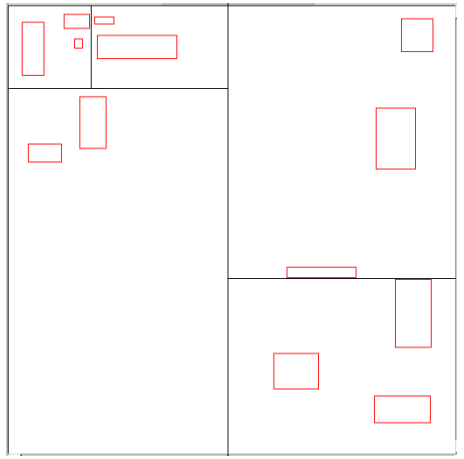


http://www.cs.tau.ac.il/~dcor/Graphics/adv-slides/RT_acceleration13.pdf -- Slide 46, Fredo Durand slides

kd-trees - Motivation

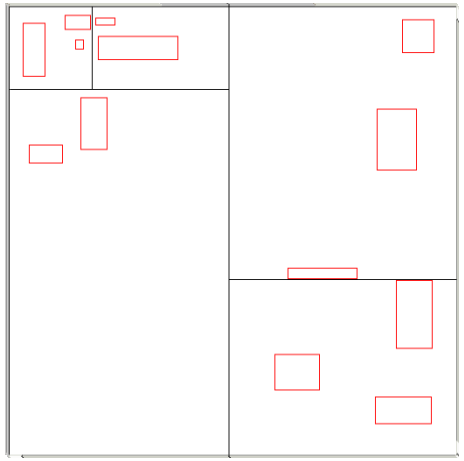
- Let's be smarter about determining the sizes of nodes in the tree
  - Tradeoff between spending time intersecting with nodes (deep tree) vs intersecting with scene primitives (shallow tree)
  - We want to isolate areas of high complexity and cut out large empty spaces, which we can step through quickly during traversal
  - Solution: kd-tree
- Definition: a kd-tree is a k-dimensional, axis-aligned binary tree
  - Axis-aligned like octree
  - Binary tree unlike octree - choose one axis to split along at each node
- The main challenge with kd-trees used for spatial partitioning is determining where to position the split plane at a given node (including which axis to split along)
  - Would like splitting plane to adapt to the locations of objects
  - Degenerate kd-tree is identical to its octree equivalent
  - Let's look in-depth at how we would select one split for a single node

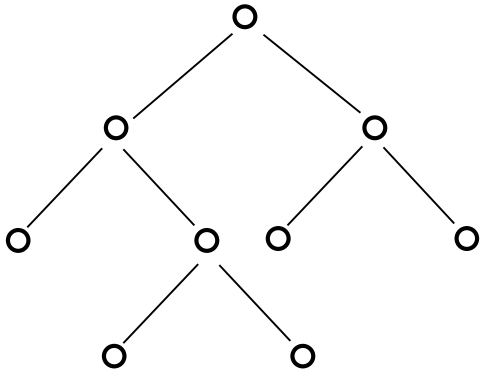
Kd-δέντρο



- Μια από τις πιο συνήθεις μεθόδους

- Έχουμε τομές //ες με τους άξονες
- Δυαδικός διαχωρισμός, εναλλάξ στους άξονες
- Εύκολη υλοποίηση, αλλά αν θέλουμε καλό δέντρο θέλει κόπο
- Διάσχιση: λίγο πιο ακριβό σε κάθε βήμα, αλλά κάνει λιγότερα βήματα





Binary Space Partition Trees

Binary Space Partition Trees (1979)

- Πιο γενικό, μπορεί να μεταχειριστεί αντικείμενα που δεν διαχωρίζονται
- Δουλεύει με τα πολύγωνα παρά με τα αντικείμενα
- Αυτόματο, χρησιμοποιεί επίπεδα χωρισμού (partitions planes) που ορίζονται από τα πολύγωνα της σκηνής

- Η μέθοδος έχει δύο βήματα:
  1. κτίσιμο δέντρου ανεξάρτητου από το viewpoint
  2. διάσχιση του δέντρου από δεδομένο viewpoint για να πάρεις την σειρά ορατότητας (visibility ordering)

Binary Space Partition Trees (1979)

- BSP tree: Οργανώστε όλο το χώρο (***partition***) σε δυαδικά δέντρα
  - ***Preprocess***: δημιουργία ενός δυαδικού δέντρου με αντικείμενα στη σκηνή
  - ***Runtime***: η σωστή διέλευση αυτού του δέντρου απαριθμεί τα αντικείμενα από πίσω προς τα εμπρός
  - ***Idea***: Διαιρέστε το διάστημα αναδρομικά σε «μισούς χώρους» με τη χρήση ***splitting planes***
    - Τα επίπεδα μπορούν χωριστούν με αυθαίρετο προσανατολισμό

Κτίσιμο του BSP δέντρου

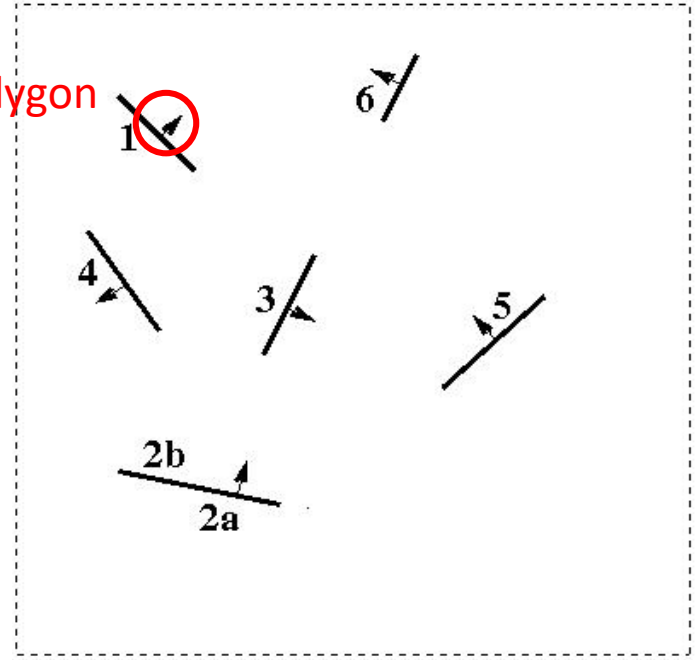
- Αρχίζουμε με ένα σύνολο πολυγώνων και κενό δέντρο
- Επιλέγουμε ένα από αυτά και το κάνουμε ρίζα του δέντρου
- Χρησιμοποιούμε το επίπεδο για να μοιράσουμε τα υπόλοιπα πολύγωνα σε 3 σύνολα: *μπροστά, πίσω, συν-επίπεδα.*
  - Πολύγωνα που σταυρώνουν το επίπεδο μοιράζονται
- Επαναλαμβάνουμε τη διαδικασία αναδρομικά με τα *μπροστά* και *πίσω* σύνολα, δημιουργώντας τα ***μπροστά*** και ***πίσω*** υποδέντρα αντίστοιχα

Κτίσιμο του BSP δέντρου (Αναδρομικά)

Σύνολο πολυγώνων σε 2D

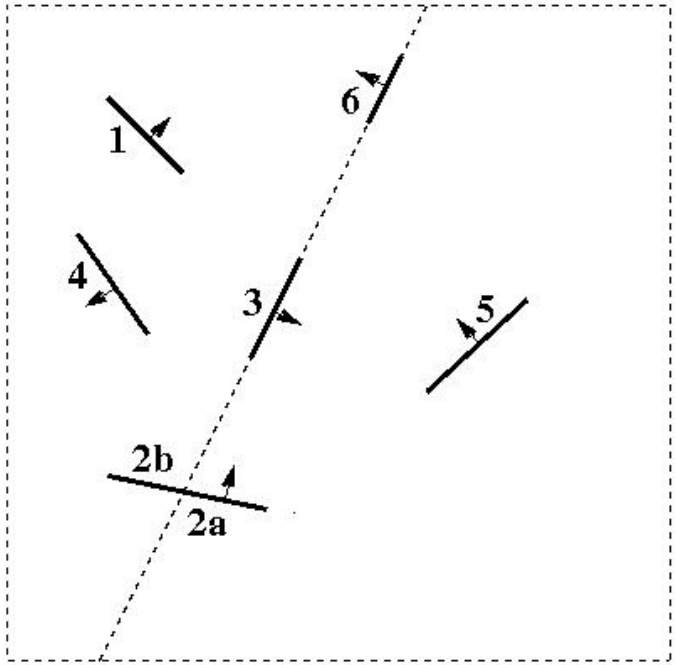
Το δέντρο

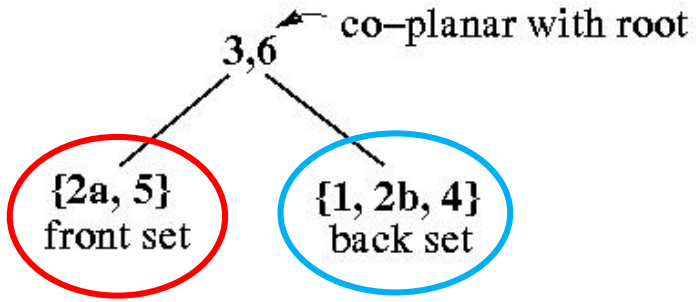
Normal of the polygon



{1, 2, 3, 4, 5, 6}

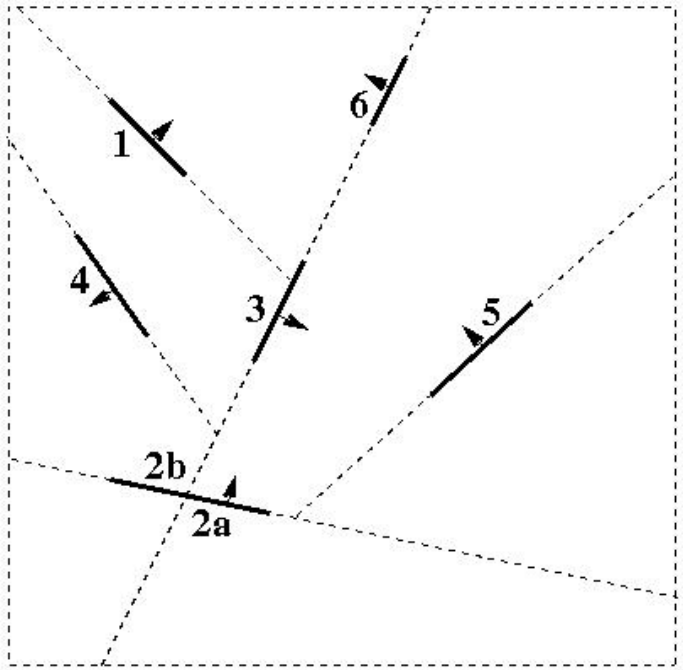
Κτίσιμο του BSP δέντρου (Αναδρομικά)

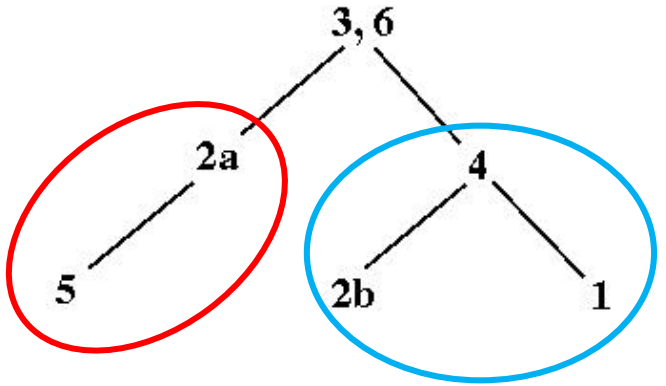




Διαλέγουμε ένα πολύγωνο και χωρίζουμε το χώρο και τα πολύγωνα

Κτίσιμο του BSP δέντρου (Αναδρομικά)



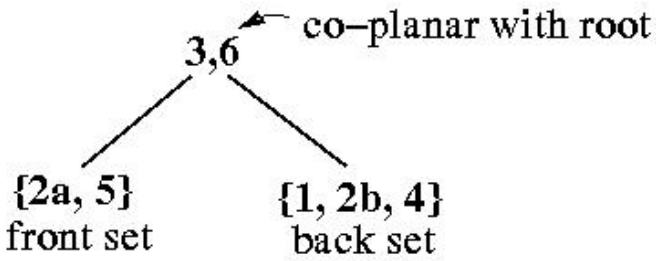


{1, 4, 2b, 3, 6, 2a, 5}

Αναδρομικά χωρίζουμε το κάθε υποδέντρο μέχρι να χρησιμοποιηθούν όλα τα πολύγωνα

C-style Κώδικας

```
typedef struct _bspTree{
        PolygonList *poly; /*list of faces on node*/
        PlaneEq plane; /* separating plane*/
        struct _bspTree *front; /*positive subspace*/
        struct _bspTree *back; /*negative subspace*/
} BSPTree;
```



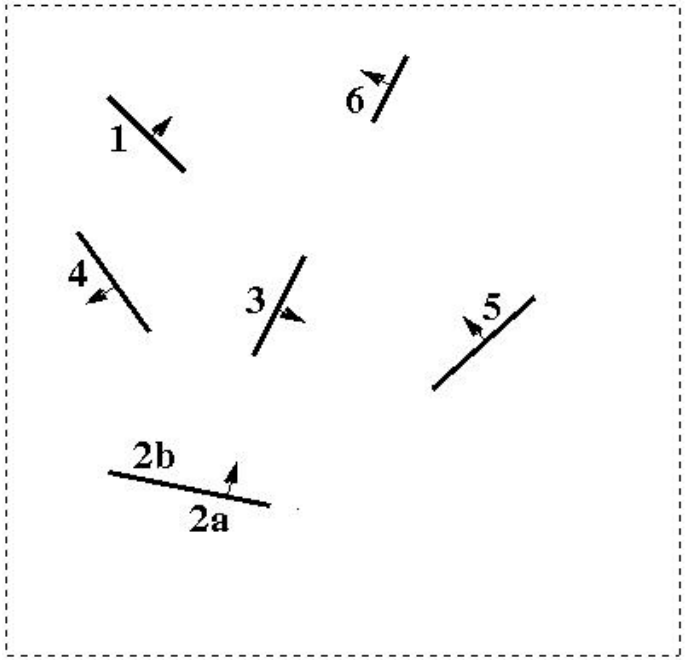

```
function makeTree(polygon_list) {
    if Empty(polygon_list)
          then return EMPTY_TREE;
    else {
          root = selectPolygon(polygon_list);
          back_list = NULL;
          front_list = NULL;
          for(each polygon in polygon_list) {
               c = splitPolygon(root->plane, polygon, front, back)
               switch(c) {
                     case ON          : appendList(root, polygon); break;
                     case IN_FRONT    : appendList(front_list, polygon); break;
                     case AT_BACK     : appendList(back_list, polygon); break;
                     case SPLIT       : appendList(front_list, polygon);
                                        appendList(back_list, polygon);
               }
          }
          bspt->root = root
          bspt->front = makeTree(front_list);
          bspt->back = makeTree(back_list);
          return bspt;
        }
}
```

Κτίσιμο του BSP δέντρου (Αυξητικά)

- Το δέντρο μπορεί επίσης να κτιστεί αυξητικά:
  - αρχίζουμε με ένα σύνολο από πολύγωνα και κενό δέντρο
  - εισάγουμε τα πολύγωνα στο δέντρο, ένα κάθε φορά
  - η εισαγωγή ενός πολυγώνου γίνεται συγκρίνοντας το σε σχέση με τα άλλα πολύγωνα στον κάθε κόμβο και σπρώχνοντας το στη σωστή πλευρά, μοιράζοντας το αν είναι απαραίτητο (splitting)
  - όταν το πολύγωνο φτάσει ένα κενό κελί, δημιουργούμε ένα κόμβο με το επίπεδο υποστήριξης του (its supporting plane)

Κτίσιμο του BSP δέντρου (Αυξητικά)

Σύνολο από πολύγωνα σε 2D



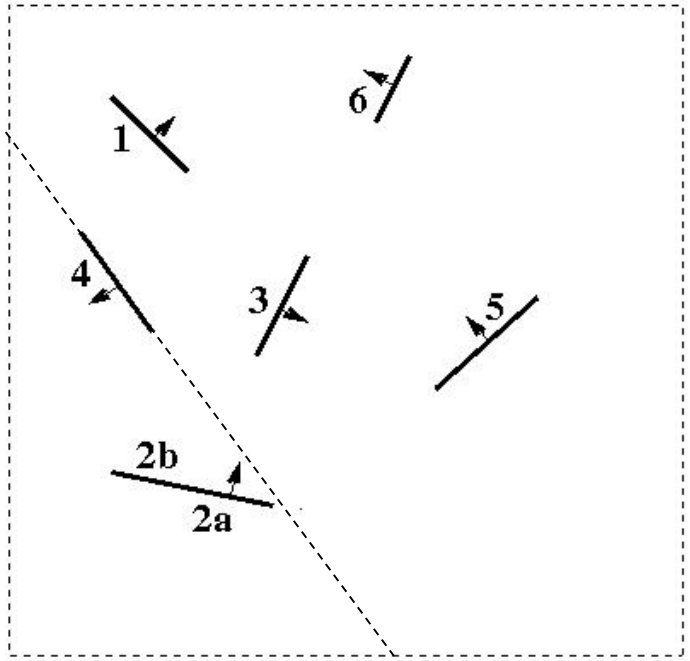
{1, 2, 3, 4, 5, 6}

Το δέντρο

4

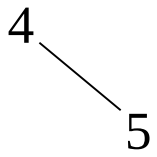
Κτίσιμο του BSP δέντρου (Αυξητικά)

Σύνολο από πολύγωνα σε 2D



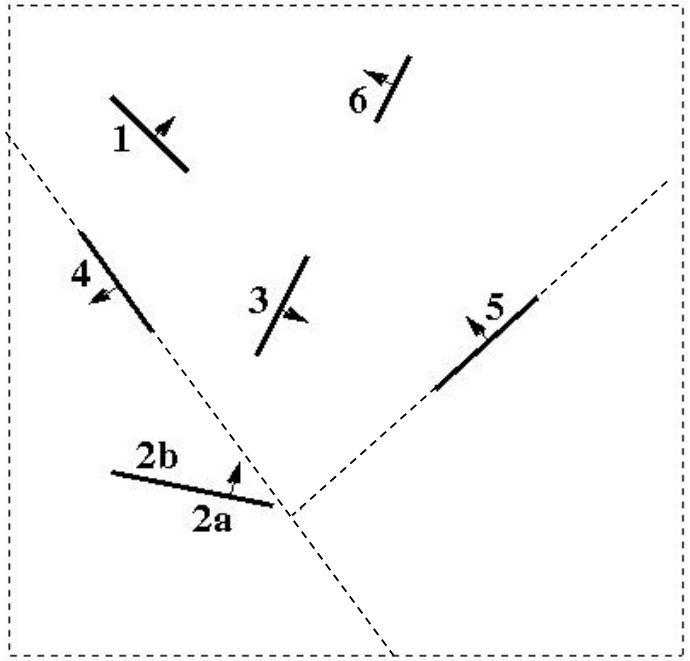
{1, 2, 3, 4, 5, 6}

Το δέντρο



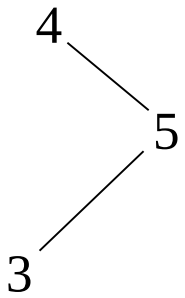
Κτίσιμο του BSP δέντρου (Αυξητικά)

Σύνολο από πολύγωνα σε 2D



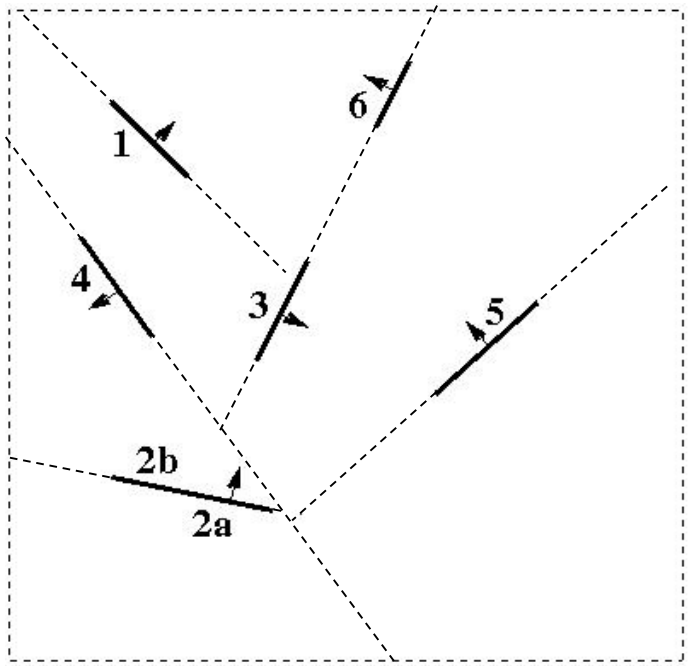
{1, 2, 3, 4, 5, 6}

Το δέντρο



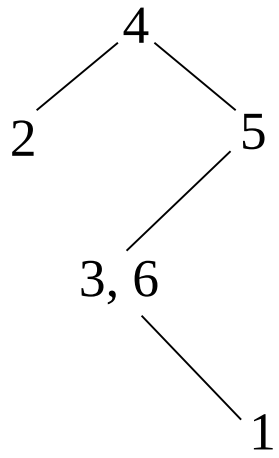
Κτίσιμο του BSP δέντρου (Αυξητικά)

Σύνολο από πολύγωνα σε 2D



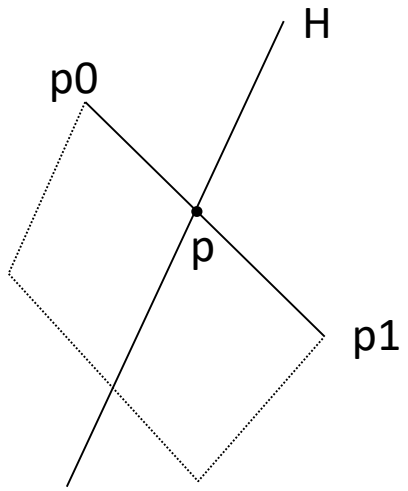
{1, 2, 3, 4, 5, 6}

Το δέντρο



Σημαντικά σημεία για την κατασκευή

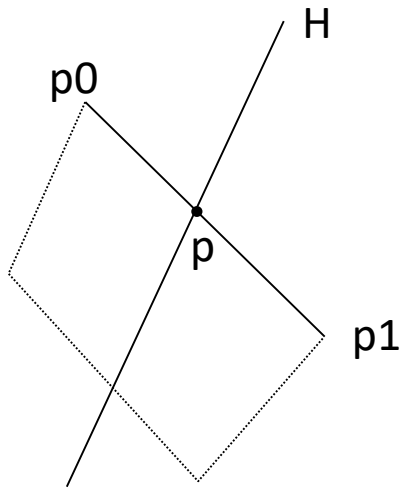
- Το δύσκολο μέρος είναι το μοίρασμα των πολυγώνων (splitting)
  - Γίνεται με σύγκριση όλων των κορυφών σε σχέση με το επίπεδο διαχωρισμού
  - Όμοια με Sutherland-Hodgeman αποκοπή



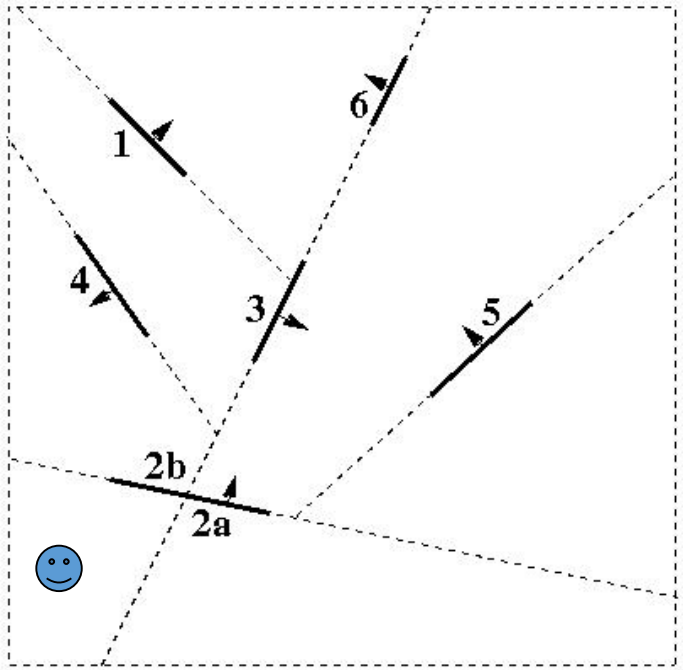
Σημαντικά σημεία για την κατασκευή

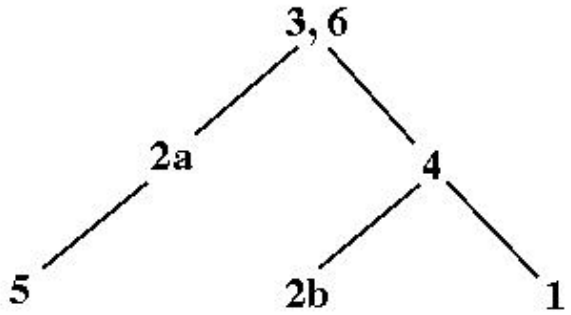
- Το δύσκολο μέρος είναι το μοίρασμα των πολυγώνων (splitting)
  - Γίνεται με σύγκριση όλων των κορυφών σε σχέση με το επίπεδο διαχωρισμού
  - Όμοια με Sutherland-Hodgeman αποκοπή
  - Για να βρούμε τα νέα σημεία όταν μοιράζουμε

$$P = p0 - (t1/(t2-t1))(p1-p0)$$

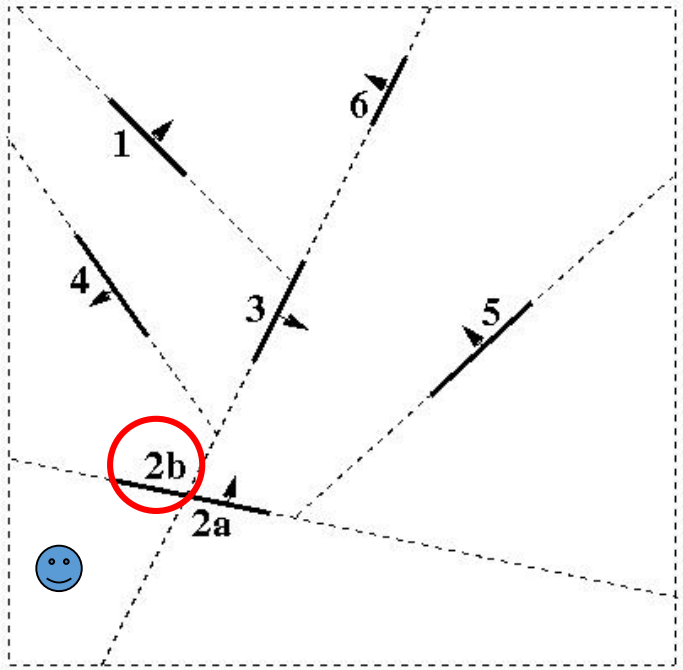


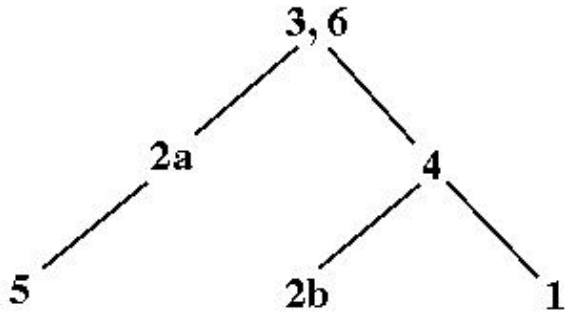
Παράδειγμα διάσχισης





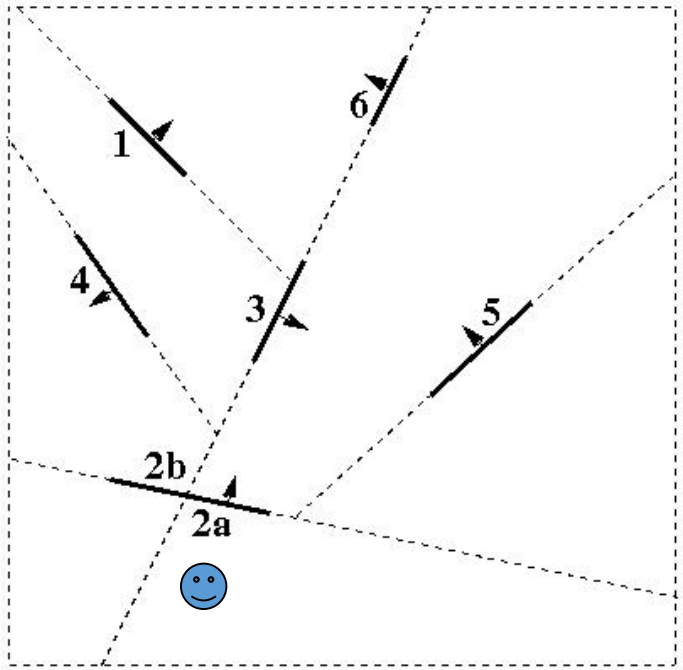
Παράδειγμα διάσχισης

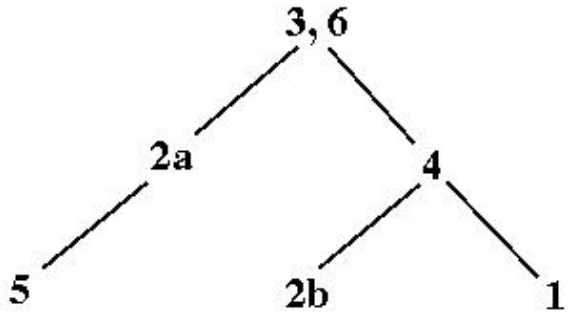




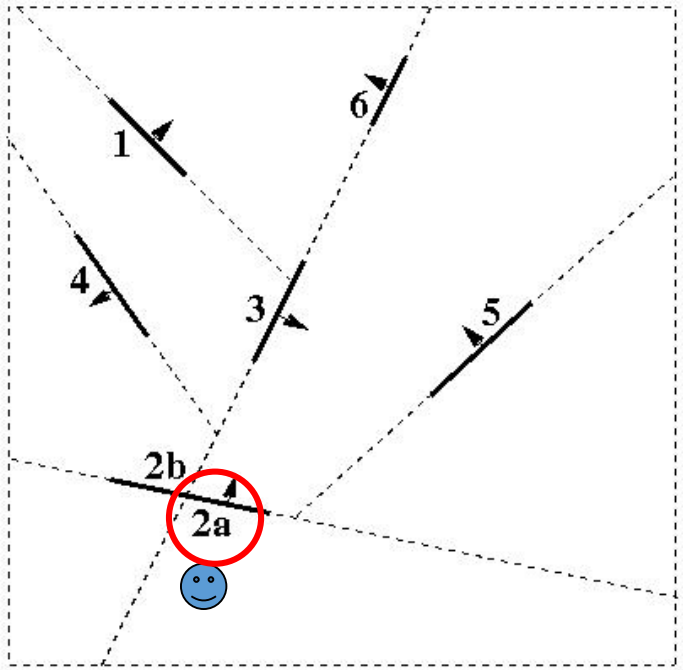
5 – 2a – 3,6 – 1 – 4 – 2b

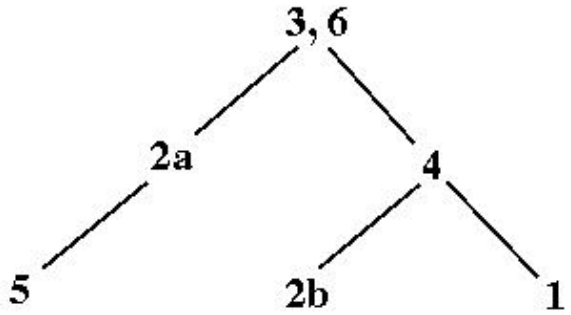
Ακόμα ένα παράδειγμα διάσχισης





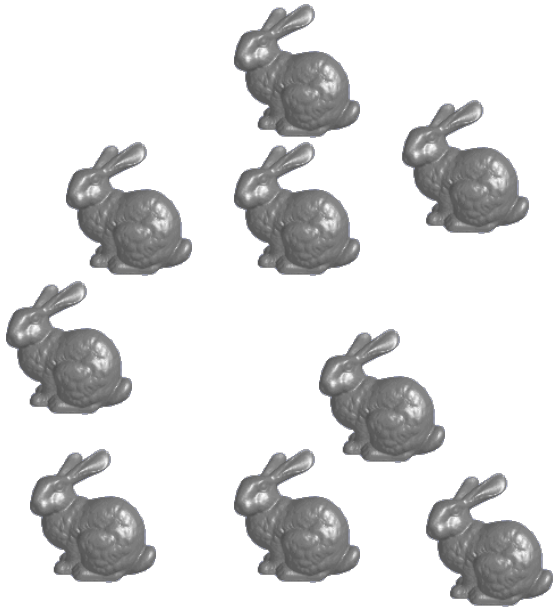
Ακόμα ένα παράδειγμα διάσχισης



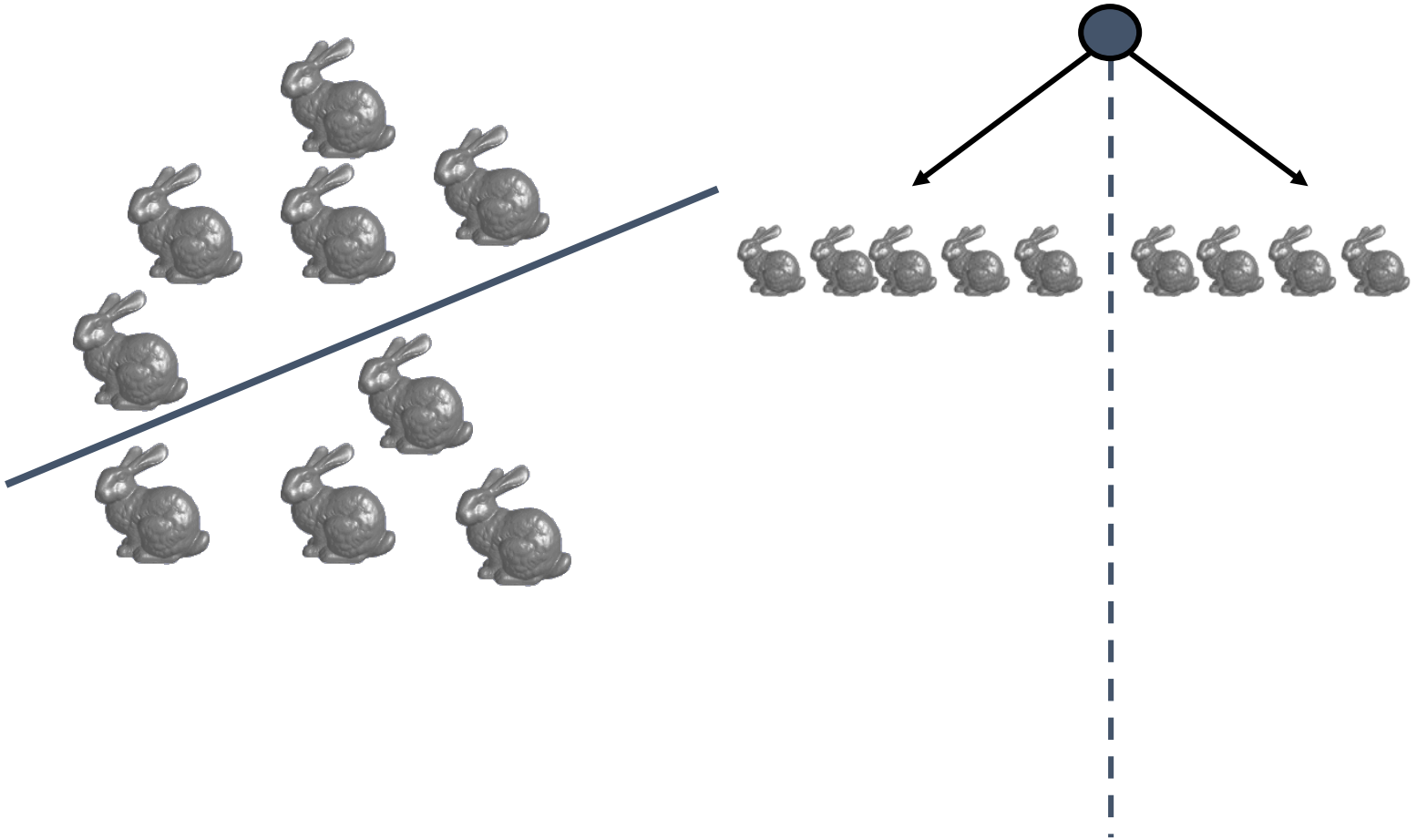


1 – 4 – 2b – 3,6 – 5 – 2a

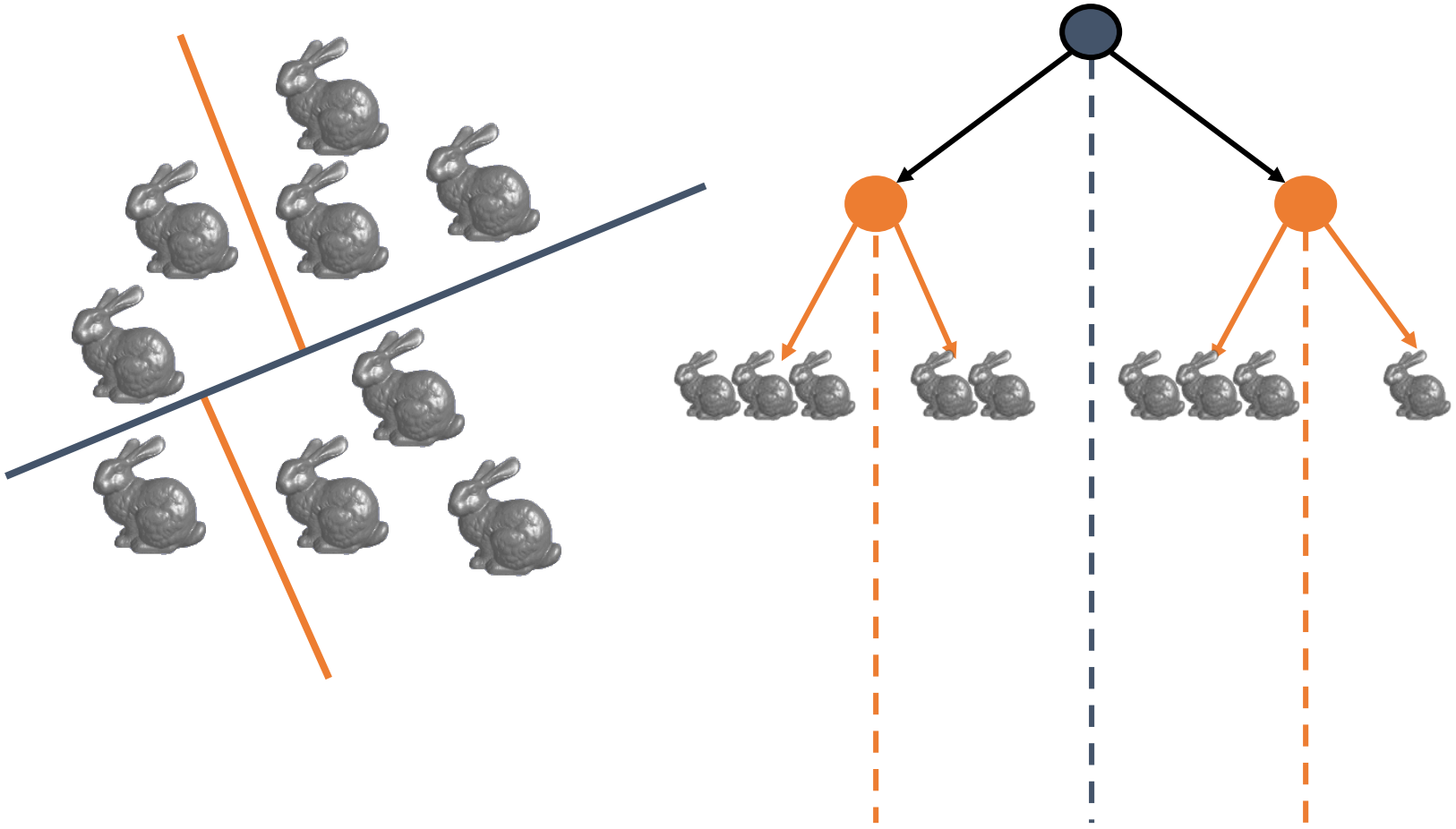
BSP Trees: Αντικείμενα



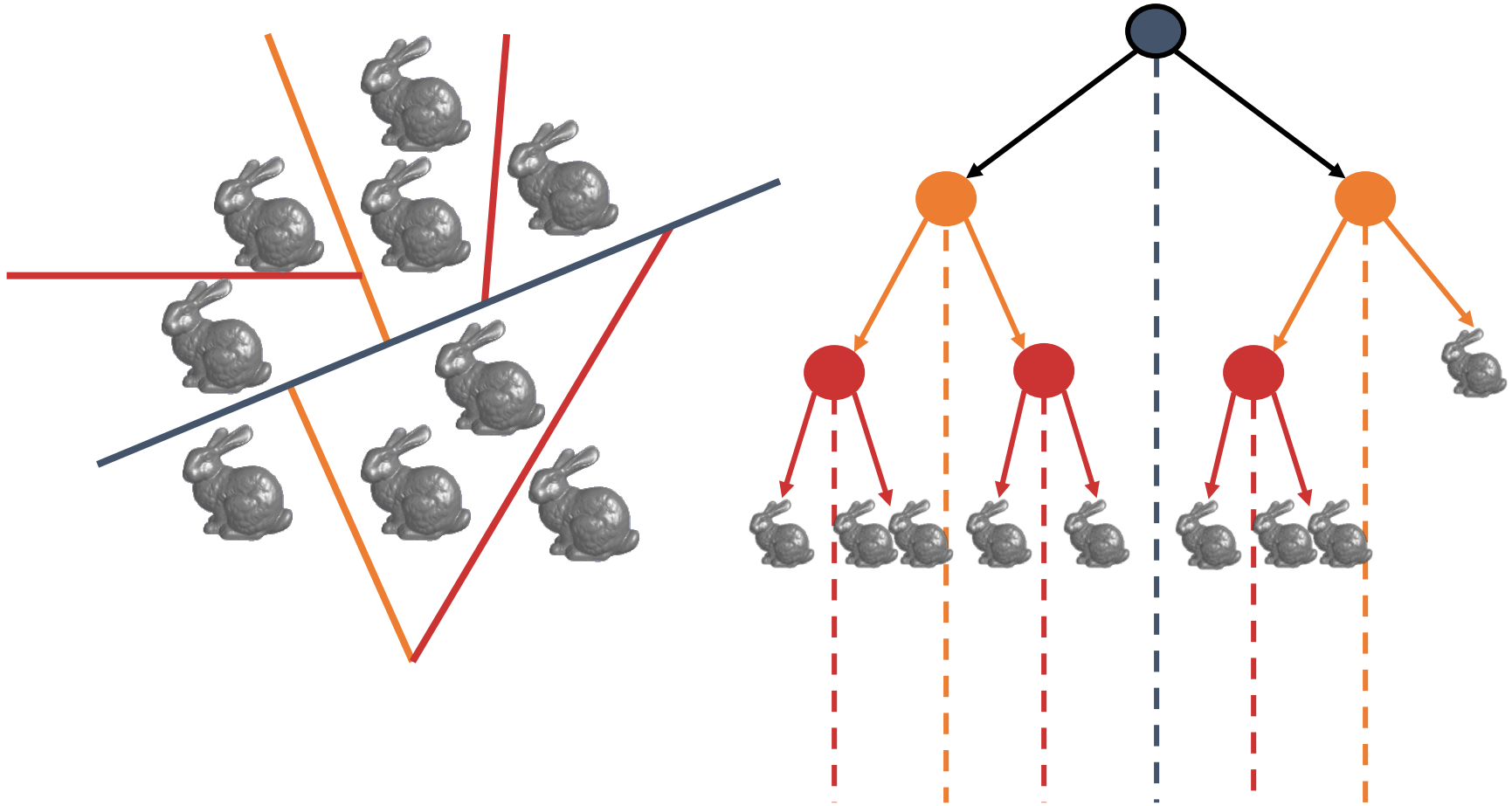
BSP Trees: Αντικείμενα



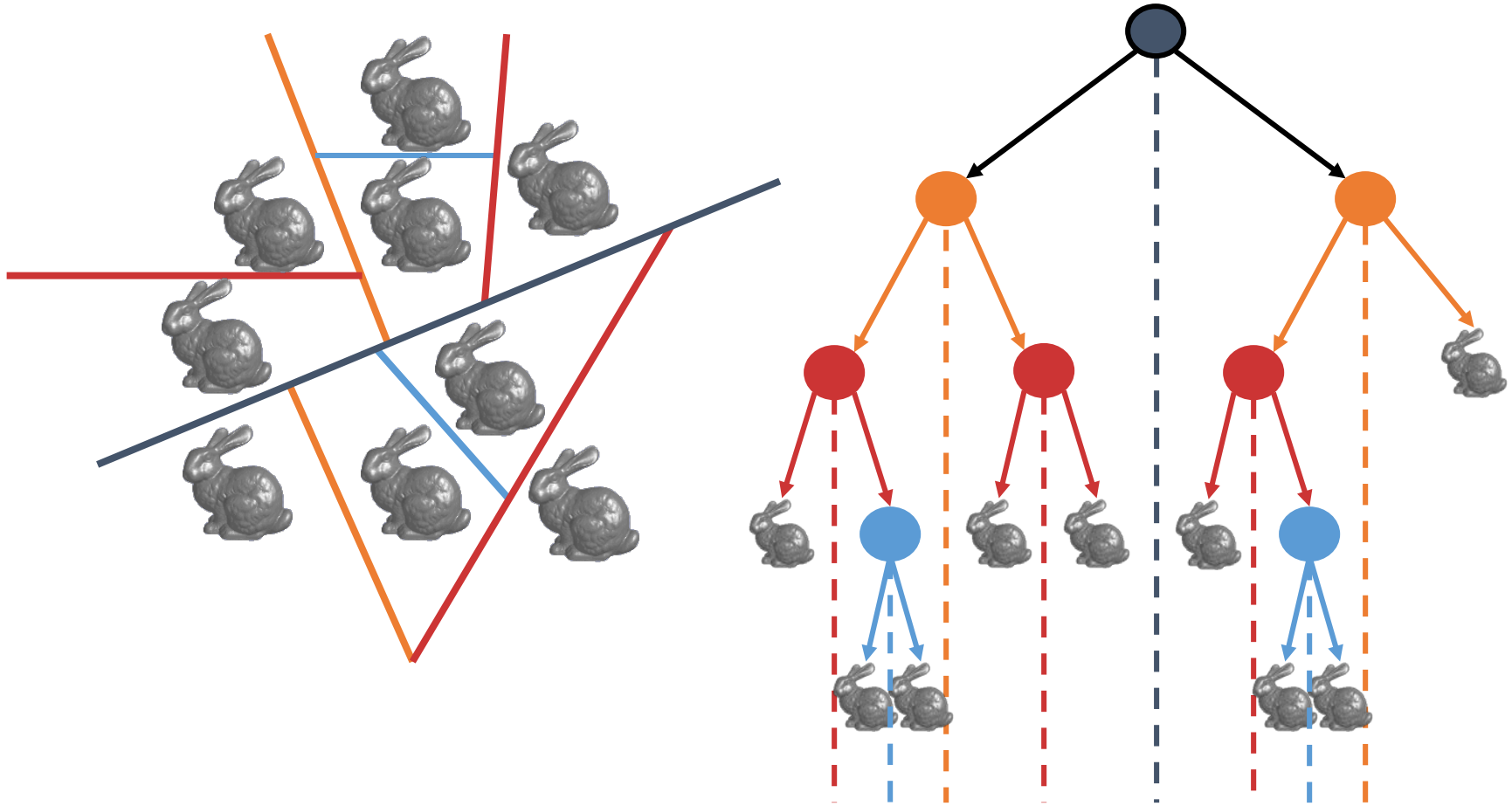
BSP Trees: Αντικείμενα



BSP Trees: Αντικείμενα



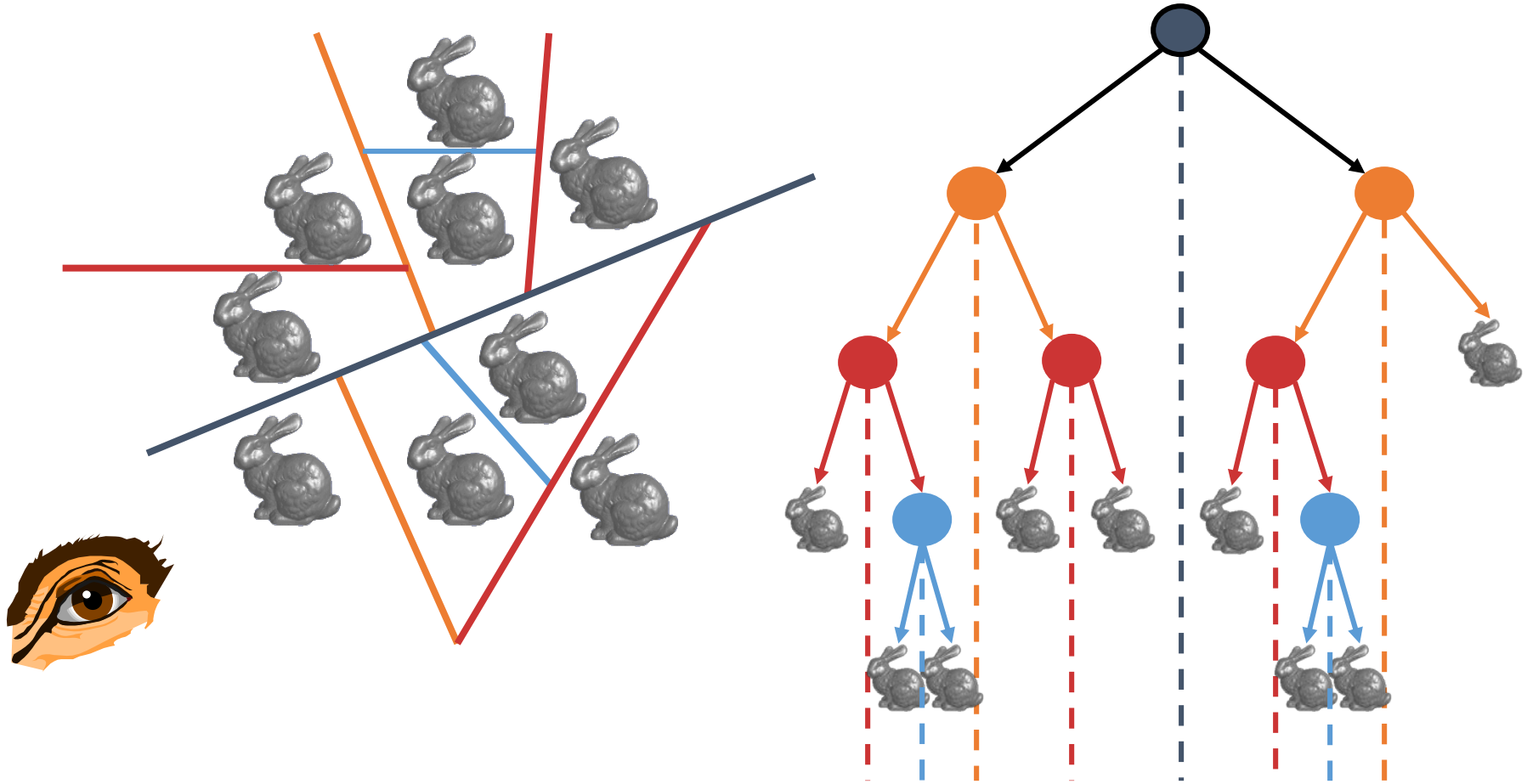
BSP Trees: Αντικείμενα



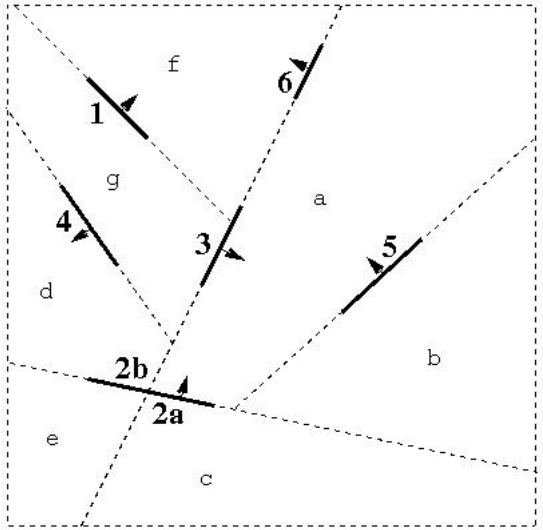
Rendering BSP Trees

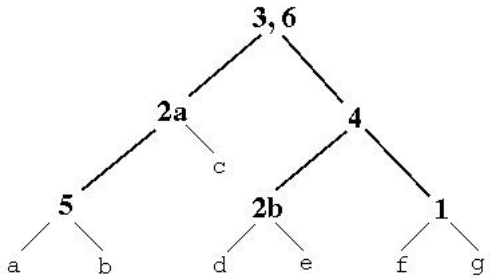
```
renderBSP(BSPtree *T)
  BSPtree *near, *far;
  if (eye on left side of T->plane)
       near = T->left; far = T->right;
  else
       near = T->right; far = T->left;
  renderBSP(far);
  if (T is a leaf node)
       renderObject(T)
   renderBSP(near);
```

Rendering BSP Trees



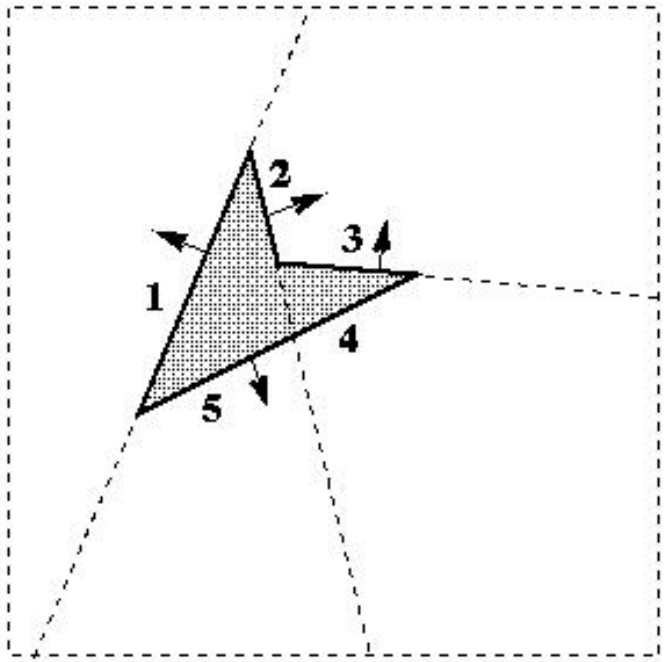
BSP σαν Ιεραρχία των χώρων

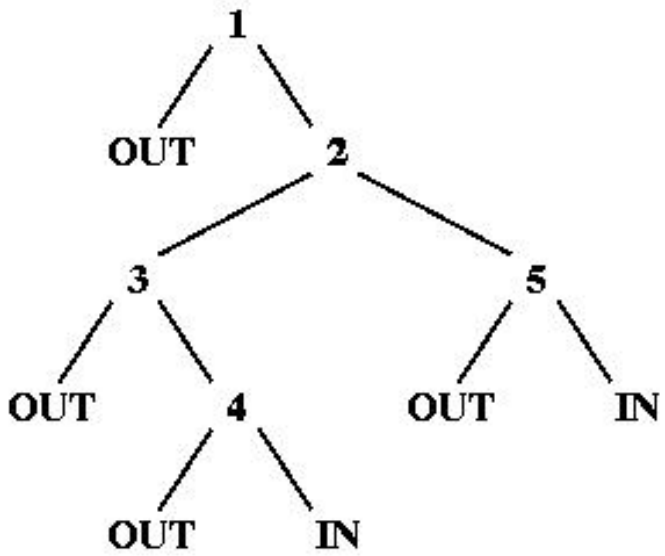




- Κάθε κόμβος αντιστοιχεί σε μια περιοχή του χώρου
  - Η ρίζα είναι ολόκληρο το R^n
  - Το επίπεδο κάθε κόμβου χωρίζει το χώρο του κόμβου σε δύο υπο-χώρους R^+, R^- που αντιστοιχούν στα θετικά και αρνητικά παιδία (might be open sub-spaces)
  - Τα φύλλα είναι ομοιογενείς περιοχές

Αναπαράσταση πολυγώνων σε 2D





Δίνουμε ετικέτες *IN* ή *OUT* στις διάφορες περιοχές

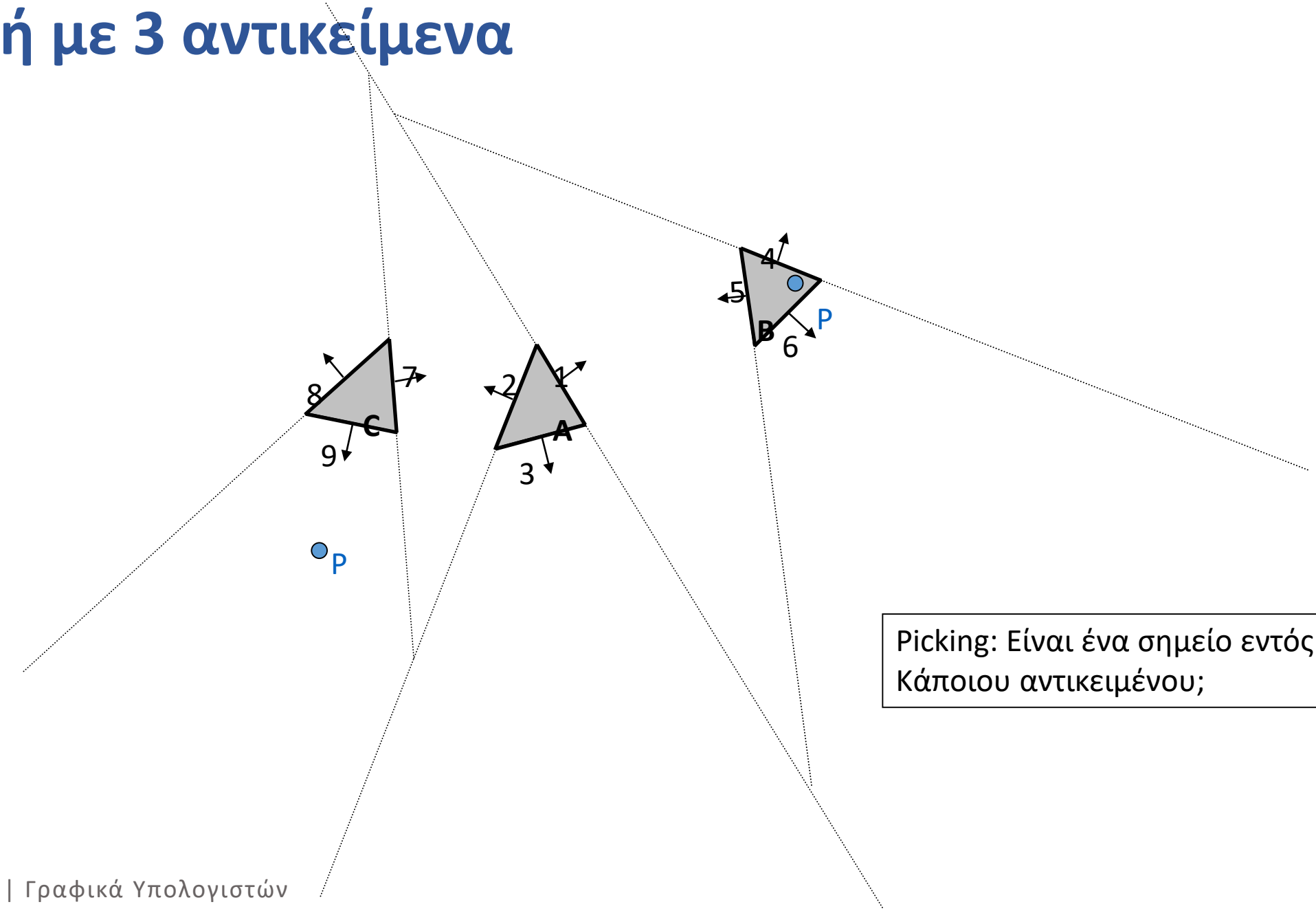
Όχι απαραίτητα μόνο ένα πολύεδρο όπως στο σχήμα,
ολόκληρη η σκηνή μπορεί να μπει σε ένα δέντρο

BSP tree: Εφαρμογές

Μπορούμε να χρησιμοποιήσουμε το BSP tree για να εκτελέσουμε ένα σύνολο από λειτουργίες στη σκηνή

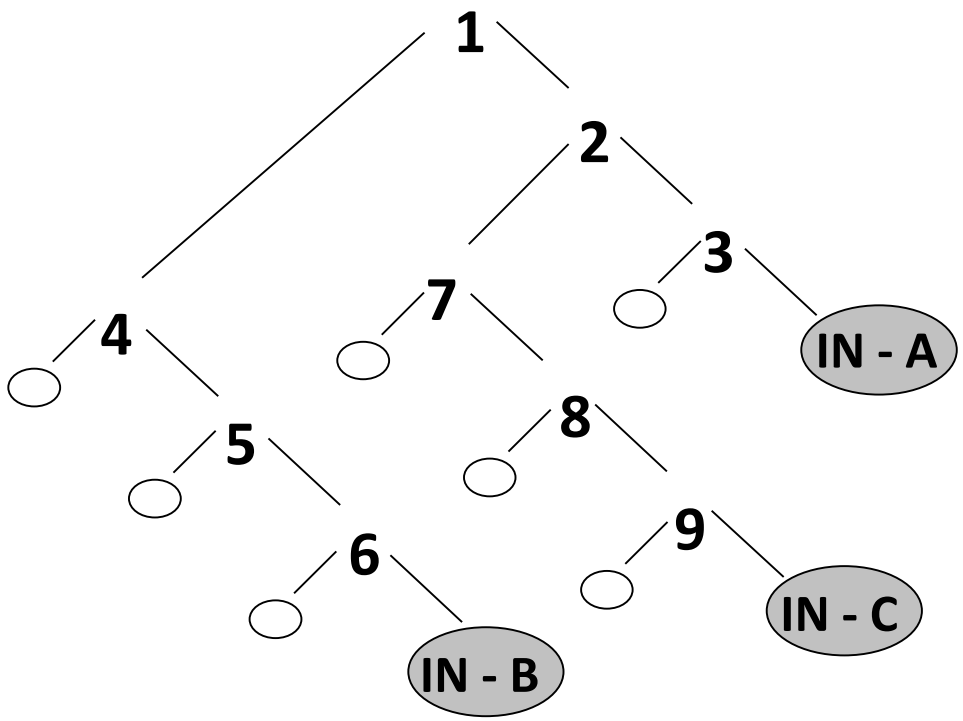
- Τοποθεσία σημείου (Point location)
  - Eg. For picking
- Κατηγοριοποίηση ακτινών (Ray classification)
  - Π.χ. για επιτάχυνση του ray tracing
- Κατηγοριοποίηση πολυγώνων (Polygon classification)
  - Eg. Collision detection
  - Σκίες
- Συγχώνευση (Merging) – beyond the scope of this class
  - View volume or visibility culling, CSG …

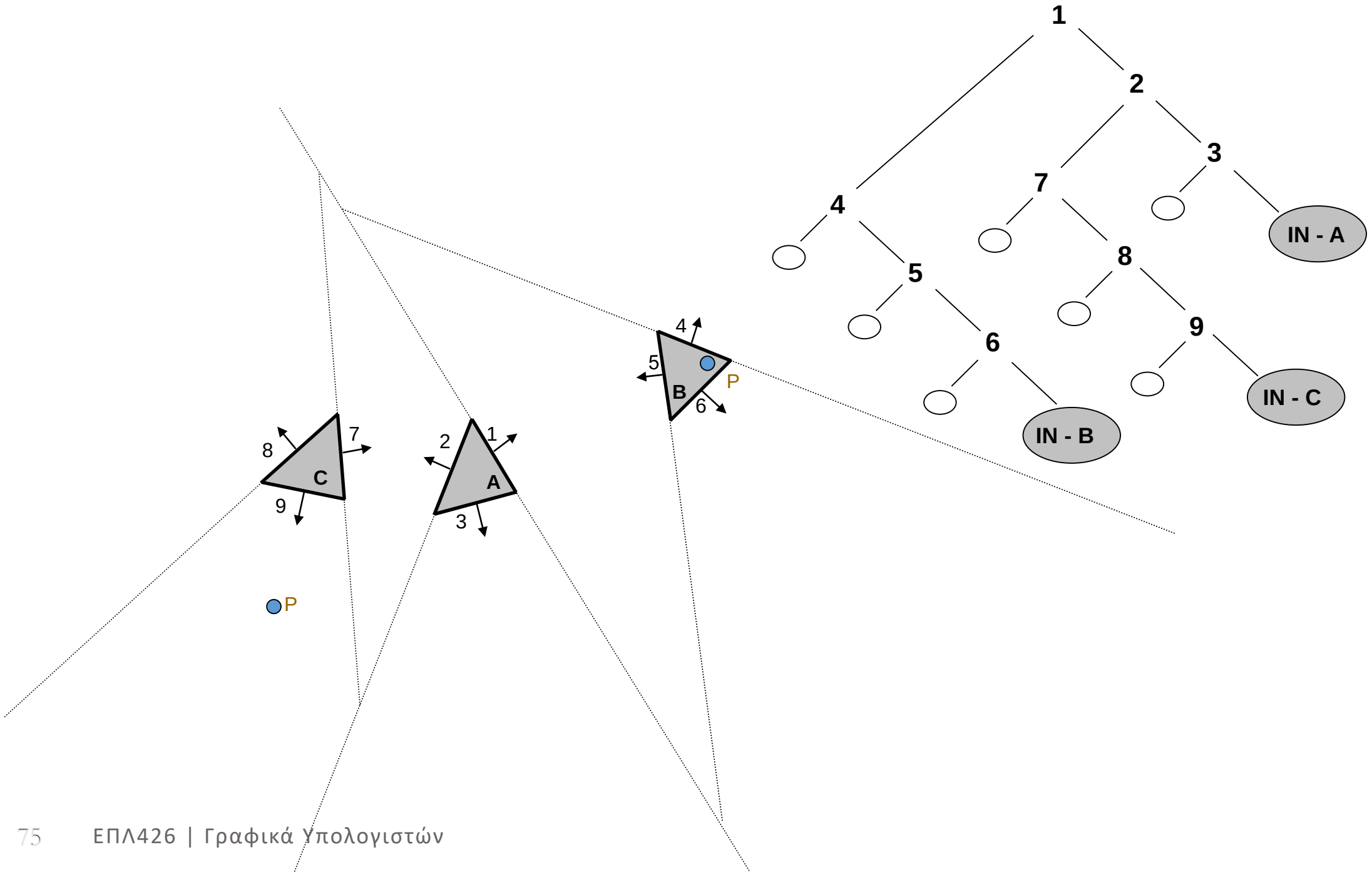
Σκηνή με 3 αντικείμενα



Picking: Είναι ένα σημείο εντός Κάποιου αντικειμένου;

Δέντρο προηγούμενου σχήματος




1
2
3
4
7
5
8
6
9
IN - A
IN - B
IN - C
4
5
B
6
P
8
7
C
9
2
1
A
3
P

Πλεονεκτήματα των BSP Trees

- Advantages:
  - Απλό, κομψό σχέδιο
  - Γράφει μόνο στο framebuffer (π.χ., painters algorithm)
    - Πολύ διαδεδομένα για video games (πλέον όχι και τόσο ☺)

Προβλήματα με BSP trees

- Η κατασκευή του δέντρου χρειάζεται σημαντικό χρόνο
  - Worst-case time to construct tree: $O(n^3)$
  - κατάλληλο κυρίως για στατικές σκηνές, ωστόσο μπορεί να χρησιμοποιηθεί και για περιορισμένες δυναμικές
- Ο διαμελισμός των πολυγώνων αυξάνει τον αριθμό των πολυγώνων στη σκηνή
  - Χρειαζόμαστε τρόπους να κτίζουμε «καλά» δέντρα
  - $O(n^3)$ in worst case
- Όταν έχουμε μεγάλο depth complexity στην σκηνή είναι δαπανηρό
  - Μπρος προς Πίσω (Gordon)

Πως κτίζουμε «καλά» δέντρα

- Ποια είναι τα επιθυμητά χαρακτηριστικά ενός καλού δέντρου;
  - Εξαρτάται από την εφαρμογή
  - Αναζήτηση
    - balanced & shallow
  - Απόδοση
    - όσο το δυνατόν λιγότερους κόμβους και πολύγωνα
- Κατάλληλη επιλογή διαχωριστικού επιπέδου σε κάθε αναδρομική κλήση
  - διαλέγουμε το καλύτερο από 5 τυχαία πολύγωνα

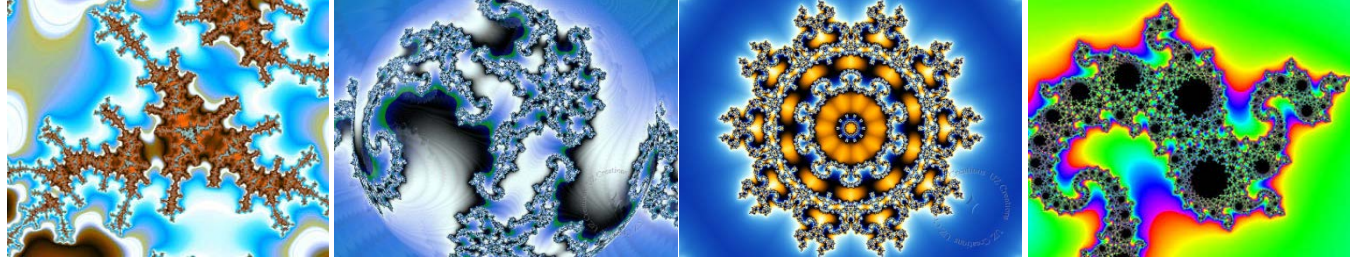
BSP Trees για δυναμικές σκηνές

- Απαιτήσεις για την υποστήριξη δυναμικών σκηνών
  - Change the camera view – OK
  - Add an object – OK , incremental building
  - Delete an object ?
- Δυνατό, αλλά δύσκολο
- Κάποια συστήματα χρησιμοποιούν BSP tree και z-buffer (depth buffer)
  - Αποθήκευση μόνο στατικής γεωμετρίας σε BSP trees
  - z-buffer για rendering των δυναμικών αντικειμένων (π.χ. πόρτες, κλπ.)

Combining Acceleration Data Structures

- Often it can be helpful to mix different acceleration data structures
  - e.g. using a grid on the top level, then have BVHs for fine-grained objects in each grid cell
  - This is fairly simple to integrate multiple acceleration data structures if you architect your code well, as you can treat your acceleration structure just like a scene object

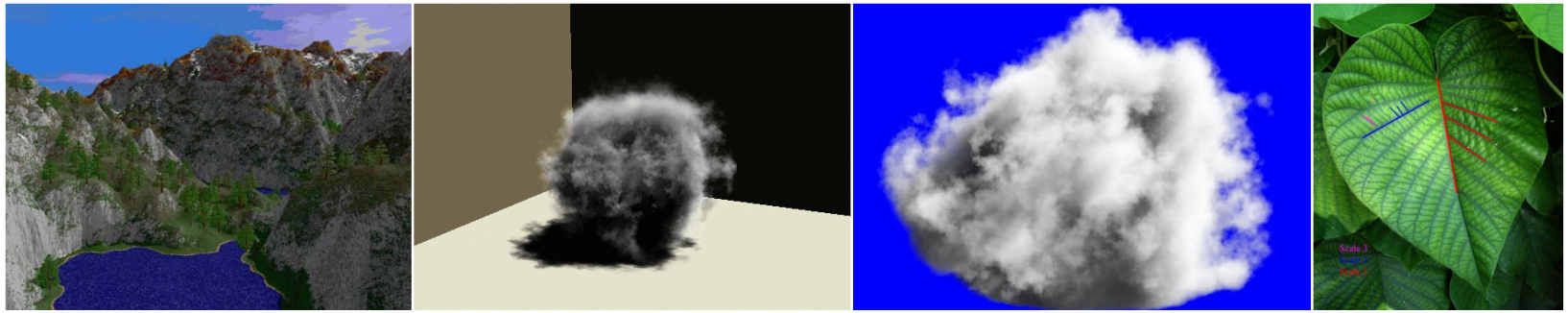
Fractal Geometry



- Όλες οι τεχνικές μοντελοποίησης που είδαμε μέχρι στιγμής χρησιμοποιούν Ευκλείδεια γεωμετρία
  - Τα αντικείμενα περιγράφηκαν με τη χρήση εξισώσεων
- Αυτό είν
- Αλλά τι
  κατακερ
  - Βουνά

"Clouds are not spheres, mountains are not cones, coastlines are not circles and bark is not smooth, nor does lightning travel in a straight line."

Benoit Mandelbrot

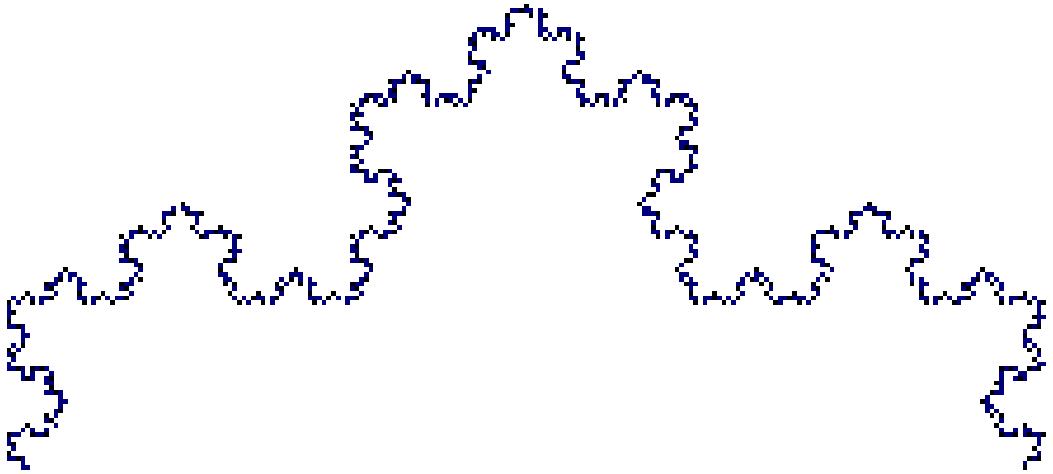


Fractal Geometry Methods & Procedural Modelling

- Τα φυσικά αντικείμενα μπορούν να περιγραφούν ρεαλιστικά χρησιμοποιώντας **fractal geometry methods**
- Οι Fractal μέθοδοι χρησιμοποιούν διαδικασίες και όχι εξισώσεις για την μοντελοποίηση αντικειμένων -**procedural modelling**
- Το κύριο χαρακτηριστικό κάθε διαδικαστικού μοντέλου είναι ότι το μοντέλο δεν βασίζεται σε δεδομένα, αλλά στην εφαρμογή μιας διαδικασίας μετά από ένα συγκεκριμένο σύνολο κανόνων

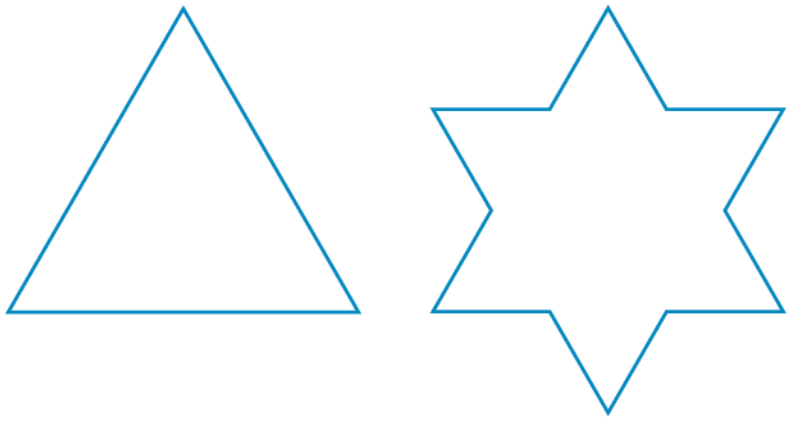
Fractals

- Ένα fractal αντικείμενο έχει δύο βασικά χαρακτηριστικά:
  - Άπειρη λεπτομέρεια σε κάθε σημείο
  - Ομοιότητα μεταξύ των μερών του αντικειμένου και των γενικών χαρακτηριστικών γνωρισμάτων του αντικειμένου

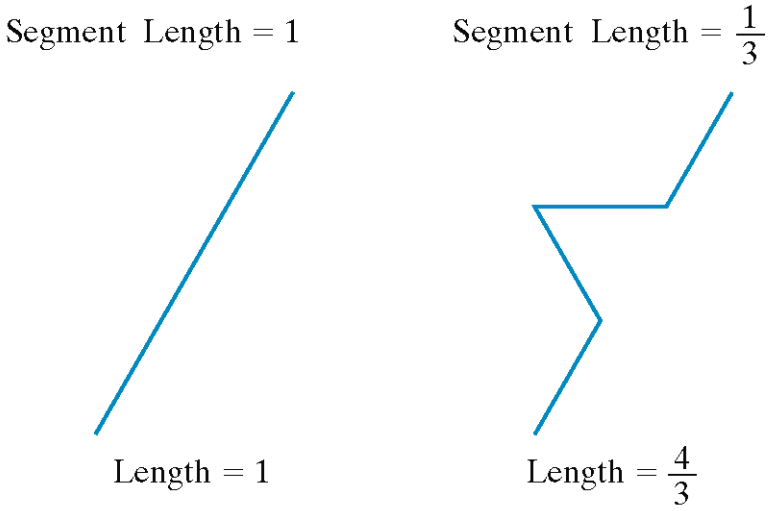


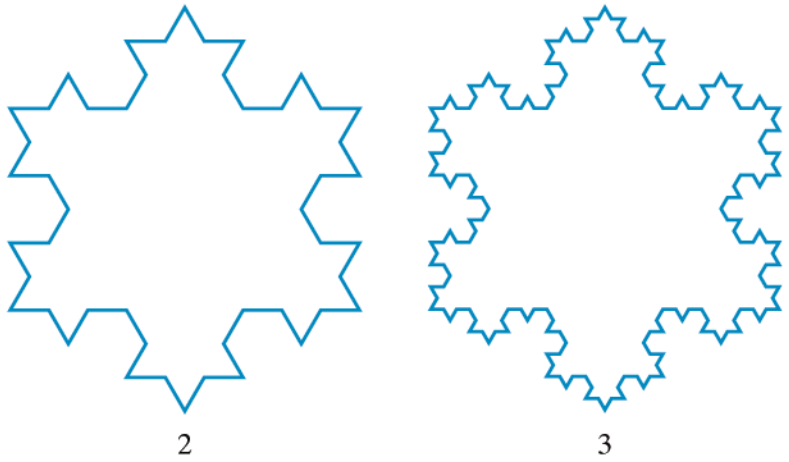
The Koch Curve

Παράδειγμα: The Koch Snowflake

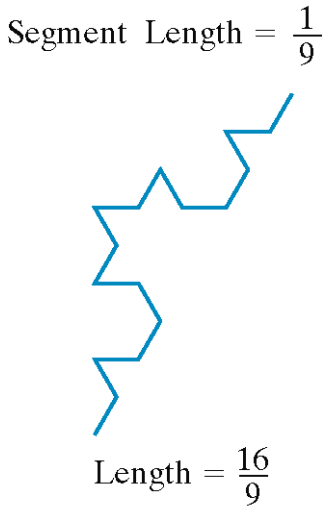


0 1



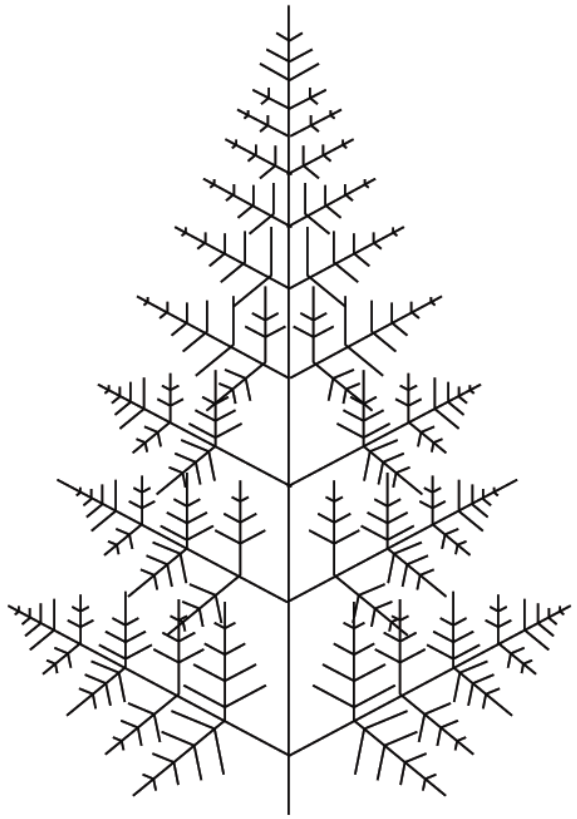


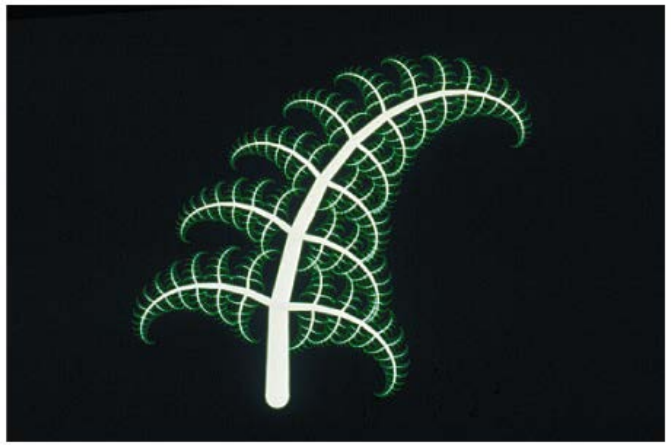
2 3



Παράδειγμα: Φτέρες

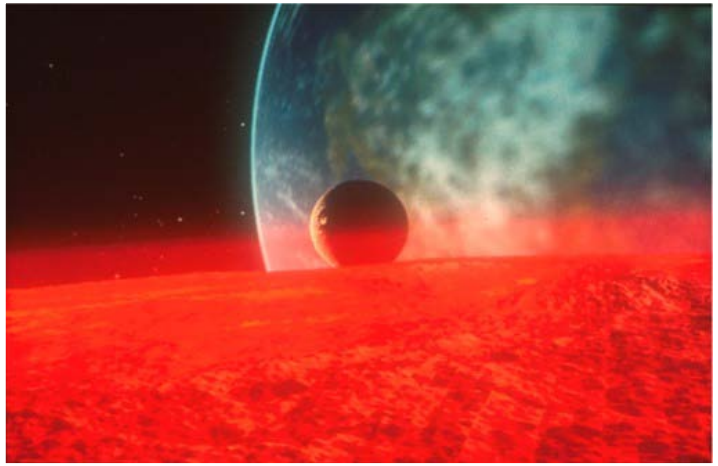
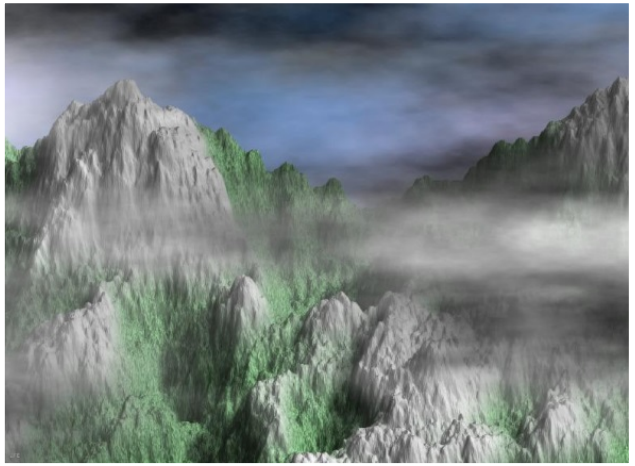
- Πολύ παρόμοιες τεχνικές μπορούν να χρησιμοποιηθούν για τη δημιουργία βλάστησης





Παράδειγμα

- Μια από τις πιο επιτυχημένες χρήσεις της τεχνικής fractals σε γραφικά είναι η δημιουργία τοπίων

Fractals σε ταινίες με Special Effects