Animation

CSE606: Computer Graphics
T K Srikanth, IIIT Bangalore
April 2025

Animation and Performance (2)

Agenda

- Key approaches - overview
- Space-based partitioning
  - Quadtrees and Octrees
  - BSP Trees, KD Trees
- Object-based partitioning
  - Bounding Volume Hierarchies
- Managing very large spaces and objects

Managing data for performance

- Spatial Data Structures
  - Data structures/databases that improve performance on spatial computations
- Level of Detail
  - Performing computations (including rendering) on appropriate levels of resolution of the geometry of the objects
- Visual Culling
  - Discard objects that cannot be seen or do not impact the scene - early on in the rendering process

Rendering Large Complex Scenes

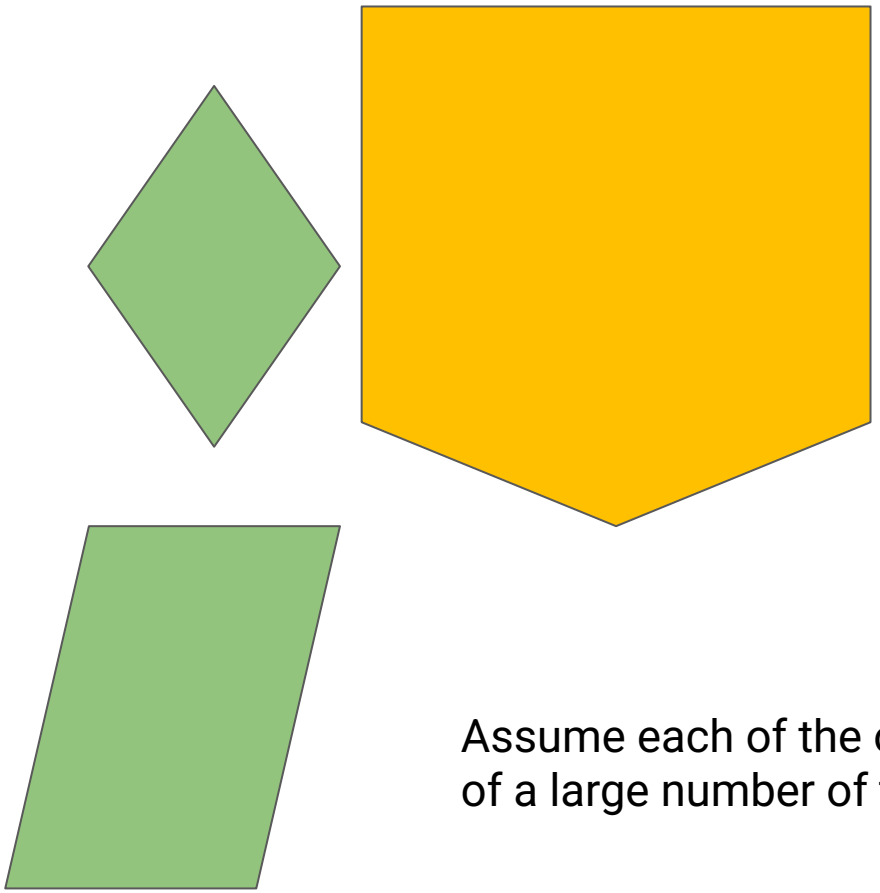
Need efficient schemes for handling large scenes for operations such as:

- Interactive rendering
  - Front-to-back or Back-to-front rendering
- Ray tracing
- Collision detection
- Visibility set computation

Want these operations to be ***O****(n)* or better (*n* = number of polygons in the scene)

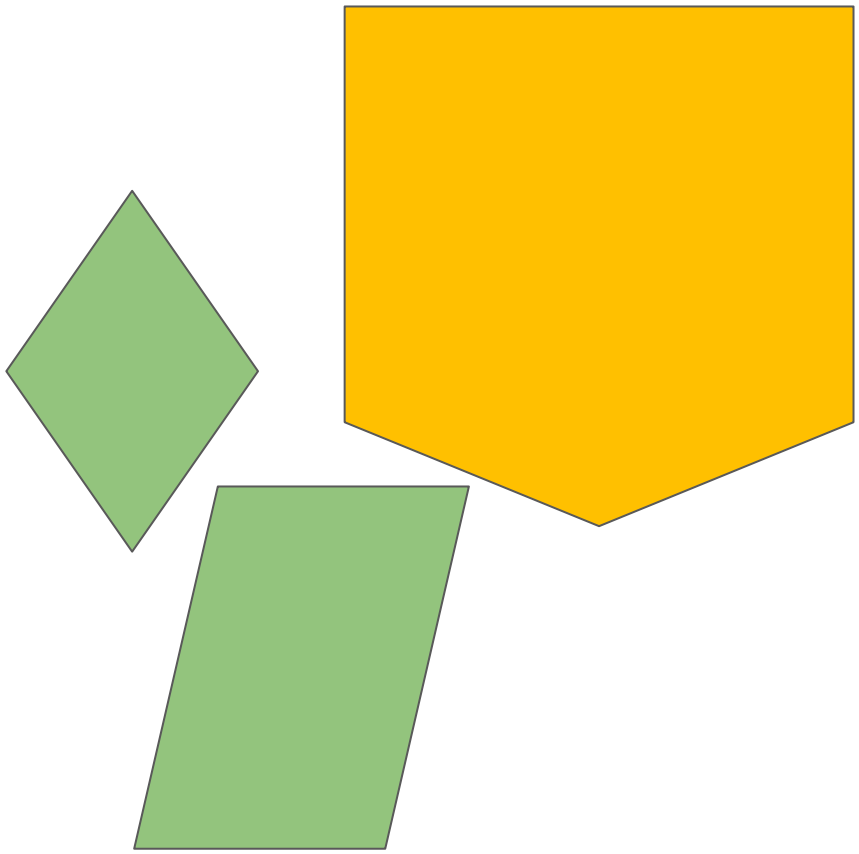
Explore ways of arranging the data in a scene into a hierarchy that will allow efficient determination of whether an object is “inside” or “outside” a region of interest, or meets some other similar spatial criteria.

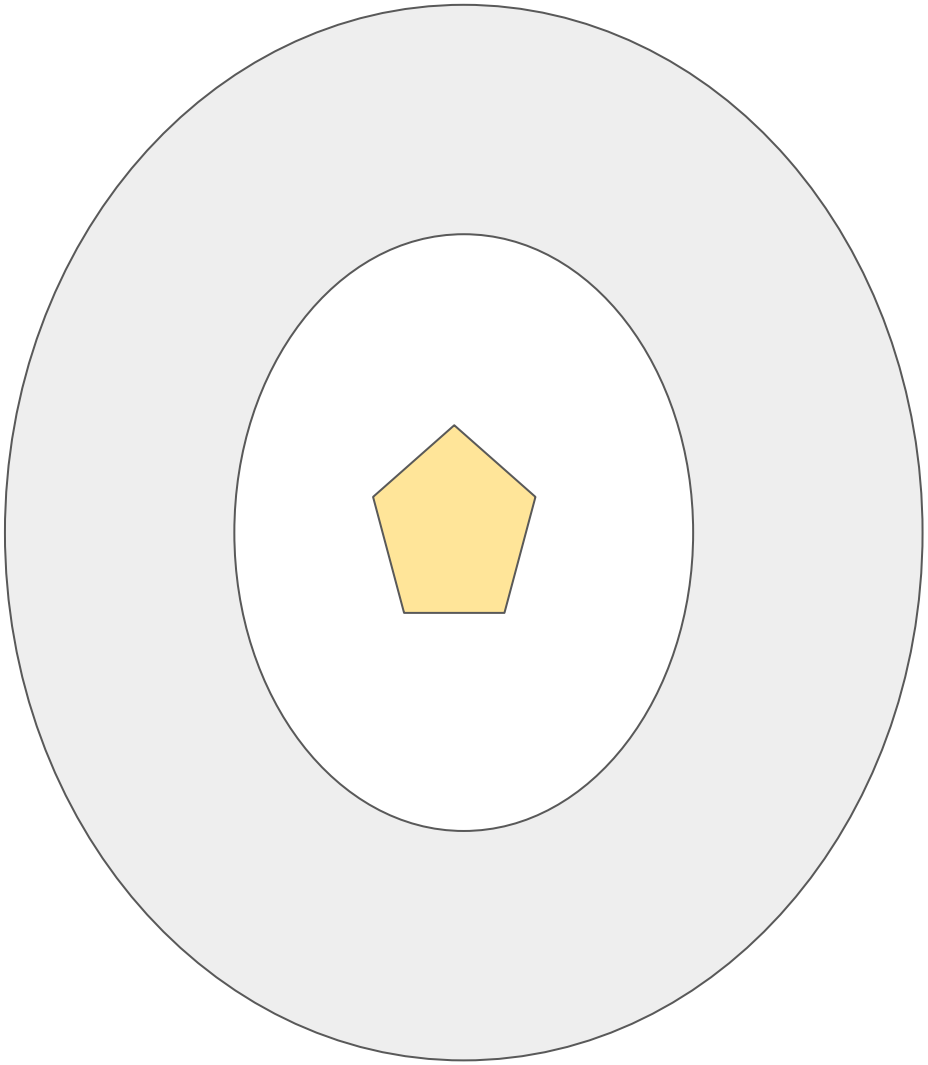
Collision Detection

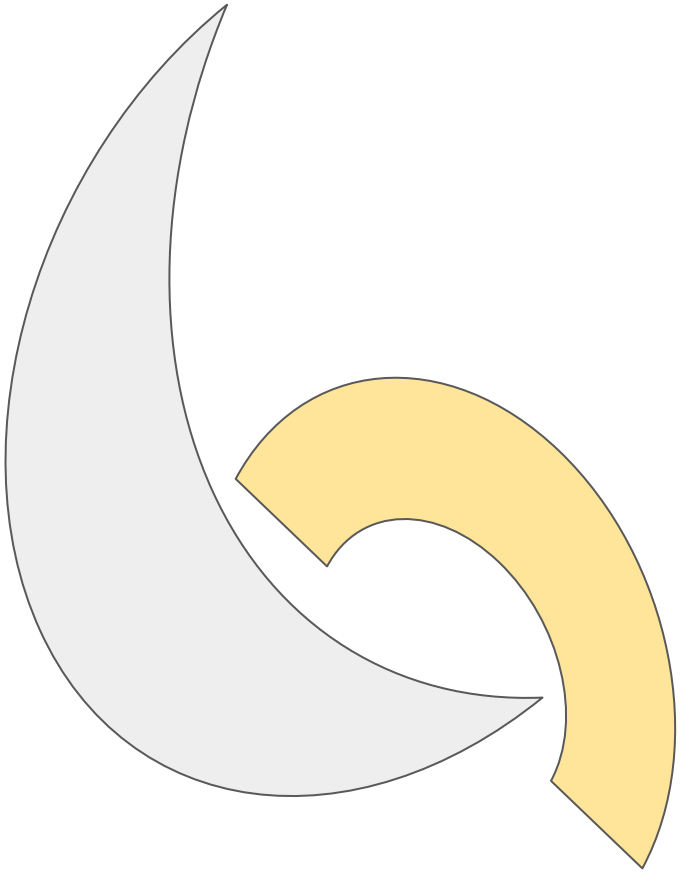


Assume each of the objects here consists of a large number of triangles

Collision Detection



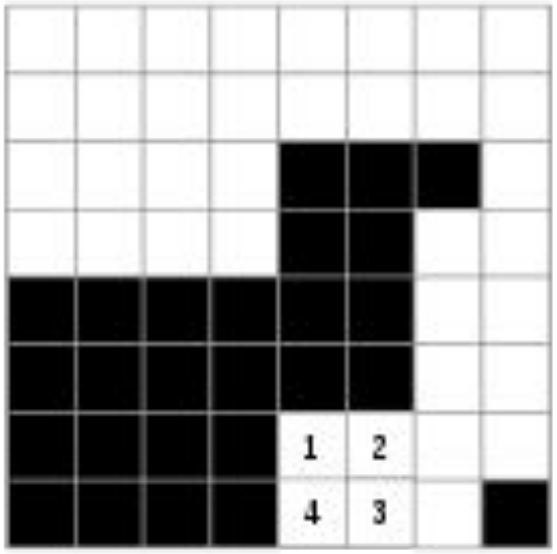


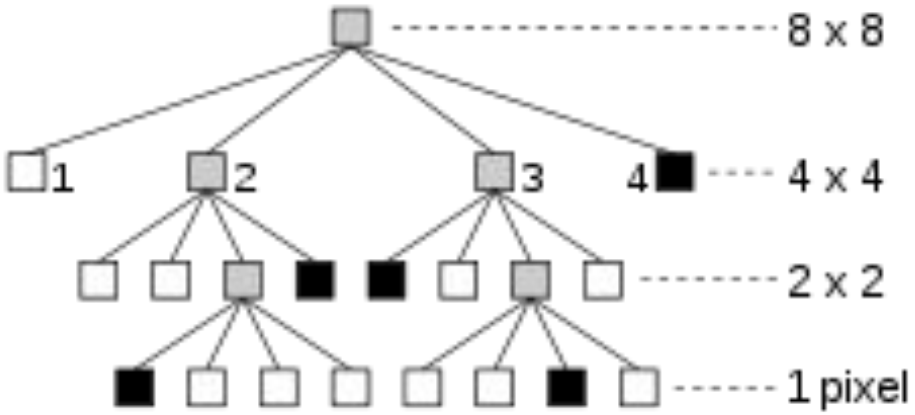


Grid-based Hierarchies: Quad-trees

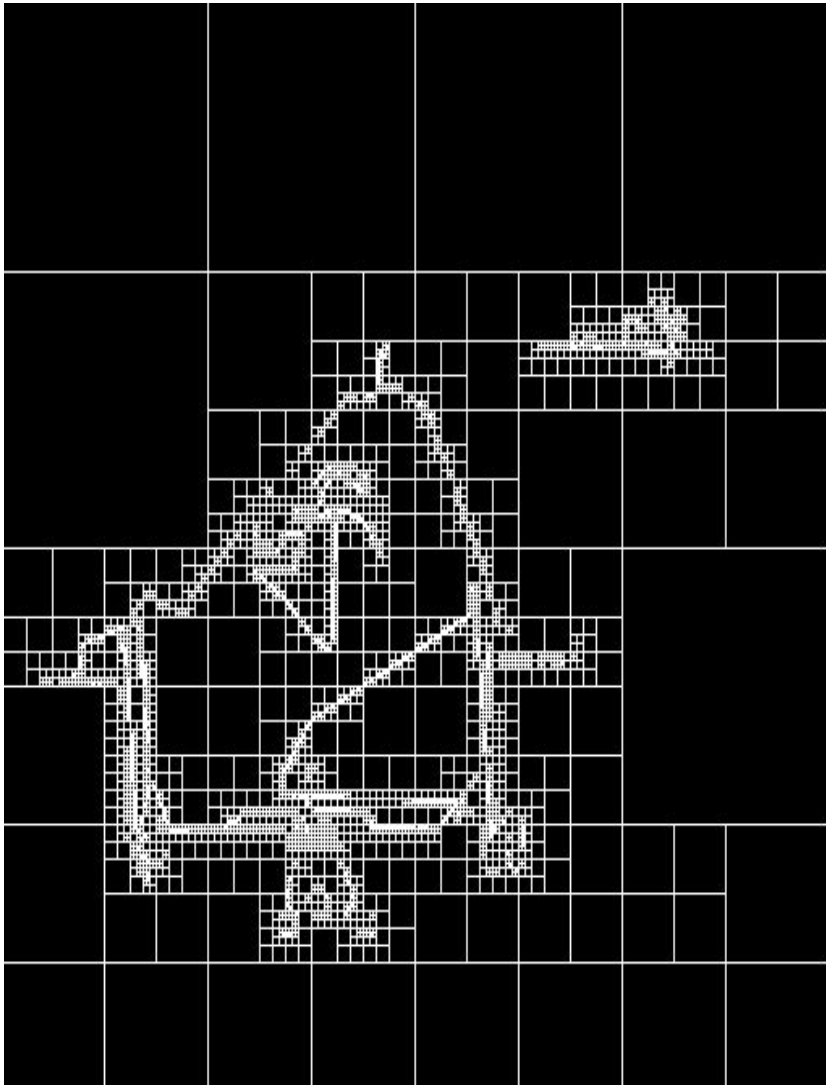
Quadtrees:

- Each rectangular region is divided uniformly into 4 sub-regions, corresponding to a 2x2 split of the region
- Thus, when arranged into a tree, each node has 4 children
- Can stop subdivision if leaves of the sub-tree have the same value
- Useful for efficiently checking intersection/containment/overlap with existing scene objects in 2D (e.g if all objects are at ground level)





Source: Wikipedia



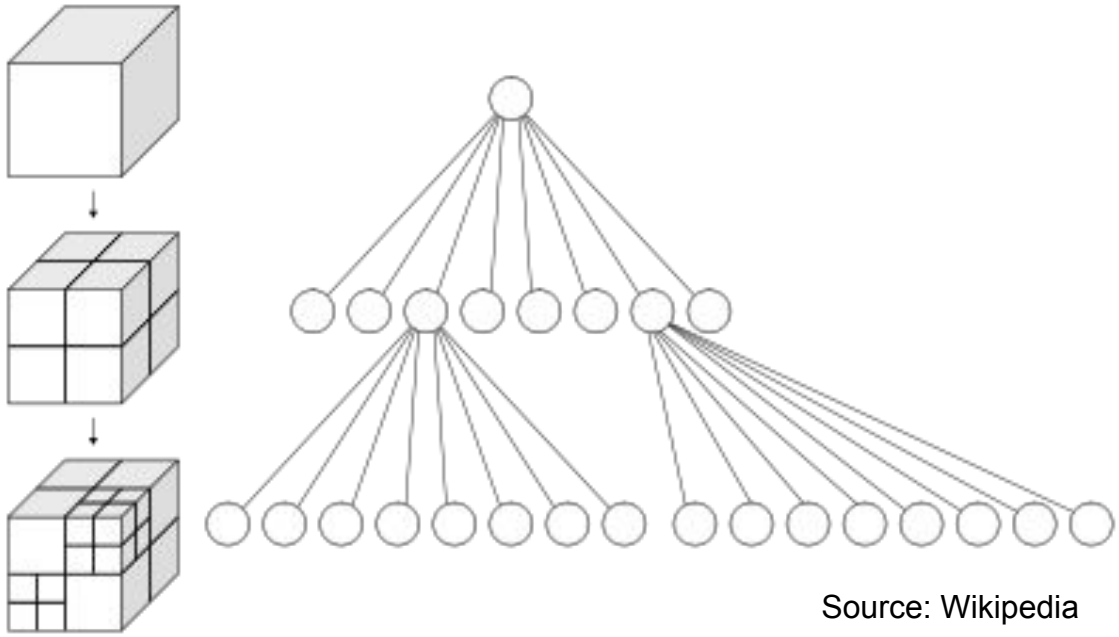
Source: Mathworks

Octrees

Octrees:

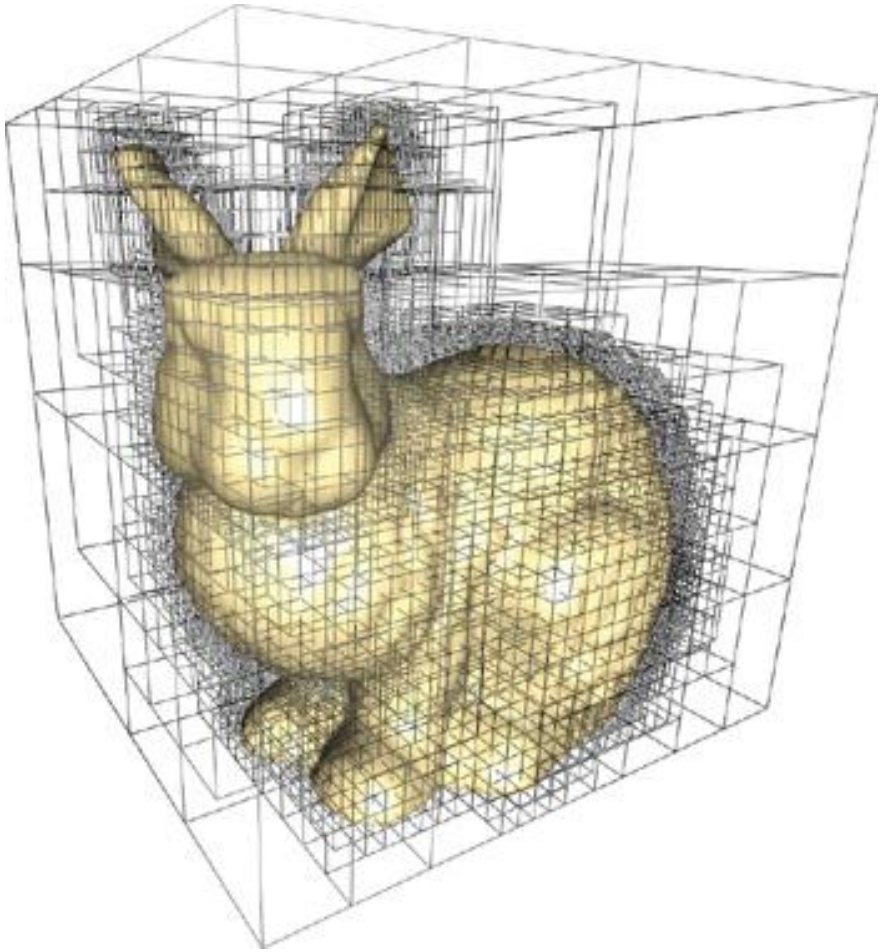
Extension of quadtrees to 3D.

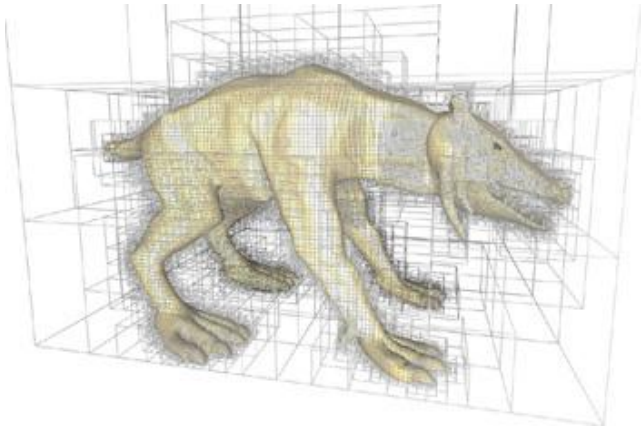
Each volumetric region (box) is divided into 8 sub-regions, 2 along each coordinate direction



Source: Wikipedia

Octrees





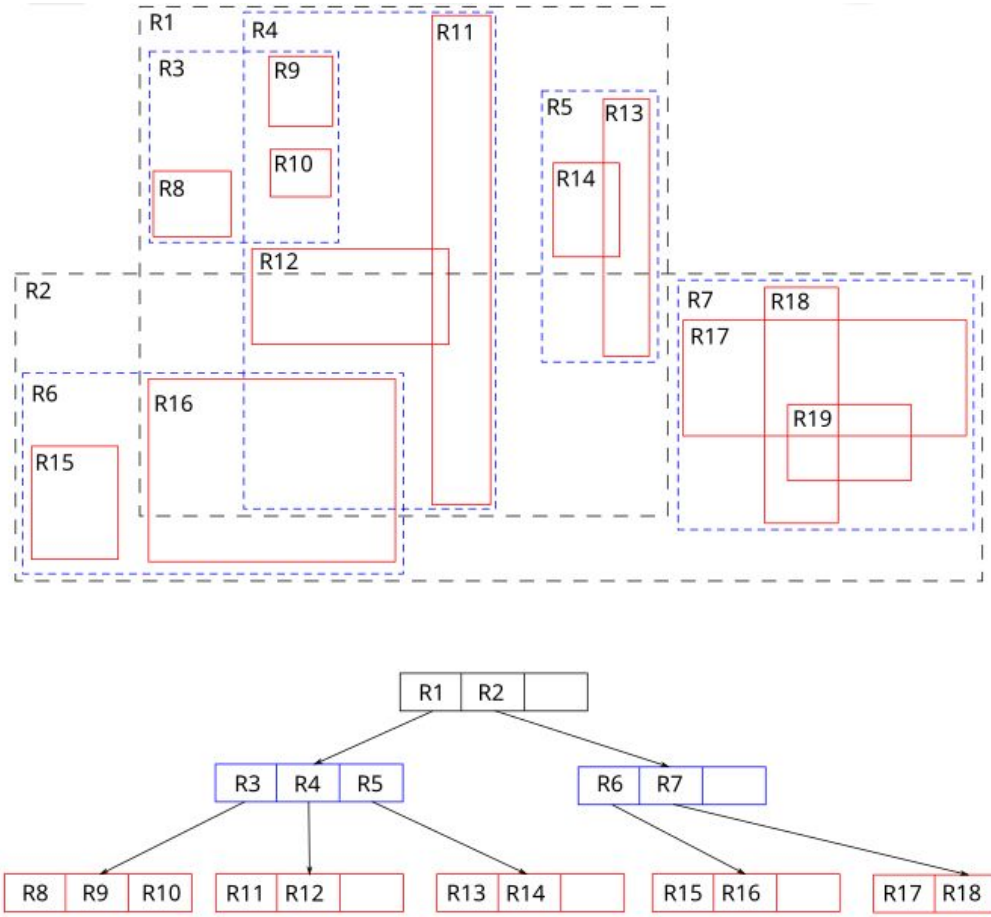
Source: Nvidia

R-Trees

Hierarchical grouping of Minimum Bounding Rectangles (MBR)

Not restricted to grid subdivision. SIbling sub-trees may overlap

Used extensively in RDBMS extensions that support spatial data types and queries



Visibility Queries

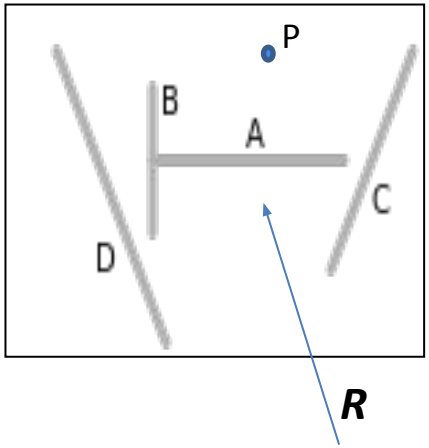
- Given a view direction (ray) ***R***, which line segments does it intersect?

- Given a point P, sort (and render) render line segments “back to front” with respect to the point.

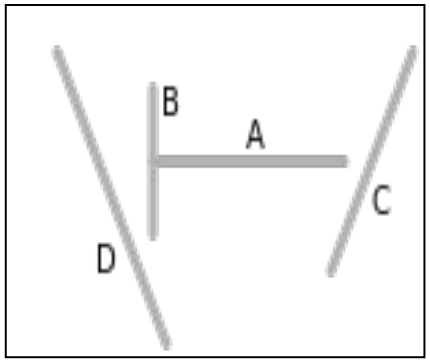
Query operations should be O(n) – linear in the number of line segments (/polygons).

Can we re-use the generated data structure for queries on the same scene?

- Under what conditions is this possible?



Determining Visibility



Based on: Fuchs, Henry; Kedem, Zvi. M; Naylor, Bruce F, "On Visible Surface Generation by A Priori Tree Structures“, SIGGRAPH 80

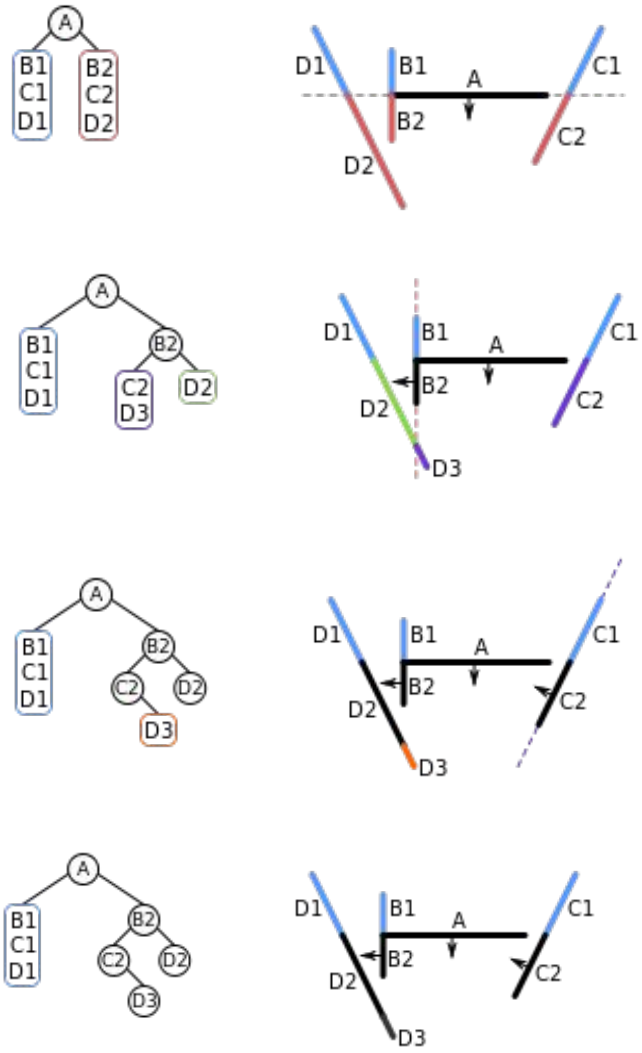
Key idea:
Each line segment s_i (when extended) partitions the region into 2 halfspaces, a “positive” H_{i+} and a “negative” halfspace H_{i-} . Assume segments are split by s_i into smaller segments if they intersect

If the view point P is in the positive halfspace H_{i+} , then no segment in H_{i-} can obscure s_i or any segment in H_{i+} . Similarly, if P is in H_{i-}

Binary Space Partitions

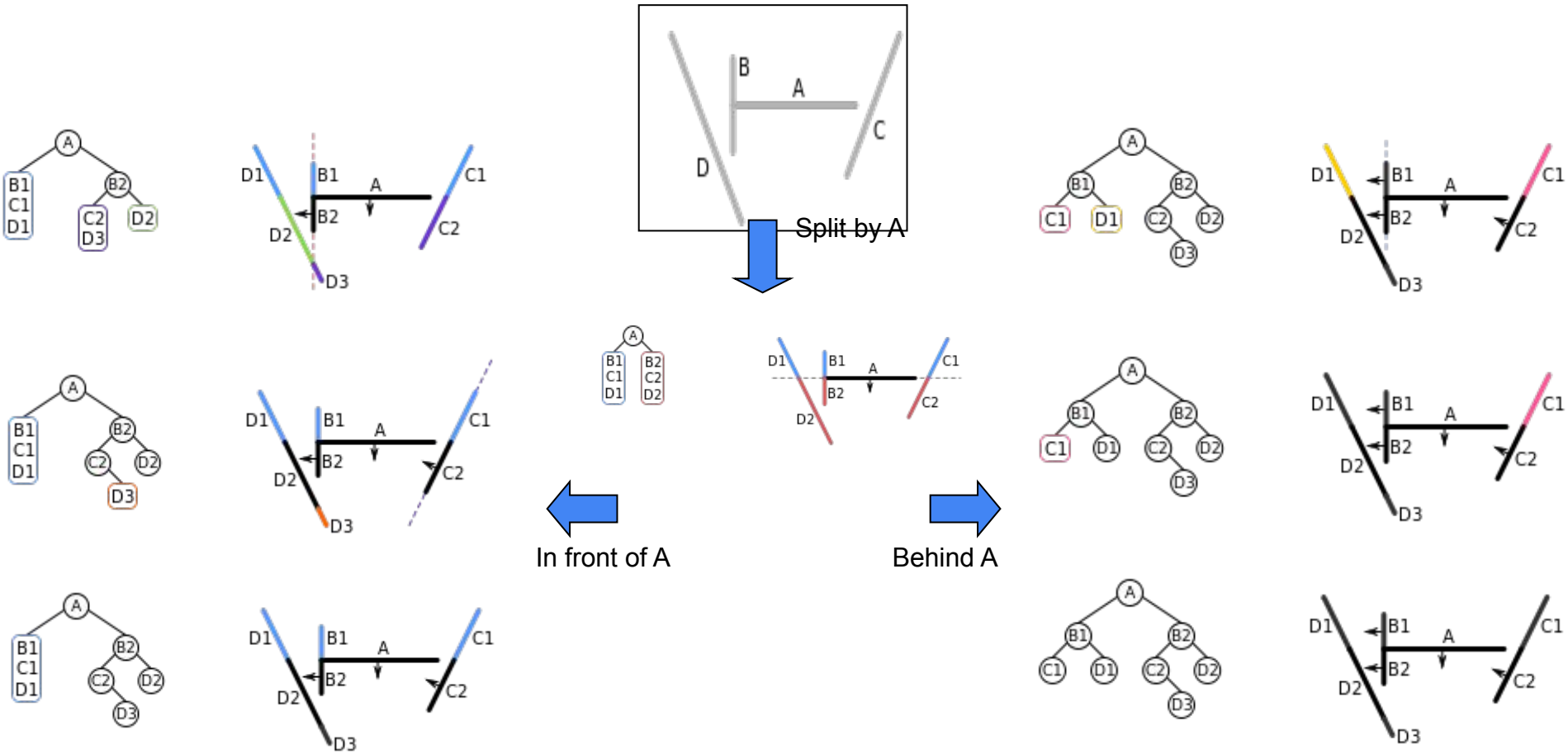
Building a Binary Space Partition tree (for line segments in 2D)

- Choose any of the line objects, and its plane as the partitioning plane. Set this object as the root of the current sub-tree
- Split all lines that intersect this plane.
- Group the elements that are “in front of” and “behind” the plane into two lists
- Recursively process the two lists and insert the output of these processes as the right and left children of the current node respectively.



Source: Wikipedia

Building a BSP Tree



Source: Wikipedia

Rendering Using a BSP Tree

Back-to-front rendering (e.g. Painter's algorithm) relative to a view point **P**

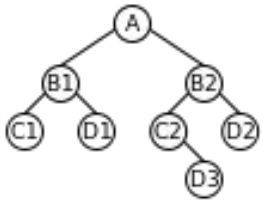
In-order traversal of the tree:

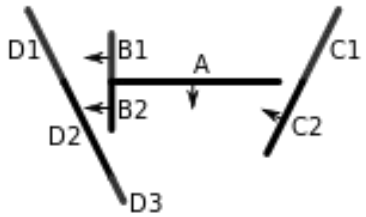
If **P** is in "front" of plane corresponding to the sub-tree root,

- Render objects behind plane (i.e. left subtree)
- Render objects on the plane (i.e. current node)
- Render objects in front of plane (i.e. left subtree)

Else

- Render objects in front of plane (i.e. right subtree)
- Render objects on the plane (i.e. current node)
- Render objects behind plane (i.e. left subtree)

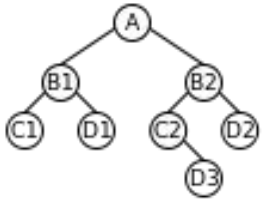


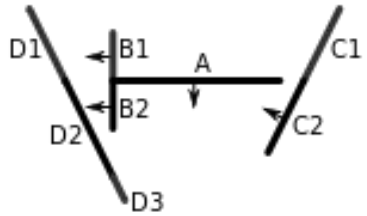


Rendering Using BSP Tree

For **Front-to-back rendering** (e.g. for Z buffer), reverse order of traversal of subtrees

- Thus, we can use the same tree, but change the traversal mechanism based on needs of the algorithm





Complexity:

- Depth of tree is O(log (n)) – if partitioning segments are chosen carefully
- Could still result in $O(n^2)$ leaves, which means construction and rendering could be $O(n^2)$
- In practice, can be reduced by use of specialized techniques based on nature of input objects

- Updates could also be expensive – when processing dynamic scenes

BSP Trees

Complexity:

- Depth of tree is O(log (n)) – if partitioning segments are chosen carefully
- Could still result in $O(n^2)$ leaves, which means construction and rendering could be $O(n^2)$
- In practice, can be reduced by use of specialized techniques based on nature of input objects

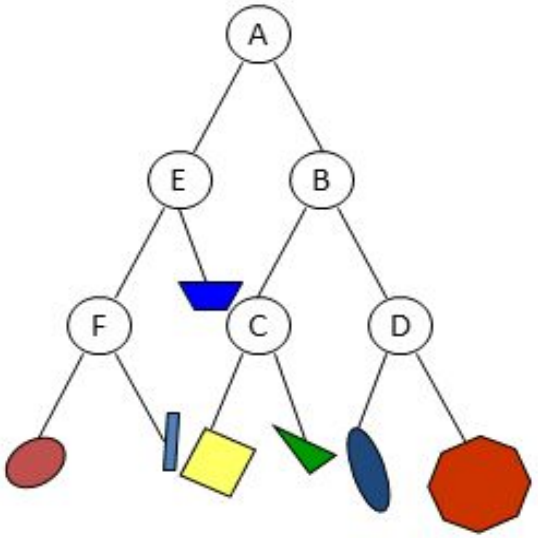
- Updates could also be expensive – when processing dynamic scenes

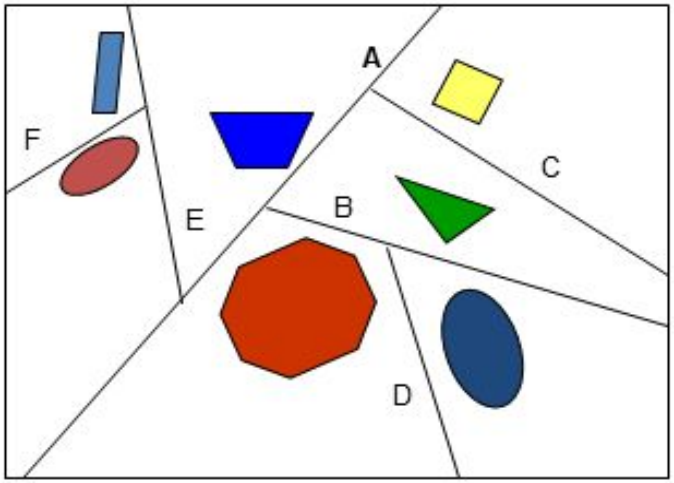
Building BSP Trees

For an arbitrary set of objects (in 2D, or all objects and viewing points on a planar surface)

- Choose partition planes that would recursively divide the space, using an appropriate rule
  - E.g. if objects are polygons, use an edge of any polygon as the partition plane
  - Or, find planes that divide the set of objects into approximately equal subsets.
- Build tree with the partition planes as interior nodes
- Partition objects based on which side of the partition plane they are in
- The set of input objects form the leaves of the tree.

This approach can also be extended to 3D.





Building BSP Trees

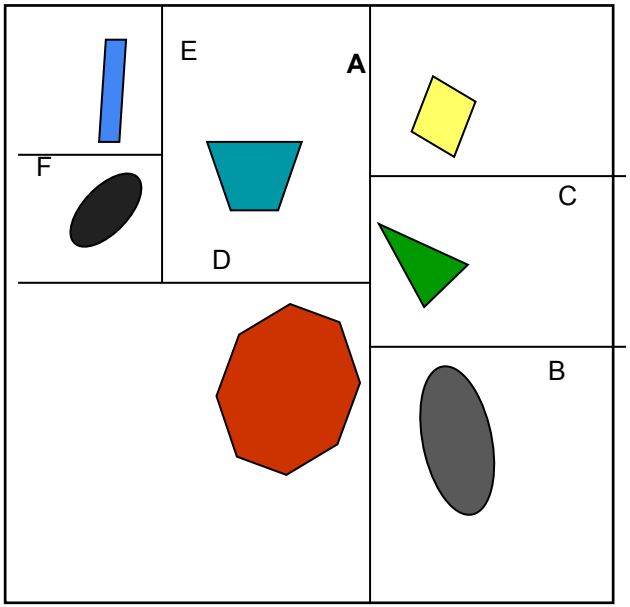
- Choice of partition plane is the key
- Depending on the application, could be based on objects or even the set of polygons in the scene
- Options
  - Divide space uniformly (quadtree/octree)
  - Split so that each partition has approximately equal elements
  - Split based on mean or median position of elements
- In some cases, would also make sense to use this partitioning to build the scene graph. Can allow application level optimization during traversal for rendering (e.g. culling – discussed later)

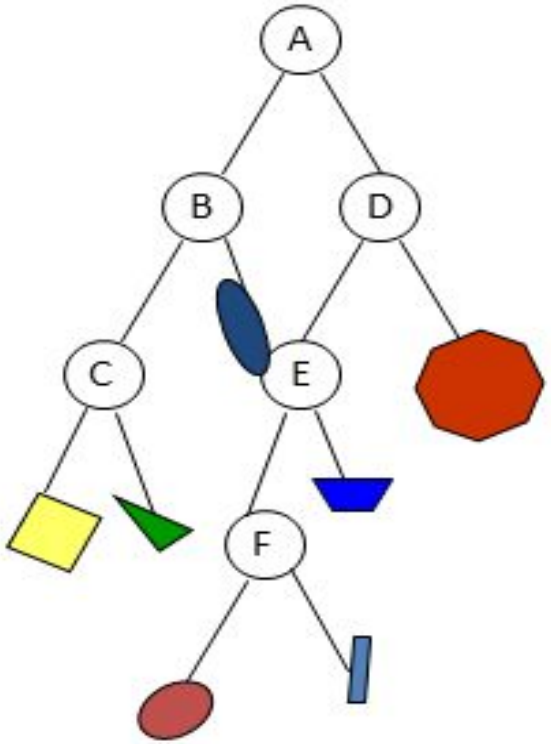
Space Partitioning – k-d Trees

A special case of BSP: *k-d* Trees

To simplify construction and testing, can use axis-aligned partition planes

Each subdivision divides a rectangular region into two sub-regions, by splitting along one of the coordinate directions.

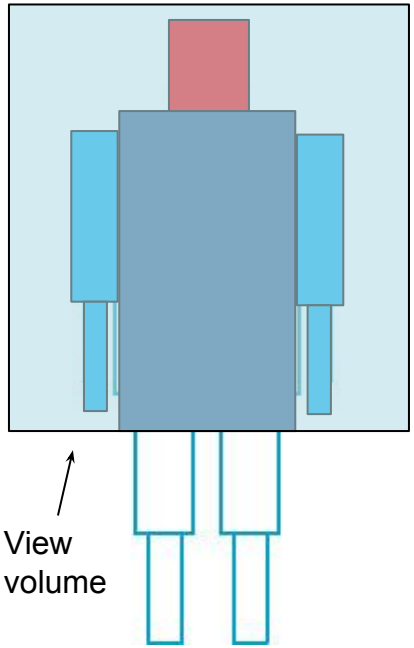


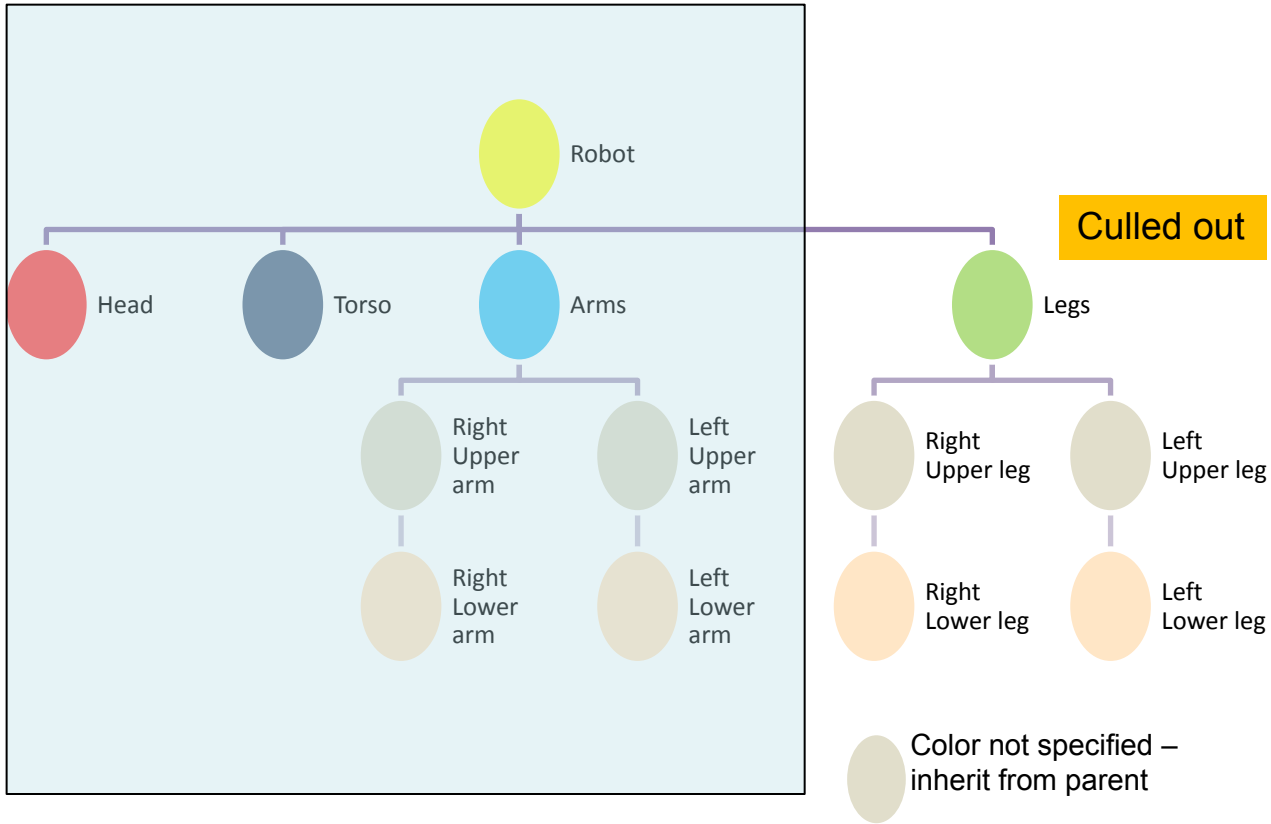


Culling

When rendering a complex scene, we would like to process only those objects that are within the view volume.

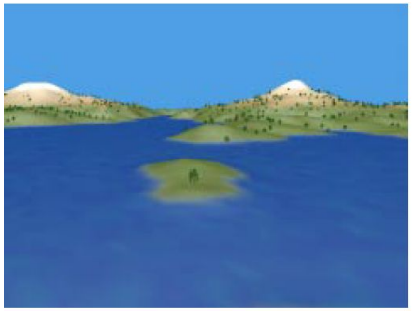
Culling out the other objects early in the render process can help improve performance, especially load on the graphics hardware.



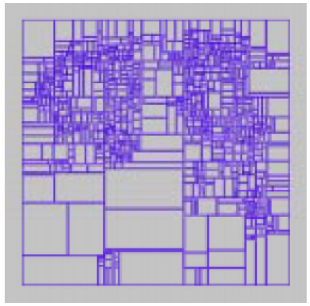


Efficient Rendering for Walkthroughs

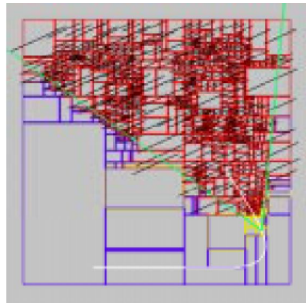
- How can we organize the data to support real-time rendering for walkthroughs?
- Preprocess complex scenes into hierarchical space partitions
- Allows for efficient computation of visibility sets and/or culling



The scene



BSP tree



Visible region

SIGGRAPH 96: Hierarchical Image Caching for Accelerated Walkthroughs of Complex Environments, J Shade, D Lischinksi, D Salesin, T DeRose, J Snyder

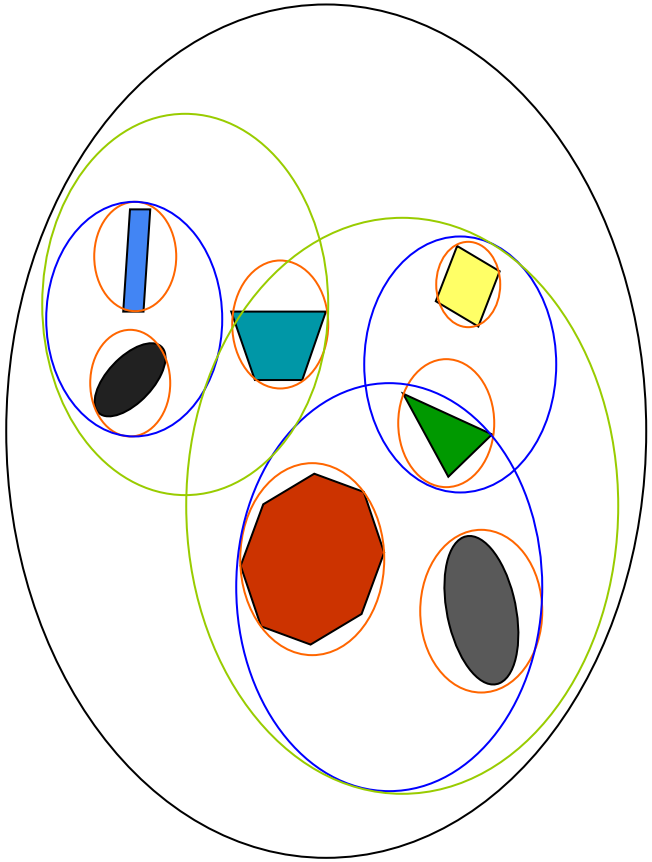
Object-based Hierarchies

- Create hierarchies based on the position and size of objects.
  - Useful for collision-detection and other inter-object interactions

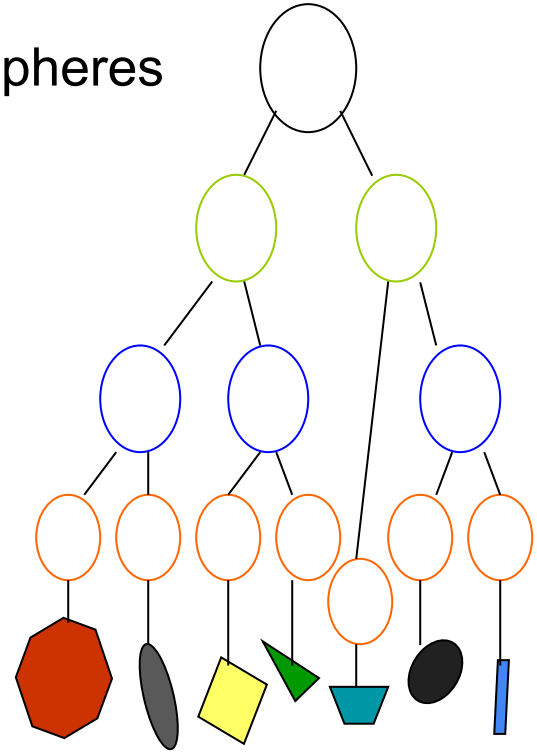
- Define regular **convex** regions that tightly bound objects of interest.
- Build up tree by grouping subtrees and computing bounding convex regions for the tree.

- Any ray that does not hit a bounding region cannot hit objects inside the region
- Two objects cannot intersect/collide if their bounding regions do not intersect

Hierarchical Bounding Regions



Bounding spheres



Bounding Volume Hierarchies

Bounding spheres

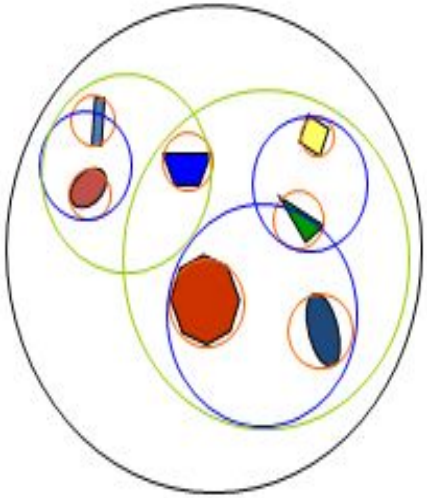
- Provides a tight cover of containing objects
- Bounds preserved through translate/rotate/scale
- Requires computation of distances for collision tests

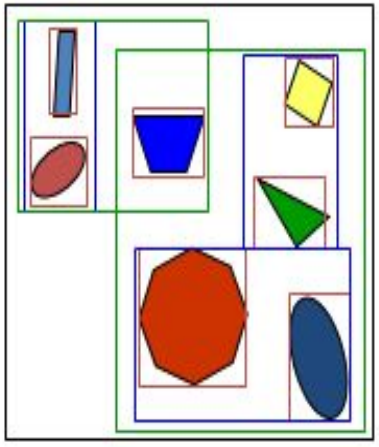
Bounding Boxes (axis-aligned)

- Computationally simple – for tree construction and collision tests
- Box needs to be recomputed after rotation to find axis-aligned bounds

Choice of bounding shape often a trade-off between simplicity and tightness of fit (and hence efficiency of collision/containment tests)

Other volumes, such as Oblique Bounding Boxes, also used





How do you check for collisions given 2 complex objects and their BVH’s?

Rendering at Scale

- What if we have very large spaces and very large number of objects? How do we partition and organize objects to allow efficient search, locate, render …
- Rendering a portion of objects in a certain area of the earth, given a database of millions of objects distributed over the surface of the earth
  - E.g. Google Earth, Uber cabs, map features, …

Locating Objects on the Earth Surface

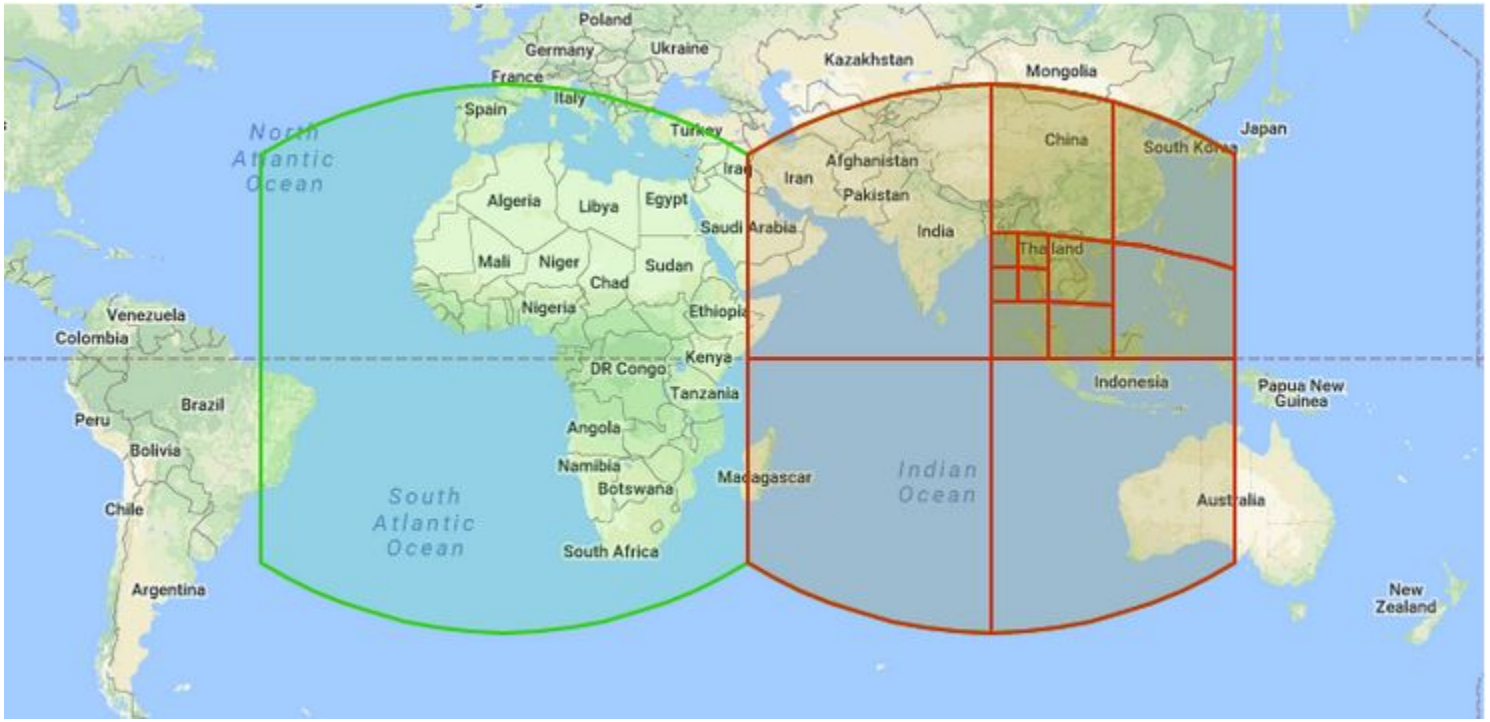
(Not strictly part of Graphics or this course, but may be of interest to some of you).

Geometry on the Sphere: S2 Geometry (s2geometry.io) Position a cube that fits the sphere

- Project each point of the sphere onto one of the 6 faces (extend ray from center to point on surface)
  - Adjust mapping to get somewhat uniform sized cells
- Build a quad-tree on each face.
- Derive a linear index for each cell using Hilbert Curves (can index all cells with 64 bits)

https://docs.google.com/presentation/d/1Hl4KapfAENAOf4gv-pSngKwvS_jwNVHRPZTTDzXXn6Q/view#slide=id.i0

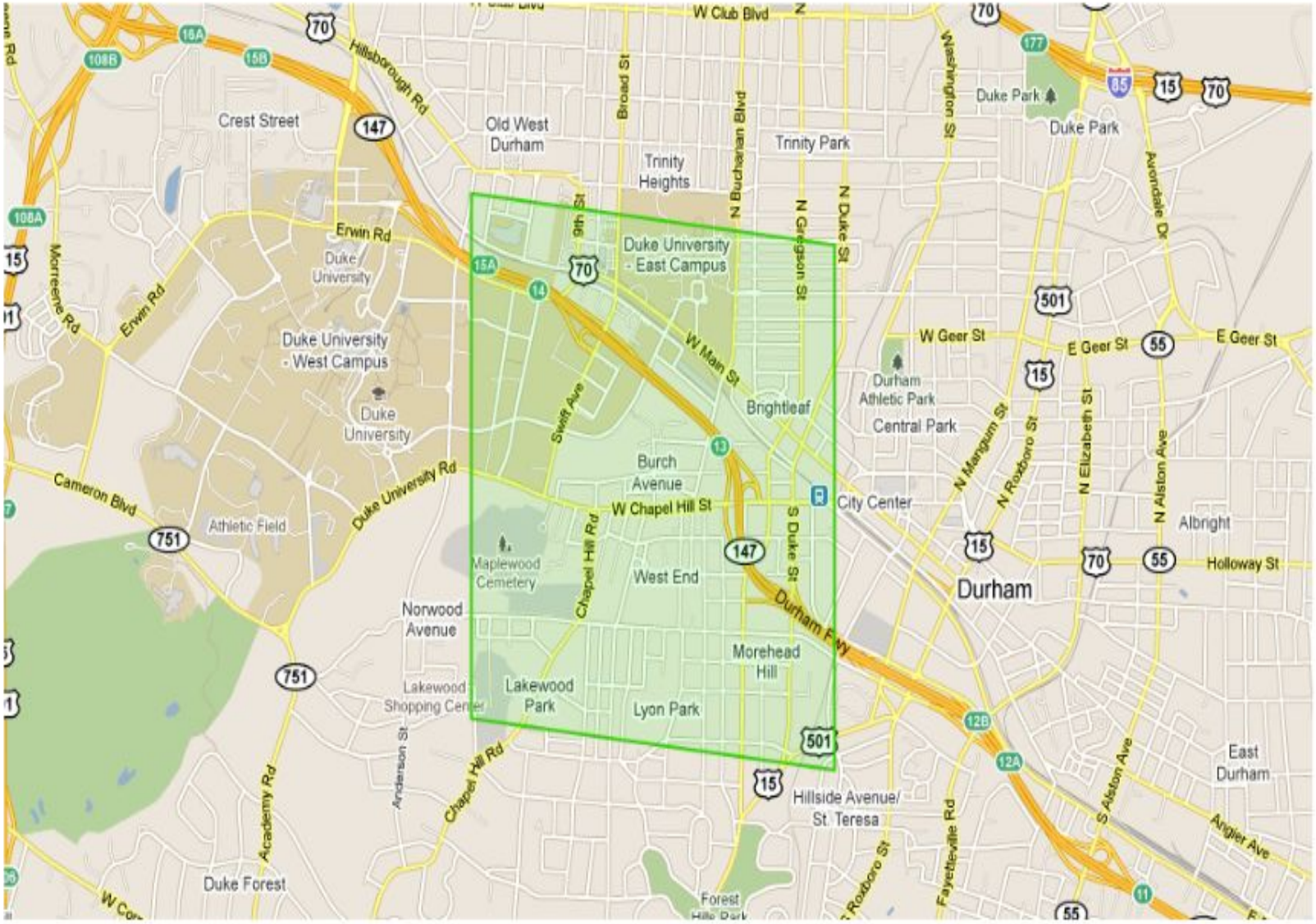
http://bit-player.org/extras/hilbert/hilbert-mapping.html


Poland
Germany
Ukraine
France
Italy
Spain
Turkey
Kazakhstan
Mongolia
China
Japan
South Korea
Afghanistan
Iran
Pakistan
India
Thailand
Saudi Arabia
Algeria
Libya
Egypt
Mali
Niger
Chad
Sudan
Nigeria
Ethiopia
Kenya
DR Congo
Tanzania
Angola
Namibia
Botswana
South Africa
Madagascar
Indonesia
Papua New Guinea
Australia
New Zealand
Venezuela
Colombia
Brazil
Peru
Bolivia
Chile
Argentina
North Atlantic Ocean
South Atlantic Ocean
Indian Ocean

S2: Quadtree size

Level	Min Area	Max Area
0	85,011,012 km^2	85,011,012 km^2
1	21,252,753 km^2	21,252,753 km^2
12	3.31 km^2	6.38 km^2
30	0.48 cm^2	0.93 cm^2

smallest cell


W Club Blvd
Hillsborough Rd
Crest Street
Old West Durham
Broad St
Trinity Heights
N Buchanan Blvd
Trinity Park
Duke Park
Duke Park
Avondale Dr
Erwin Rd
Duke University
Duke University - East Campus
N Gregson St
N Duke St
Morreene Rd
Erwin Rd
Duke University - West Campus
W Main St
Brightleaf
Durham Athletic Park
Central Park
W Geer St
E Geer St
E Geer St
N Mangum St
N Roxboro St
N Elizabeth St
N Alston Ave
Duke University
Swift Ave
Burch Avenue
W Chapel Hill St
City Center
Cameron Blvd
Duke University Rd
Athletic Field
Maplewood Cemetery
Chapel Hill Rd
West End
S Duke St
Durham
Albright
Holloway St
Durham Fwy
Norwood Avenue
Morehead Hill
Lakewood Shopping Center
Lakewood Park
Lyon Park
Anderson St
Chapel Hill Rd
Academy Rd
Hillside Avenue/ St. Teresa
East Durham
S Alston Ave
Angier Ave
Fayetteville Rd
Duke Forest
Forest

Hexagonal Cells

Uber H3: Hexagonal Hierarchical Space Index (eng.uber.com/h3/)

Use hexagons as cells with 16 levels. Hexagons have some nice properties such as distance to adjacent cells, etc.