Animation (2)

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

Scene Graph

CS 606 Computer Graphics

Term 2, 2022-23

T K Srikanth

IIIT Bangalore

Scene Graph: Animation

- Scene Graphs provide a convenient way to manage scenes and their animation
  - Most commonly used approach – gaming engines, simulation displays etc.
- Represent the scene as a hierarchical structure, with (relative) transforms at each node
- For each time step of the animation
  - Update the positions (and other properties) of the nodes of the graph
  - Render the nodes
- Both above involve pre-order traversal of the graph, and using a run-time stack for maintaining transforms and other properties of each sub-tree

Graph of Scene Objects

Similar to position, other properties could also be hierarchically defined

- Material properties (color, material) : sometimes a node inherits properties of parent, and sometimes overrides them
- Can tag these to the nodes of the tree/graph and process them appropriately during traversal

In addition to the model being rendered, objects such as lights and camera also have geometric properties as well as specific properties

If we can add all these objects into a single graph, then traversing this graph allows the scene to be rendered with the correct geometry, material, lighting, camera view.

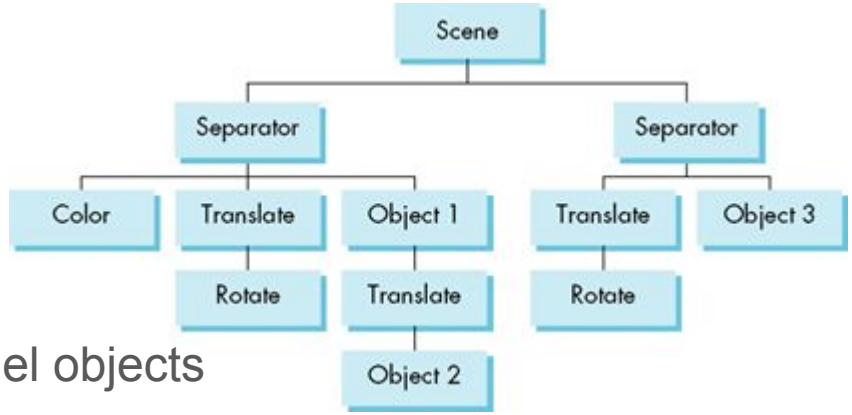
A structure such as this is needed in any case, since OpenGL does not store display information once rendered (immediate mode). Data of the scene needs to be resent to the pipeline after any change in scene – geometry, material, lighting, camera view.

Scene Graph

A graph that represents the objects of a scene and their relationships: camera, lighting and the hierarchical model that is to be rendered

Comprises the following types of nodes:

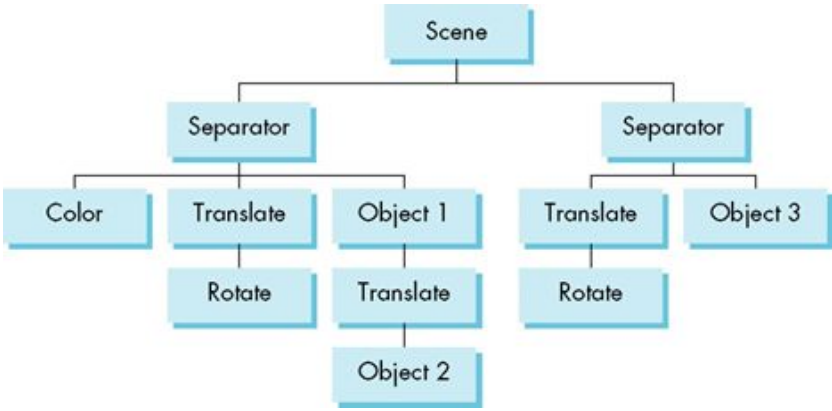
- Geometry: defining the shapes – primitives, mesh surfaces, parametric surfaces, curves etc.
- Lighting: type, position and orientation
- Camera: type, position and orientation
- Transform
- Appearance: material, color etc.
- Group/Separator: logical grouping of model objects
  - Isolate state changes
  - Equivalent to push/pop



Scene Graph

Traversal of the graph is used for multiple purposes:

- Display
- Interactive pick
- Computing bounding boxes
- Searching
- Writing to stream



Specific actions in an SG traversal are application specific and need to be designed (for a particular kind of application)

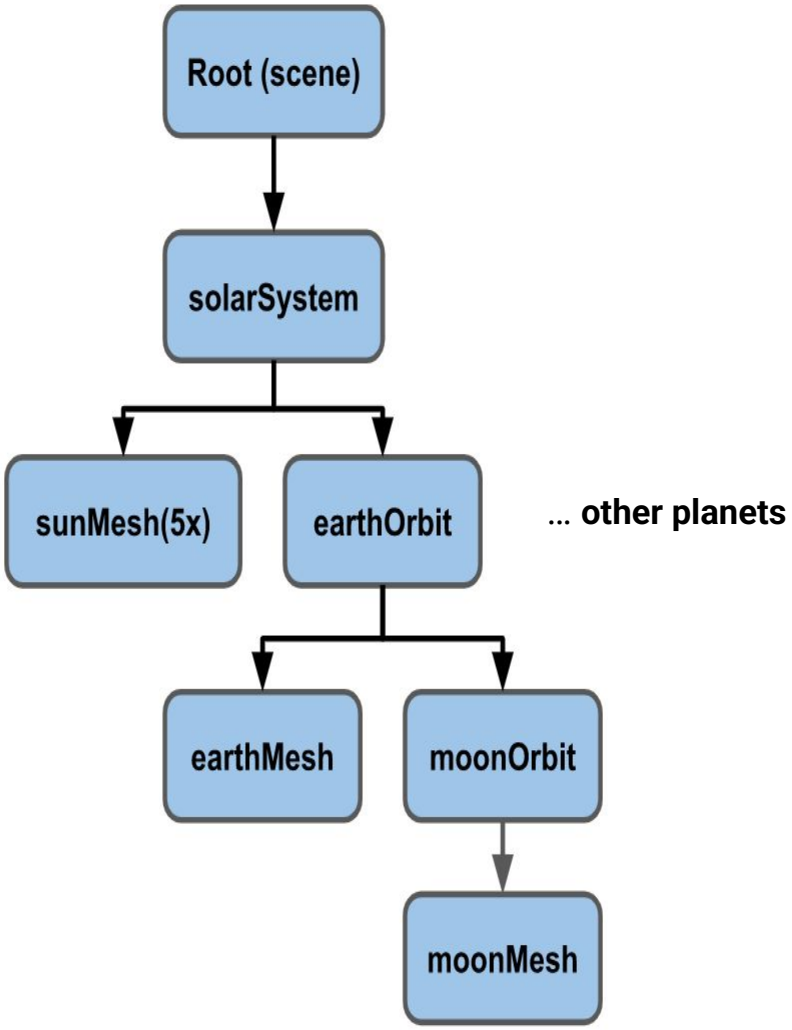
Concept of scene graph is used in most modeling, graphical editing and gaming systems. Encouraged to use this where possible

Solar System: Example

How would you create a dynamic model of the solar system

- The Earth
  - completes an elliptical orbit around the sun in 365 earth-days
  - Has its axis of rotation tilted at 23.5 degrees to the plane of orbit
  - Rotates on its axis in 1 day
  - Has a satellite – the moon
    - Revolves around the earth in ~28 days
    - Rotates once for each revolution around the earth
- Similar descriptions for each of the planets (including images of the surface appearance)

Scene Graph: Solar System



https://threejsfundamentals.org/threejs/lessons/threejs-scenegraph.html

Scene Graphs - Implementation

General structure of base class of the graph nodes:

```
class SceneNode {
    virtual void render(int mode) {}
    virtual void update(float time) {}  //
    // list of children (of type SceneNode)

    // add/remove child from parent node

    // Transforms
    … set and get Translate vector/values
    .. Set and get Rotation values
}
```

Implement appropriate classes for sub-types such as:

- Cylinder, sphere
- Mesh surface
- Bezier surface…
- Groups of nodes, etc.

(The data elements stored and the render method would typically be different depending on the geometry)

Scene Nodes

Derived classes are created for the different geometry types. Examples:

```
class TriMeshNode : public SceneNode {
  // contains information for a triangular surface mesh (e.g. ply file)
  // construct and manage the geometric data

  // override render and update:
  //    Typically render will invoke GL methods to render tris
  //    Camera and lights could be updated where relevant

 //  update would compute the new transforms associated with the node
}
```

Note: This will vary depending on the library you use. Many libraries provide their variant of this concept.

Scene Graph & Program Flow

Controller:

Instantiates Model objects and adds them to the Scene Graph (as needed by the application)

Sets up callback from View or other events (e.g. timer) where the event will be handled

Within the timer or other callback, repeatedly invokes the following:

```
{
        // update time;

        sceneGraph.update(time);

        sceneGraph.render();
}
```

Scene Graph - Guidelines

- Implement the derived classes (of SceneNode), implementing update and render methods
- Set up the scene graph: identify dependencies between object and build up the correct hierarchy
  - Read in all data files, or otherwise create the model nodes
  - Ensure the data read in, or objects created during the program, are done only once
- Compute and store properties such as normals, texture coords etc, with the object
  - Generating coords for the model as well as normals and texture coords should generally be done only once per object – at the time the scene object is created
- The main display function (callback triggered by changes in OpenGL state, or other events) invokes **update** followed by **render**
- Ideally, add lights and camera to the Scene Graph (and have appropriate render methods) to simplify the overall management.

Kinematics

- Forward kinematics often difficult to compute, since consistency has to be maintained between the values of the different degrees of freedom of the joints.
  - E.g. movement of knee joint has to be coordinated with movement of hip joint in a simple humanoid model
  - Can get very complicated as the number of DOFs increase

- Typically, we are interested in the movement of a specific part of the object such as the torso of a humanoid or the end effector of an articulated robot

- Can we specify the path of the end effector and derive the values of the joint DOF's? Also called "goal oriented motion"

Inverse Kinematics

For an articulated model, such as a robot arm, we can compute the position of all objects as a function of time, if we can compute the joint angles at each point in time

In particular the position of the tip of the arm can be determined from the set of joint angles.

If **p** is the position of the tip, and **θ** is the vector of joint angles**,**

$$\mathbf{p = g(\theta)}$$

How do we compute the reverse or **inverse kinematics** (IK)

- Given a desired state of the model, compute the joint angles
- E.g. compute the positions of the joint angles given the position of the tip
- That is, derive $\mathbf{\theta = g^{-1}(p)}$

Inverse Kinematics

How do we compute the reverse or **inverse kinematics** (IK)

- Given a desired state of the model, compute the joint angles
- E.g. compute the positions of the joint angles given the position of the tip
- That is, derive

$$\theta = g^{-1}(p)$$

Need to find a *feasible solution*.

This may not have a unique solution. No unique inverses for the forward kinematics equations. Usually solved by iterative numerical methods.

Inverse Kinematics

- IK usually solved by setting up the Jacobian of the kinematics system
  - i.e. the matrix of partial derivatives of the position function w.r.t each of the state variables (DOF's)

$$X = f(\theta)$$
$$dX = J(\theta)d\theta$$

or

$$d\theta = J^{-1}(dX)$$

This can be used to iterate towards the goals by a series of incremental steps, computing the intermediate θ values at each stage.

Dynamics

We use **dynamics** to bring in effects of forces: friction, gravity, collision, etc.

E.g. modeling a ball bouncing in a room:

- Compute parabola that describes the path of the center, based on initial position and velocity
- Compute intersections with the walls
- Animate (in time steps) from current position till next intersection
- Determine new direction and velocity based on physics of reflection (directions, elasticity, …)
- Repeat process

Dynamics

- Study of motion of rigid bodies taking forces into account
- Modelled with the Newton-Euler equations (handles translation and rotations) and the external forces and torques on the system

$$m\dot{\mathbf{v}} = \mathbf{F}$$
$$\mathbf{I}\dot{\boldsymbol{\omega}} + \boldsymbol{\omega} \times \mathbf{I}\boldsymbol{\omega} = \boldsymbol{\tau}.$$

- Set of forces may need to be augmented with *friction*. Significantly complicates the problem
- Need to simultaneously solve the set of equations that incorporate kinematics, dynamics, constraints, friction etc.
- Critical for realistic simulations, games, VR, etc.

Motion Capture

- Markers/sensors attached to human or other “real” object
- Motion captured using cameras
- Complex algorithms used to infer motion and connect back to key parameters of model being animated

Animation Tools

Sample movie using Blender:

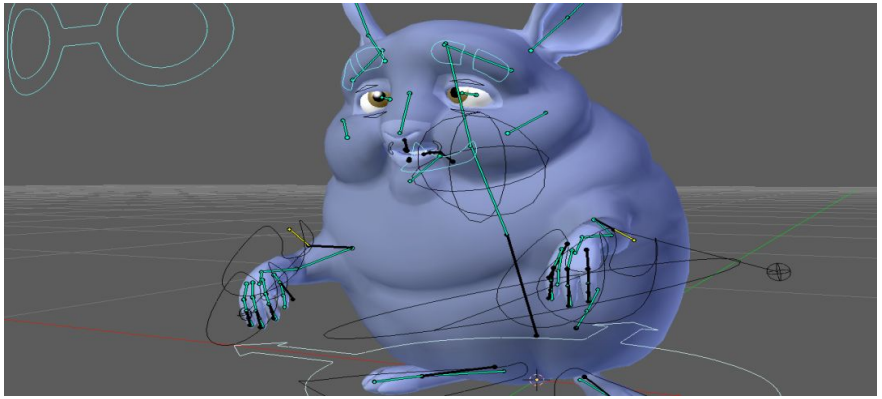
https://www.youtube.com/watch?v=ZIliNIrx1RY

https://cloud.blender.org/p/cosmos-laundromat/

As examples of animation and tools, see Maya demos:

https://www.youtube.com/watch?v=EpzhQu1lZYs

https://www.youtube.com/watch?v=IRPAmoJodJM



Blender.org

Animation and Performance

CS 606 Computer Graphics

Term 2, 2022-23

T K Srikanth

IIIT Bangalore

Animation: Realistic Rendering

Aspects of rendering

- Visual appearance: photo-realism (or non-photo realistic)
- Animation: rigid and non-rigid, physically realistic motion and deformation
- Real-time: performance of frame generation process

Some examples:

SIGGRAPH 2017 Animation selections

SIGGRAPH 2018 Animation winner: Hybrids

SIGGRAPH 2019: Animation selections

SIGGRAPH Asia 2021: Animation Selections

SIGGRAPH Asia 2022: Animation Selections

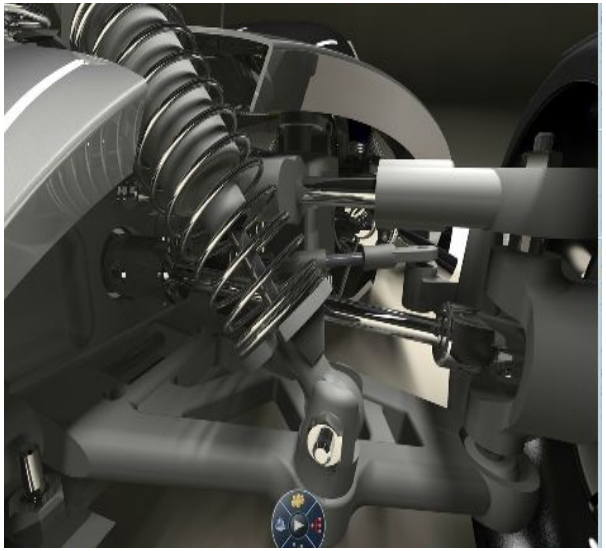
SIGGRAPH 2022 Electronic Theater Preview

Agenda

- Animating Complex Scenes
- Kinematics and Dynamics
- Space-based Partitioning - 1
    - Quadtrees and Octrees

Rendering Complex Scenes

Wikipedia



Microsoft

Autodesk

Scenes with High Level of Detail

Dreamworks

Rendering Large Complex Scenes

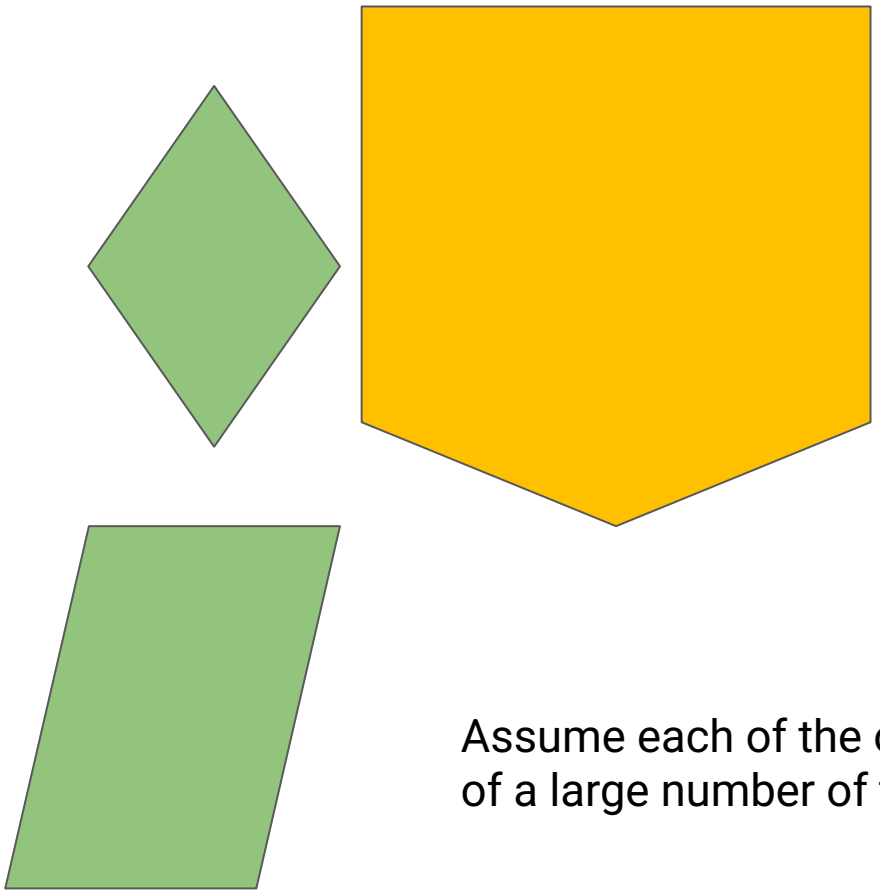
Need efficient schemes for handling large scenes for operations such as:

- 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)

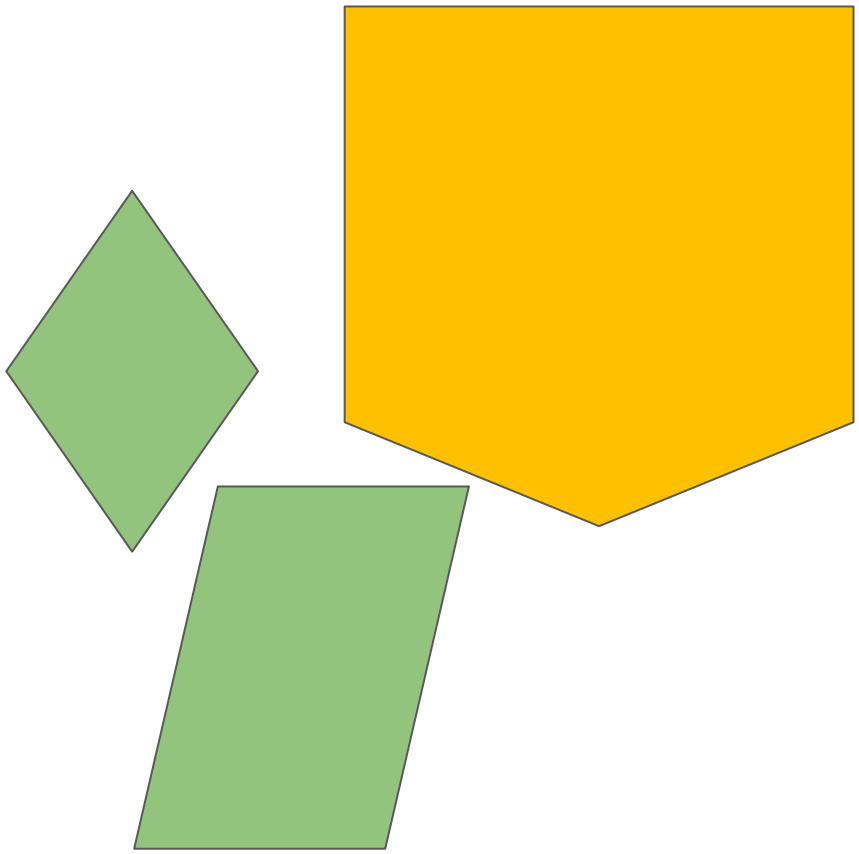
Explore ways of arranging the data in a scene into a hierarchy that will allow efficient determination of whether an object is “inside” or “outside” a region of interest, or meets some other similar spatial criteria.

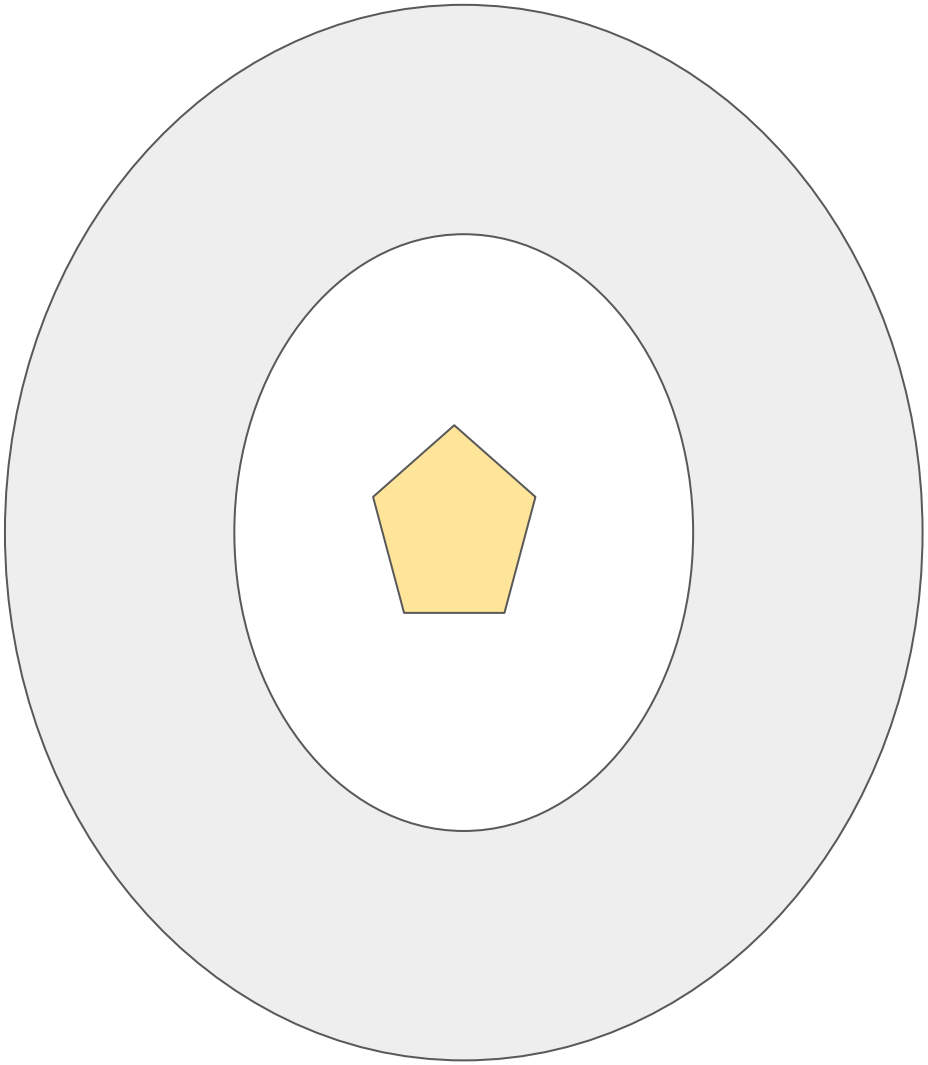
Collision Detection

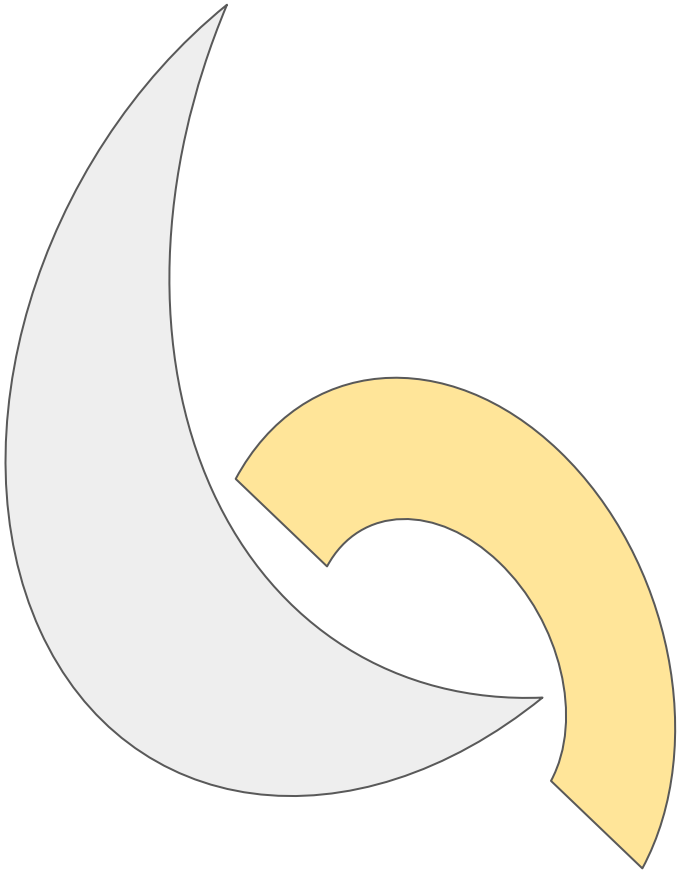


Assume each of the objects here consists of a large number of triangles

Collision Detection



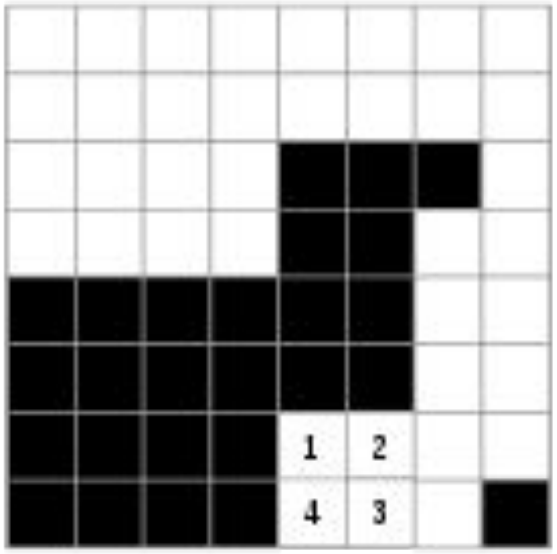


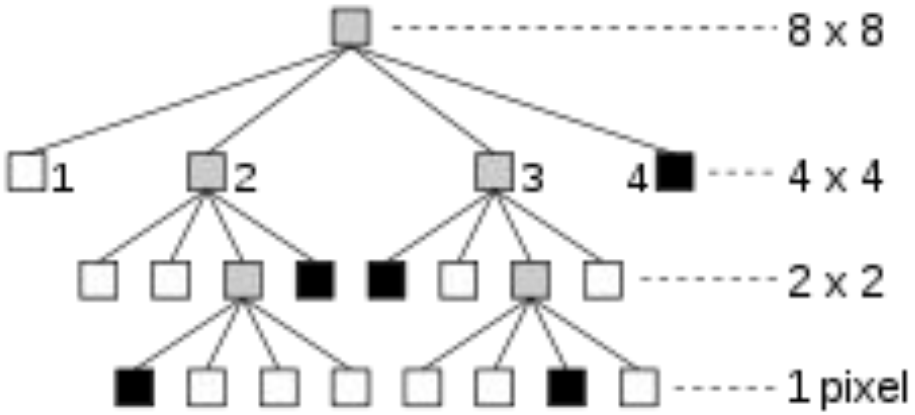


Grid-based Hierarchies: Quad-trees

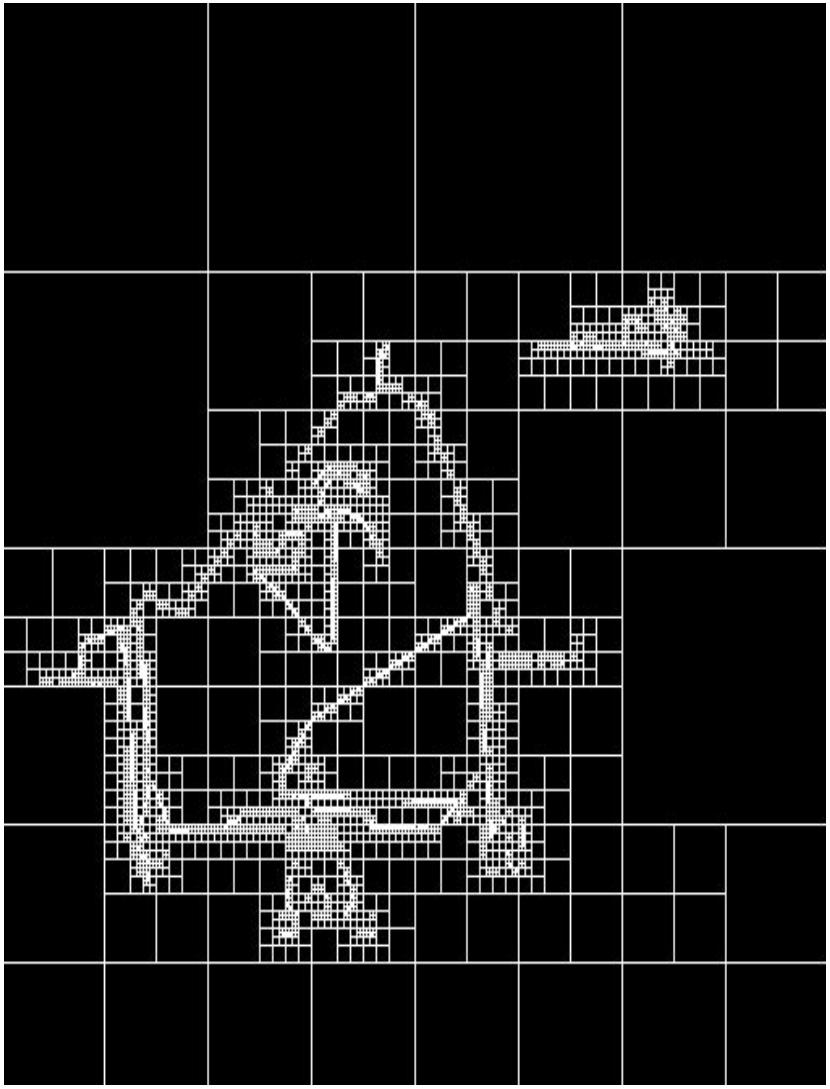
Quadtrees:

- 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



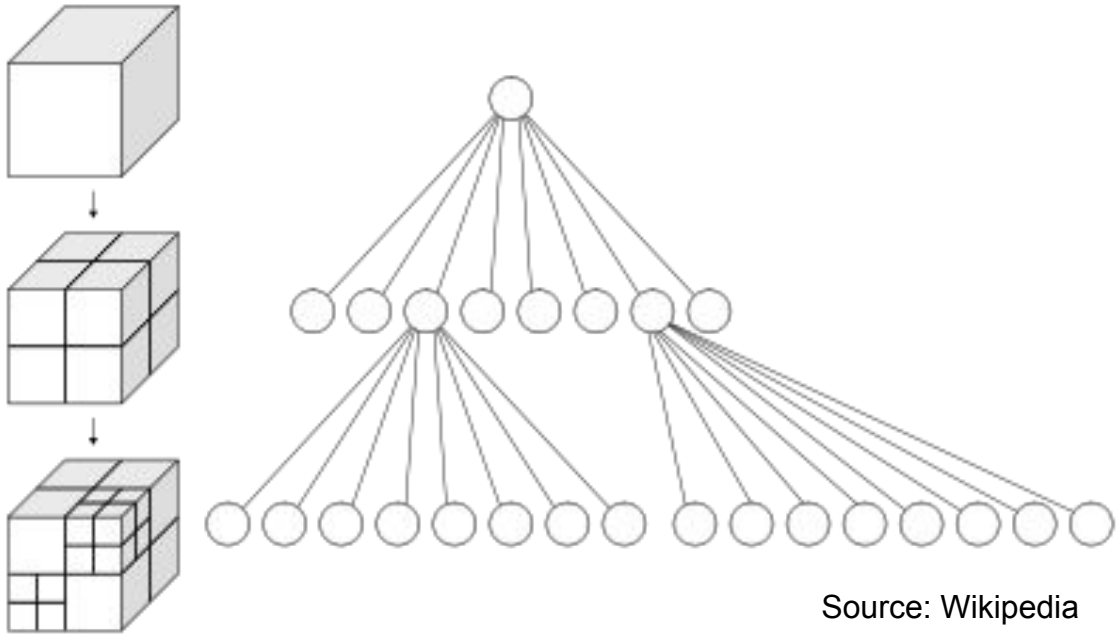
Source: Mathworks

Octrees

Octrees:

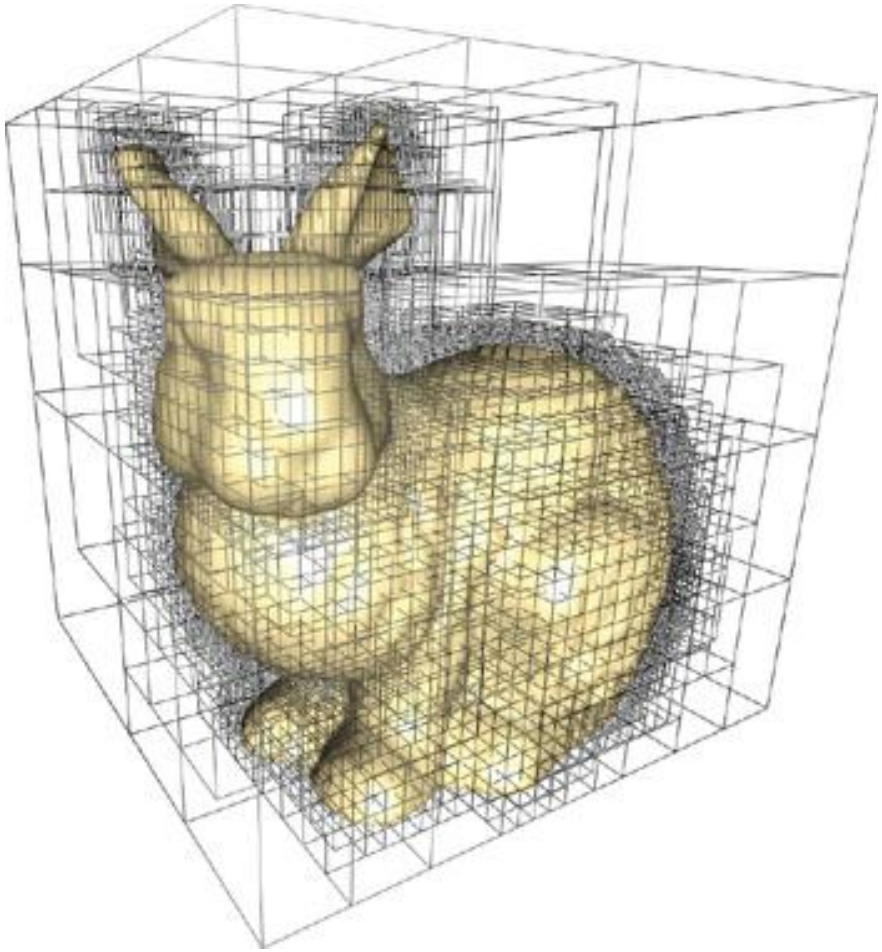
Extension of quadtrees to 3D.

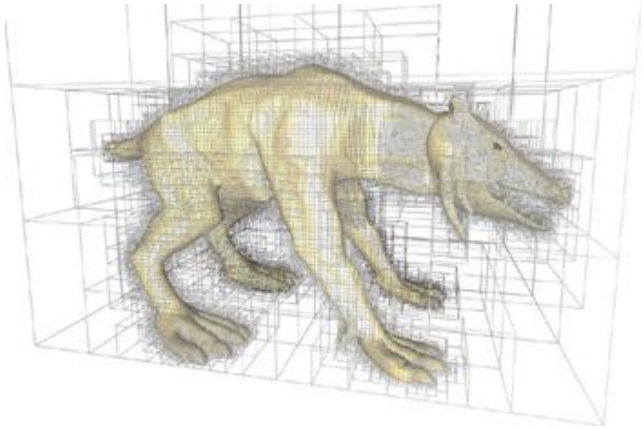
Each volumetric region (box) is divided into 8 sub-regions, 2 along each coordinate direction



Source: Wikipedia

Octrees





Source: Nvidia

Animation and Performance (2)

Agenda

- Space-based partitioning
  - Quadtrees and Octrees
  - BSP Trees, KD Trees

- Object-based partitioning
  - Bounding Volume Hierarchies

- Managing very large spaces and objects

Rendering Large Complex Scenes

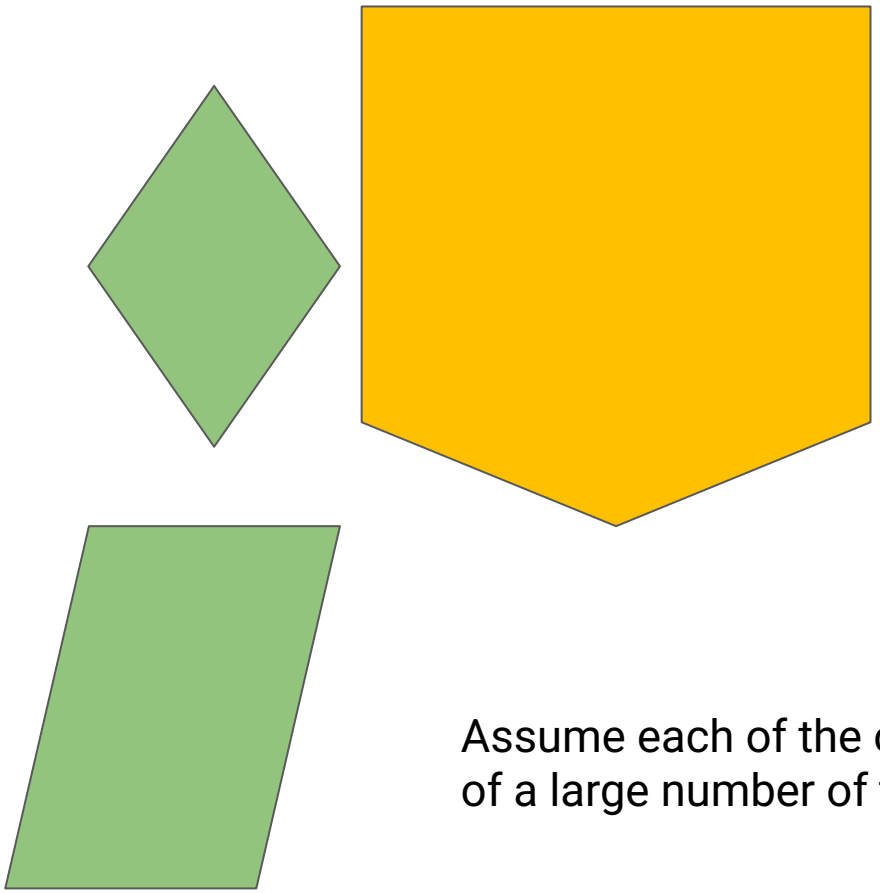
Need efficient schemes for handling large scenes for operations such as:

- 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)

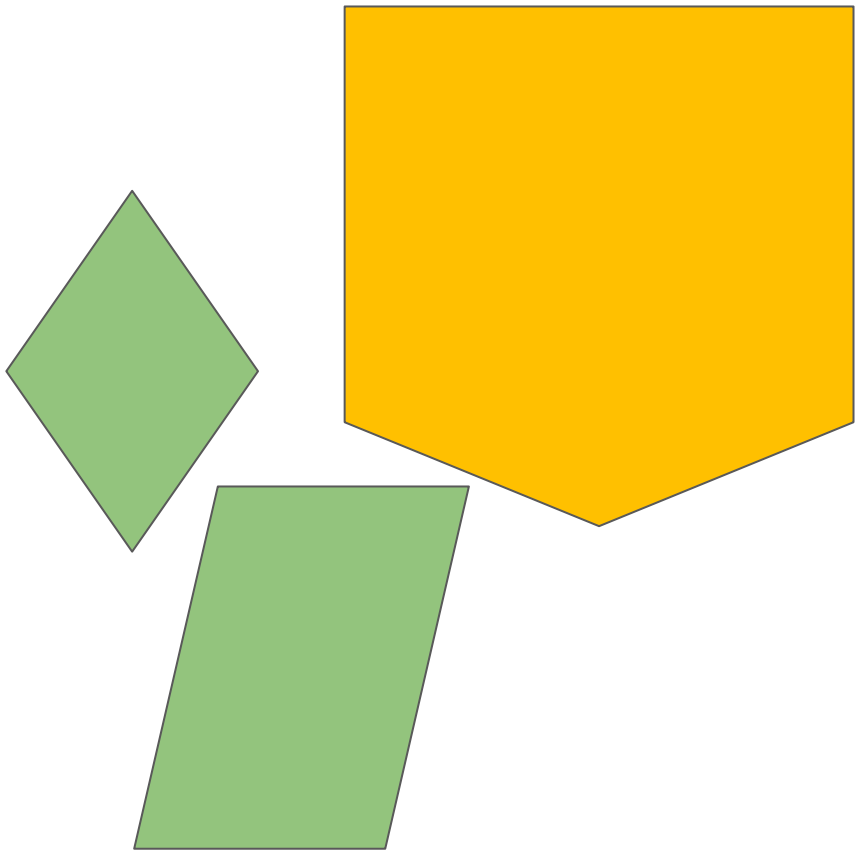
Explore ways of arranging the data in a scene into a hierarchy that will allow efficient determination of whether an object is “inside” or “outside” a region of interest, or meets some other similar spatial criteria.

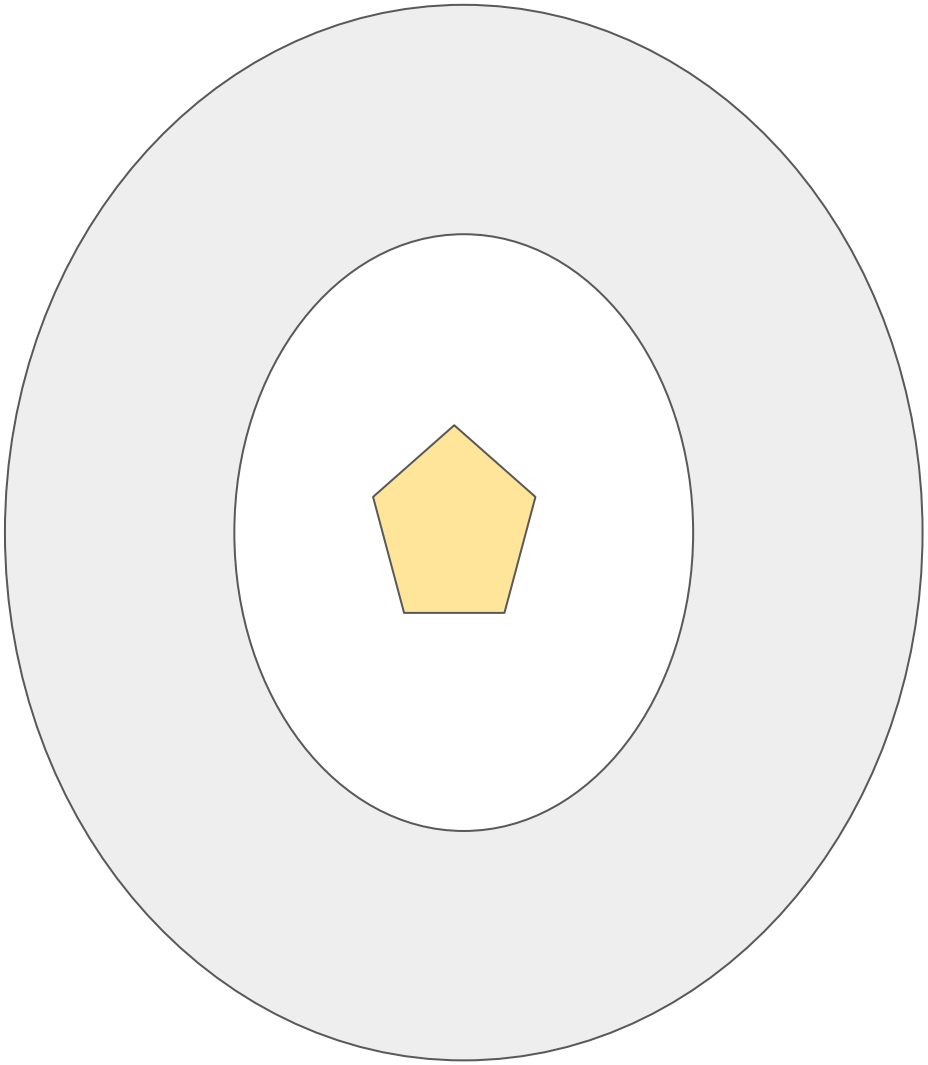
Collision Detection

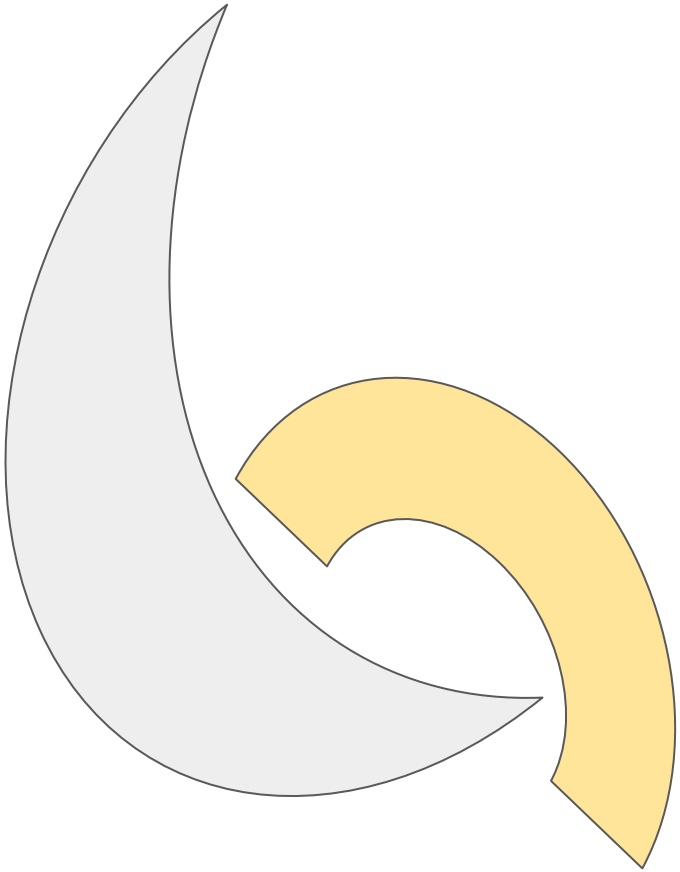


Assume each of the objects here consists of a large number of triangles

Collision Detection



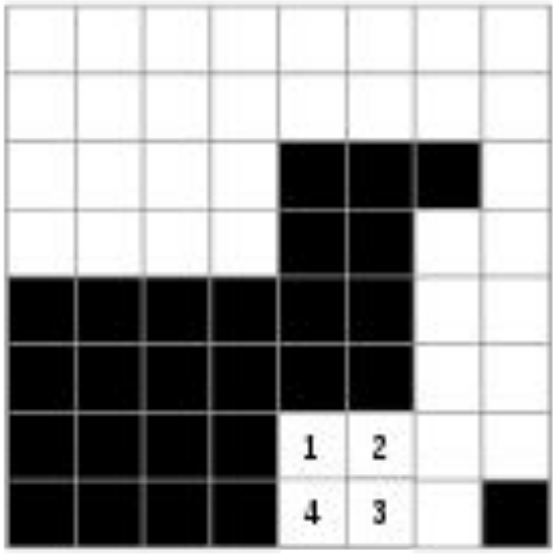


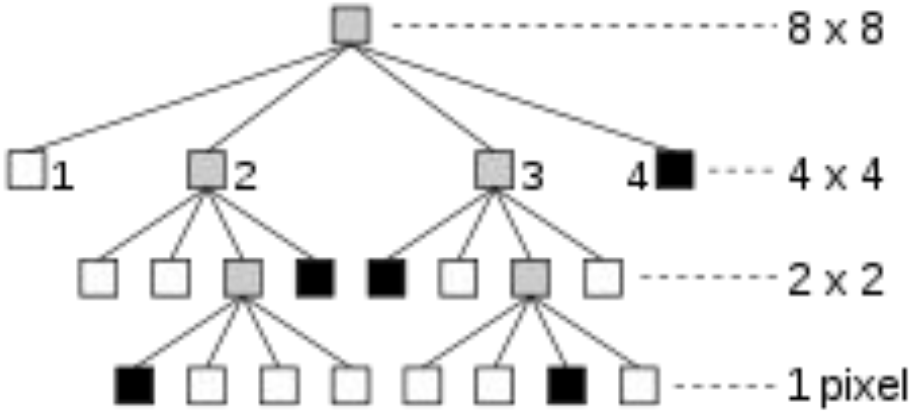


Grid-based Hierarchies: Quad-trees

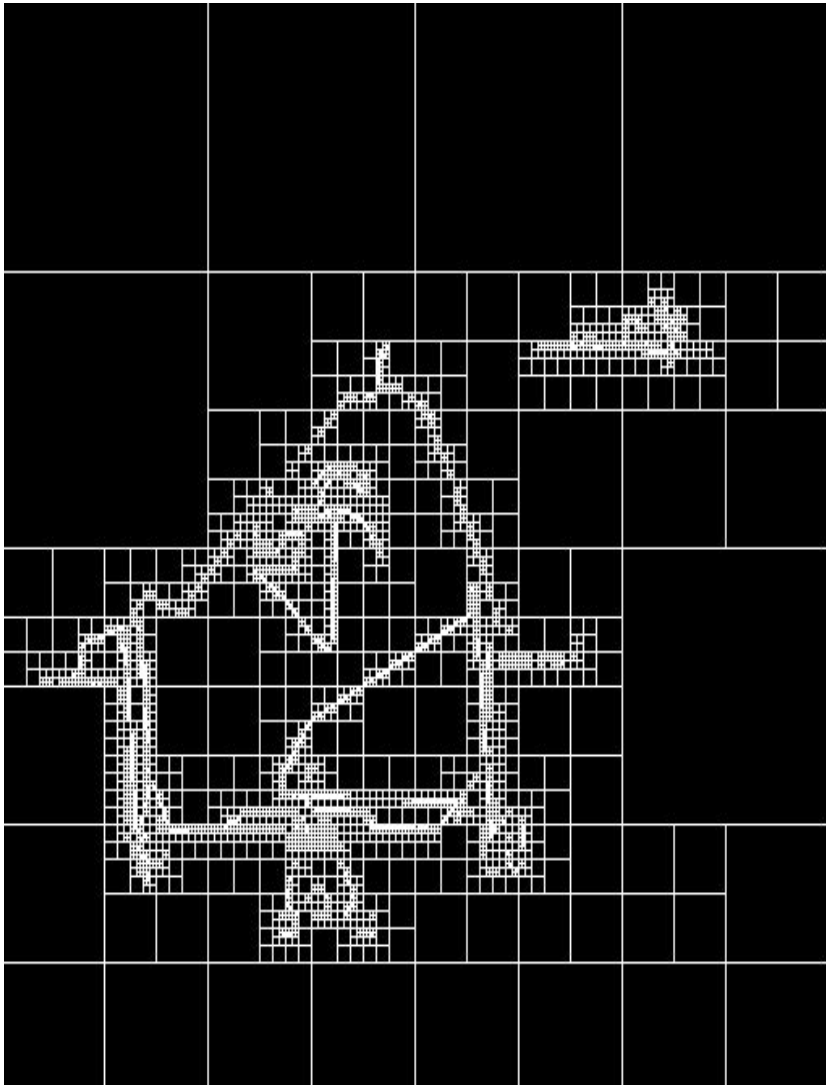
Quadtrees:

- 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



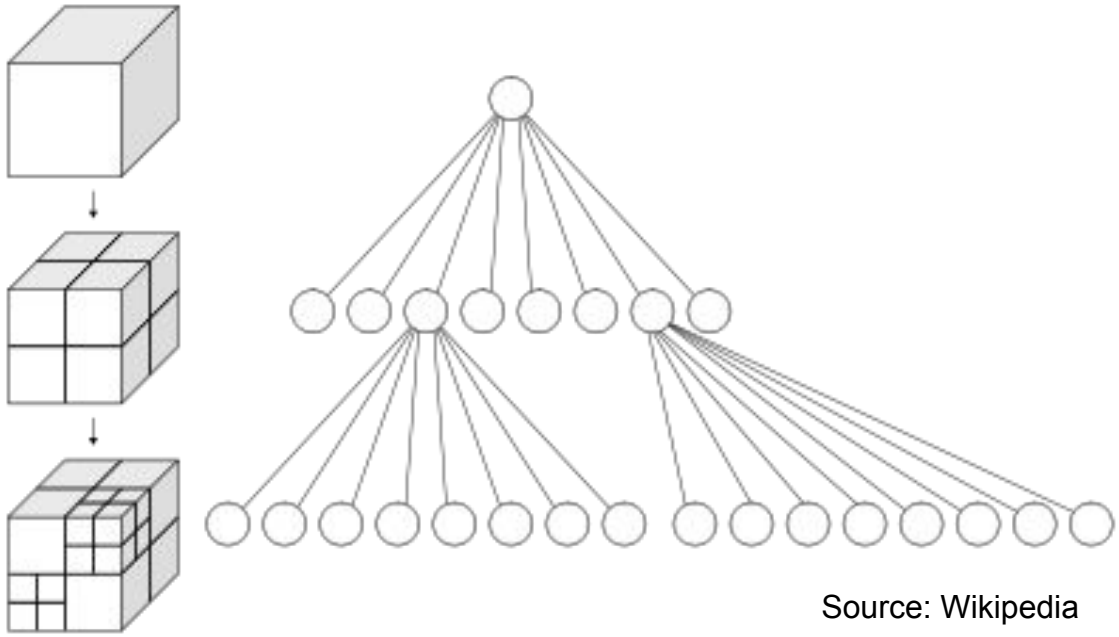
Source: Mathworks

Octrees

Octrees:

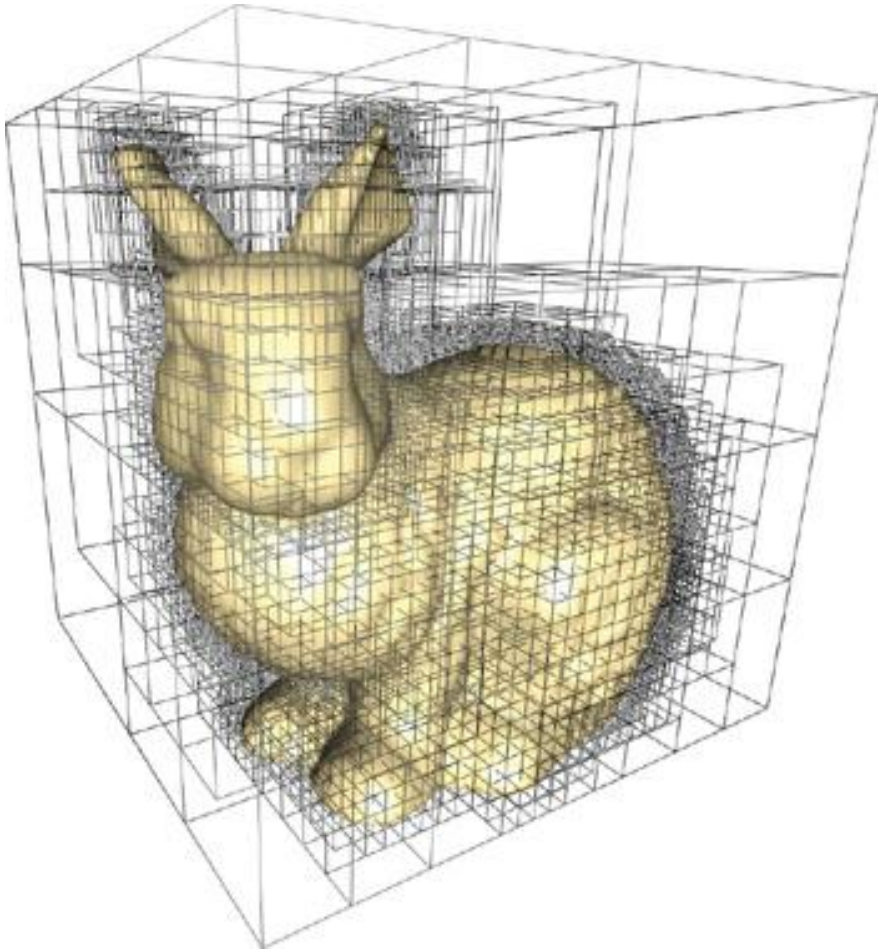
Extension of quadtrees to 3D.

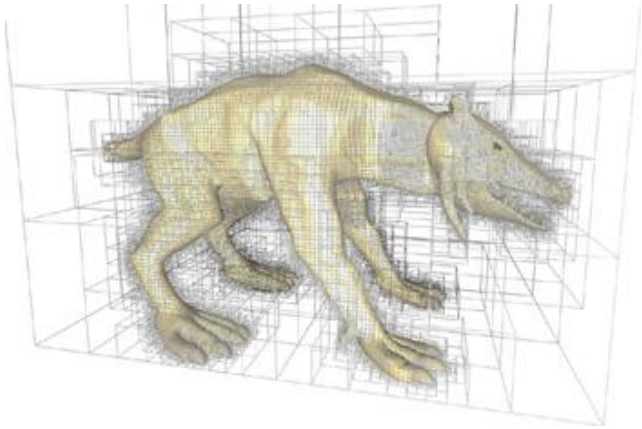
Each volumetric region (box) is divided into 8 sub-regions, 2 along each coordinate direction



Source: Wikipedia

Octrees





Source: Nvidia

Visibility Queries

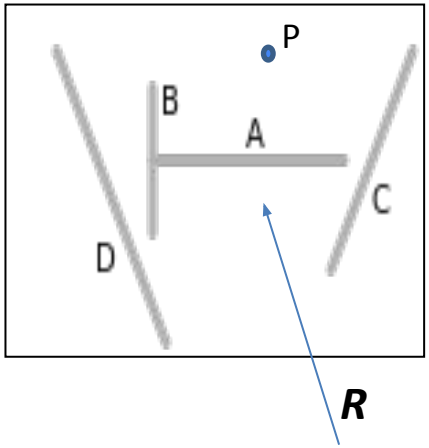
- Given a view direction (ray) ***R***, which line segments does it intersect?

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

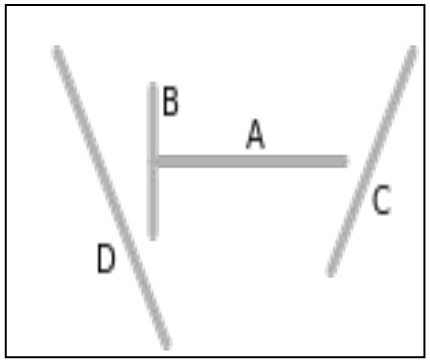
Query operations should be O(n) – linear in the number of line segments (/polygons).

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

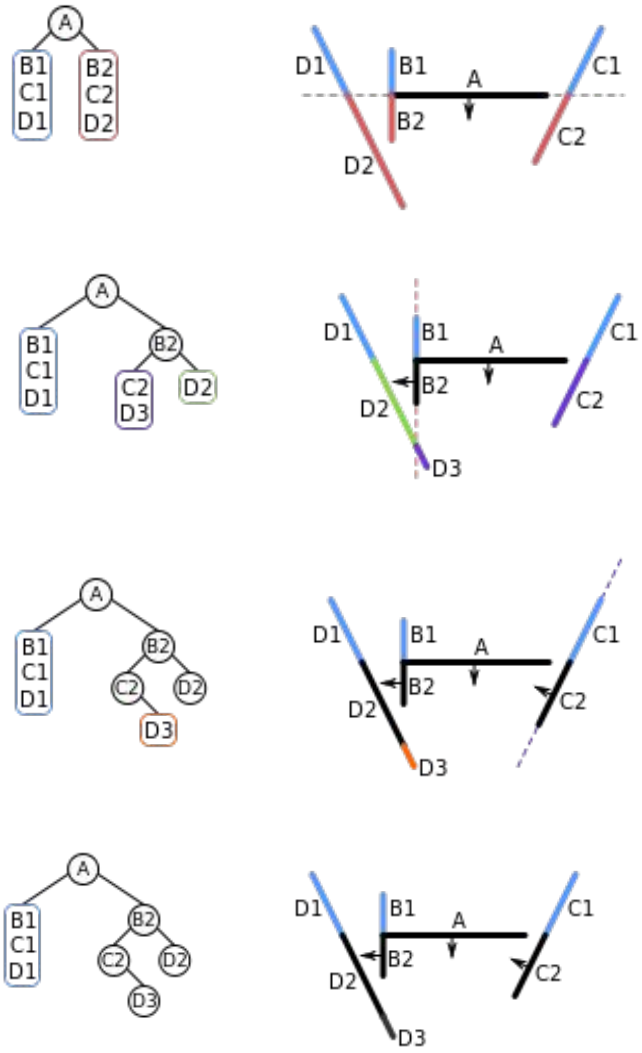
Key idea:
Each line segment s_i (when extended) partitions the region into 2 halfspaces, a “positive” H_{i+} and a “negative” halfspace H_{i-} . Assume segments are split by s_i into smaller segments if they intersect

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

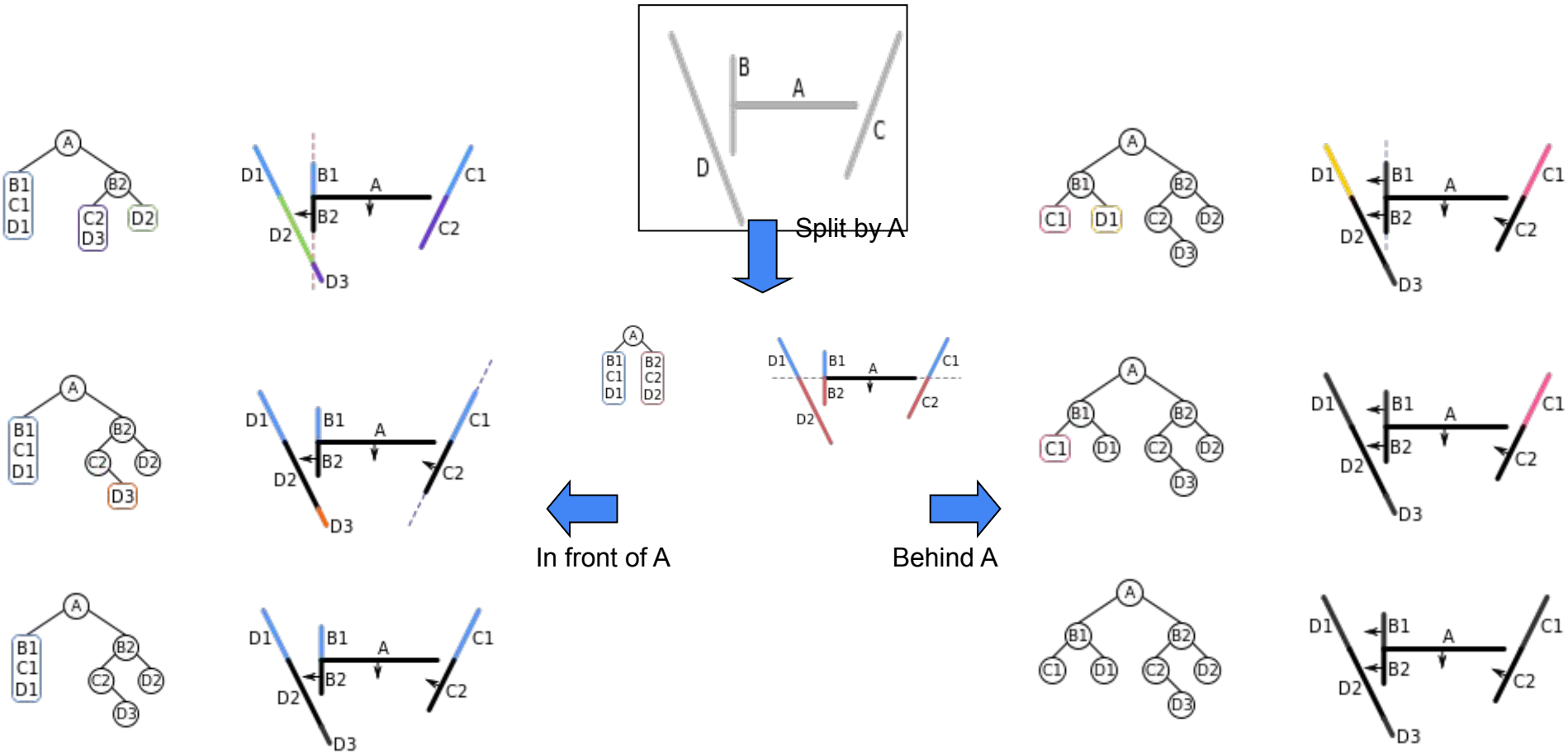
Building a Binary Space Partition tree (for line segments in 2D)

- 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**

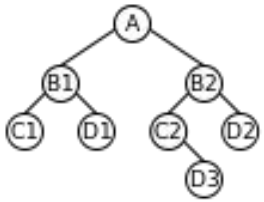
In-order traversal of the tree:

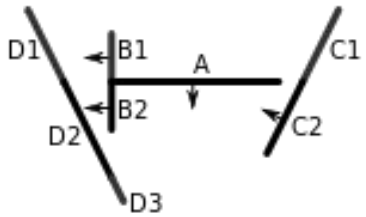
If **P** is in "front" of plane corresponding to the sub-tree root,

- 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)

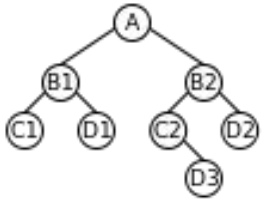


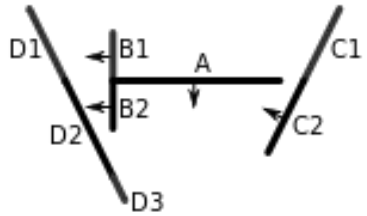


Rendering Using BSP Tree

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

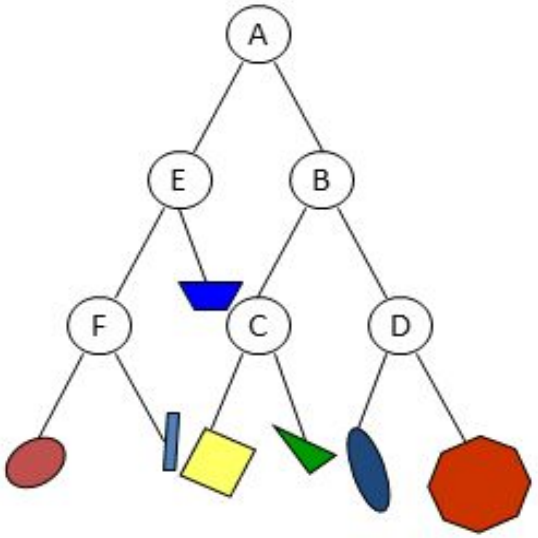
- Updates could also be expensive – when processing dynamic scenes

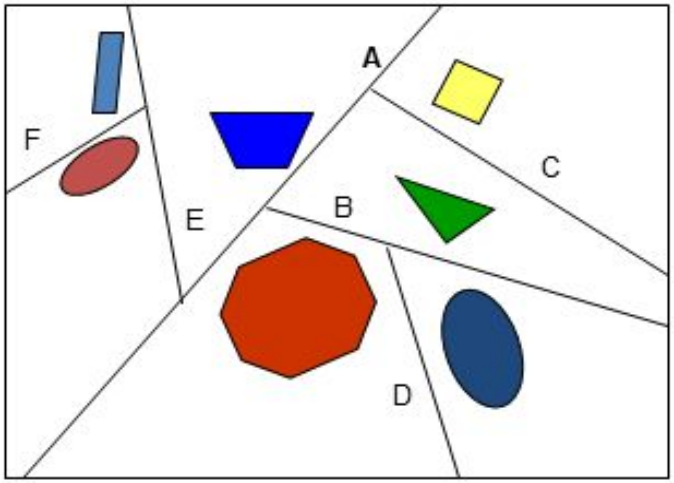
Building BSP Trees

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

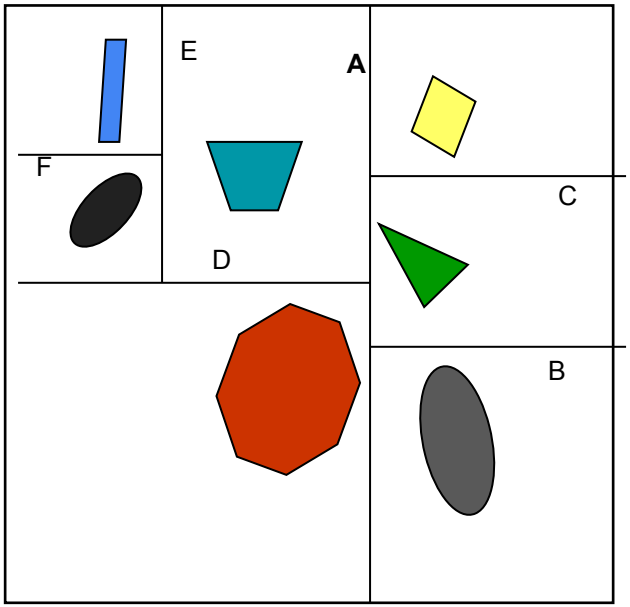
- Choice of partition plane is the key
- Depending on the application, could be based on objects or even the set of polygons in the scene
- Options
  - Divide space uniformly (quadtree/octree)
  - Split so that each partition has approximately equal elements
  - Split based on mean or median position 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)

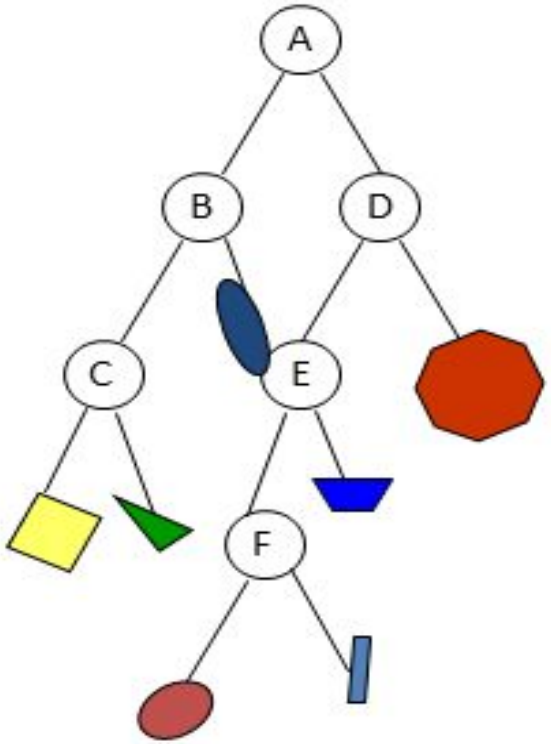
Space Partitioning – k-d Trees

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.

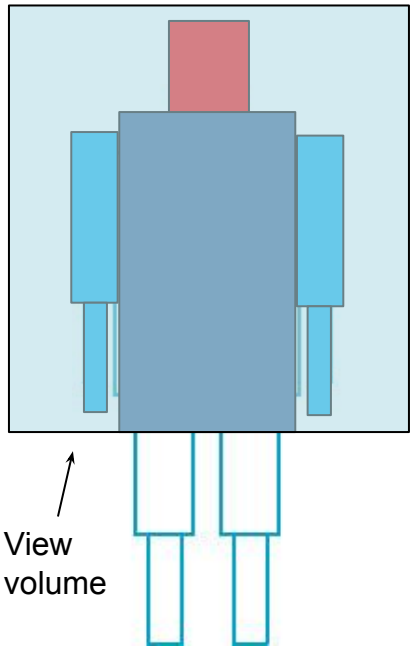


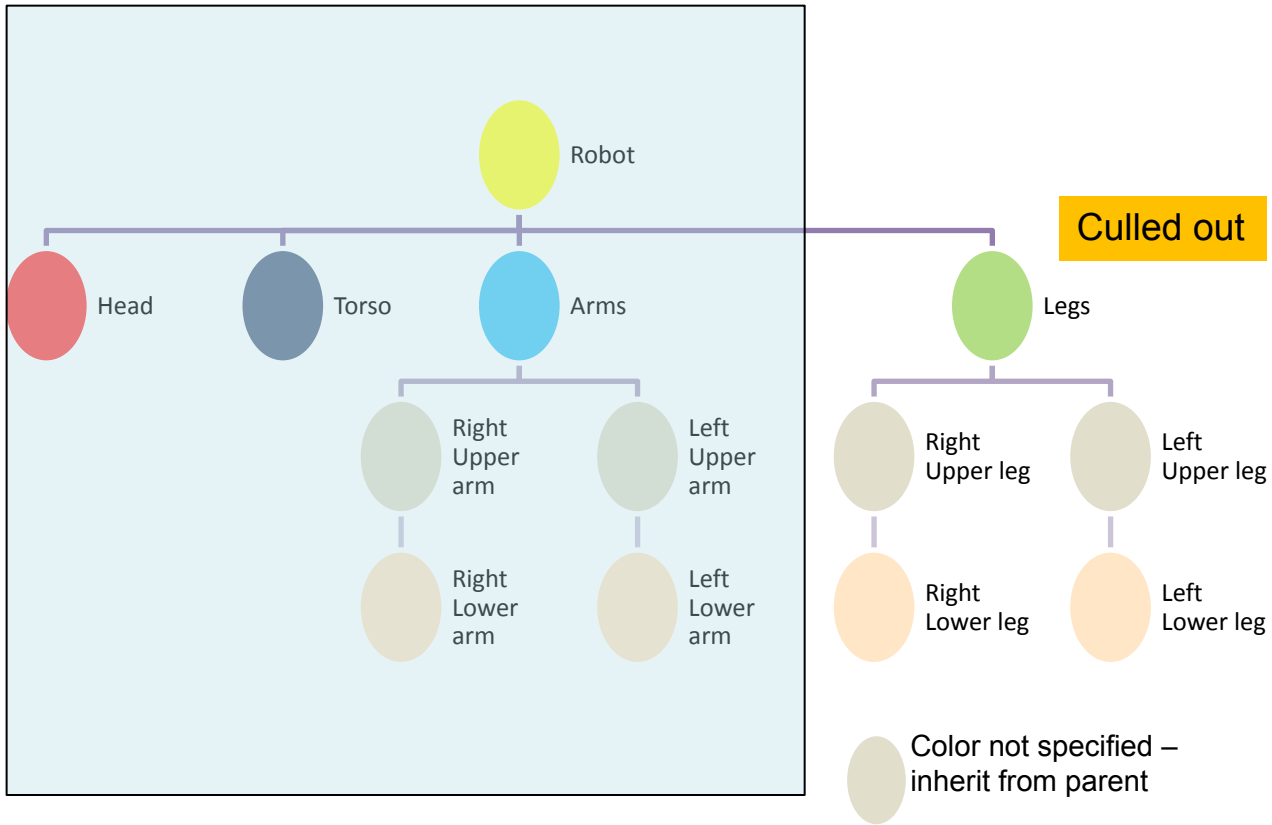


Culling

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

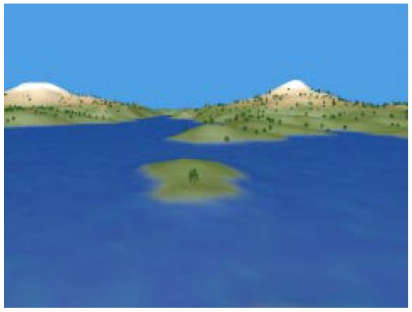
Culling out the other objects early in the render process can help improve performance, especially load on the graphics hardware.



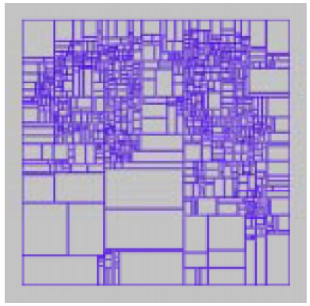


Efficient Rendering for Walkthroughs

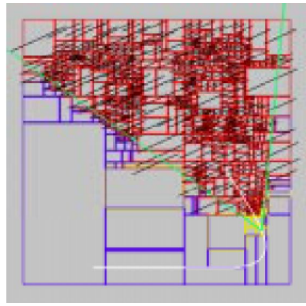
- How can we organize the data to support real-time rendering for walkthroughs?
- Preprocess complex scenes into hierarchical space partitions
- Allows for efficient computation of visibility sets and/or culling



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

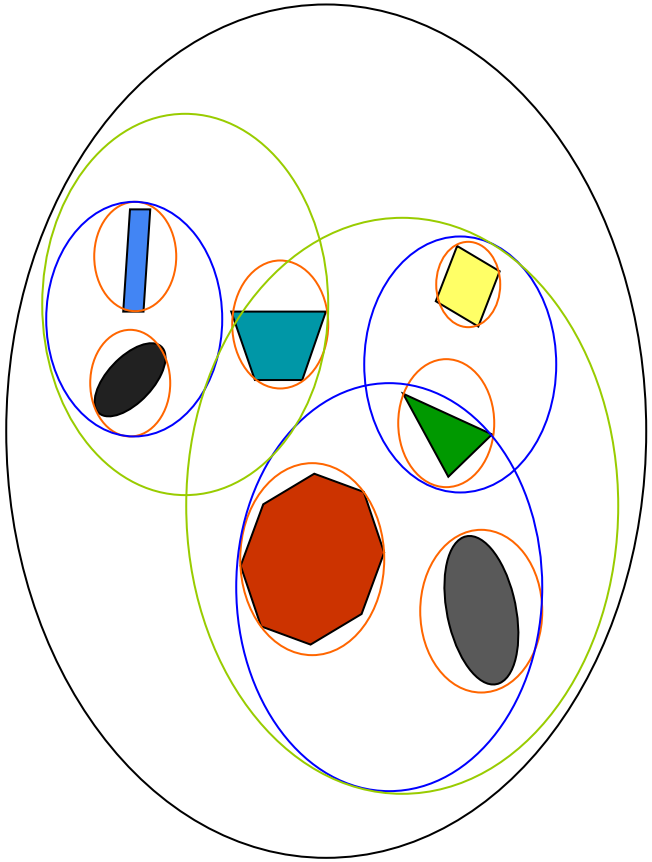
Object-based Hierarchies

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

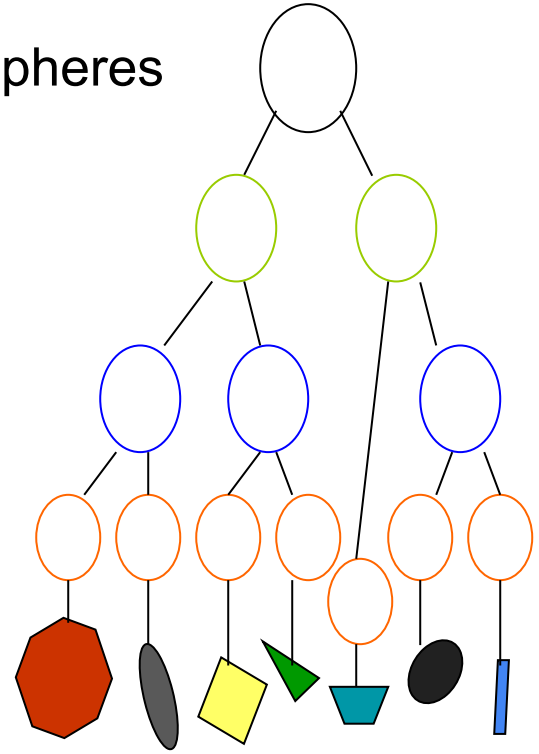
- Define regular **convex** regions that tightly bound objects of interest.
- Build up tree by grouping subtrees and computing bounding convex regions for the tree.

- 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

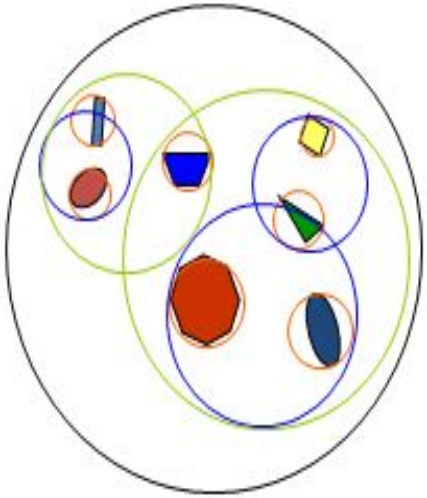
- Provides a tight cover of containing objects
- Bounds preserved through translate/rotate/scale
- Requires computation of distances for collision tests

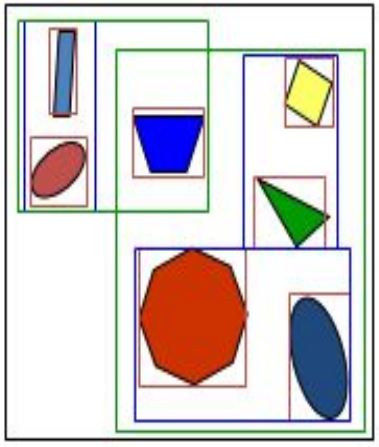
Bounding Boxes (axis-aligned)

- 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

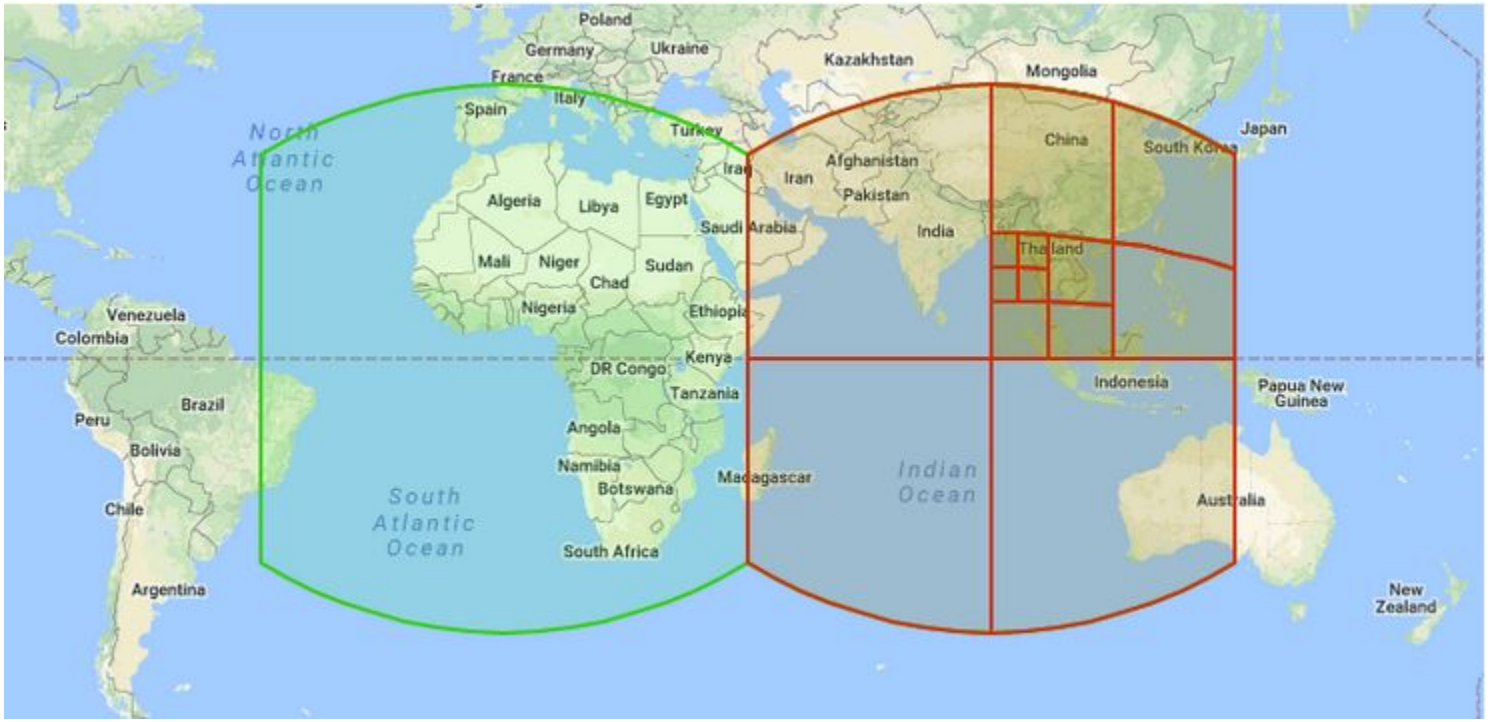
(Not strictly part of Graphics or this course, but may be of interest to some of you).

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

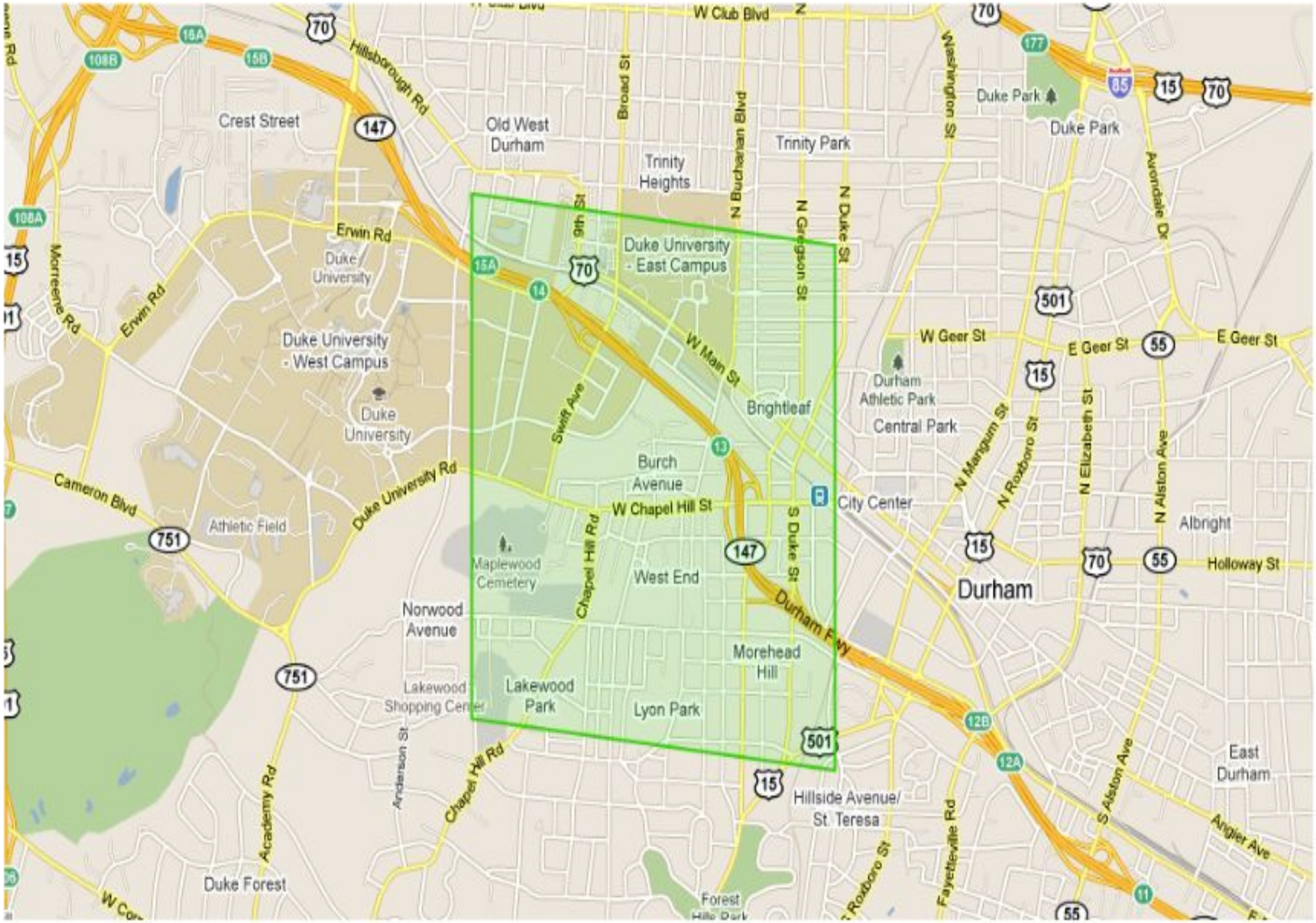
http://bit-player.org/extras/hilbert/hilbert-mapping.html


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
Washington St
Duke Park
Erwin Rd
9th St
Duke University - East Campus
N Gregson St
N Duke St
Duke University
Duke University - West Campus
Morreene Rd
W Main St
W Geer St
E Geer St
Durham Athletic Park
Central Park
Brightleaf
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
Durham Fwy
N Mangum St
N Roxboro St
N Elizabeth St
N Alston Ave
Albright
Holloway St
Avondale Dr
Norwood Avenue
Morehead Hill
Lakewood Shopping Center
Lakewood Park
Lyon Park
Anderson St
Hillside Avenue/ St. Teresa
East Durham
S Alston Ave
Angier Ave
Academy Rd
Duke Forest
Fayetteville Rd

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.