CSE528 Computer Graphics: Theory, Algorithms, and Applications

Hong Qin

Rm. 151, NEW CS Building

Department of Computer Science

Stony Brook University (SUNY at Stony Brook)

Stony Brook, New York 11794-2424

Tel: (631)632-8450; Fax: (631)632-8334

qin@cs.stonybrook.edu; or qin@cs.sunysb.edu;

http://www.cs.stonybrook.edu/~qin

Geometry Representations

- What is geometric modeling
  - Representation of existing objects (mathematical tools to represent shape geometry of real-world objects, both natural and manufactured ones)
  - Reverse engineering (from physical prototypes to digital prototypes)
  - Design of new objects (shape editing, deformation, manipulation)
  - Rendering - leading to visual interpretation
- Application of geometric modeling
  - Graphics, CAD, CAGD, CAM/CAE, robotics, vision, virtual reality, scientific visualization, animation, physical simulation, computer games, etc.

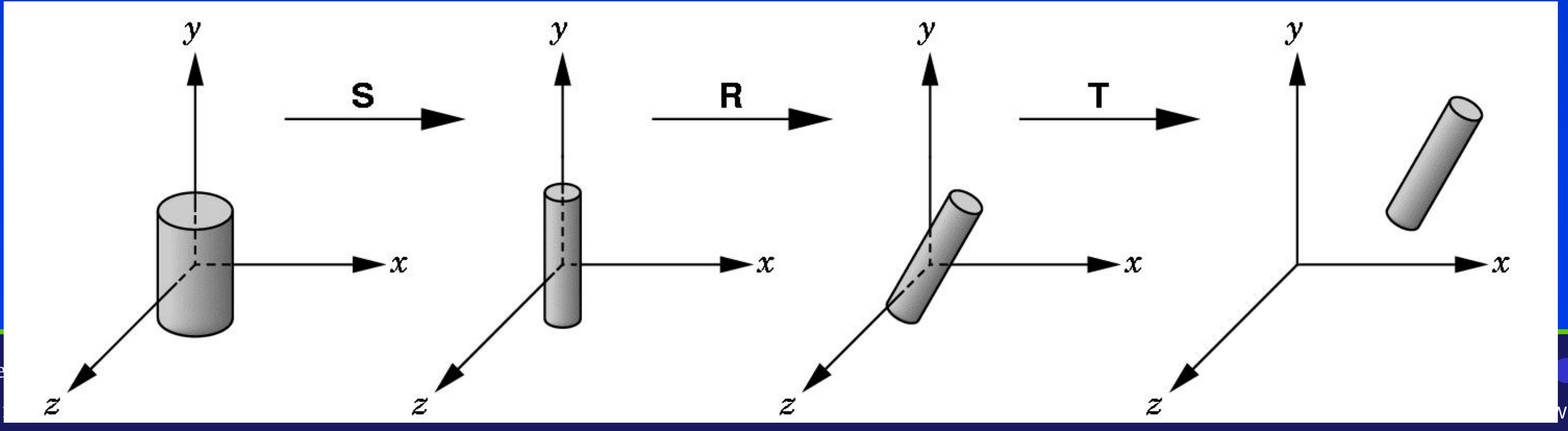
Hierarchical Models

Now we are learning and examining the geometric modeling techniques from the data structure's perspective

Representing Objects in Graphics

- Objects represented as symbols
- Defined in model coordinates; transformed into world coordinates (M = TRS)

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity(); glTranslatef(…);
glRotatef(…); glScalef(…);
glutSolidCylinder(…);
```



Geometry Representations

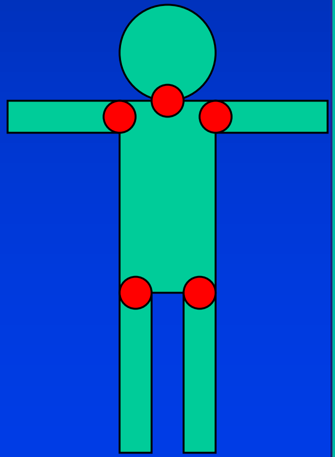
- Modeling primitives
  - Polygon
  - Sphere, ellipsoid, torus, superquadrics
  - NURBS, surfaces of revolutions, smoothed polygons
  - Particles
  - Skin & bones
- Approaches to modeling complex shapes
  - Tools such as extrude, revolve, loft, split, stitch, blend
  - Constructive solid geometry (CSG)
  - Hierarchy; kinematic joints
  - Inverse kinematics
  - Keyframes

Overview

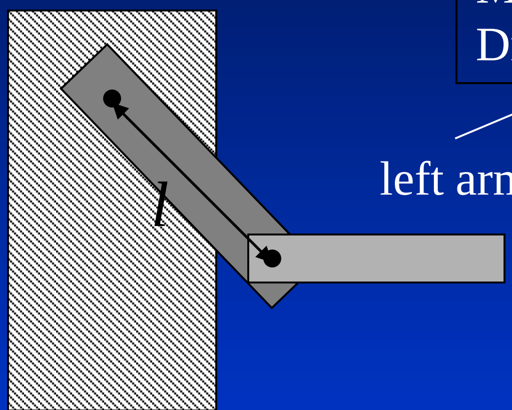
- Data structures for interactive graphics
  - CSG-tree
  - BSP-tree
  - Quadtrees and Octrees
- Modeling
- Computer animation

Hierarchical Modeling

- Hierarchical model: a group of parts (including meshes) are related by a tree (or graph) structure
  - Properties of children are derived from their parents
  - Most useful for animating articulated objects (human figures, low-life animals, robots, etc.)
- Consider a walking (humanoid, classic) robot:
  - How would you move the robot around?
  - Does the entire robot move in the same way?
  - Does the position of one part of the robot depend on other parts?



Two-link Robot Example



Move body
Draw body

left arm

Rotate about shoulder
Draw upper arm

Translate *(l,0,0)*
Rotate about origin of lower arm
Draw lower arm

Important issues:

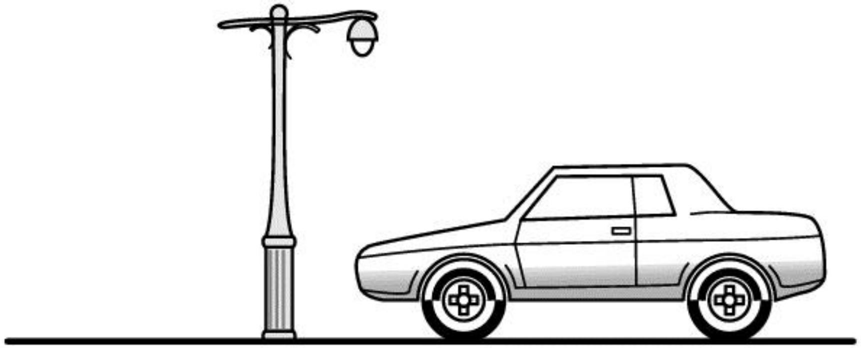
- Every node has its own local coordinate system.
- This makes specifying transformations much easier.
- What are we assuming about the "upper arm" coordinate system?

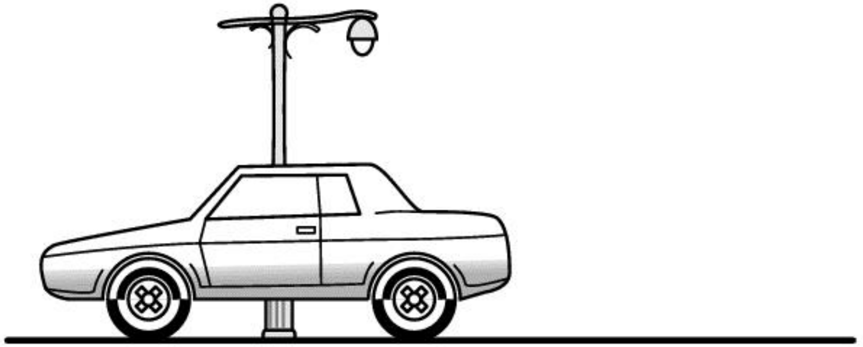
Hierarchical Models

- Generally represented as a tree, with transformations and instances at any node
  - Can use a general graph, but resolving inheritance conflicts is a problem
- Rendered by traversing the tree, applying the transformations, and rendering the instances
- Particularly useful for animation
  - Human is a hierarchy of body, head, upper arm, lower arm, etc…
  - Animate by changing the transformations at the nodes
- Other things can be inherited (colors, surface properties, etc.)

Hierarchical Models

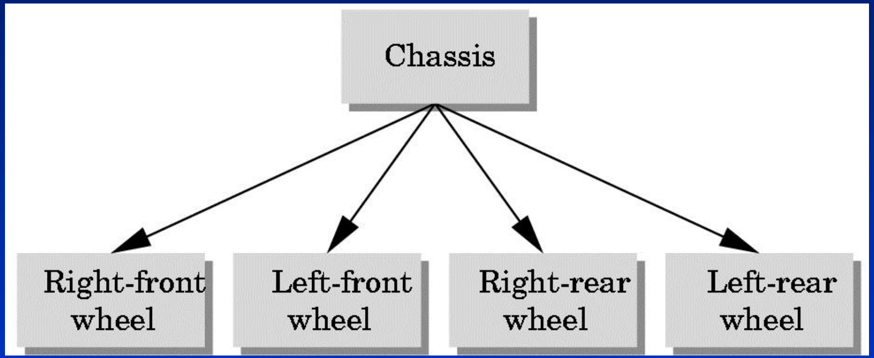
- When animation is desired, objects may have parts that move with respect to each other
  - Object represented as hierarchy
  - Often there are joints with motion constraints
  - For example, represent wheels of car as sub-objects with rotational motion (car moves **2 pi r** per rotation)

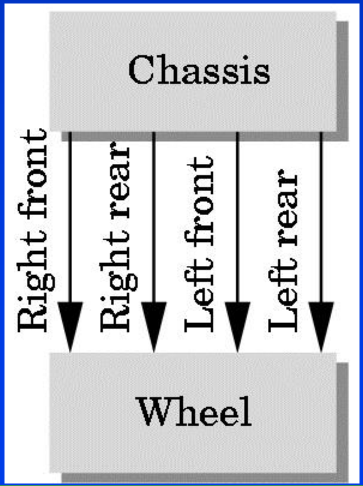




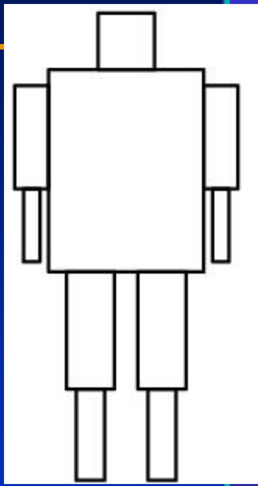
Directed Acyclic Graph (DAG) Models

- Could use tree to represent object
- Actually, a DAG (directed acyclic graph) is better: can re-use objects
- Note that each arrow needs a separate modeling transform
- In object-oriented graphics, also need motion constraints with each arrow





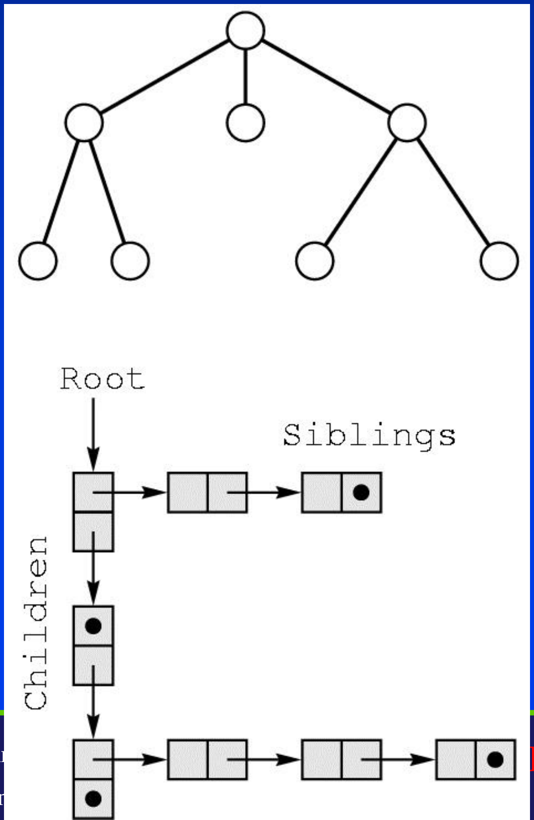
Robot

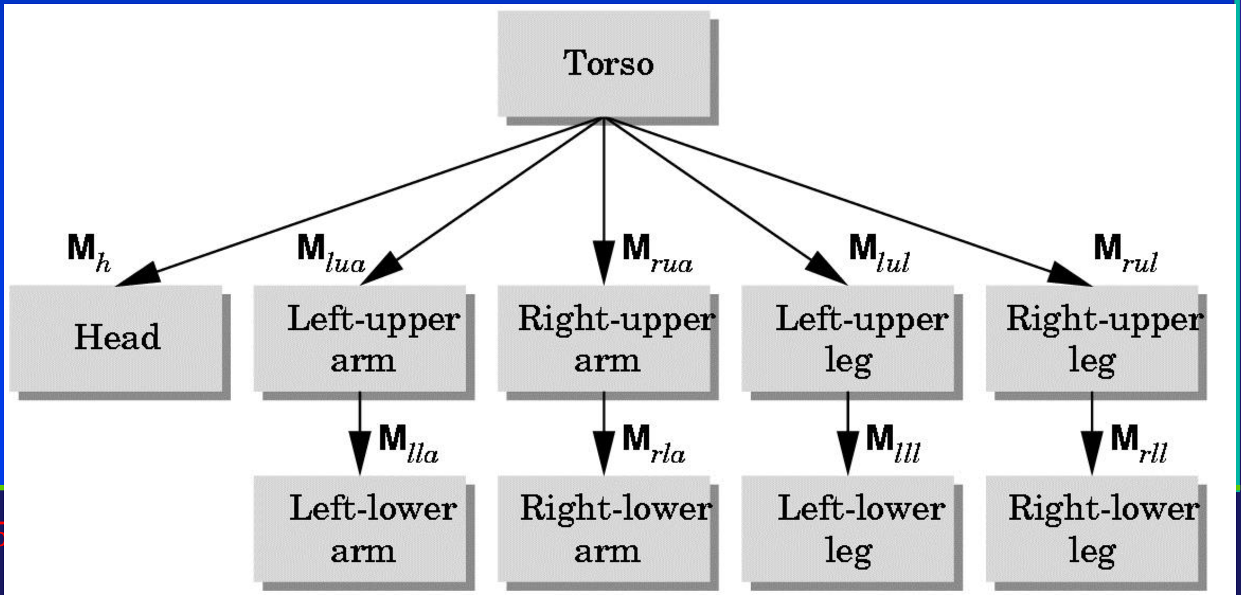


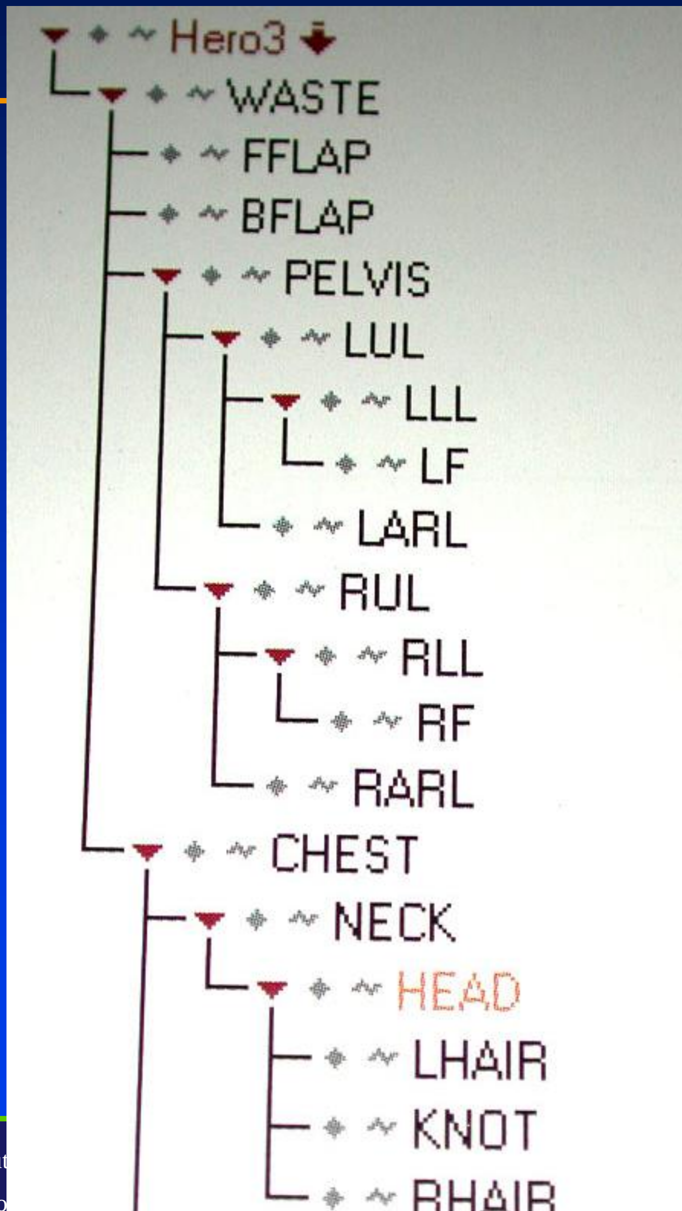
- Traverse DAG using DFS (or BFS)
- Push and pop matrices along the way

(e.g., left-child right-sibling)

(joint position parameters?)

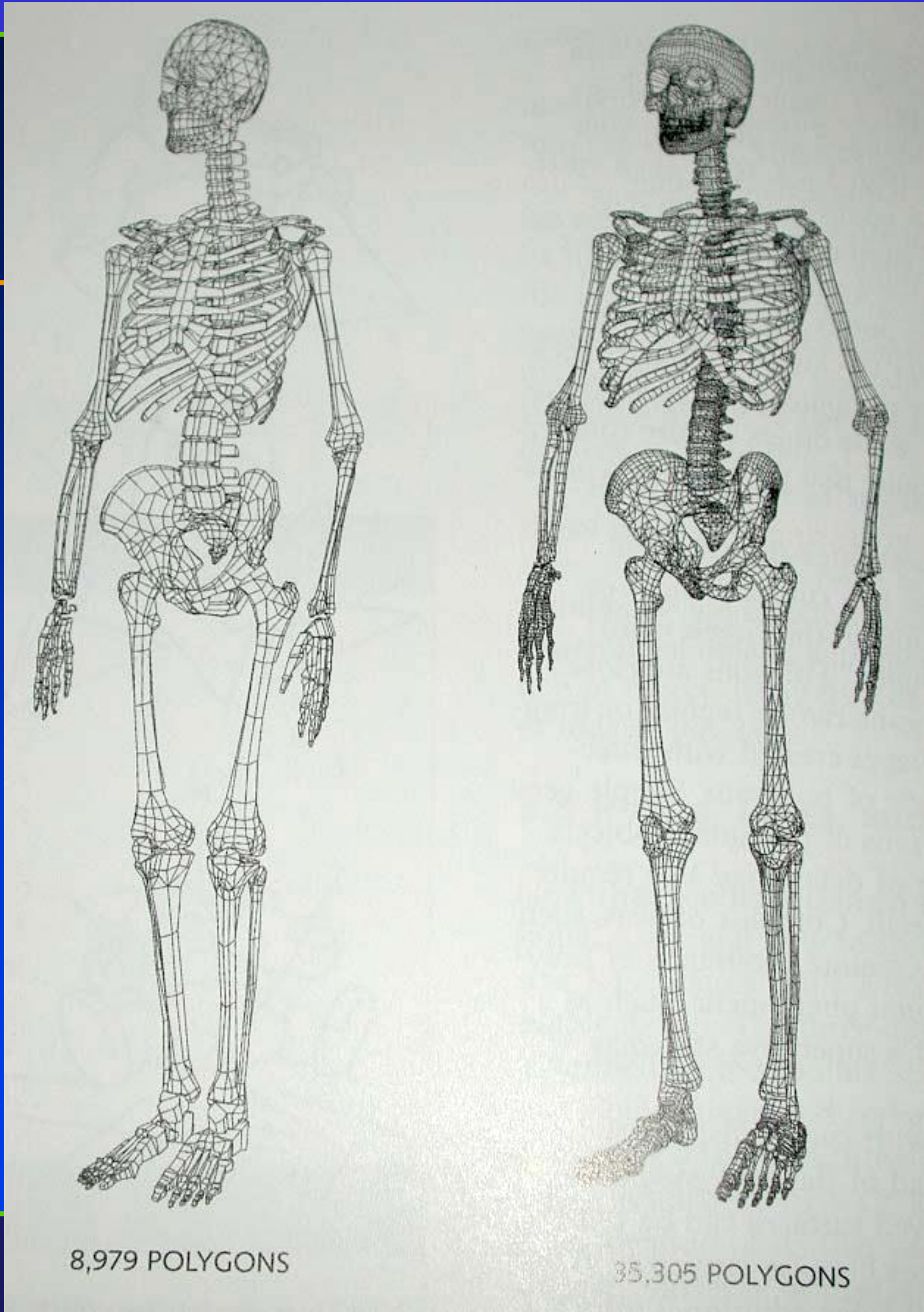





Hero3
WASTE
FFLAP
BFLAP
PELVIS
LUL
LLL
LF
LARL
RUL
RLL
RF
RARL
CHEST
NECK
HEAD
LHAIR
KNOT
RHAIR

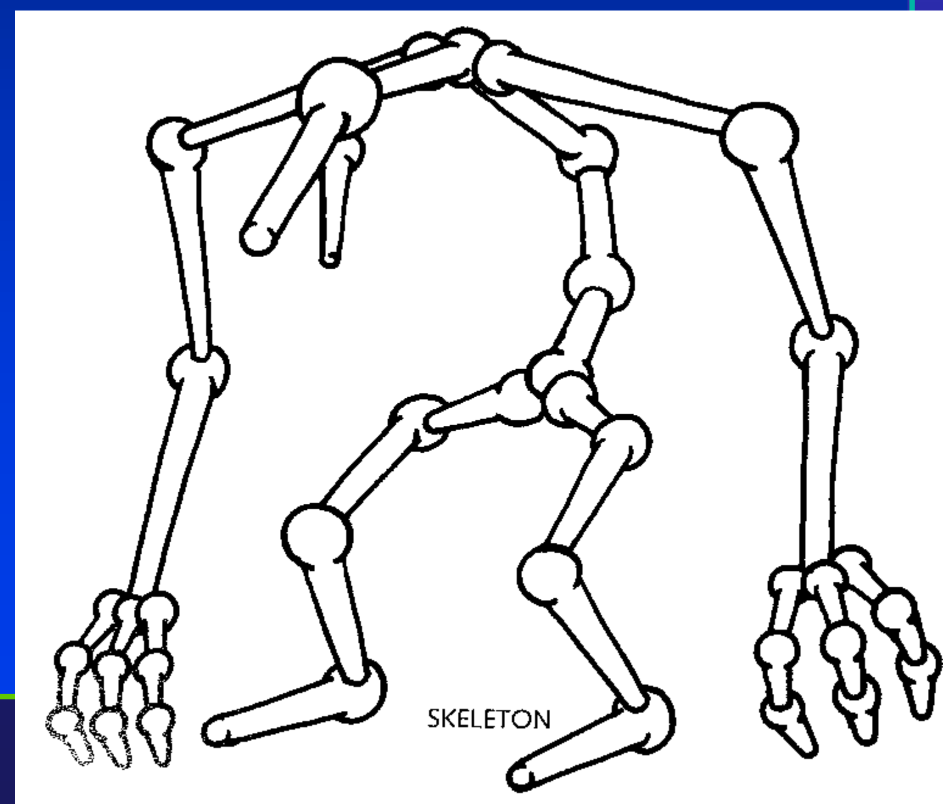
Primitives

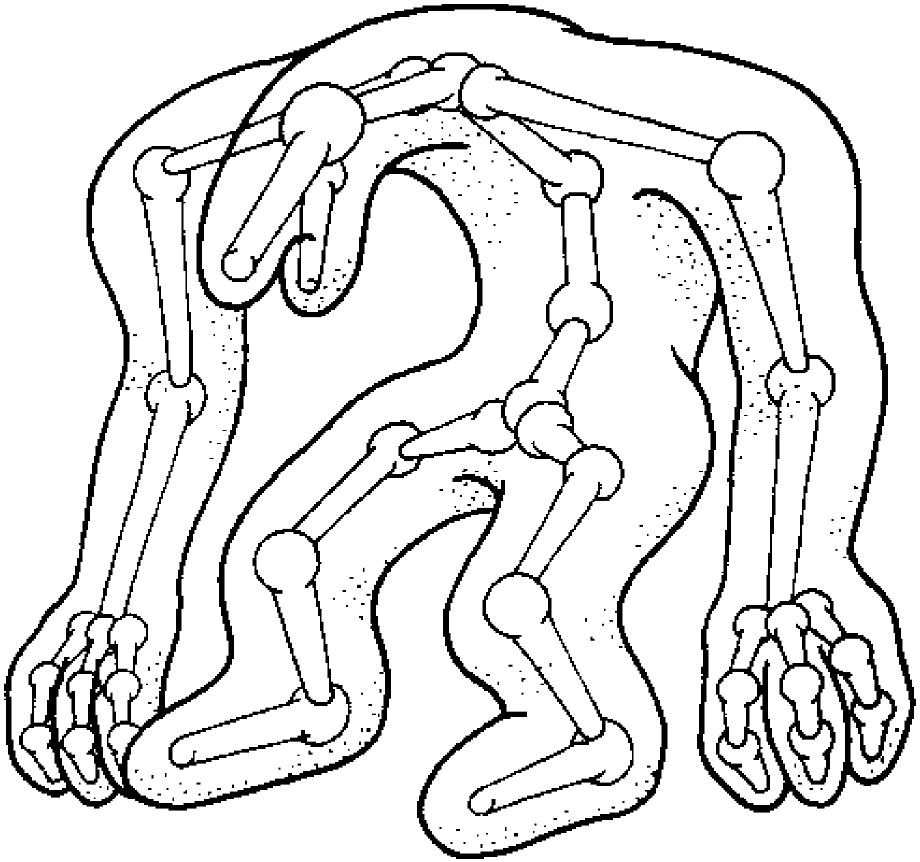
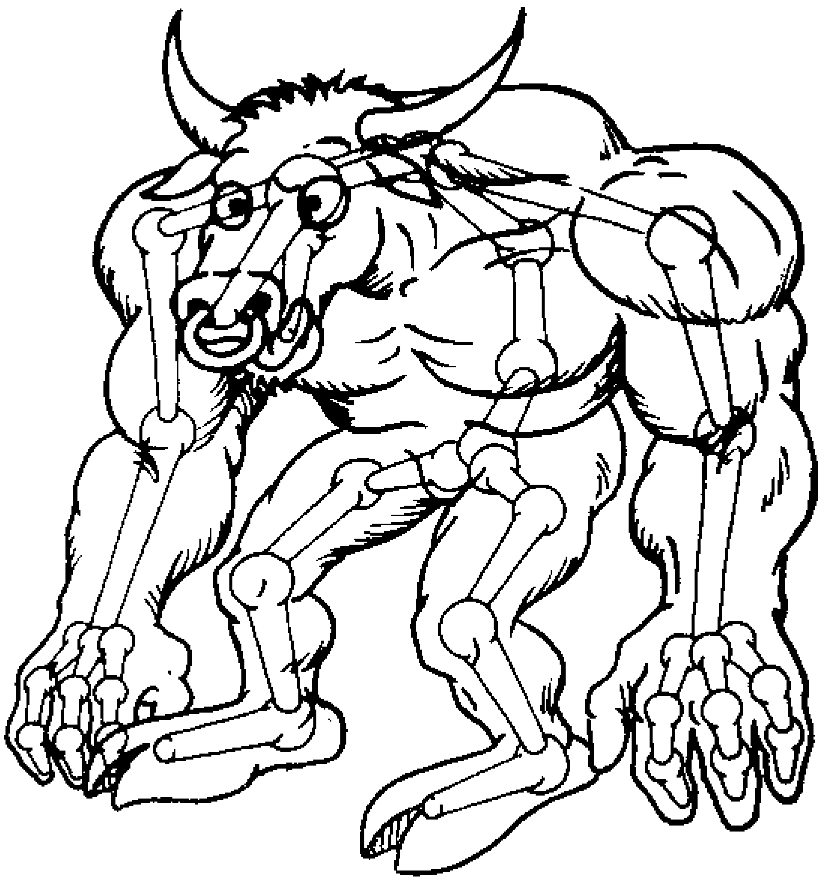
- The basic type of primitive is the polygon
- Number of polygons: tradeoff between render time and model accuracy

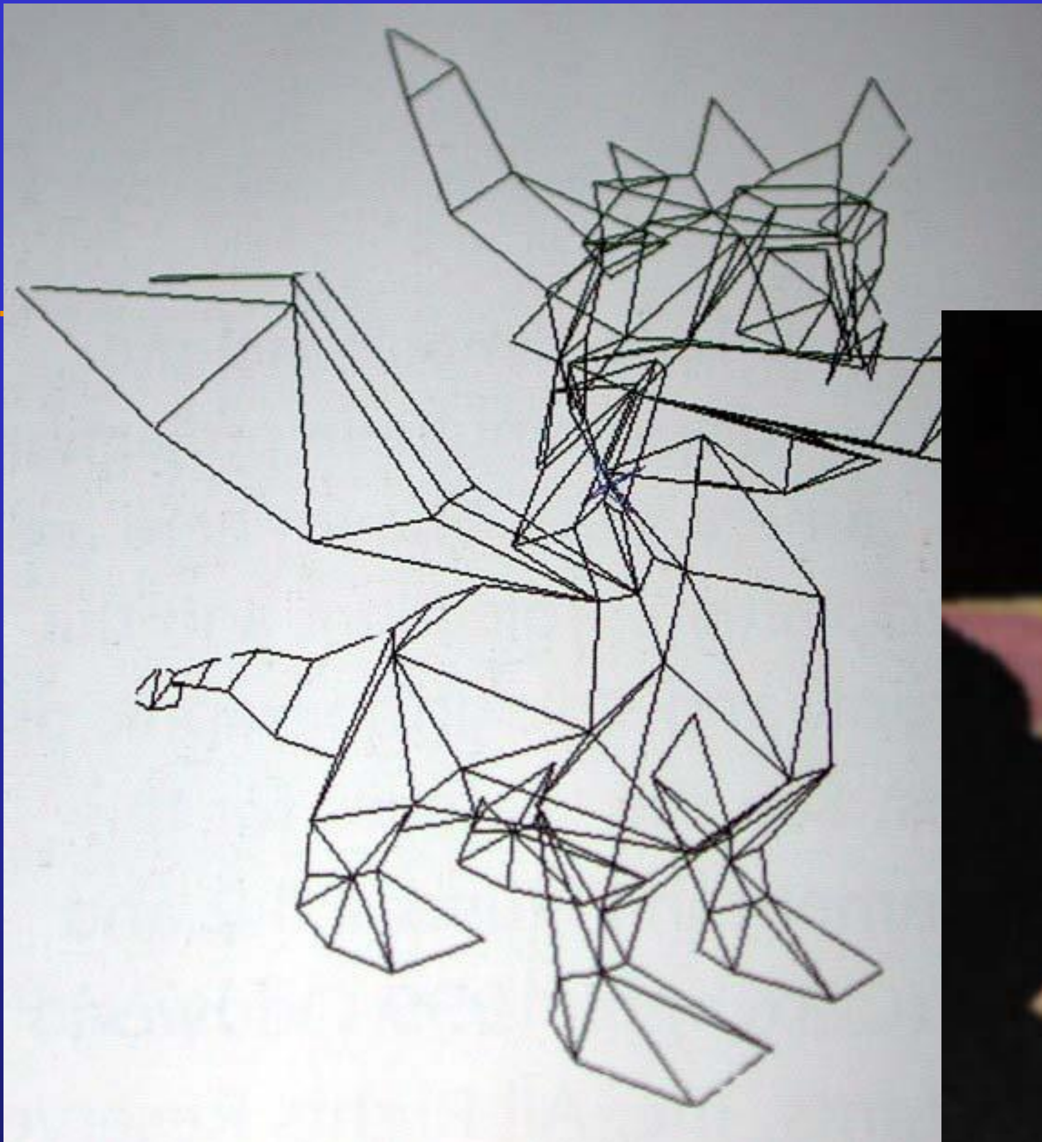


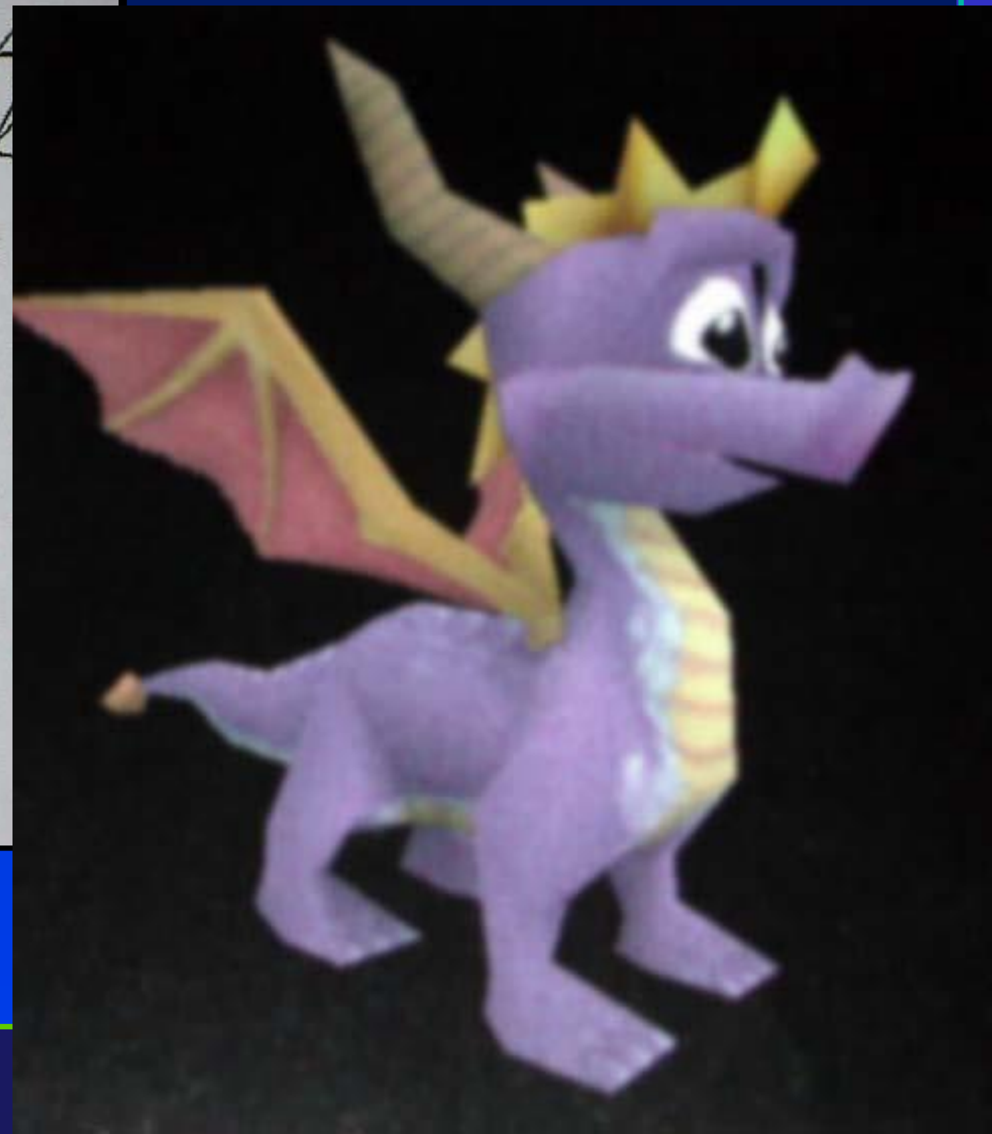
Bones and Skin

- Skeleton with joined "bones"
- Can add "skin" on top of bones
- Automatic or hand-tuned skinning



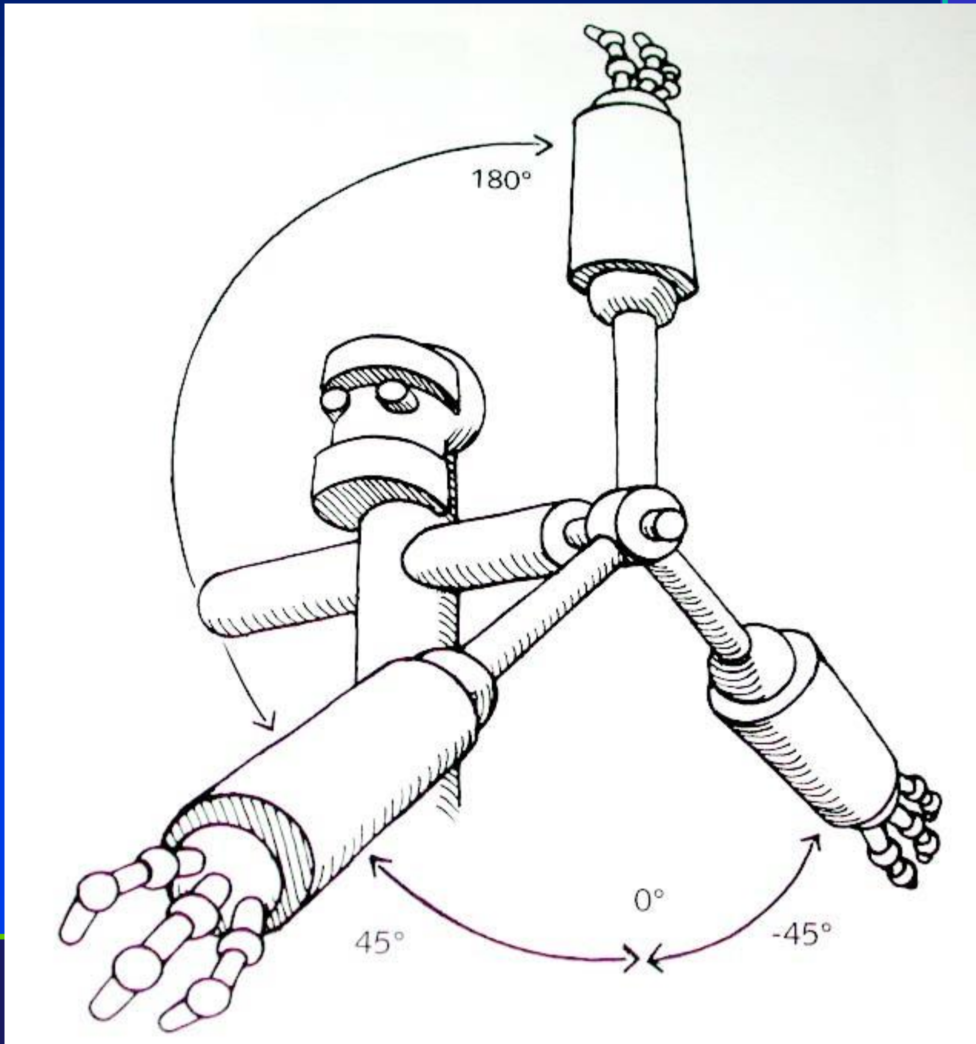







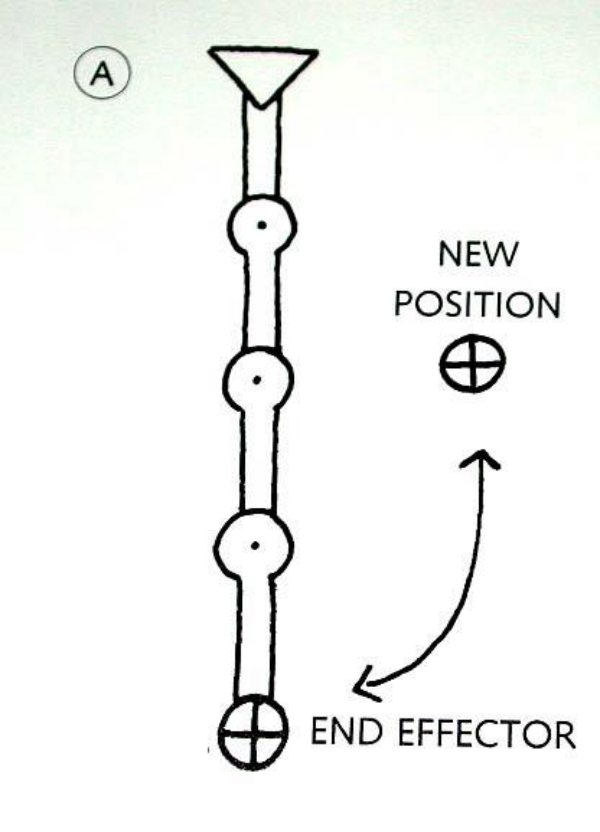
Animation

- Suppose you want the robot to pick up a can of oil to drink, how?
- You could set the joint positions at each moment in the animation (**kinematics**)



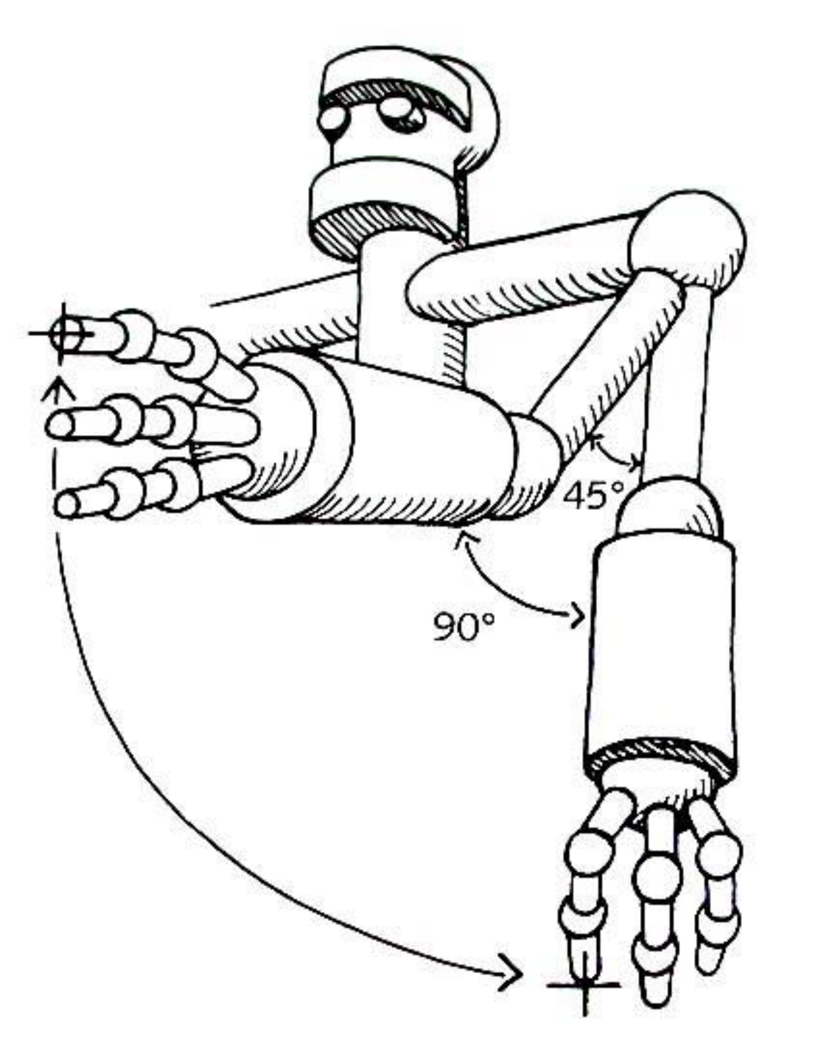
Inverse Kinematics

- You can't just invert the joint transformations
- Joint settings aren't even necessarily unique for a hand position!
- **Inverse kinematics**: figure out from the hand position where the joints should be set.



Using Inverse Kinematics

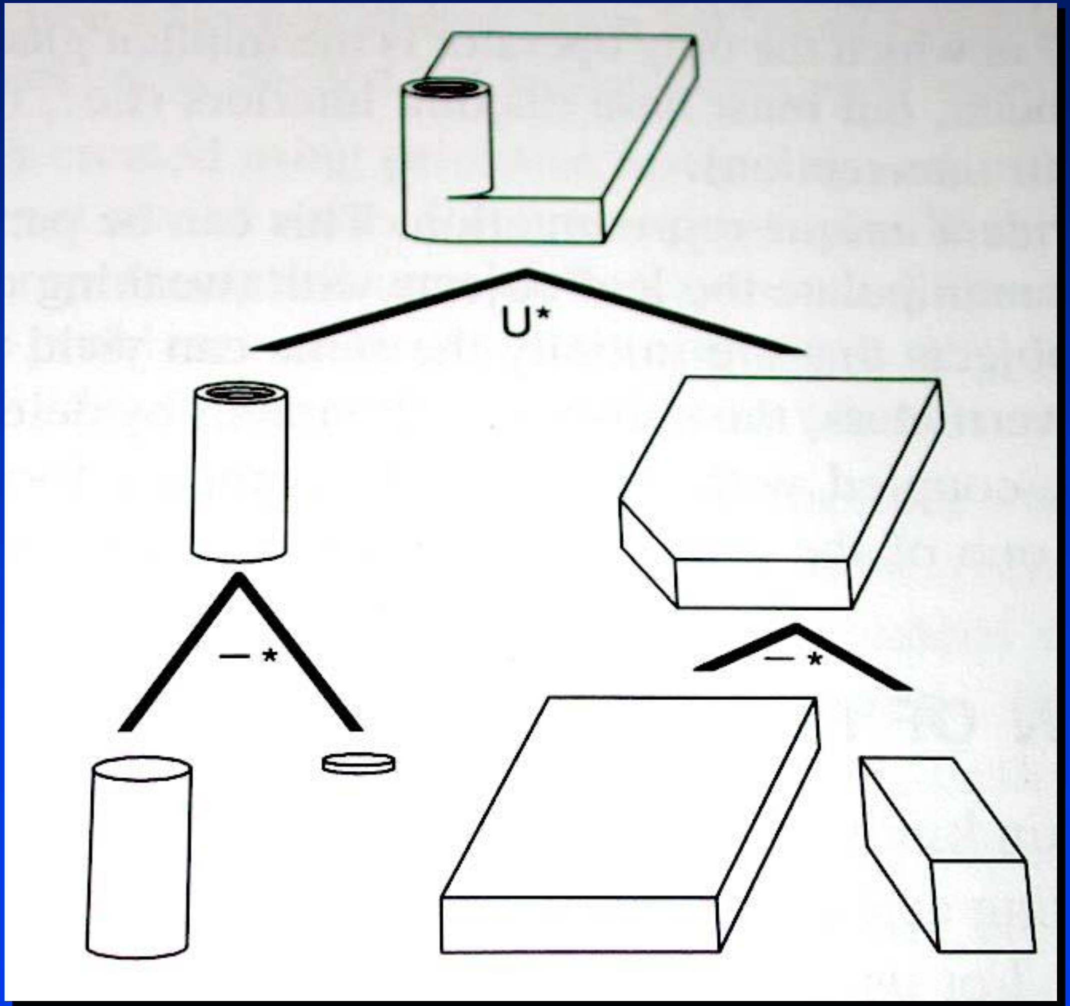
- Specify joint constraints and priorities
- Move end effector (or object pose)
- Let the system figure out joint positions



Data Structure for Modeling

- Data structure for geometry representations
- How to represent complex objects made up of union, intersection, difference of other objects
- Spatial data structure
- Tree-based decomposition of space

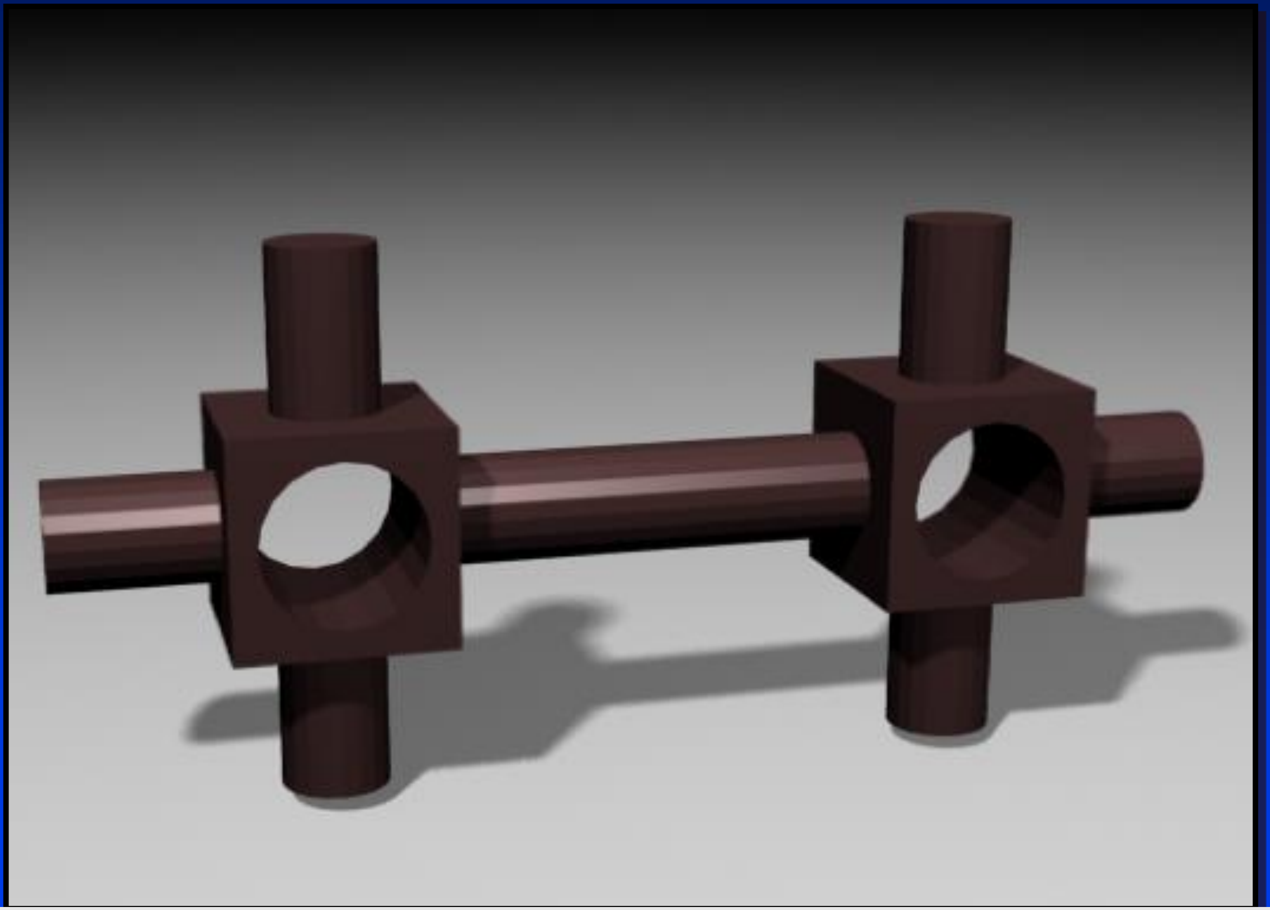
CSG Tree



Constructive Solid Geometry (CSG)

- Based on a tree structure, like hierarchical modeling, but now:
  - The internal nodes are set operations: union, intersection or difference (sometimes complement)
  - The edges of the tree have transformations associated with them
  - The leaves contain only geometry
- Allows complex shapes with only a few primitives
  - Common primitives are cylinders, cubes, etc, or quadric surfaces
- Motivated by computer aided design and manufacture
  - *Difference* is like drilling or milling
  - A common format in CAD products

Constructive Solid Geometry (CSG)



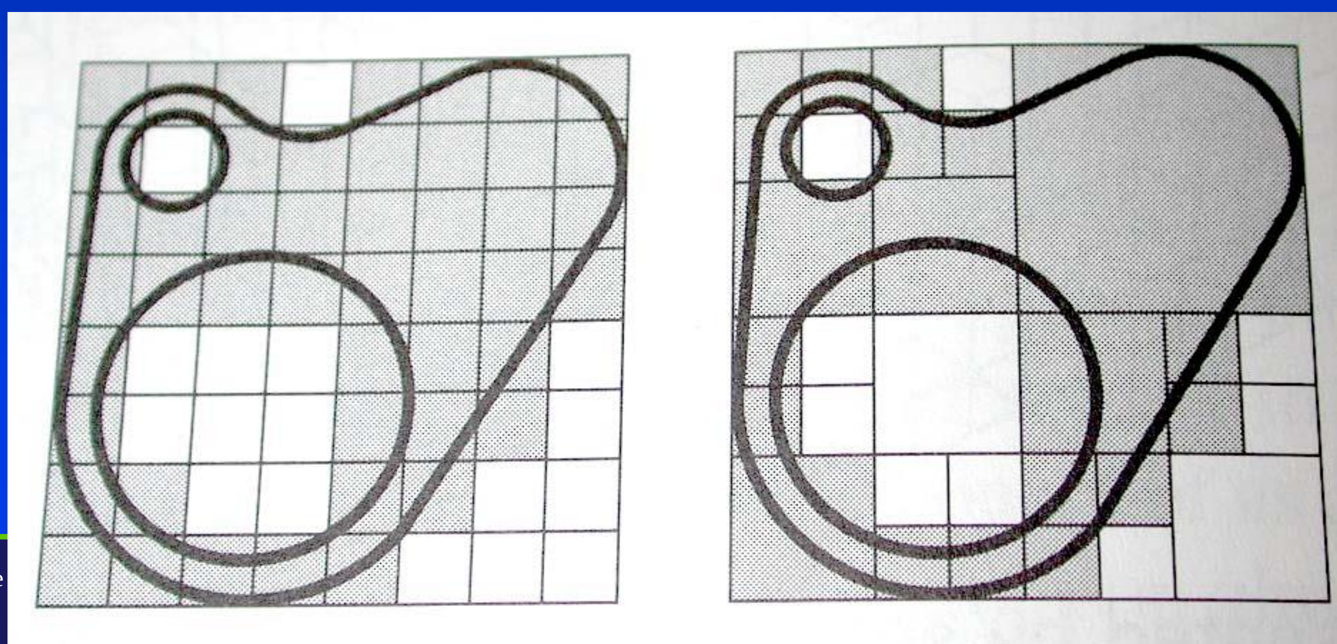
Object made by CSG Converted to polygons

Quadtrees and Octrees

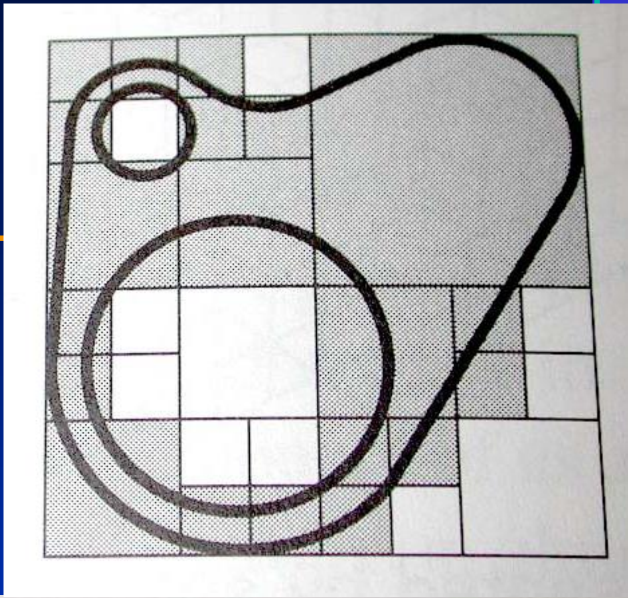
- Build a tree where successive levels represent better resolution (smaller voxels)
- Large uniform spaces result in shallow trees
- Quadtree is for 2D (four children for each node)
- Octree is for 3D (eight children for each node)

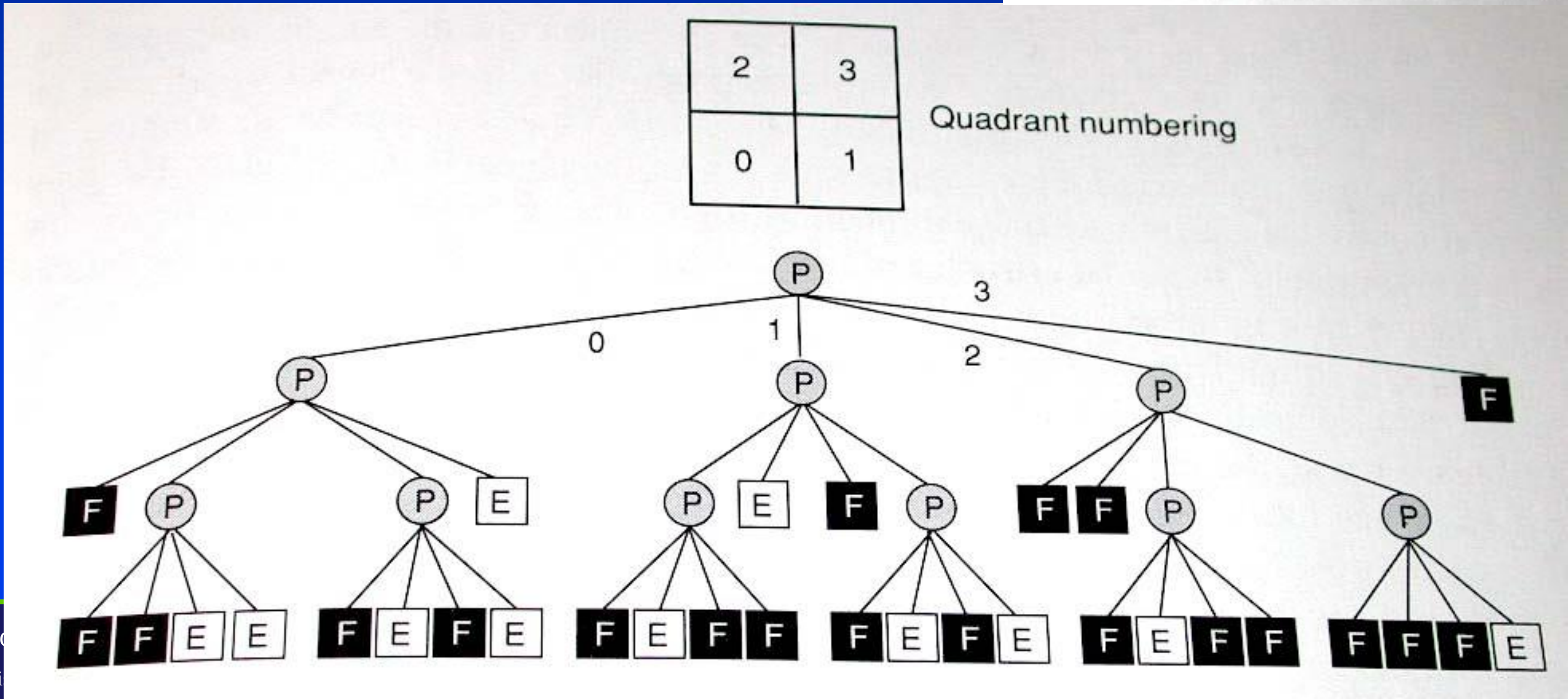
Quadtree

- Quadtree: divide space into four quadrants. Mark as Empty, Full, or Partially full.
- Recursively subdivide partially full regions
- Saves much time, space over 2D pixel data!

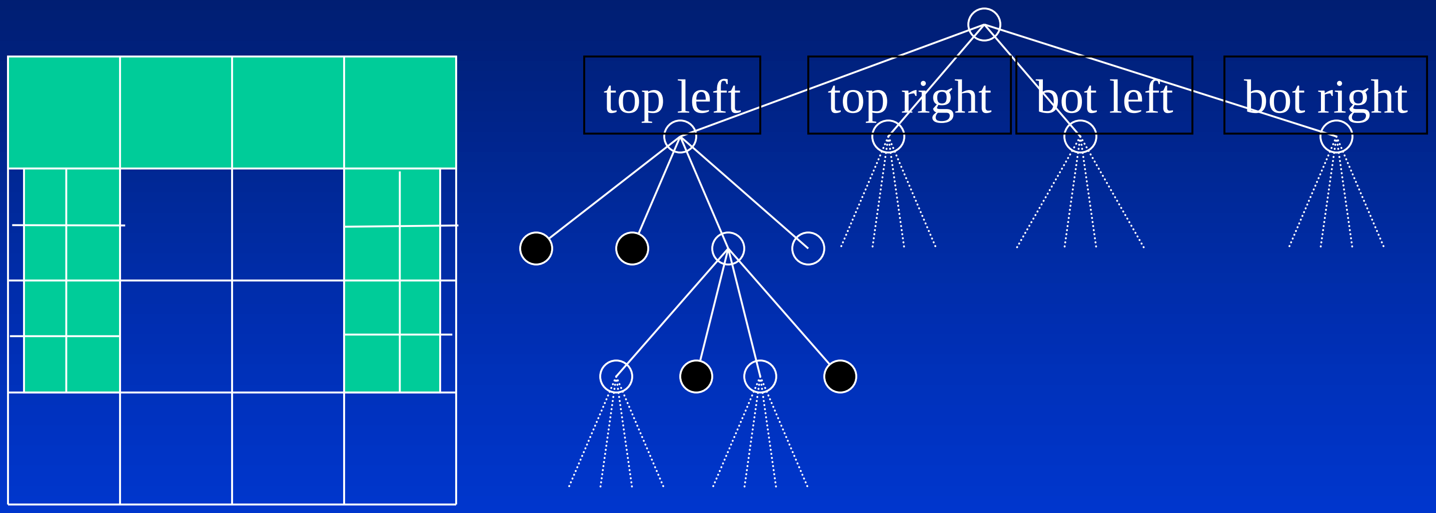


Quadtree Structure





Quadtree Example



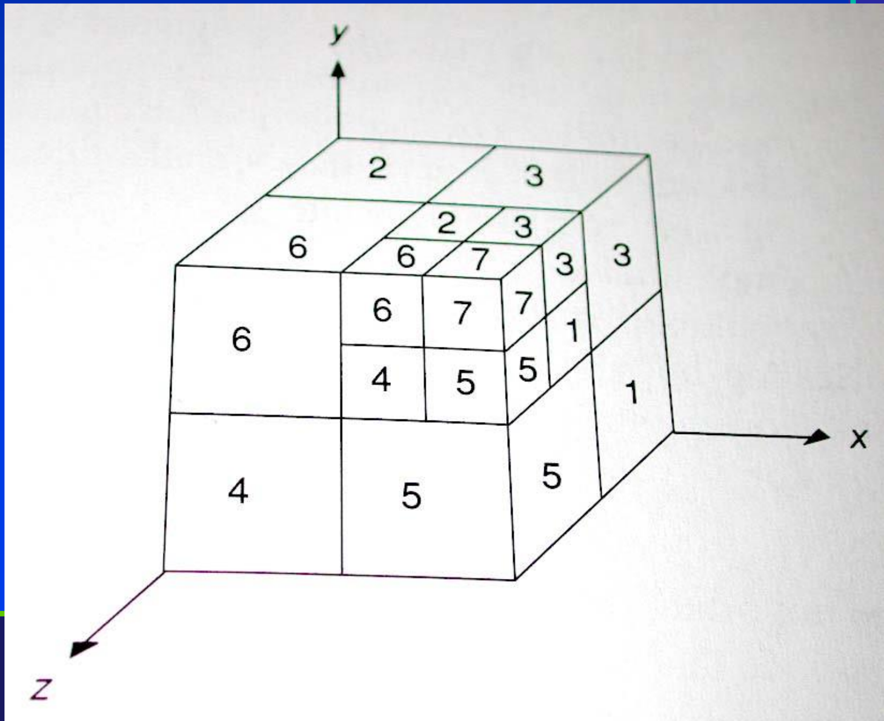
Octree principle is the same, but there are 8 children

Quadtree Algorithms

- How would you
  - render a quadtree shape?
  - find the intersection of a ray with a quadtree shape?
  - Take the union of two quadtrees?
  - Intersection?
  - Find the neighbors of a cell?

Octrees

- Generalize to cut up a cube into 8 sub-cubes, each of which may be E, F, or P (and subdivided)
- Much more efficient than a 3D array of cells for 3D volumetric data

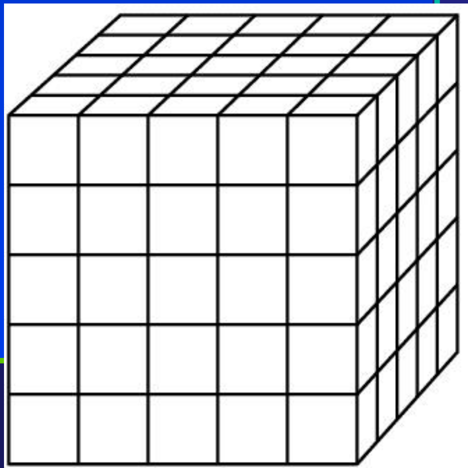


Spatial Data Structure

- Beyond graphics spatial representations
- Octree also serves as a spatial data structure itself – specifically designed for storing spatial information
- Frequently used to store information about where polygons, or other primitives, are located in a scene
- Speeds up many computations by making it fast to determine when something is relevant or not (another example is BSP-tree which speeds up visibility test)
- Other spatial data structures include BSP trees, KD-Trees, Interval trees, ….

Handling Large-scale Spatial Datasets

- Example application: image-based rendering
  - Suppose you have many digital images of a scene, with depth information for pixels
  - How to find efficiently the points that are in front?
- Other applications:
  - Speeding up ray-tracing with many objects
  - Rendering contours of 3D volumetric data such as MRI scans



Spatial Enumeration

- Basic idea: describe something using space it occupies, break the volume of interest into lots of tiny cubes, and use cubes inside the object to represent (approximate) the object
- Works well for medical data (e.g., MRI or CAT scans)
- Enumerates the volume - data is associated with each voxel (volume element)
  - Problems: for anything other than small volumes or low resolutions, the number of voxels explodes
  - Note that the number of voxels grows with the *cube* of linear dimension

Rendering Octrees

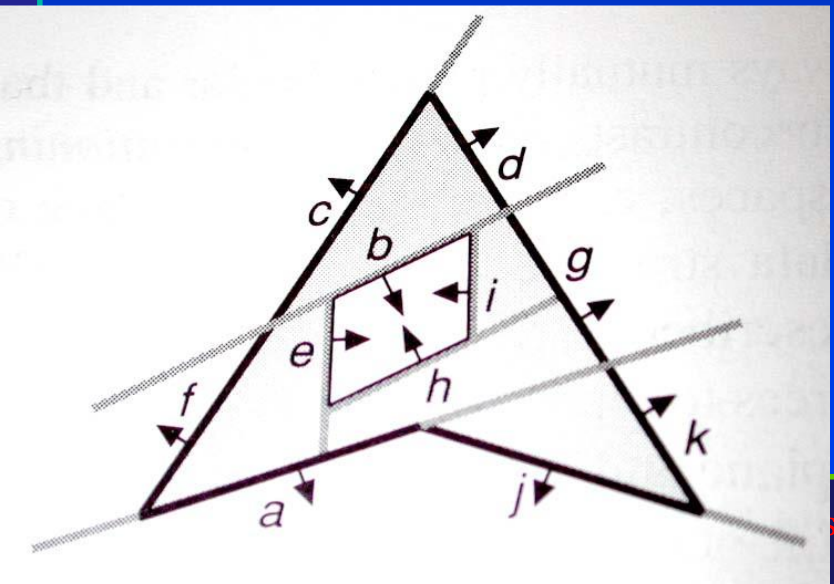
- Relying on volume rendering techniques to handle octrees and associated data directly
- Converting to polygons by a few methods:
  - Just take faces of voxels that are on the boundary
  - Find iso-surfaces within the volume and render those
  - Typically do some interpolation (smoothing) to get rid of the artifacts from the voxelization
- Typically render with colors that indicate something about the data, but other methods exist

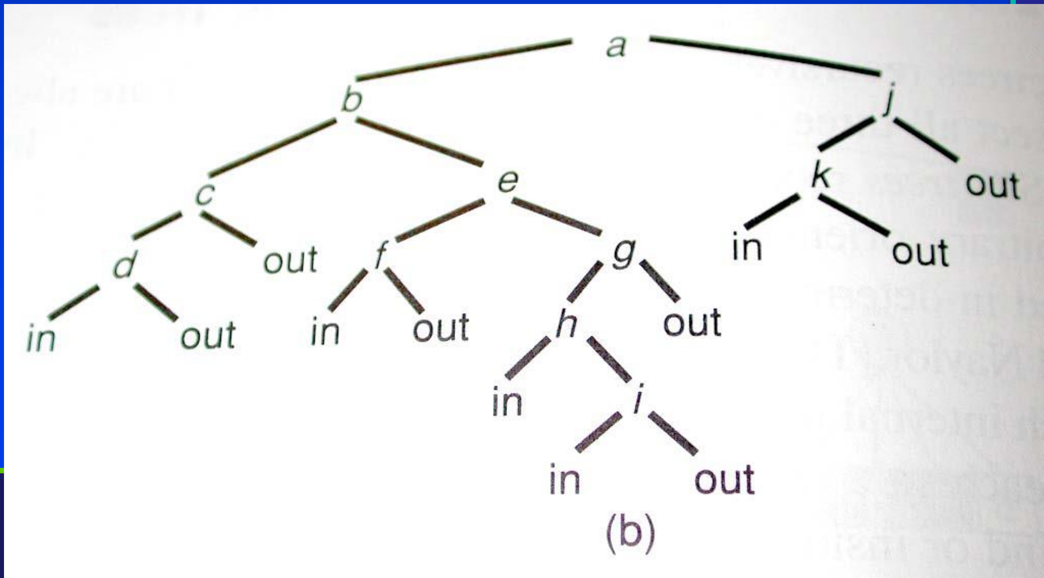
Applications of Octrees

- Contour finding in MRI data
- 3D scanning and rendering
- Efficient ray tracing
- Intersection, collision testing

BSP Tree for Shape Modeling

- Right is “front” of polygon; left is “back”
- In and Out nodes show regions of space inside or outside the object
- (Or, just store split pieces of polygons at leaves)





Building a BSP Tree

- Inserting a polygon:
  - If tree is empty make it the root
  - If polygon to be inserted intersects plane of polygon of current node, split and insert half on each side recursively.
  - Else insert on appropriate side recursively
- Problem: the number of faces could grow dramatically
  - Worst case ($O(n^2)$)…but usually it doesn't grow too badly in practice…

Traversing a BSP Tree

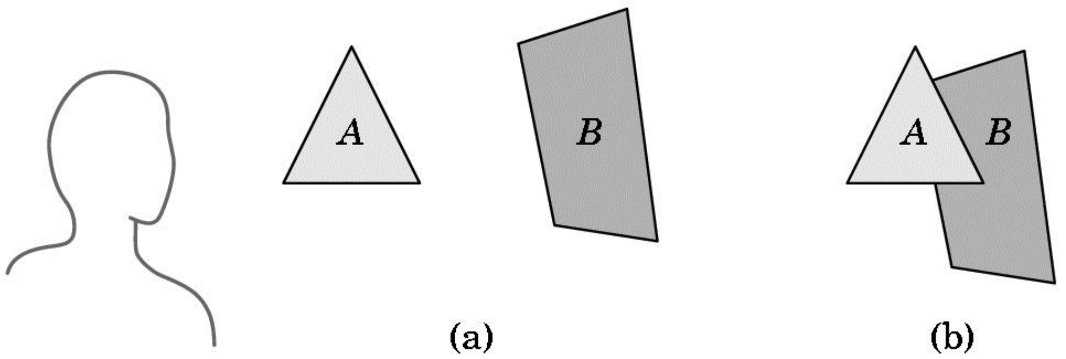
- **Binary Space Partition tree**: a binary tree with a polygon at each node
  - Children in left subtree are behind polygon
  - Children in right subtree are in front of polygon
- Traversing a BSP-tree:
  - If null pointer, do nothing
  - Else, draw far subtree, then polygon at current node, then near subtree
  - Far and near are determined by location of viewer
- Runtime of traversal?
- Drawbacks?

Hidden Surface Removal (HSR)

- How to render in 3D with hidden surface removal when you don't have a hardware depth-buffer?
- Can you think of any other ways of removing hidden surfaces *quickly*?
- *Principle: a polygon can't be occluded by another polygon that is* behind *it.*

BSP-Tree

- The *painter's algorithm* for hidden surface removal works by drawing all faces, from back to front
- How to get a listing of the faces in back-to-front order?
- Put them into a binary tree and traverse the tree (but in what order?)



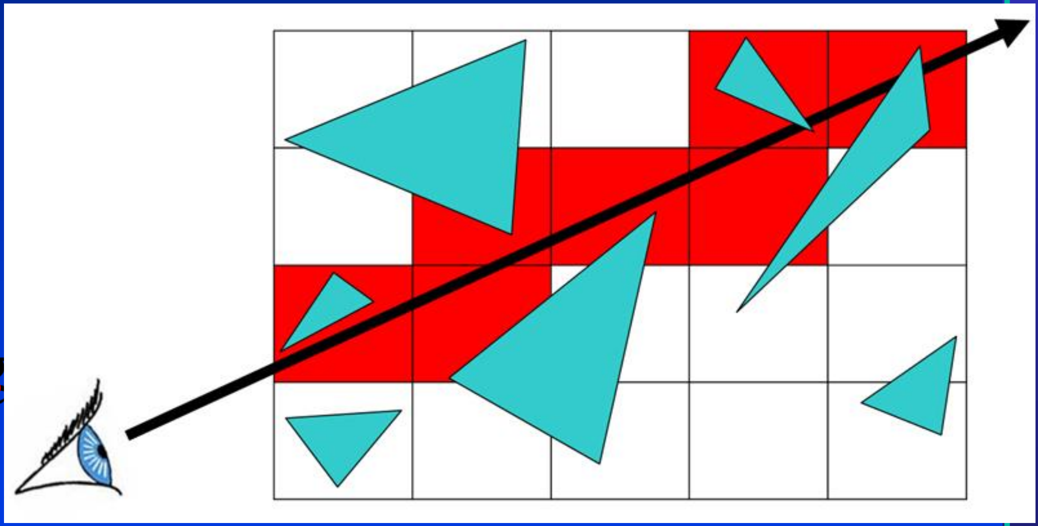
BSP-Tree Summary

- Returns polygons not necessarily in sorted order, but in an order that is correct for back-to-front rendering
- Widely used when Z-buffer hardware may not be available (e.g., game engines)
- Guarantees back-to-front rendering for alpha blending
- Works well (linear-time traversals) in the number of *split* polygons
- [And we hope the number of polygons doesn't grow too much through splitting]

Bounding Volume Hierarchy (BVH)

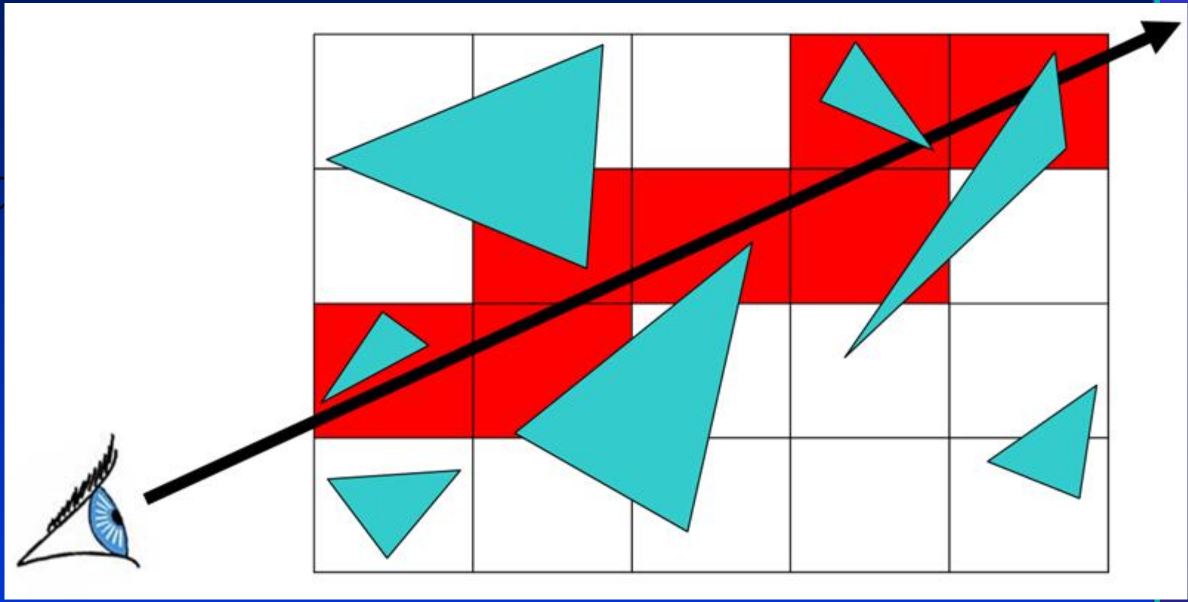
Bounding Boxes

- Bounding boxes
- Data structures for spatial acceleration
  - Regular grid
  - Adaptive grids
  - Hierarchical bounding volumes
- Flattening the transformation hierarchy



Regular Grid ONLY

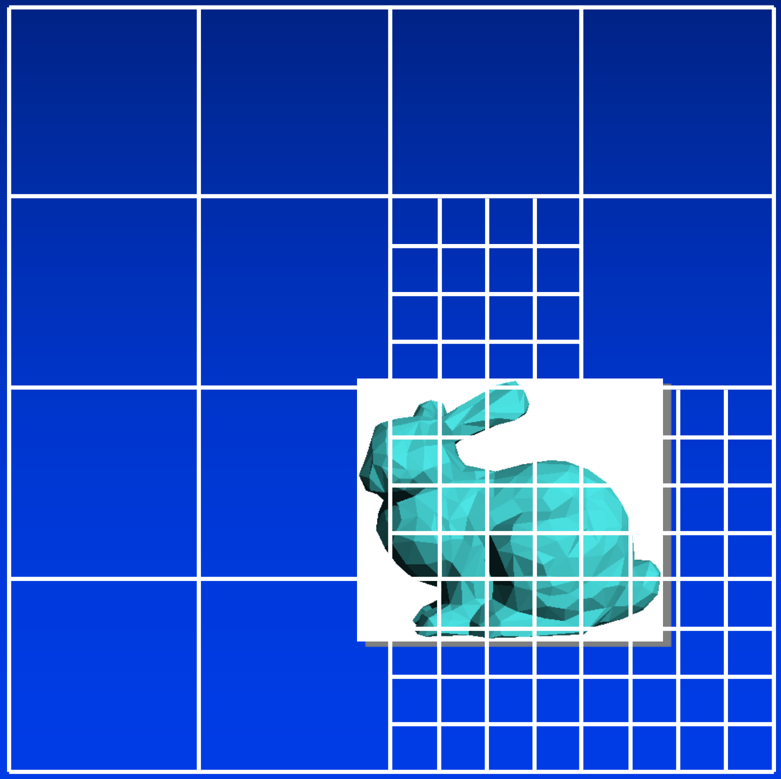
- Advantages?
  - easy to construct
  - easy to traverse



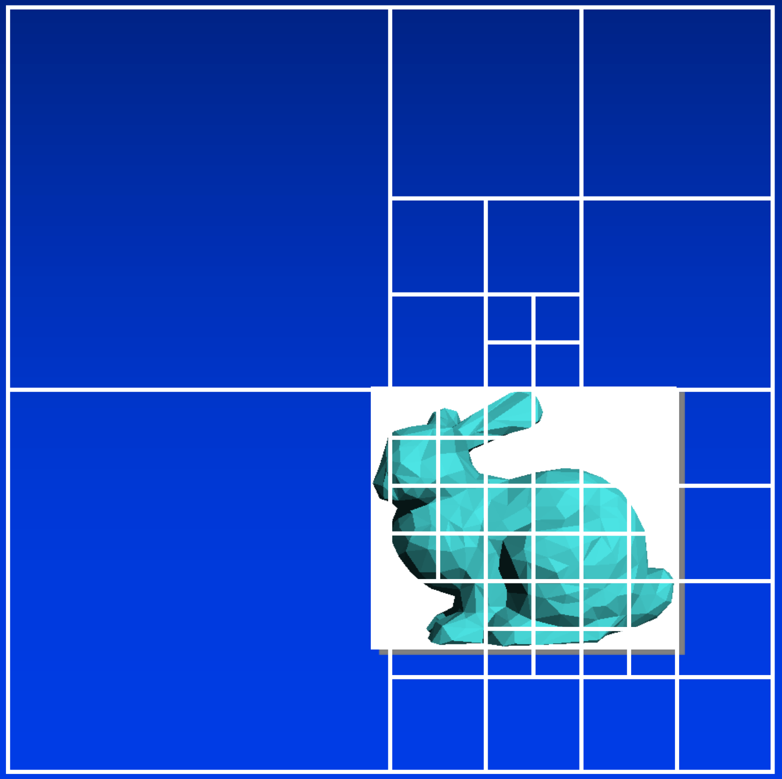
- Disadvantages?
  - may be only sparsely filled
  - geometry may still be clumped

Adaptive Grids

- Subdivide until each cell contains no more tha *n* elements, or maximum depth *d* is reached



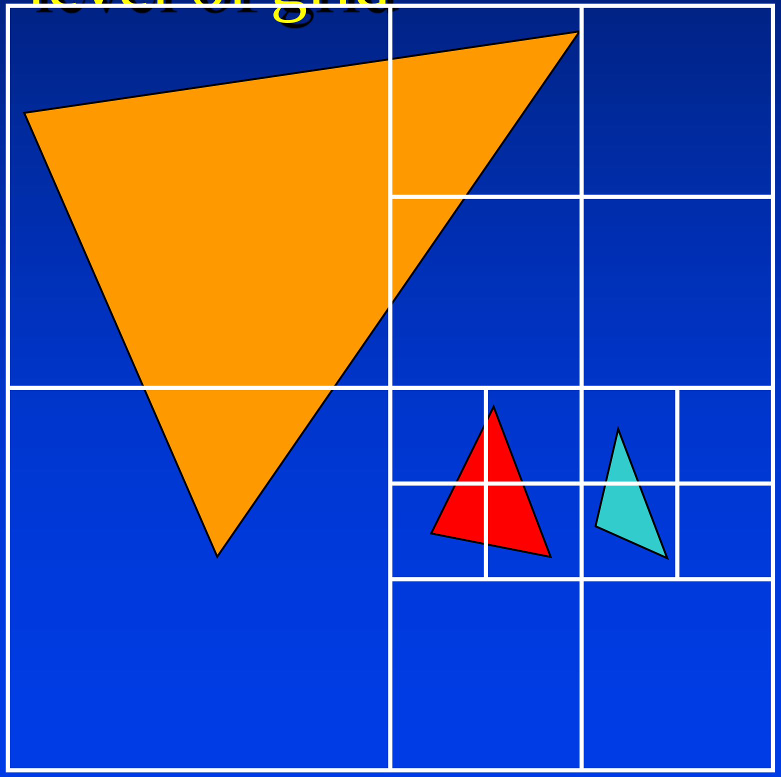
Nested Grids



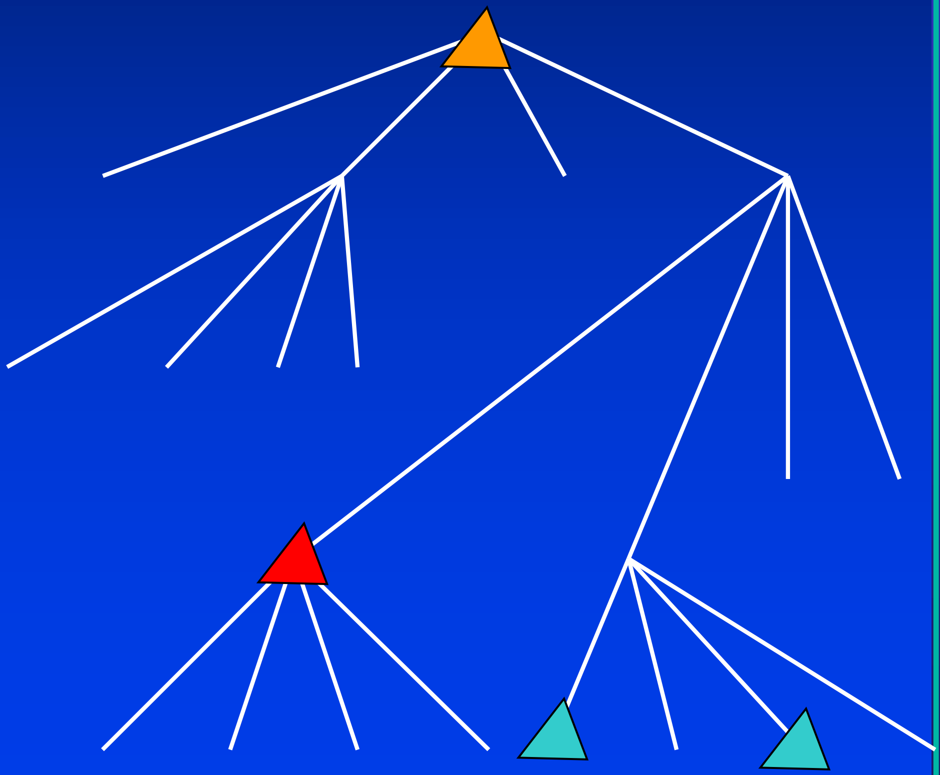
Quadtree/(Octree)

Primitives in an Adaptive Grid

- Can live at intermediate levels, or be pushed to lowest level of grid

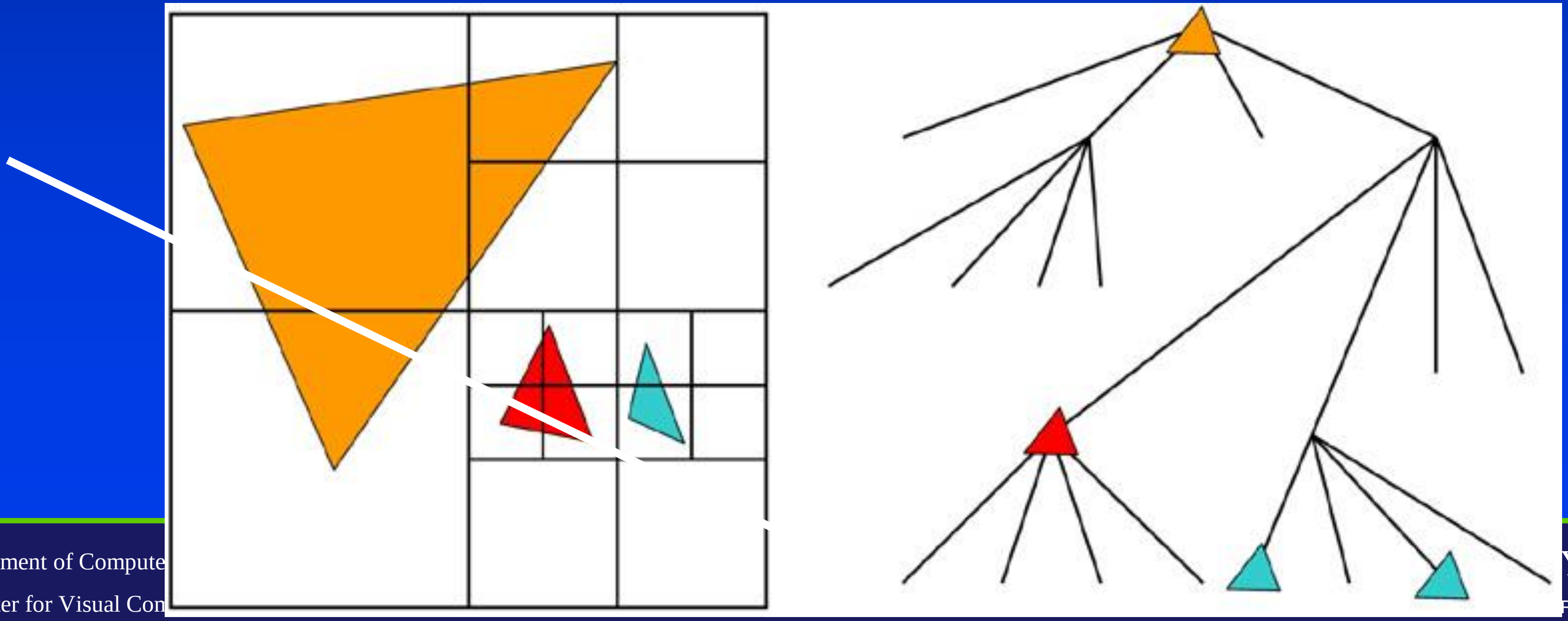


Quadtree/(Octree)



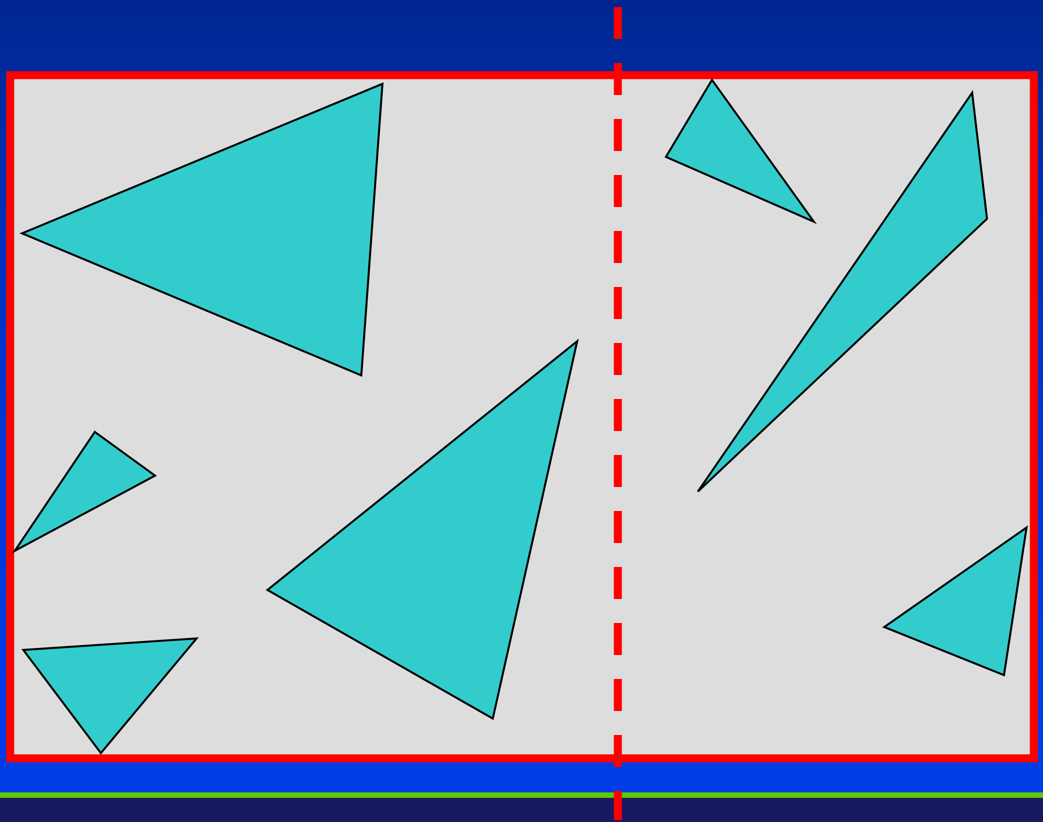
Adaptive Grid Discussion

- Advantages?
  - grid complexity matches geometric density
- Disadvantages?
  - more expensive to traverse (especially octree)



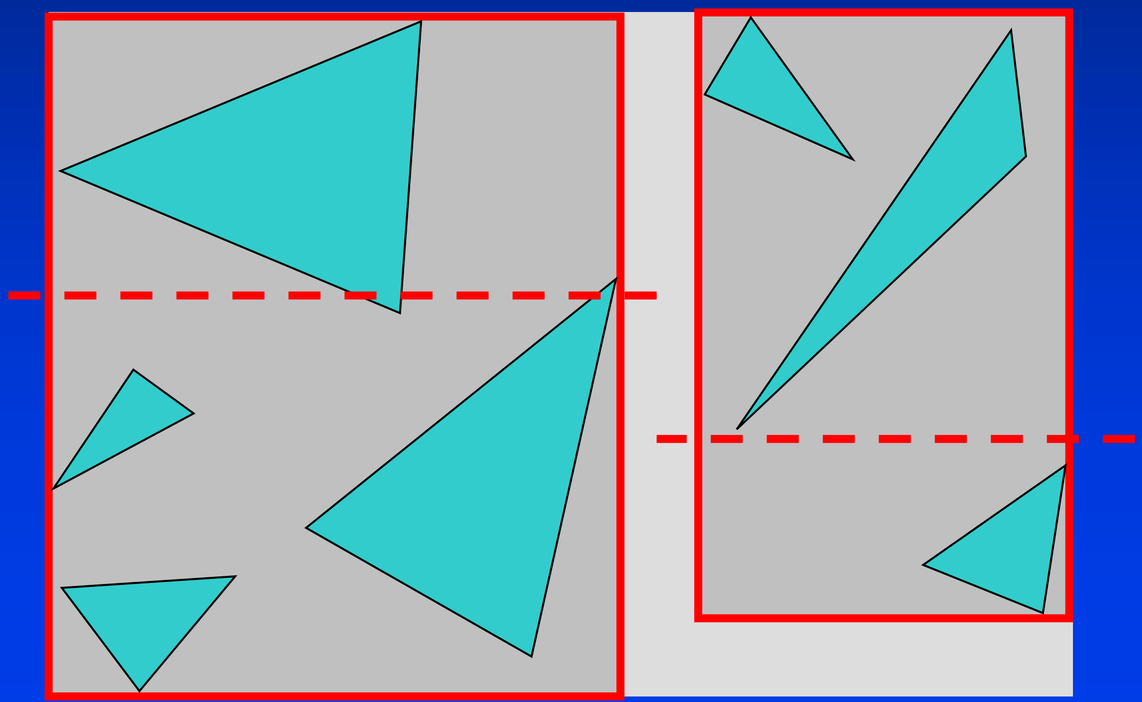
Bounding Volume Hierarchy

- Find bounding box of objects
- Split objects into two groups
- Recurse



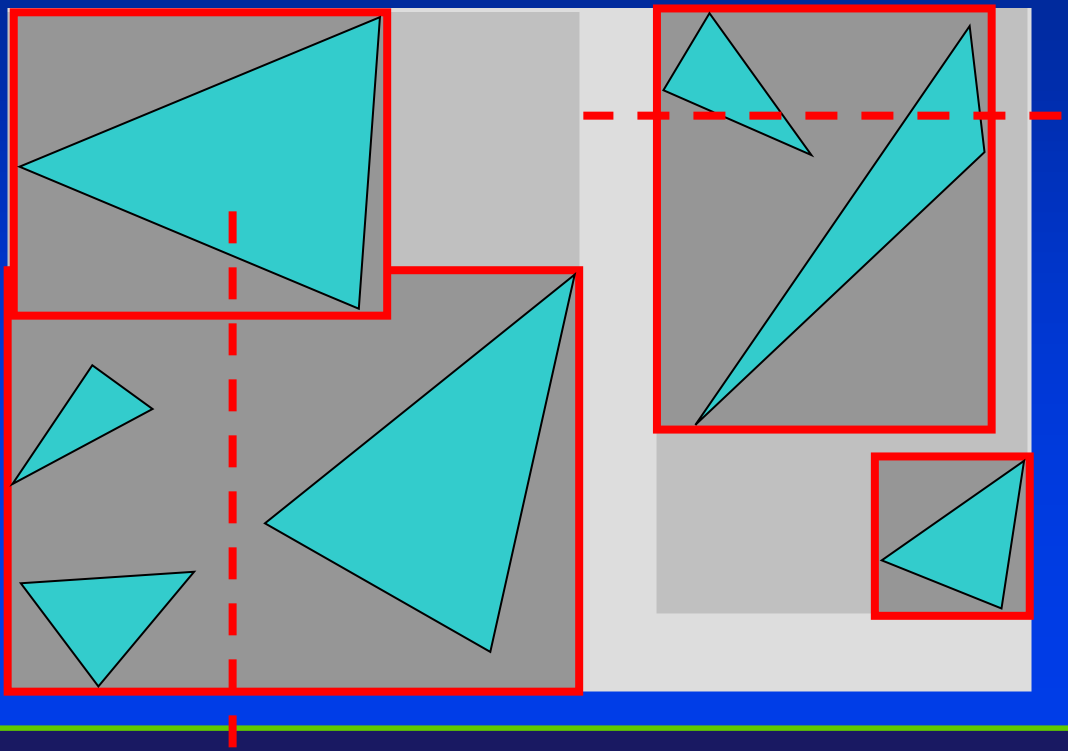
Bounding Volume Hierarchy

- Find bounding box of objects
- Split objects into two groups
- Recurse



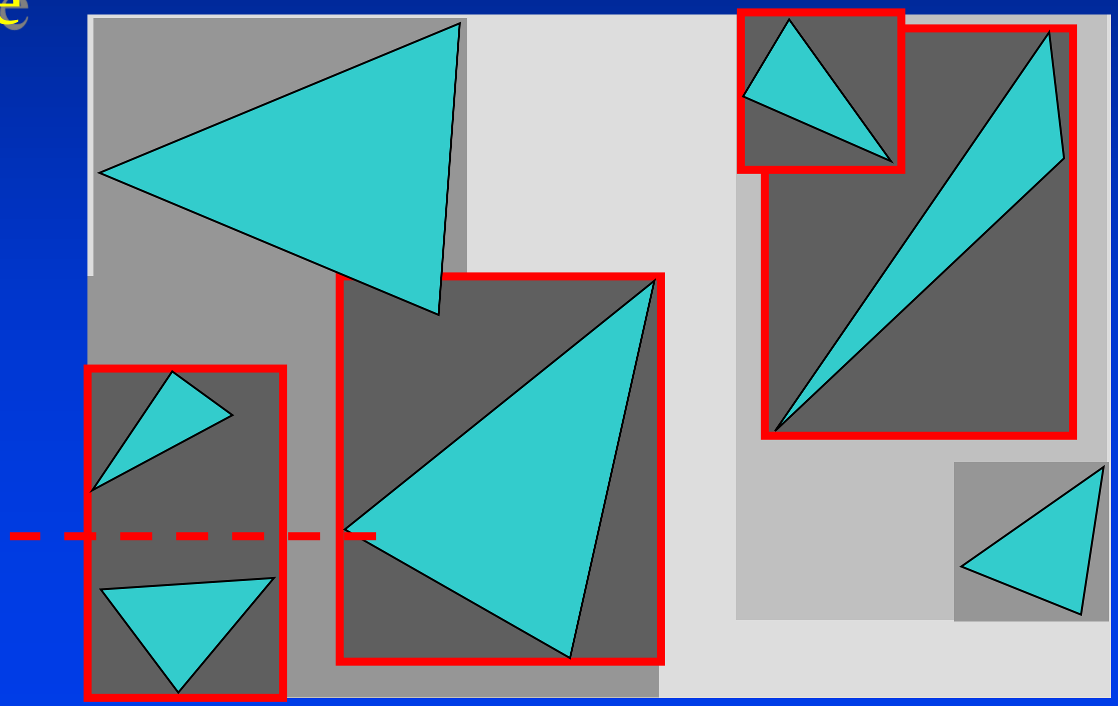
Bounding Volume Hierarchy

- Find bounding box of objects
- Split objects into two groups
- Recurse



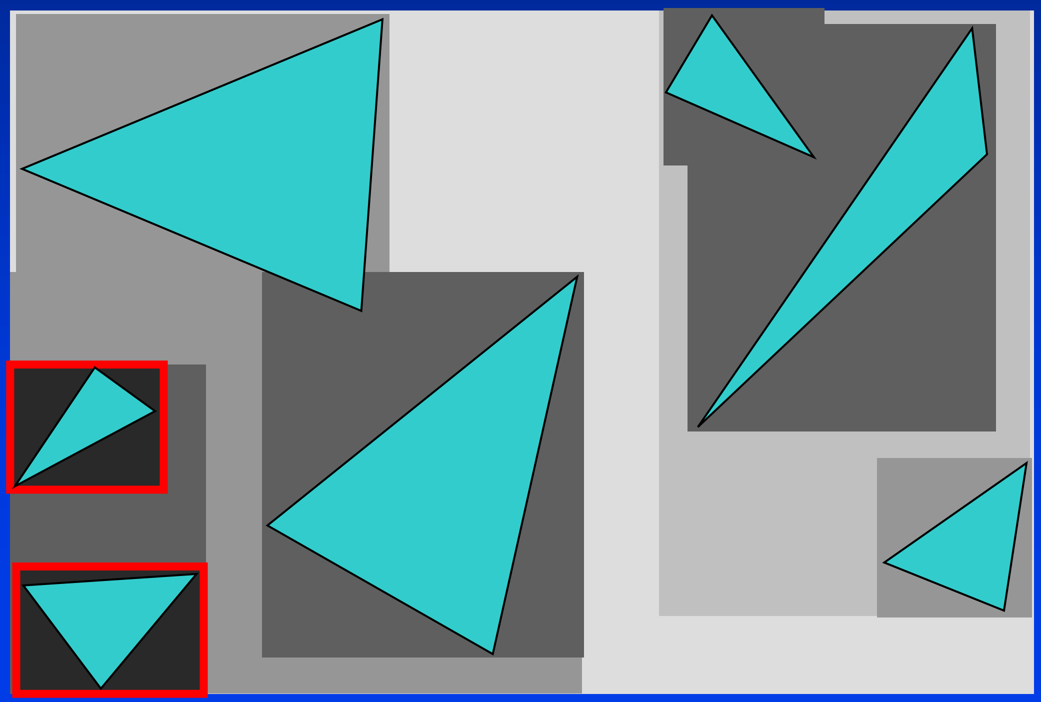
Bounding Volume Hierarchy

- Find bounding box of objects
- Split objects into two groups
- Recurse



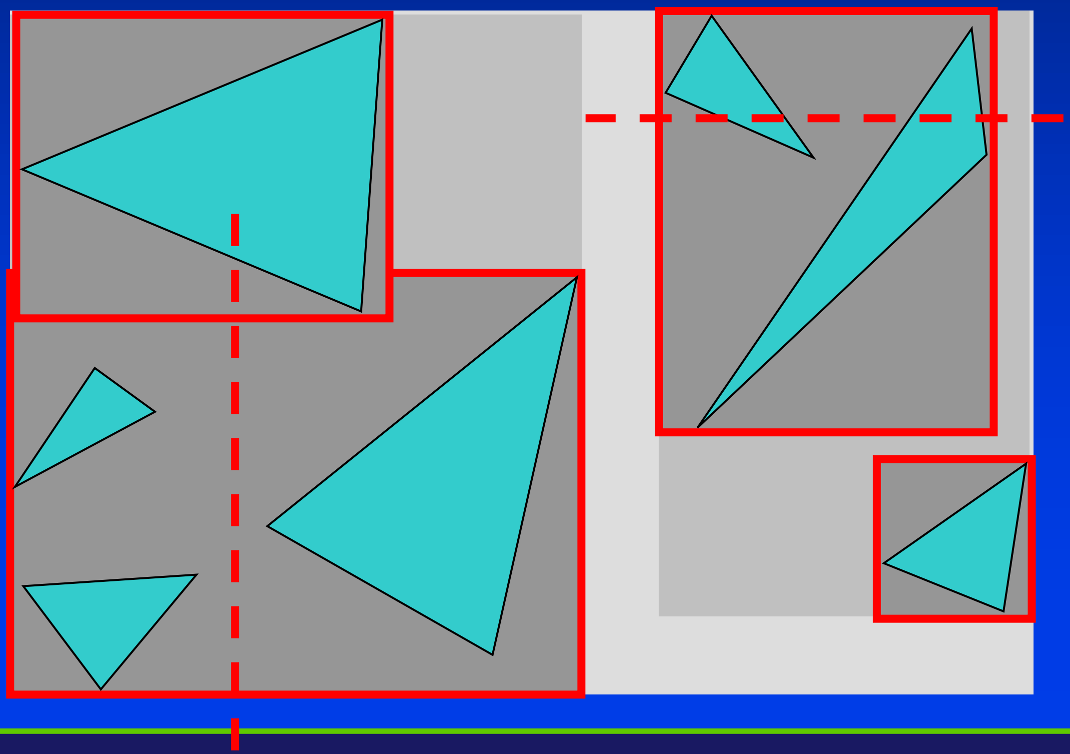
Bounding Volume Hierarchy

- Find bounding box of objects
- Split objects into two groups
- Recurse



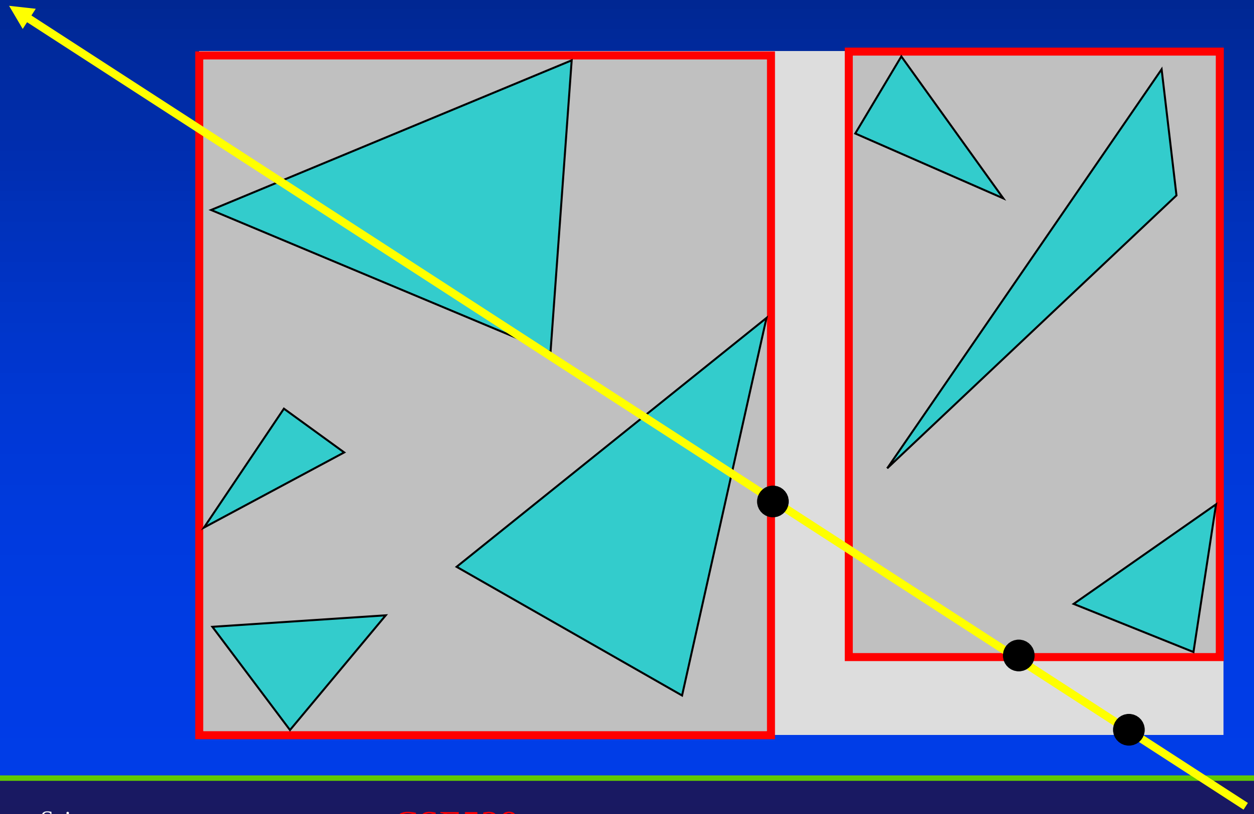
Where to split objects?

- At midpoint *OR*
- Sort, and put half of the objects on each side *OR*
- Use modeling hierarchy



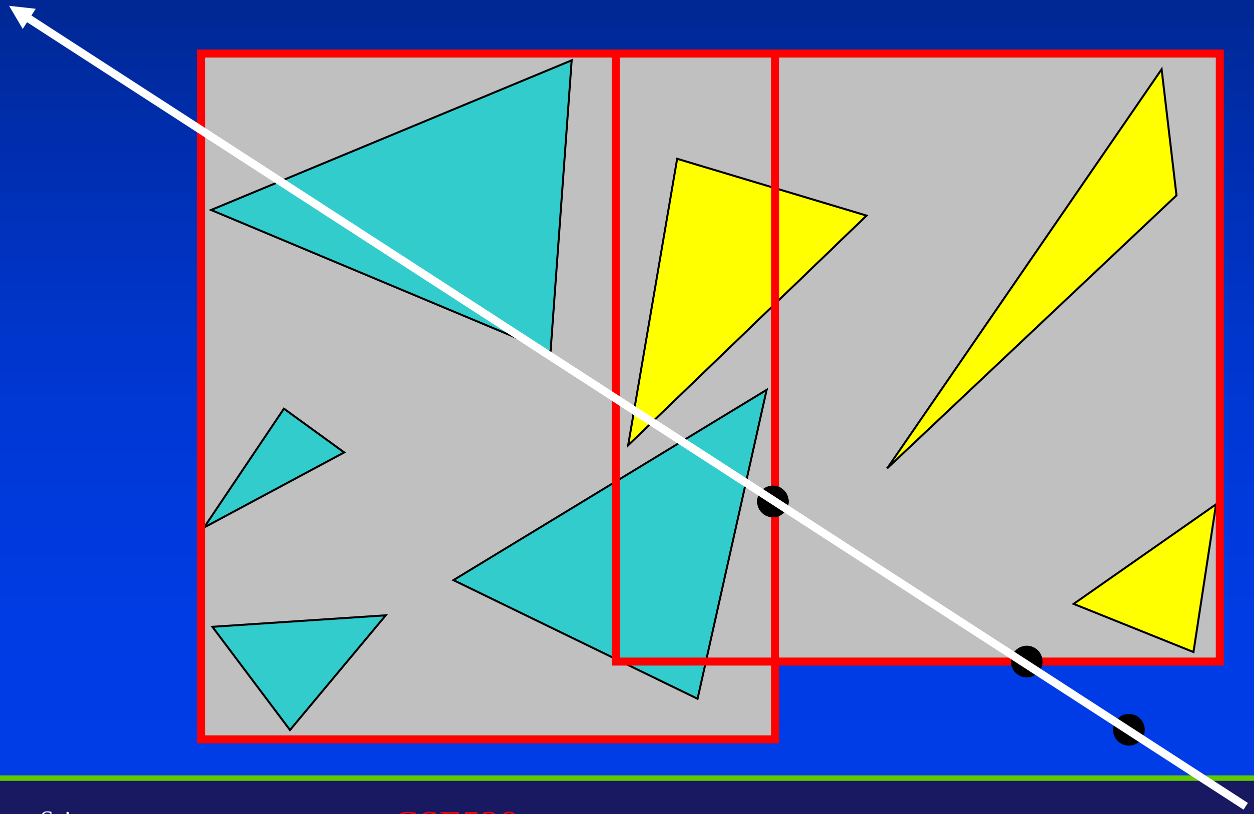
Intersection with BVH

- Check sub-volume with closer intersection first

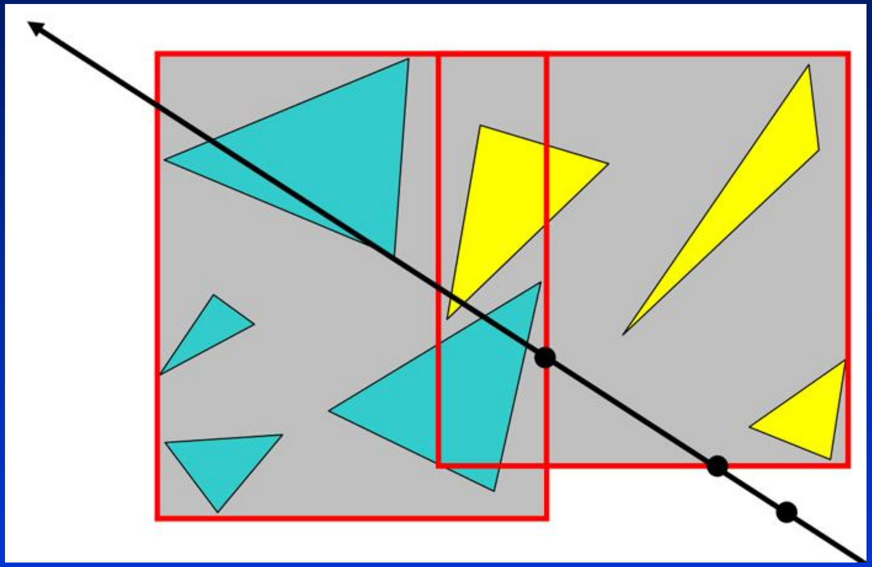


Intersection with BVH

- Don't return intersection immediately if the other subvolume may have a closer intersection



Bounding Volume Hierarchy Discussion



- Advantages
  - easy to construct
  - easy to traverse
  - binary
- Disadvantages
  - may be difficult to choose a good split for a node
  - poor split may result in minimal spatial pruning

Summary

- 3D modeling uses advanced primitives and ways of cutting, joining them
- Inverse kinematics determines joint position from end effector motions
- Keyframe animation involves important poses and inbetweening
- 3D morphing animates surface control points
- 3D spatial subdivision trees include CSG-trees, BSP-trees, Quadtrees, and Octrees

OpenGL Example for Hierarchical Modeling

- OpenGL defines `glPushMatrix()` and `glPopMatrix()`
  - Takes the current matrix and pushes it onto a stack, or pops the matrix off the top of the stack and makes it the current matrix
  - Note: Pushing does not change the current matrix
- Rendering a hierarchy (recursive):

```
RenderNode(tree)
      glPushMatrix()
            Apply node transformation
            Draw node contents
            RenderNode(children)
      glPopMatrix()
```

OpenGL Examples

- OpenGL defines *display lists* for encapsulating commands that are executed frequently

```
list_id = glGenLists(1);
glNewList(list_id, GL_COMPILE);
glBegin(GL_TRIANGLES);
     draw some stuff
glEnd();
glEndList();
```

And later

```
glCallList(list_id);
```

Instancing

- Sometimes you need many copies of the “same” object
  - Like chairs in a room
- Define one chair, the base or the prototype
- Create many *instances* (copies) of it, and apply a different transformation to each
- Appears in scene description languages (Renderman, Inventor) as “defining” a label for an object
- What does it save?

Parametric Instancing

- Many things, called primitives, are conveniently described by a label and a few parameters
  - Cylinder: Radius, length, does it have end-caps, …
  - Bolts: length, diameter, thread pitch, ….
  - Other examples?
- This is a modeling format:
  - Provide software that knows how to draw the object given the parameters, or knows how to produce a polygonal mesh
  - How you manage the model depends on the rendering style
  - Can be an exact representation

Rendering Instances

- Generally, provide a routine that takes the parameters and produces a polygonal representation
  - Conveniently brings parametric instancing into the rendering pipeline
  - May include texture maps, normal vectors, colors, etc
  - OpenGL utility library (GLu) defines routines for cubes, cylinders, disks, and other common shapes
  - Renderman does similar things, so does POVray, …
- The procedure may be dynamic
  - For example, adjust the polygon resolution according to distance from the viewer

Display Lists

- Why use display lists?
- Almost any command can go in a display list
  - Viewing transformation set-up
  - Lighting set-up
  - Surface property set-up
- But some things can't
  - Causes strange bugs – always check that a command can go in a display list
- The list can be:
  - `GL_COMPILE`: things don't get drawn, just stored
  - `GL_COMPILE_AND_EXECUTE`: things are drawn, and also

Display Lists (Pros vs. Cons)

- You should use display lists when:
  - You do the same thing over and over again
  - The commands are supported
  - Nothing changes about the way you do it
- Advantages:
  - Can't be much slower than the original way
  - Can be much much faster
- Disadvantages:
  - Can't use various commands that would offer other speedups

Questions?