CMT107 Visual Computing

IV.1 Object Representation

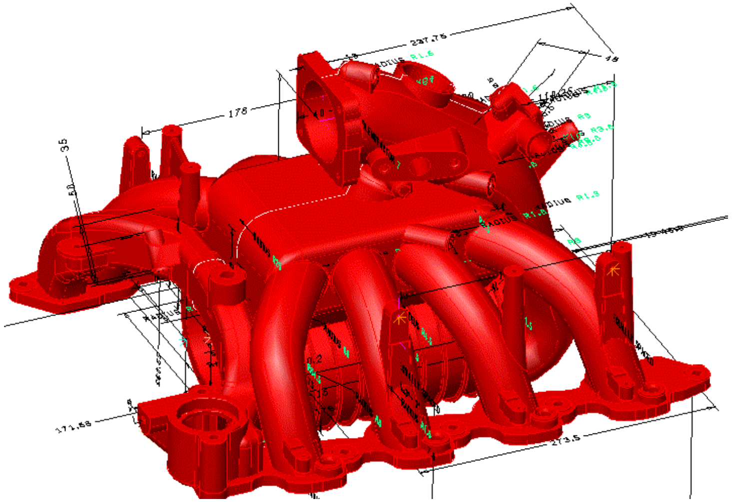
Xianfang Sun

School of Computer Science & Informatics
Cardiff University

Overview

- Constructive solid geometry
- Boundary representation
- Mesh representation
  - Rendering meshes with OpenGL
- Volumetric representation: voxels

Example Models and Scenes

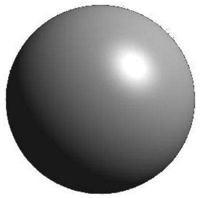


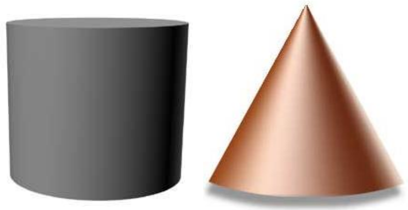
Geometric Modelling

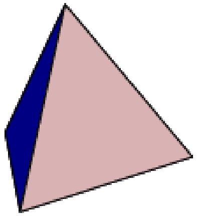
- Need data-structures and algorithms to model shapes
  - *Scene* – description of the whole environment
  - *Model* – description of an object in the environment
  - Suitable for *creating, editing, analysing* and *rendering*
- Object representations
  - Constructive solid geometry (CSG)
  - Boundary representation (B-rep)
  - Mesh representation
  - Volumetric representation: voxels

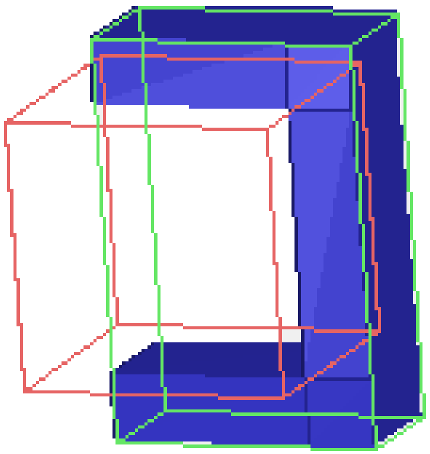
Constructive Solid Geometry

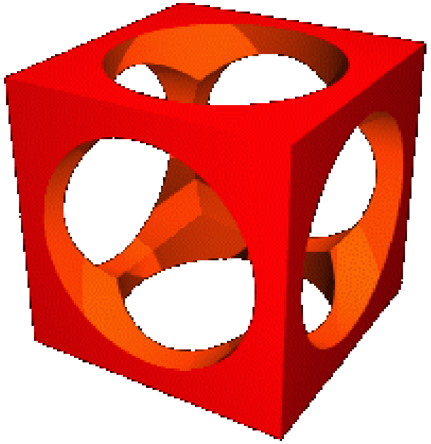
- Use set of volumetric primitives
  - Block, Tetrahedron, sphere, cylinder, cone, ...
- Construct objects using Boolean operations
  - Union, intersection, difference

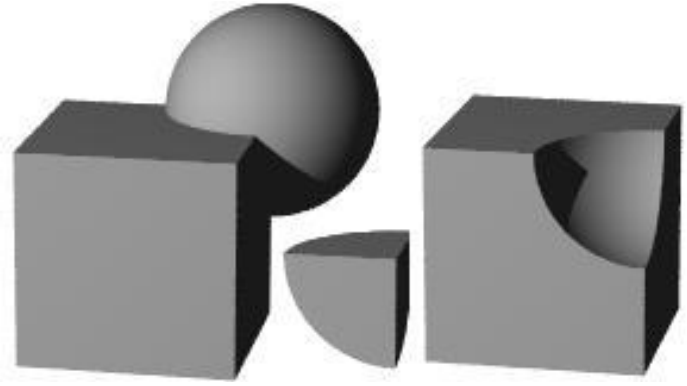






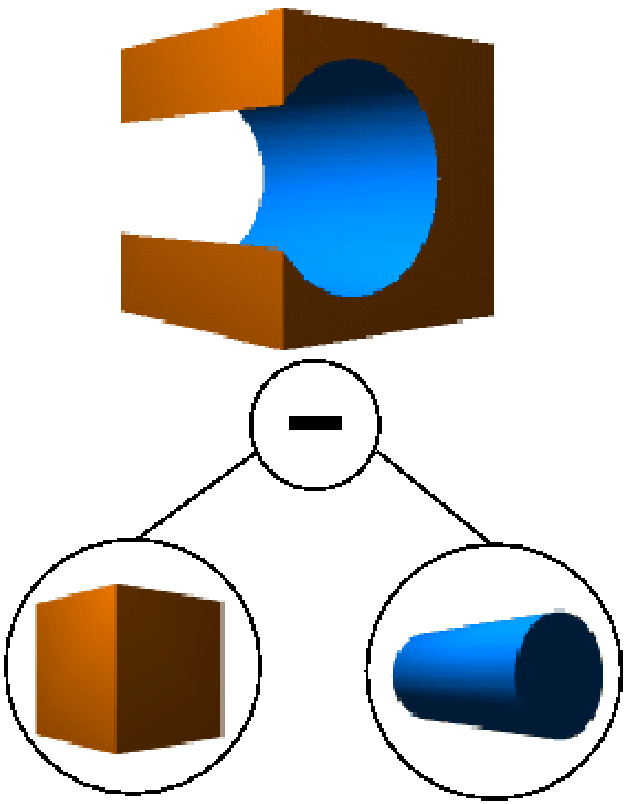


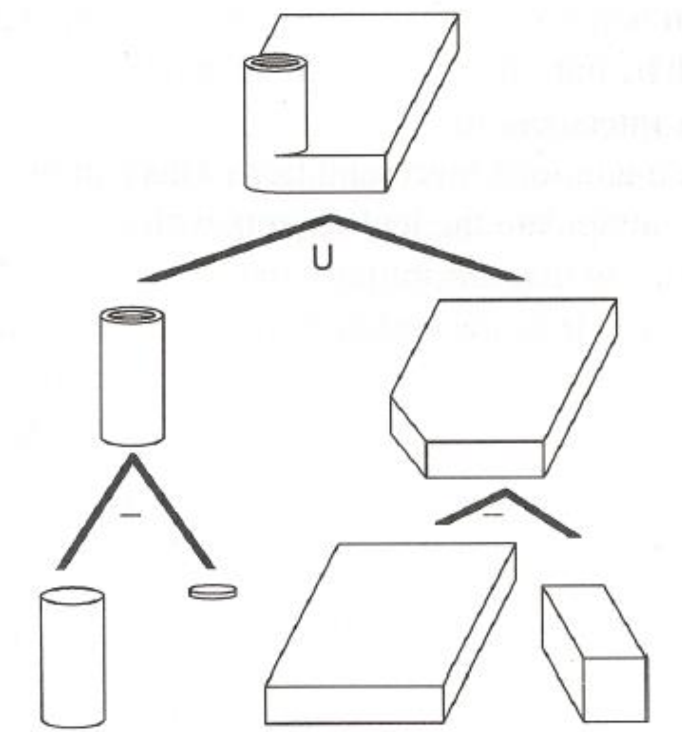




CSG Tree

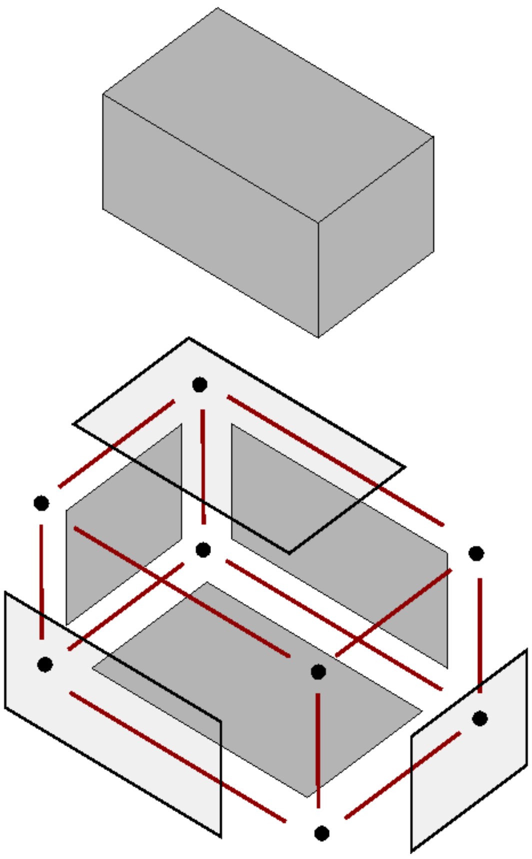
- CSG operations stored as tree (or sequence) of operations on primitives
- Common for CAD – *feature based modelling*





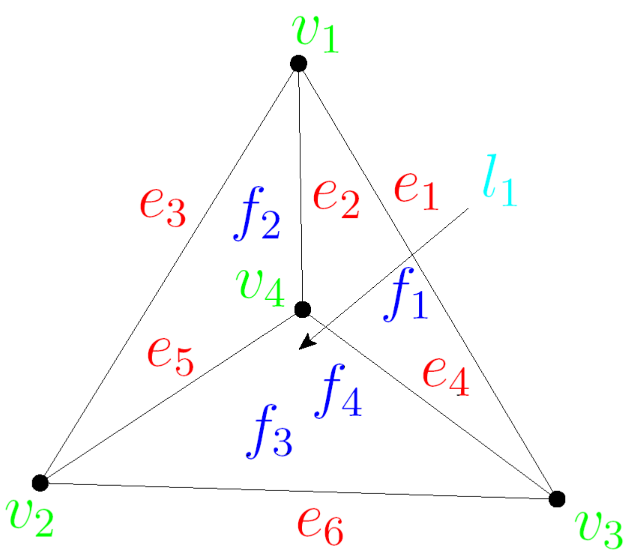
Boundary Representation

- Explicitly represent boundary of object:
  - Basic elements are (natural) *faces, edges, vertices* with a geometry (shape)
  - Also record topology (connectivity/ boundary relations) of elements
- Mathematically: an algebraic complex (topology) with a geometric realisation (geometry)
- Algorithmically: a graph data structure (topology) where nodes have shape (geometry) attributes



B-Rep: An Algebraic Complex

- *Cells* (elements) = $\{v_1, v_2, v_3, v_4, e_1, e_2, e_3, e_4, e_5, e_6, f_1, f_2, f_3, f_4, l_1\}$
- *Rank* (dimension) $=$
  $\{(0, \{v_1, v_2, v_3, v_4), (1, \{e_1, e_2, e_3, e_4, e_5, e_6), (2, \{f_1, f_2, f_3, f_4\}), (3, \{l_1\})\}$



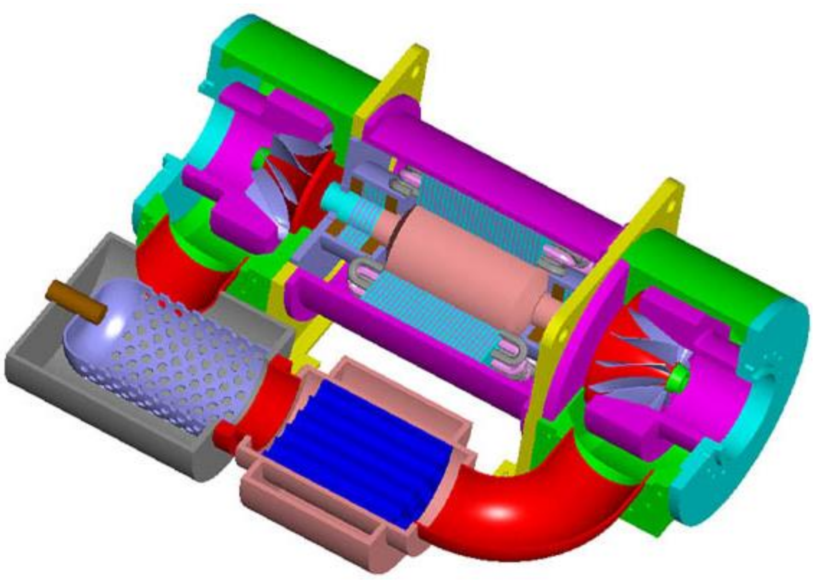
- *Bound* (topology) $= \{(e_1, \{v_1, v_3\}), (e_2, \{v_1, v_4\}), (e_3, \{v_1, v_2\}), (e_4, \{v_3, v_4\}), (e_5, \{v_2, v_4\}), (e_6, \{v_2, v_3\}), (f_1, \{e_1, e_2, e_4\}), (f_2, \{e_2, e_3, e_5\}), (f_3, \{e_1, e_3, e_6\}), (f_4, \{e_4, e_5, e_6\}), (l_1, \{f_1, f_2, f_3, f_4\})\}$

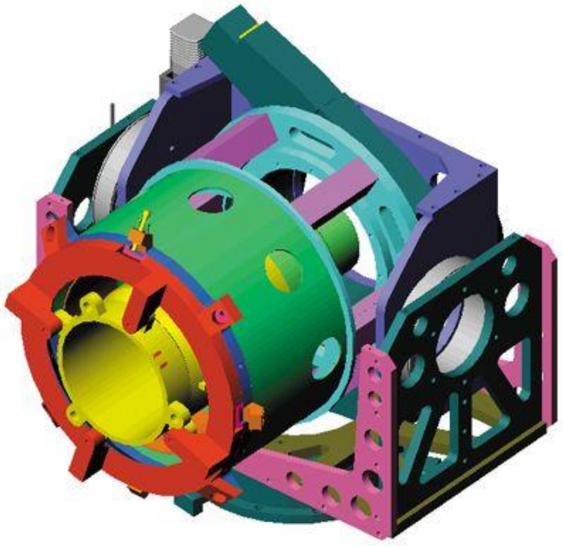
B-Rep Geometry

- Describe shape of each face, edge and vertex
  - Vertex geometry: position
  - Edge geometry: curve

    E.g. straight line, circle, ellipse, free-form curve, . . .
  - Face geometry: surface

    E.g. plane, sphere, cylinder, cone, torus, free-form, . . .





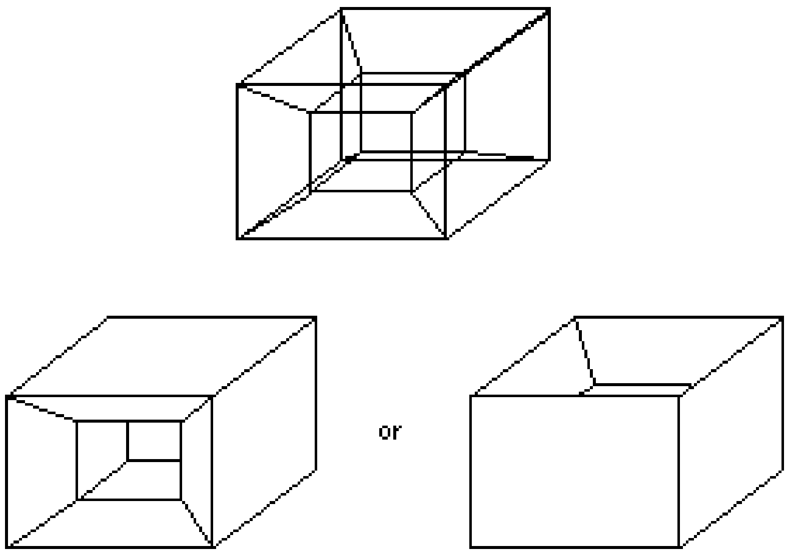
B-Rep Data Structure

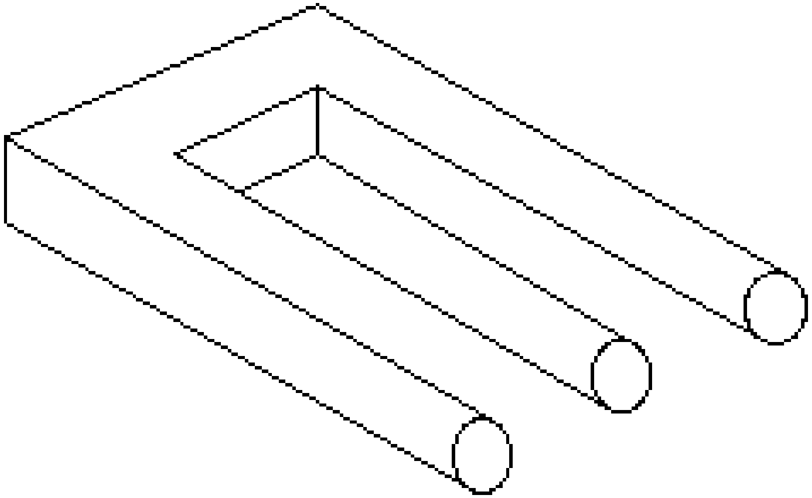
- B-Rep graph data structure representing the topology:

BODY	Solid made of a list of LUMPS
LUMP	Connected volume, bounded by a list of SHELLS
SHELL	Connected surface, consisting of a list of FACES
FACE	Natural surface, bounded by a LOOP
LOOP	Connected curves, consisting of a list of COEDGES
COEDGE	Directed edge as part of a loop, consisting of an EDGE (also called half-edge)
EDGE	Natural edge, bounded by VERTICES
VERTEX	Boundary of an edge

B-Rep Issues

- Consistency of geometry and topology
  - No explicit way to ensure boundary relations are preserved by geometry
- Ambiguous and impossible models

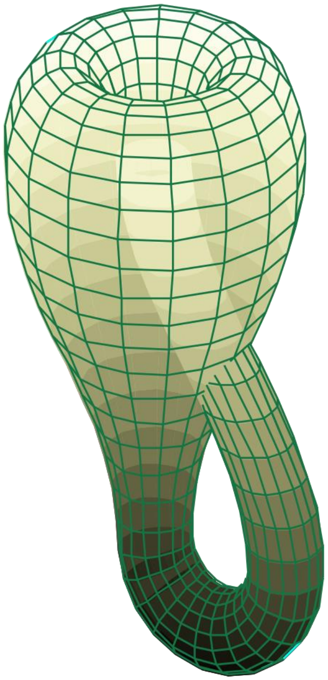




- Topology allows us to determine impossible models
- Orientation and topology distinguish ambiguous models

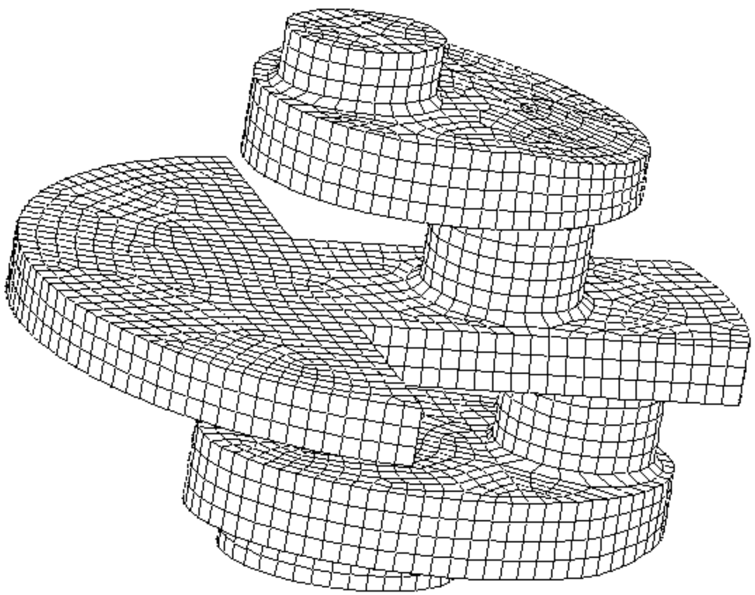
B-Rep Orientation

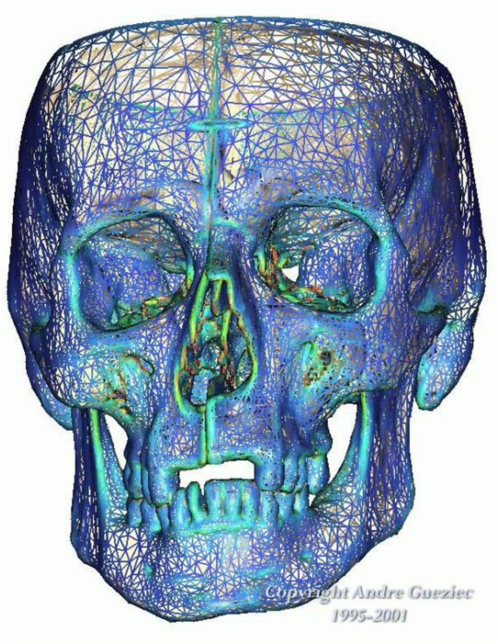
- Orient face: distinguish between inside and outside
  - Surface normals always point towards the outside
- Orient each loop
  - Move around each loop such that the inside lies to the left when viewed from outside the model
  - COEDGEs indicate direction of loop by ordering edge end-points
  - EDGE lies on two faces as indicated by two COEDGEs
- Non-manifold objects: EDGE can lie on more than two faces
  - Causes problems for orientation, etc. (so not allowed in standard B-rep)



Mesh Representation

- Describe model as a polygonal mesh (often triangular)
  - Collection of polygons (facets)
  - Similar, but simpler than B-rep
  - Linear approximation of object
  - Fast and quality good enough for real-time rendering





Polygons

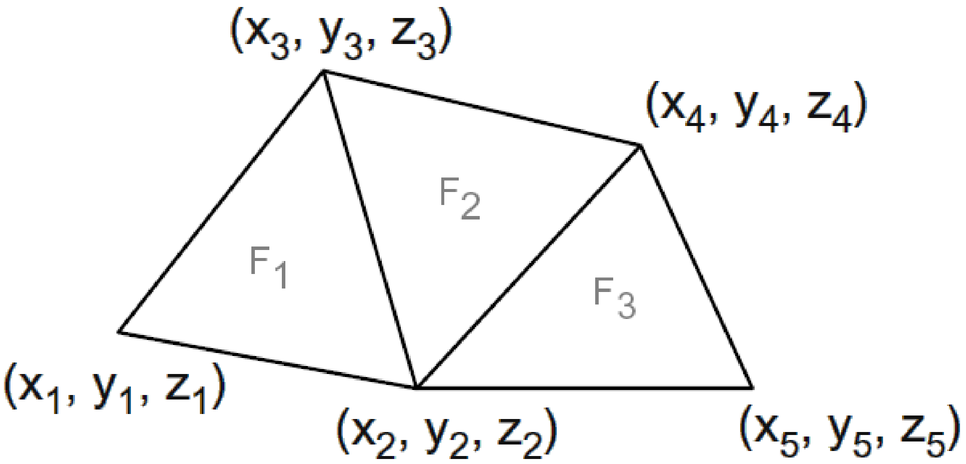
- Polygons are specified by a sequence of vertices
- Polygons are not just line segments, but have an interior
  - Simple polygon: lines do not intersect
  - Convex polygon: given two points inside the polygon, the line segment joining the points lies inside polygon
  - Flat polygon: polygon lies in a plane
- Orientation / sidedness:
  - Polygons have a front and a back
  - If vertices are in anti-clockwise order on display, we see the front
    (default OpenGL convention; consistent with B-rep orientation)

Polygon Normal

- If a polygon is simple, convex and flat, its normal can be calculate using any 3 non-collinear points p_l , p_m, and p_n
  - Suppose $l<m<n$
  - $v_1 = p_m - p_l$, $v_2 = p_n - p_m$,
  - $n = v_1 \times v_2$
  - normal n points outside the front.
- Polygon normal vector and the viewer direction vector can determine whether the viewer is looking at the front or back of the polygon
  - If the angle between normal vector and viewer direction vector are less than 90°, it's at the front
  - If the angle is great than 90°, it's at the back
  - If the angle is 90°, the viewer is on the polygon plane.

List of Faces

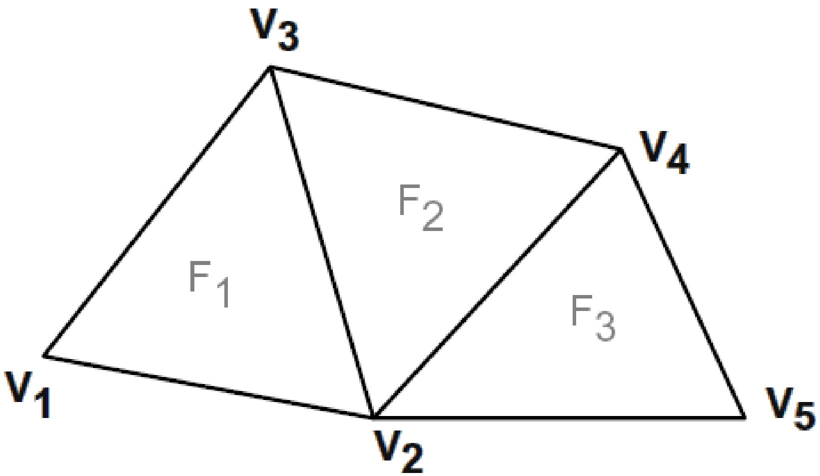
- Each face lists vertex coordinates
  - Redundant vertices
  - No adjacency or other structural information (topology)
  - Orientation from sequence of vertices



FACE TABLE	
F_1	(x_1, y_1, z_1) (x_2, y_2, z_2) (x_3, y_3, z_3)
F_2	(x_2, y_2, z_2) (x_4, y_4, z_4) (x_3, y_3, z_3)
F_3	(x_2, y_2, z_2) (x_5, y_5, z_5) (x_4, y_4, z_4)

Vertex and Face Tables

- Each face lists vertex references
  - Shared vertices
  - No adjacency or other structural information (topology)
  - Orientation from sequence of vertices



VERTEX TABLE			
V_1	X_1	Y_1	Z_1
V_2	X_2	Y_2	Z_2
V_3	X_3	Y_3	Z_3
V_4	X_4	Y_4	Z_4
V_5	X_5	Y_5	Z_5

FACE TABLE			
F_1	V_1	V_2	V_3
F_2	V_2	V_4	V_3
F_3	V_2	V_5	V_4

- Can add half-edges, shells, lumps, bodies for representing solids

Rendering Meshes with OpenGL

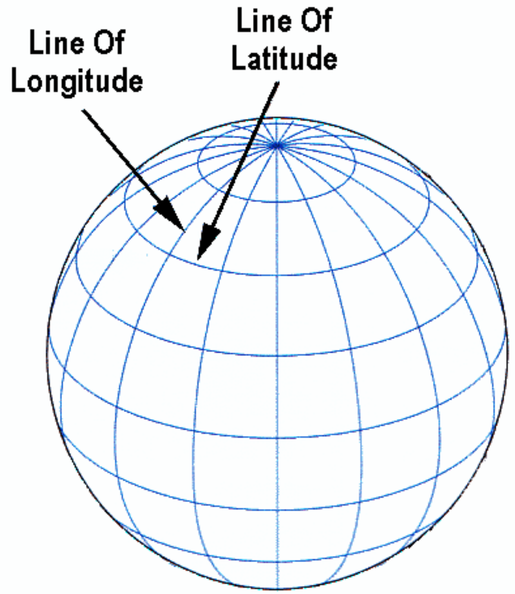
- Two simple OpenGL drawing functions:

```
✓ glDrawArrays(mode,first,count);
✓ glDrawElements(mode, count, type, indices);
```

  - mode: GL_POINTS, GL_LINES, GL_TRIANGLES, etc.
  - first: the starting index in the enabled arrays.
  - count: the number of elements to be rendered
  - type: type of the values in indices, GL_UNSIGNED_BYTE, GL_UNSIGNED_SHORT, or GL_UNSIGNED_INT
  - indices: a pointer to the location where the indices are stored.
- glDrawArrays() is used for “List of Faces”
  - Example see CG02.java in the labs
- glDrawElements() is used for “Vertex and Face Tables”
  - Example see CG03.java in the labs

Modelling a Sphere

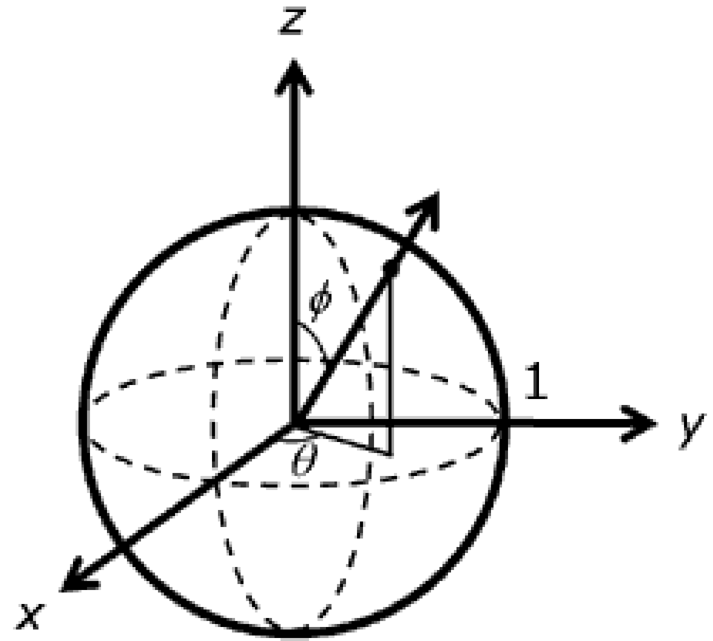
- A sphere can be modelled by covering the surface with triangles
  - use lines of longitude and latitude to divide the surface into triangles (around north and south poles) and quadrangles
  - each quadrangles is divided into two triangles for rendering by OpenGL



Spherical Coordinates

- Points on a unit sphere in spherical coordinates :

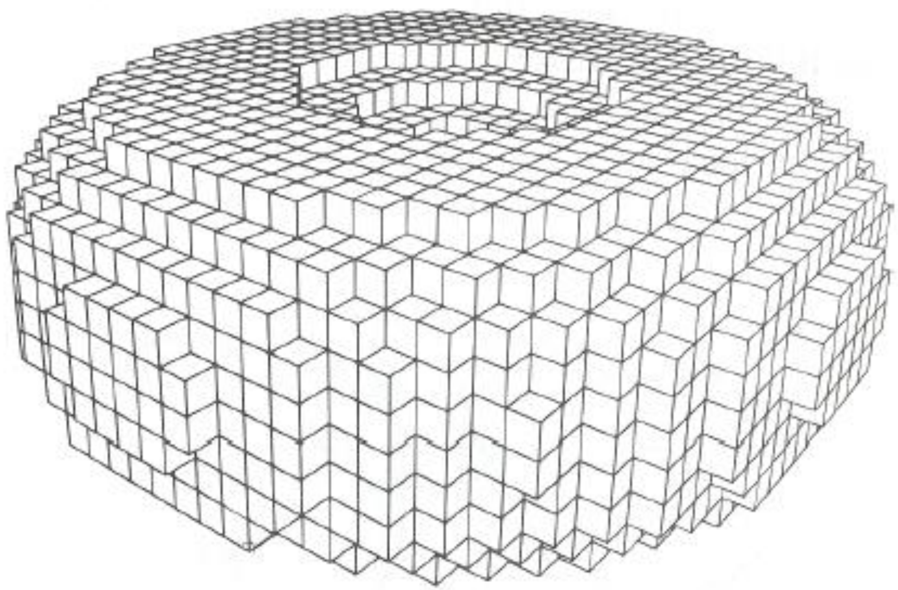
$$x(\phi,\theta) = \sin\phi\cos\theta$$
$$y(\phi,\theta) = \sin\phi\sin\theta$$
$$z(\phi,\theta) = \cos\phi$$
$$(\phi,\theta) \in [0,\pi]\times[0,2\pi]$$



- Maps each (ϕ,θ) on a point on the unit sphere
  (but be careful at the north and south poles)
- More details see sphere.java in the labs…

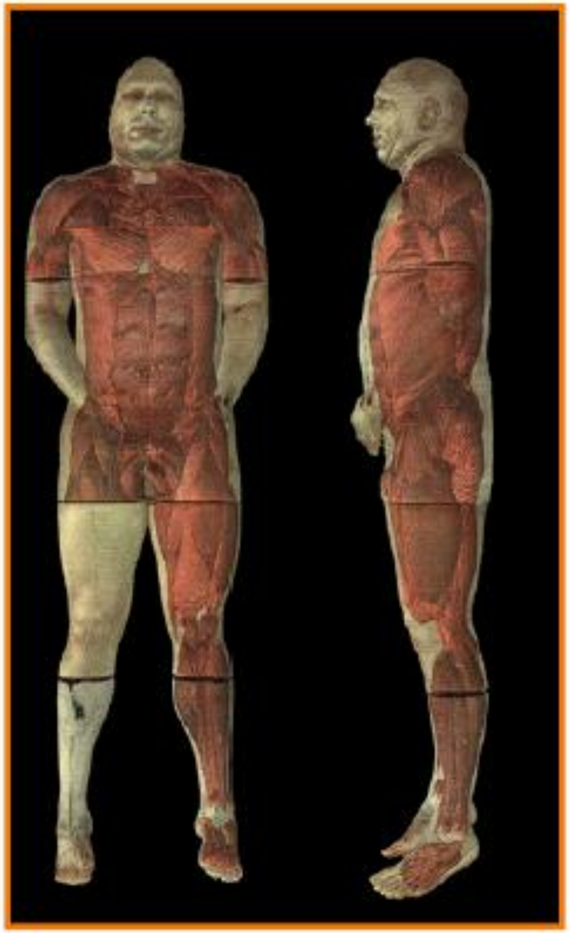
Volumetric Representation: Voxels

- Partition space into uniform 3D grid
  - Grid cells are called voxels (volume elements) (also see pixels)
- Store *properties* of solid object with each voxel
  - Occupancy
  - Colour
  - Density
  - Temperature
  - . . .



FvDFH Figure 12.20

Voxel Examples



Visible Human
(National Library of Medicine)

SUNY Stoney Brook

Voxel Issues

- Advantages:
  - Simple inside/outside test
  - Simple and robust boolean operations
  - Represent interior of the object
- Disadvantages:
  - Memory consuming

    (can use octree for hierarchical construction to save memory)
  - Non-smooth
  - Time consuming to manipulate and render

Summary

- Explain the following model representations:
  - constructive solid geometry
  - boundary representation
  - mesh representation
  - volumetric representation
  - How is the model represented?
  - Which data structures are used?
  - What are advantages/disadvantages of these representations?
- What is a simple / convex / flat polygon?
- What do we understand by the orientation of a polygon/loop/edge?

CMT107 Visual Computing

IV.2 Scene Representation

Xianfang Sun

School of Computer Science & Informatics
Cardiff University

Overview

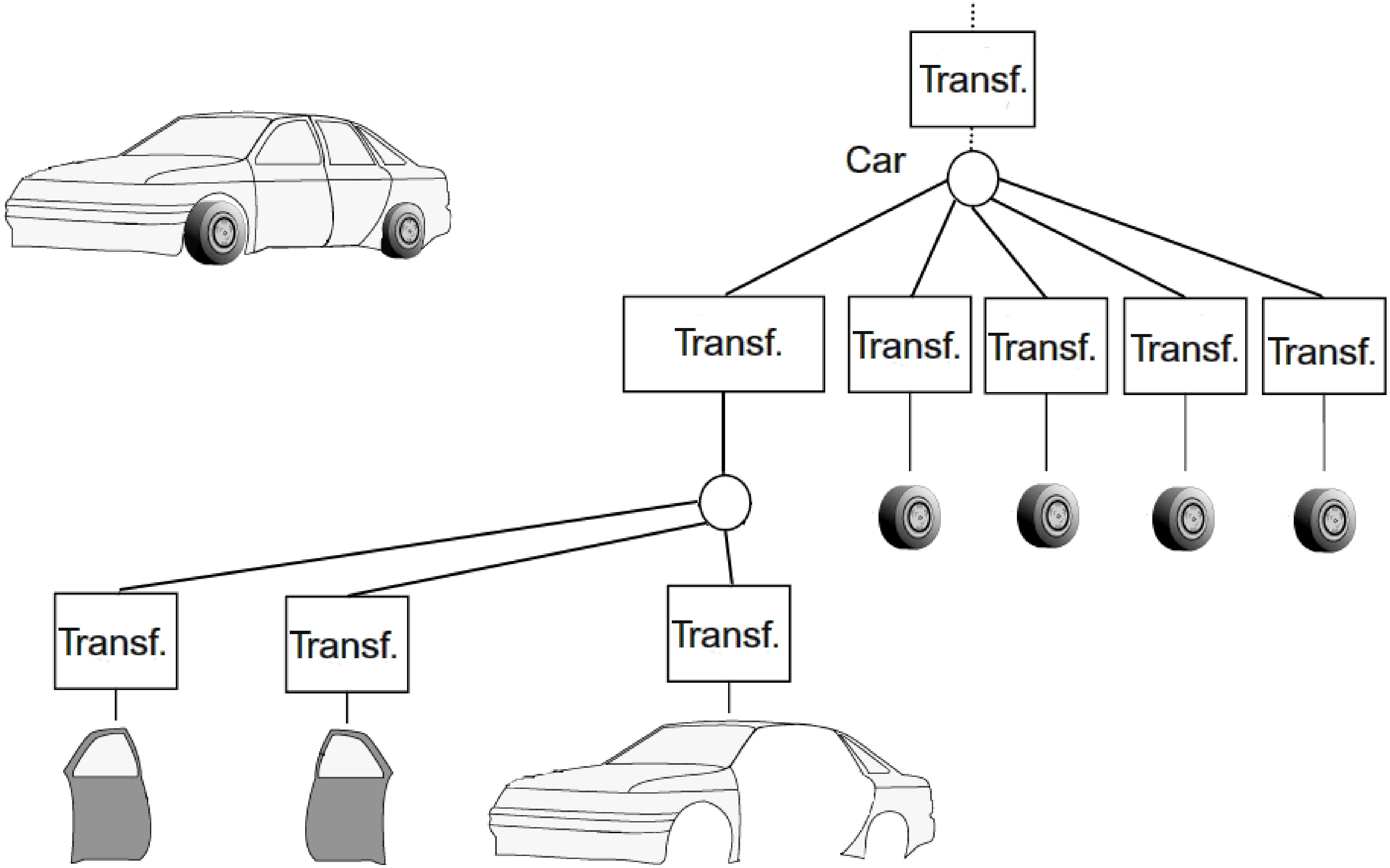
- Hierarchical modelling
  - Scene graphs
  - Constructing scene graphs
- Spatial data structures
  - Uniform grids
  - Octrees
  - kD-trees
  - BSP-trees
- Multi-resolution models

Hierarchical Modelling

- A scene is the complete description of the environment
  - A view is a particular part of the scene visible from the camera position
  - A scene consists of many models, only some are visible
- A scene can be represented by a hierarchical structure
  - A node represents some part of the scene
  - Top node is the whole scene
  - Leaf nodes are the actual geometric models
- Objects specified in *local coordinates*
  - Add transformation to hierarchy to specify location in scene

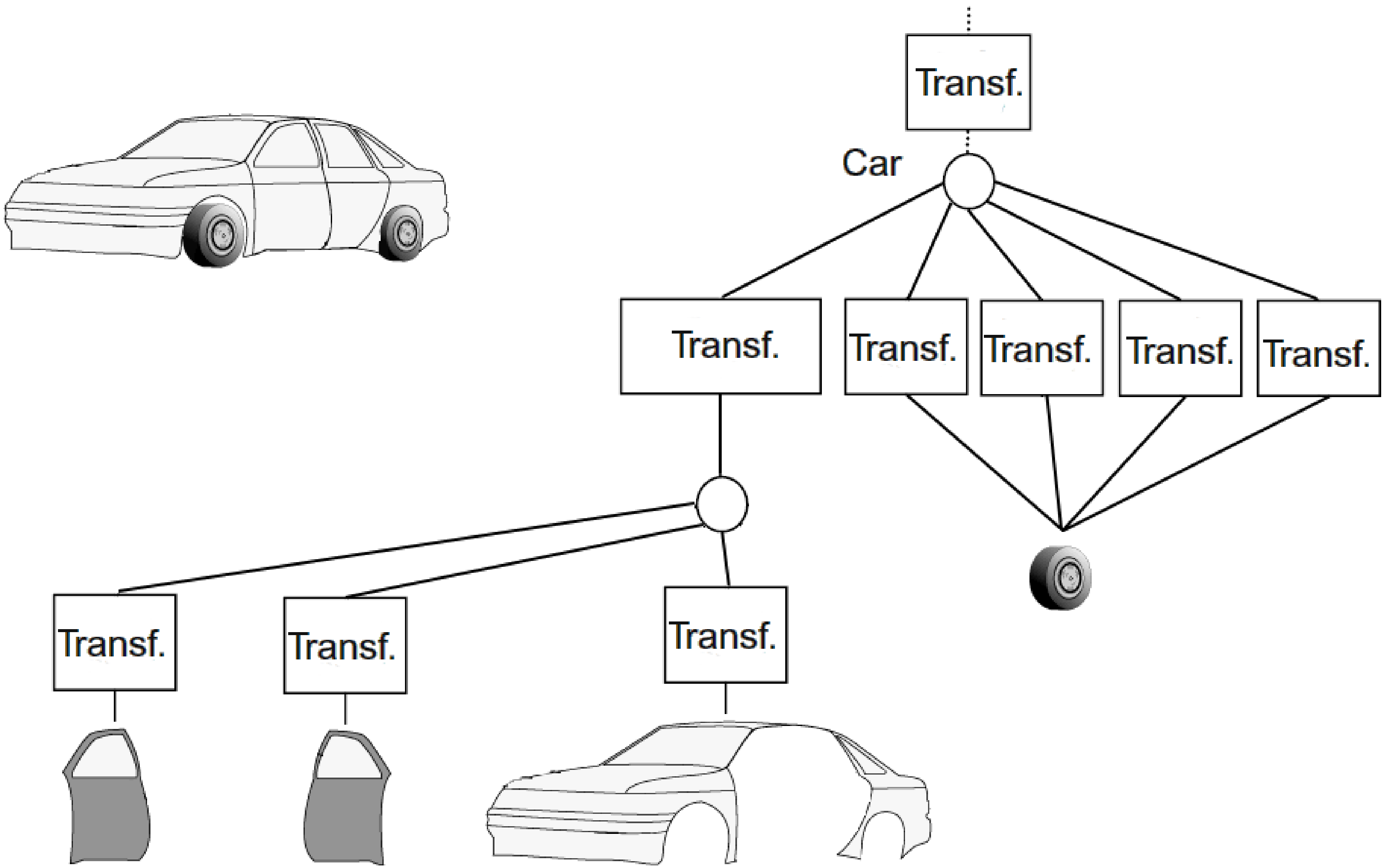
Scene Tree Example

- Scene tree for a simple car



Scene Graph Example

- Scene graph: combine congruent objects



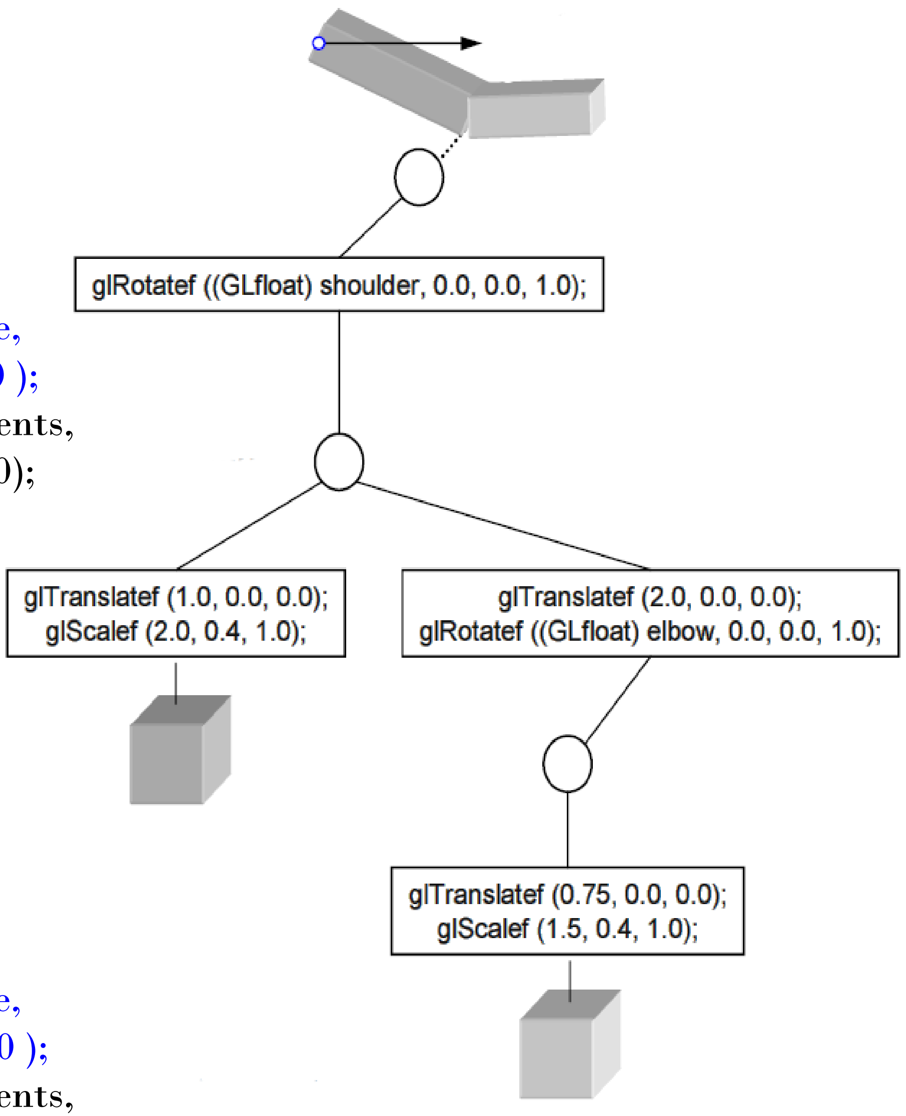
Scene Graphs

- Scene Graphs are in general *acyclic directed graphs*
  - Explicitly represented by graph data structure
  - Or implicitly by sequence of instructions / function calls
- Attributes and inheritance
  - Graph may contain material, transformation, . . . nodes representing object attributes
  - Attributes are usually inherited by all sub-nodes
- Also suitable for animations:
  - Make transformations dependent on parameter, e.g. time, motion control parameters, . . .

Robot Arm—OpenGL Implementation

```
T.initialize();
T.scale(0.5f, 0.5f, 0.5f);
T.scale(2f,0.4f,1f);
T.translate(1,0,0);
T.rotateA(-50f, -0.2f, 0f, 1f);
gl.glUniformMatrix4fv( ModelView, 1, true,
                       T.getTransformv(), 0 );
gl.glUniformMatrix4fv( NormalTransform, 1, true,
                       T.getInvTransformTv(), 0 );
gl.glDrawElements(GL_TRIANGLES, numElements,
                       GL_UNSIGNED_INT, 0);

T.initialize();
T.scale(0.5f, 0.5f, 0.5f);
T.scale(1.5f,0.4f,1);
T.translate(0.75f,0,0);
T.rotateZ(50);
T.translate(2.00f, 0.0f, 0);
T.rotateA(-50f, -0.2f, 0f, 1f);
gl.glUniformMatrix4fv( ModelView, 1, true,
                       T.getTransformv(), 0 );
gl.glUniformMatrix4fv( NormalTransform, 1, true,
                       T.getInvTransformTv(), 0 );
gl.glDrawElements(GL_TRIANGLES, numElements,
                       GL_UNSIGNED_INT, 0);
```



Hierarchy Construction

- Problem: find *optimal hierarchy* for scene graph
  - Choose bounding volumes
    spheres, boxes, convex hulls, . . .
  - Construct hierarchy of objects based on some heuristic (depends on application)
- Consider solutions for special cases
  - Spatial closeness of models
  - Standard spatial data structures

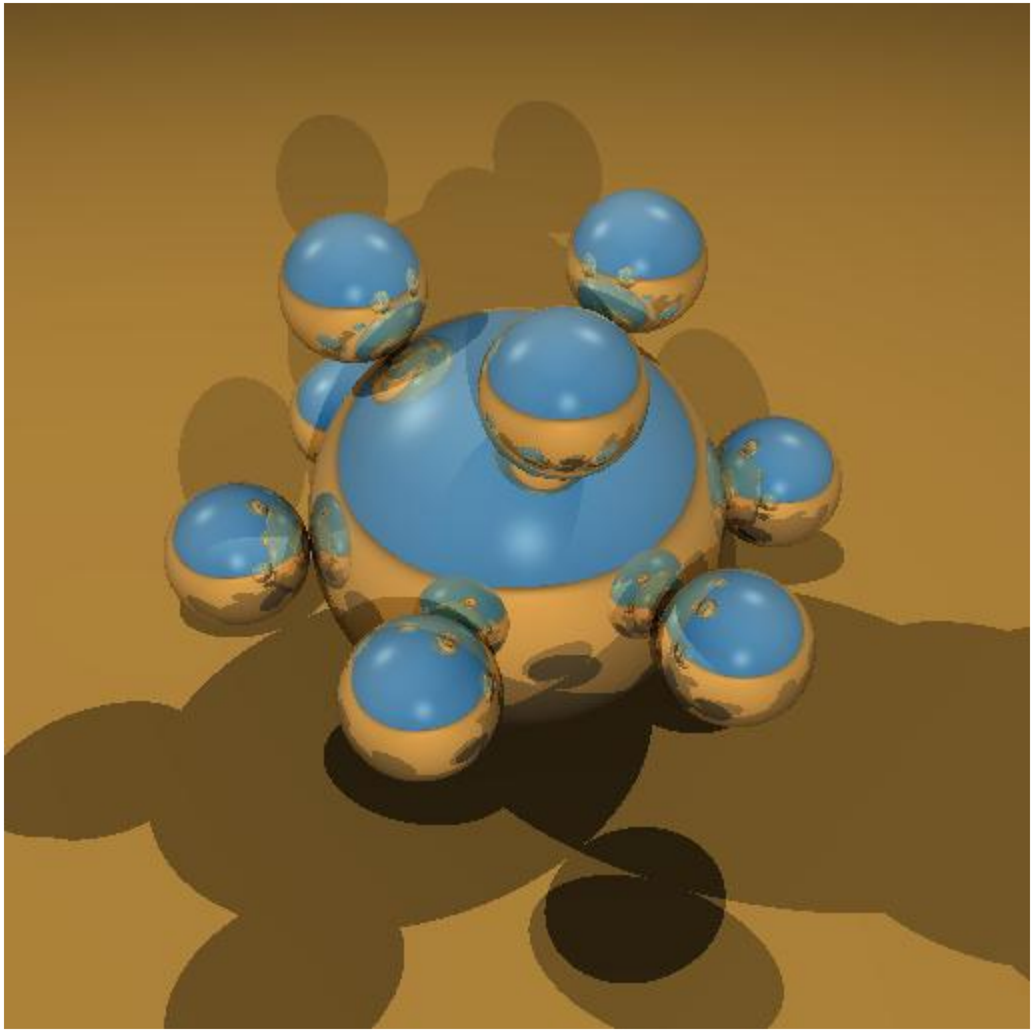
Spatial Data Structures

- Represent *spatial relations* in scene graph
  - Which models are visible from a camera position?
  - Which models can be accessed from a certain position?
  - With which models did a model collide?
  - . . .
- The more information about the spatial relations between models is known, the faster the scene can be processed
  - *Partition* space and place objects within subregions
  - Create *hierarchy* of spatially close models
  - Helps algorithms to determine relevant models quickly

Example 3D Scene

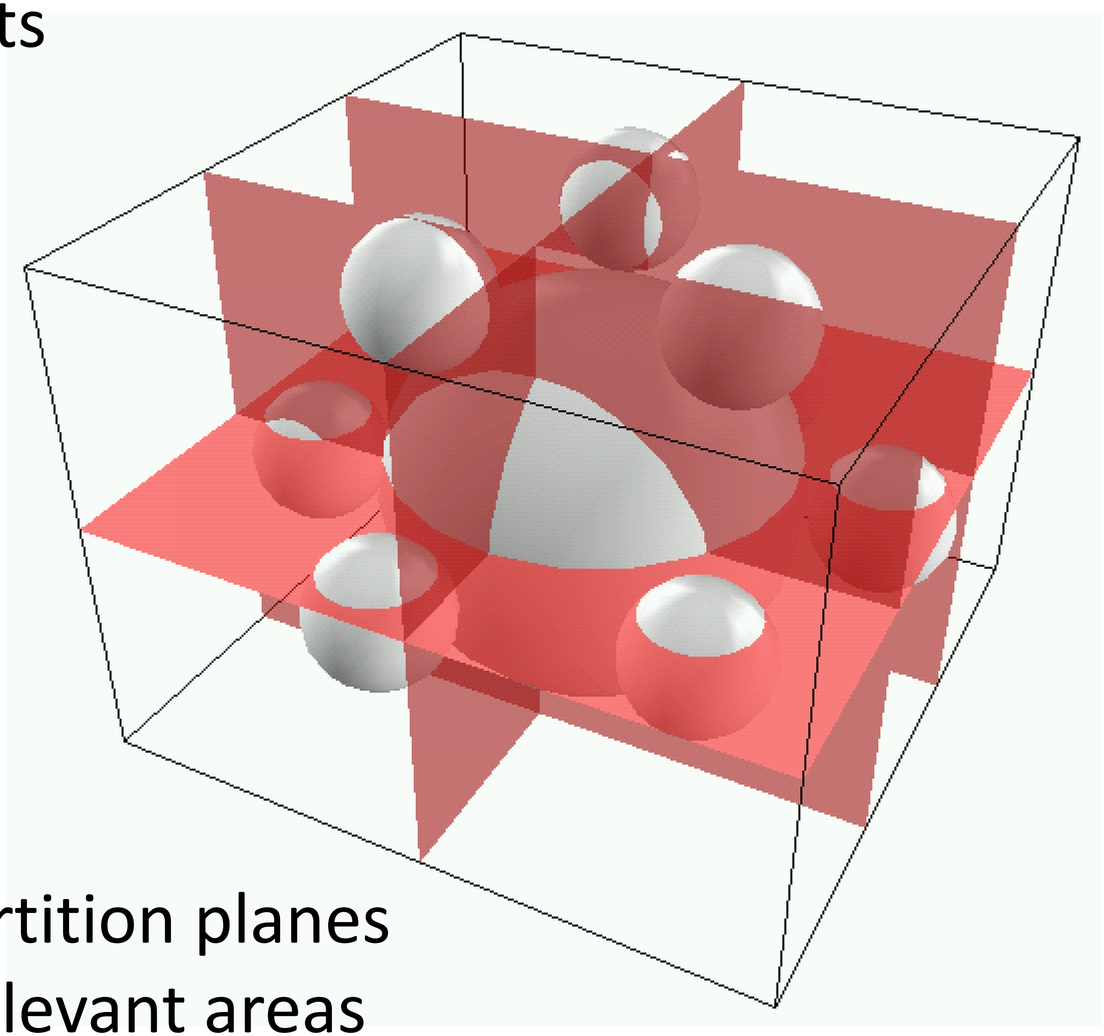
- 3D scene example

  (ray-tracing)



Uniform Grids

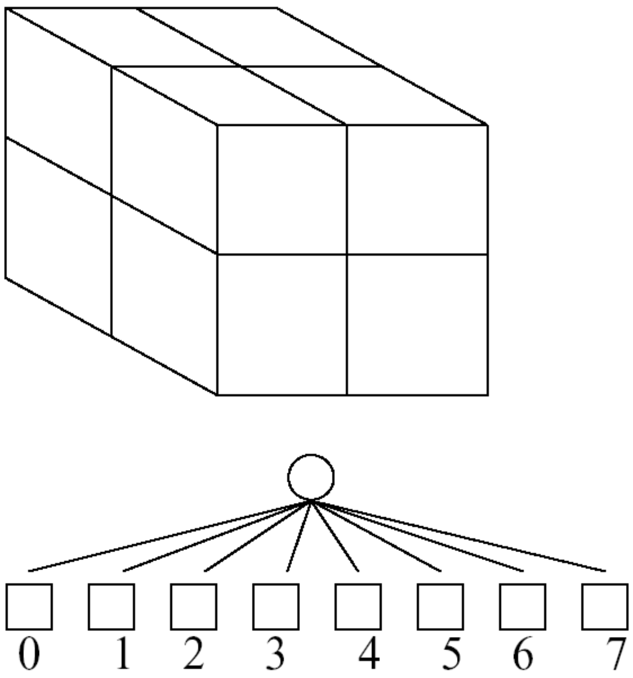
- Partition space *uniformly* using a 3D grid
  - 3D array of model lists

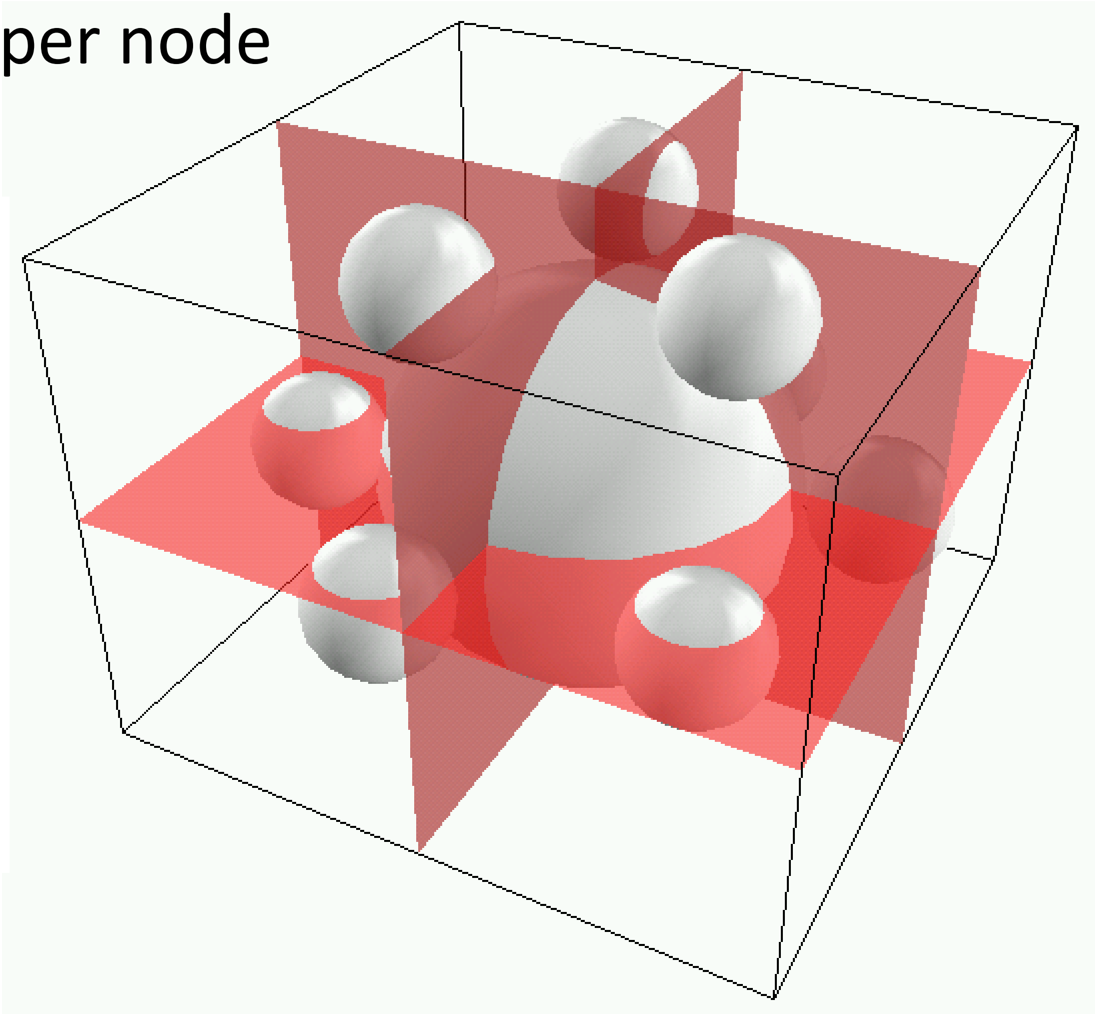


  - Cut models along partition planes
  - Or add them to all relevant areas

Octrees

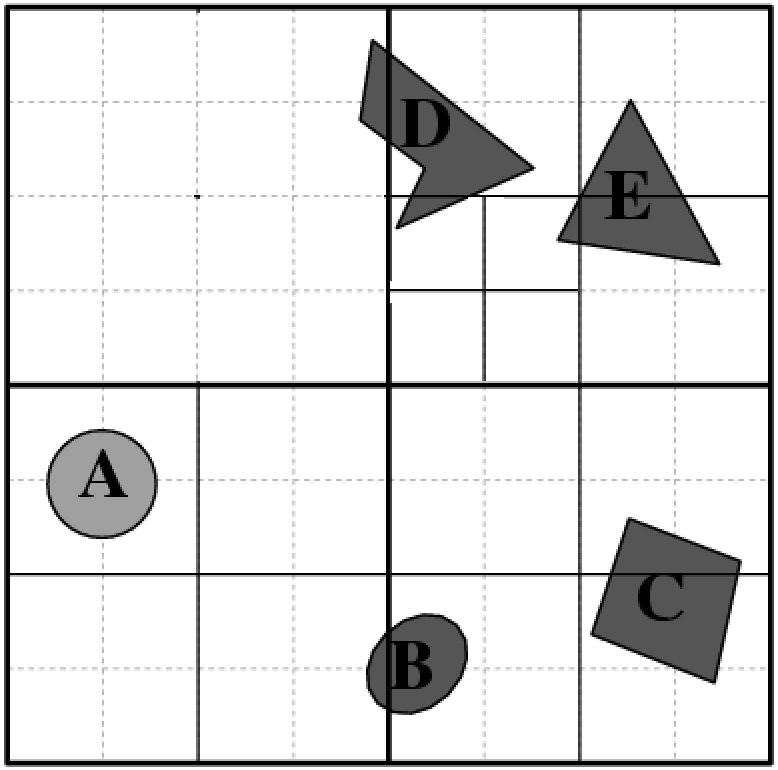
- Partition space using a 3D *hierarchical grid*
  - Tree with 8 children per node





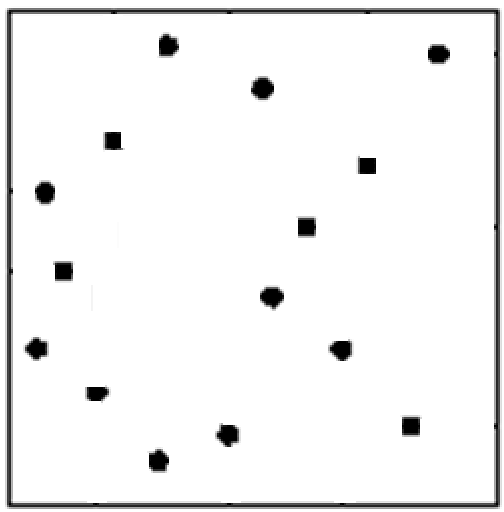
Octrees for Scene Graph Hierarchy

- Octree construction (Quadtree in 2D)
  - Generate octree for models until no cell contains more than one model
  - Group models/nodes in the same cell at the same level

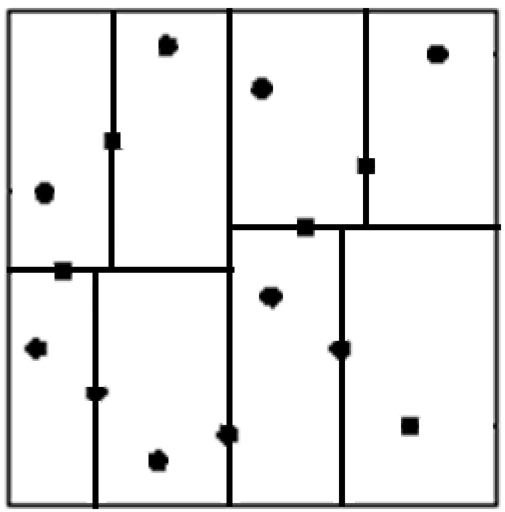


kD-Trees

- Input: n points in k dimensions
- Output: tree that *partitions space at axis-aligned planes*
  - Each point is contained in its own box-shaped region



input

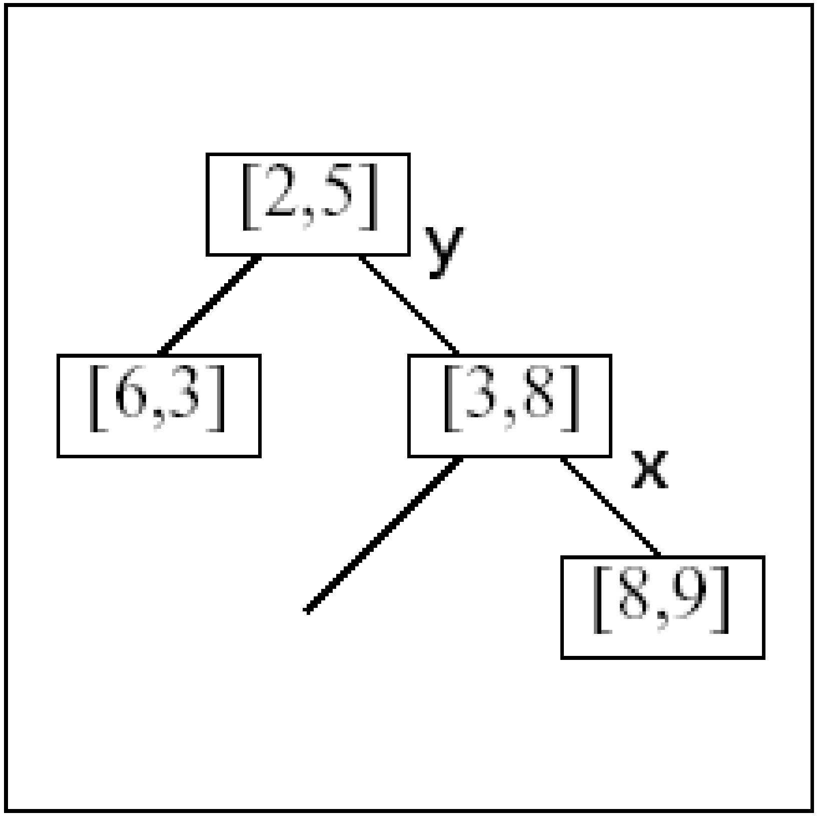


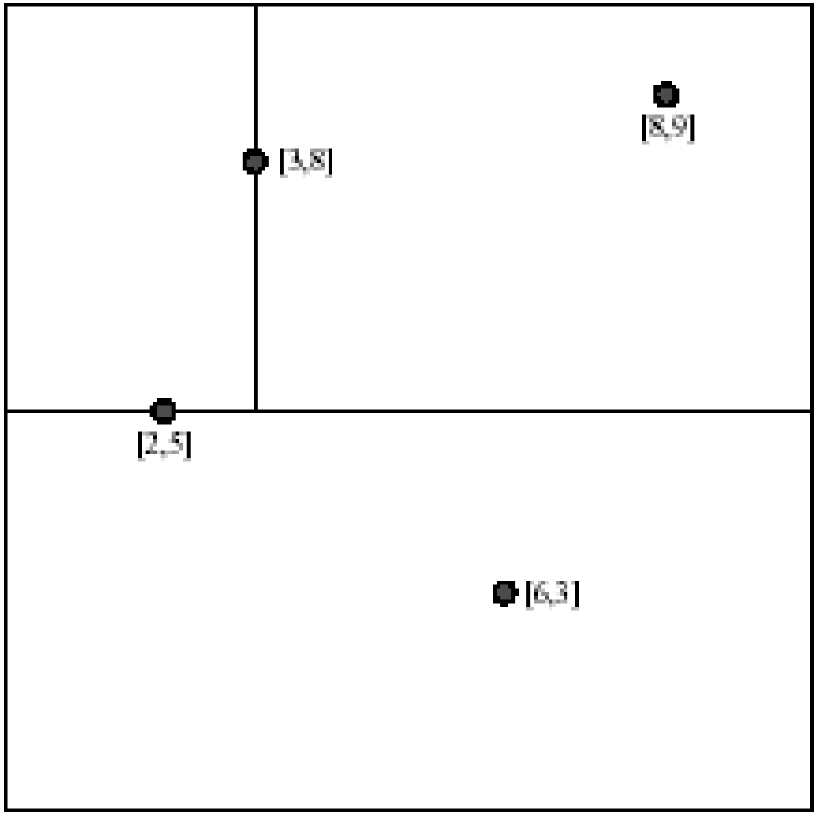
output

kD-Trees

- Generalisation of *binary search trees*
  - At each node find a point which *separates* remaining points into two (approximately) equal sized sets
- In k dimensions, repeat per level:
  - *Choose* one dimension
  - *Sort* points in 1D
  - *Split* points at median
- Choice of dimension:
  - Regular, e.g. x, y, z, x, y, . . .
  - Dimension where distance between points is maximal
  - Some other clever strategy. . .

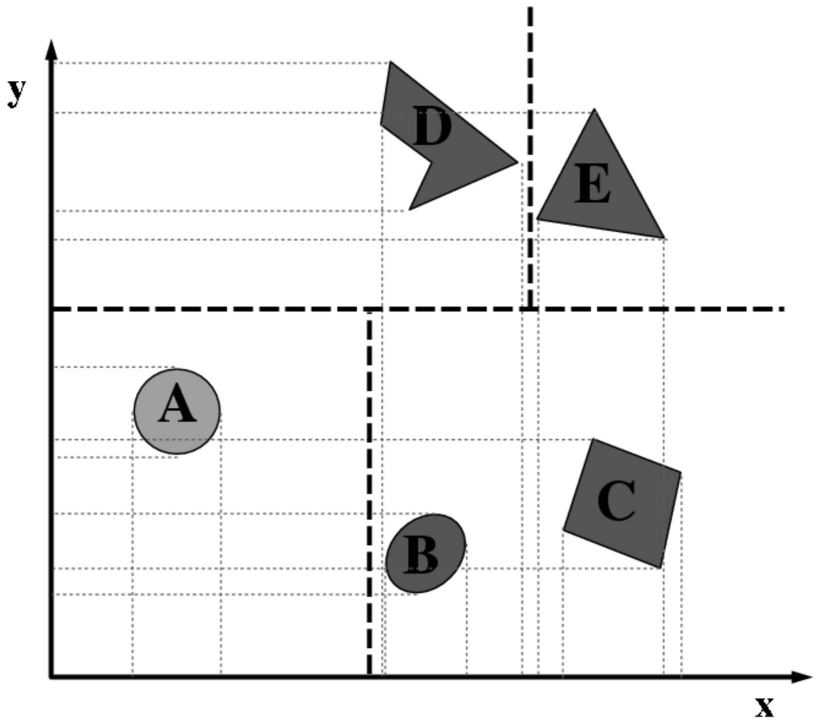
kD-Trees





kD-Tree Generalisation

- kD-Trees can be generalised to handle models
  - Median cut in x, then y, . . .
  - Search for best gaps for a small set of plane orientations

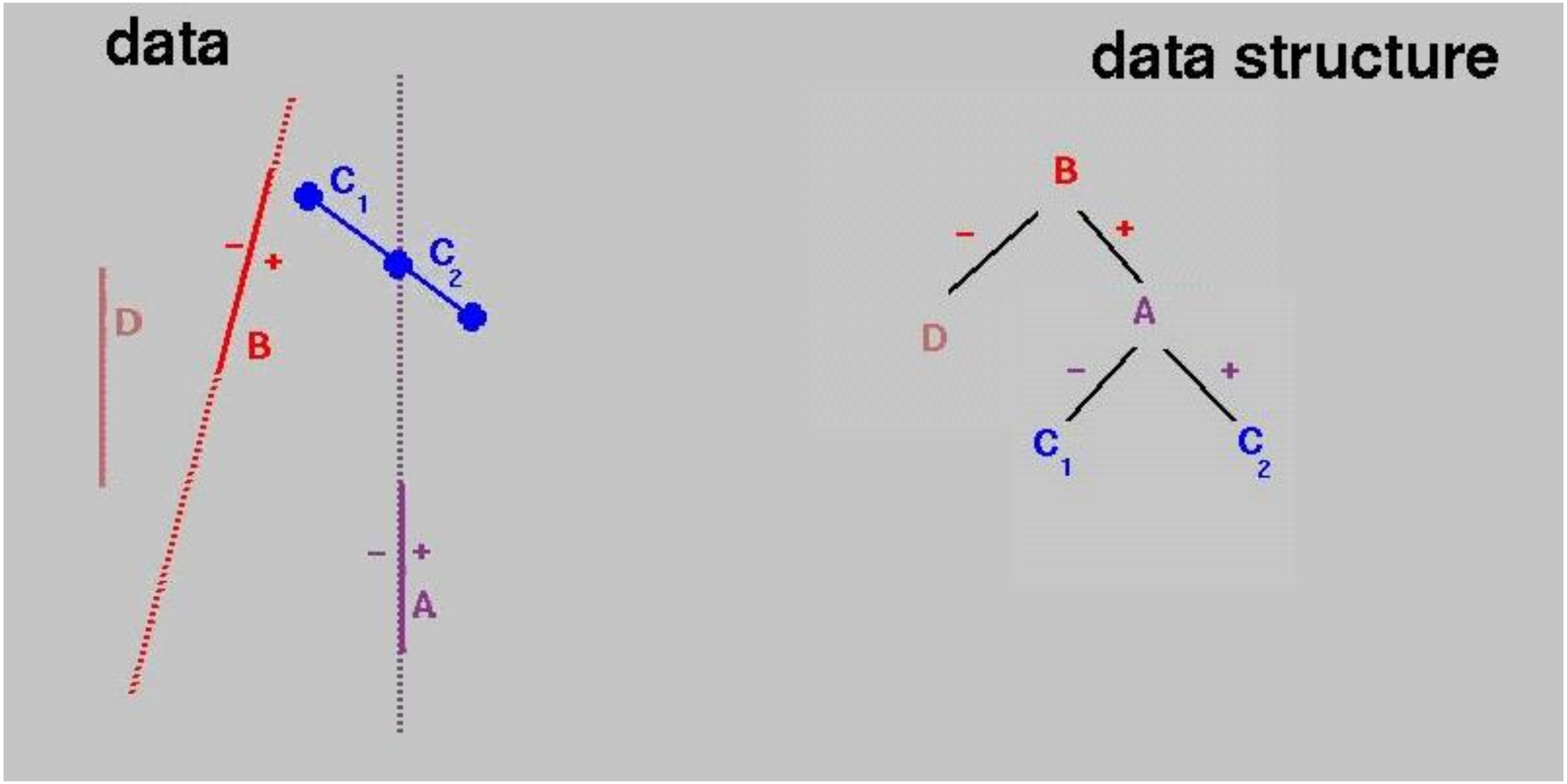


- kD-Tree gives hierarchy for scene graph

BSP-Trees

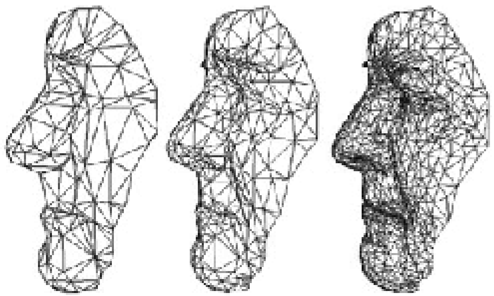
- Use a *binary space partitioning* (BSP) tree to order models
- Identify planes to *partition objects* into those in front of and those behind these planes hierarchically
  - For polygons we can choose one of them to define a *partitioning plane*
  - Polygons intersecting the plane are cut in two
  - Recursively continue splitting the polygon sets
- Particularly useful when view point changes, but objects remain at same position (partitioning does not change)
- kD-tree is a special case of BSP-tree

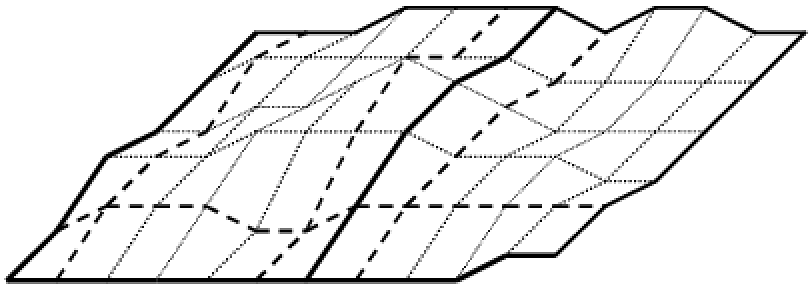
BSP Tree Example

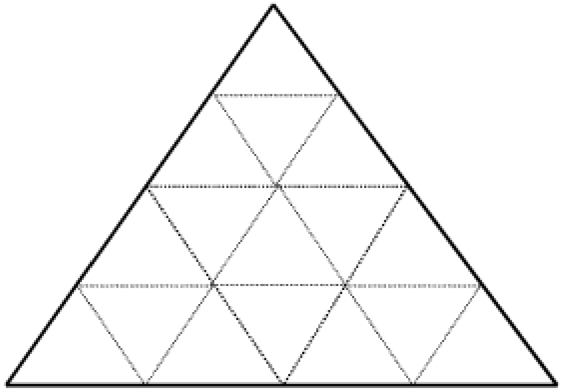


Multi-resolution models

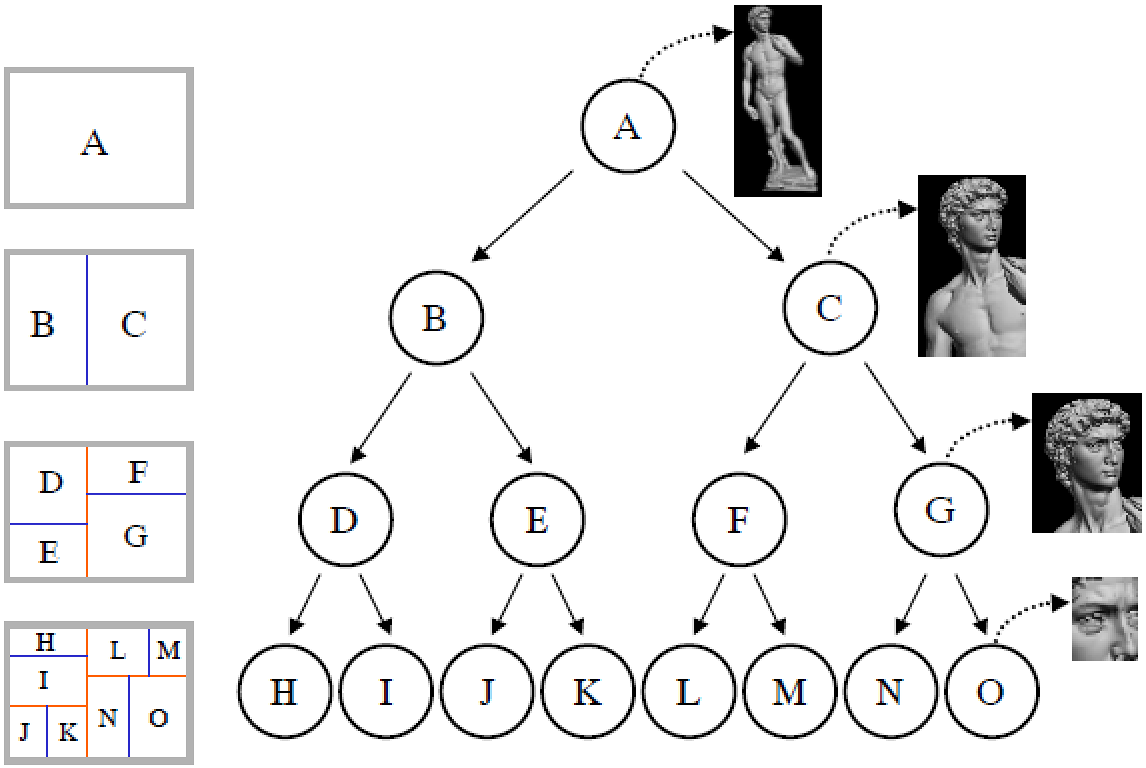
- Hierarchical representation also suitable for simple *multi-resolution models*
  - Represent model at different levels of detail (LOD) for efficient rendering and processing



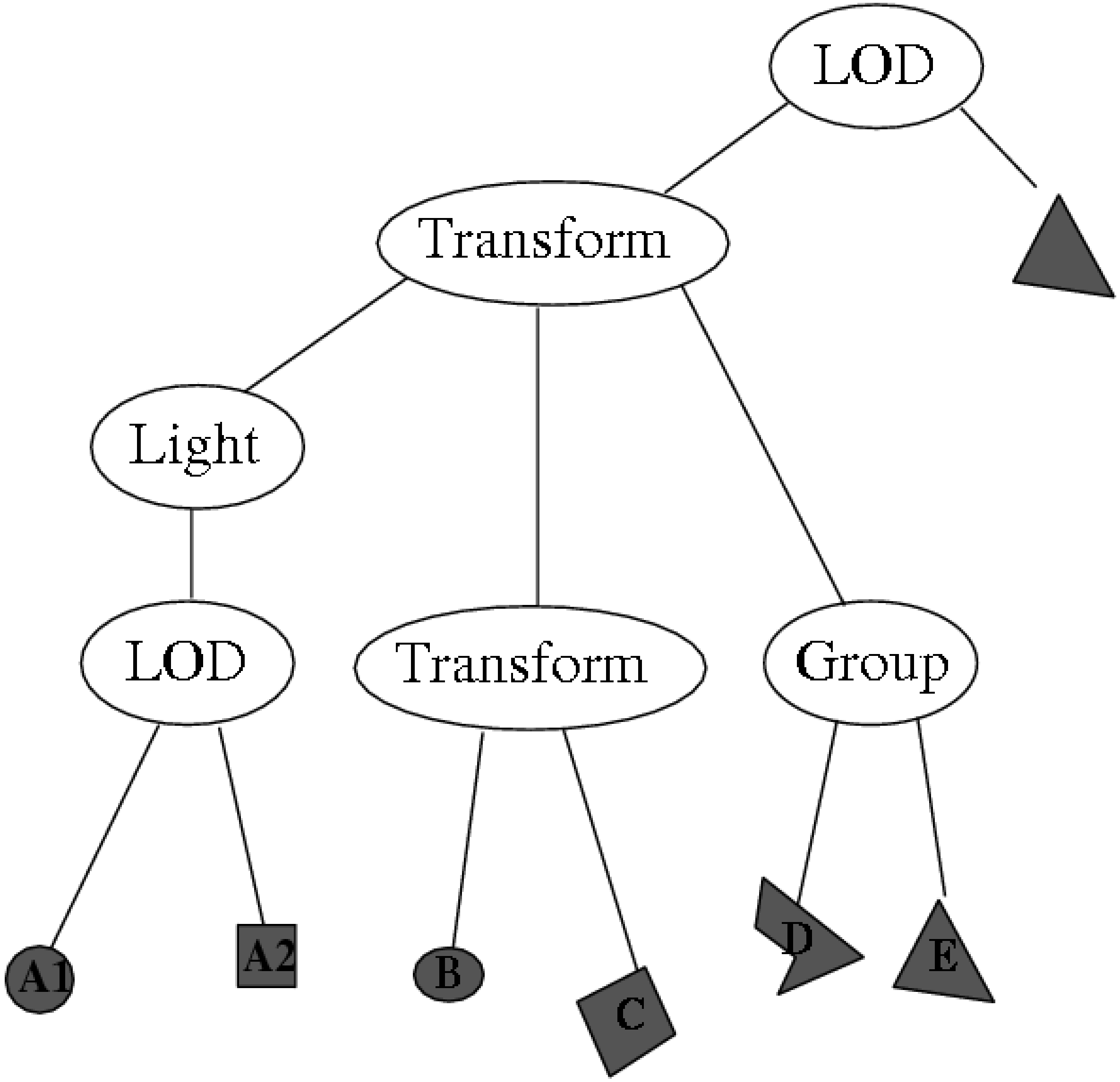




Multi-resolution Scene Graph



Multi-resolution Scene Graph



Scene Graph Issues

- *Minimise transformations*
  - Each transformation is expensive during rendering, etc.
  - Need automatic algorithms to reduce transformation nodes
- *Minimise attribute changes* (materials, etc.)
  - Each state change is expensive during rendering
- Many more scene graph optimisation problems. . .

Summary

- What is a scene graph / tree?
- Explain the principles of the following spatial data structures:
  - Uniform grid
  - Octree
  - kD-tree
  - BSP-tree
  - Given a set of objects, how are these data structures constructed?
  - How can these data structures be used to improve scene graph performance?