Scene Graphs

COMP 575 / COMP 770
Scene Graphs

• Good background at Wikipedia: http://en.wikipedia.org/wiki/Scene_graph

1. A scene graph is a collection of nodes in a graph or tree structure
2. Nodes in a scene graph (generally) represent entities or objects in the scene
3. Define logical relationships
   e.g. between a knight and a horse so that the knight is considered an extension to the horse;
4. Define spatial relationships
5. Spatial hierarchies and memory overhead
Data structures with transforms

- Representing a drawing (“scene”)
- List of objects
- Transform for each object
  - can use minimal primitives: ellipse is transformed circle
  - transform applies to points of object
Example

• Can represent drawing with flat list
  – but editing operations require updating many transforms
Groups of objects

• Treat a set of objects as one
• Introduce new object type: group
  – contains list of references to member objects
• This makes the model into a tree
  – interior nodes = groups
  – leaf nodes = objects
  – edges = membership of object in group
Example

• Add group as a new object type
  – lets the data structure reflect the drawing structure
  – enables high-level editing by changing just one node
The Scene Graph (tree)

• A name given to various kinds of graph structures (nodes connected together) used to represent scenes

• Simplest form: tree
  – just saw this
  – every node has one parent
  – leaf nodes are identified with objects in the scene
Concatenation and hierarchy

• Transforms associated with nodes or edges
• Each transform applies to all geometry below it
  – want group transform to transform each member
  – members already transformed—concatenate
• Frame transform for object is product of all matrices along path from root
  – each object’s transform describes relationship between its local coordinates and its group’s coordinates
  – frame-to-canonical transform is the result of repeatedly changing coordinates from group to containing group
Instances

• Simple idea: allow an object to be a member of more than one group at once
  – transform different in each case
  – leads to linked copies
  – single editing operation changes all instances
Example

• Allow multiple references to nodes
  – reflects more of drawing structure
  – allows editing of repeated parts in one operation
The Scene Graph (with instances)

• With instances, there is no more tree
  – an object that is instanced multiple times has more than one parent
• Transform tree becomes DAG
  – directed acyclic graph
  – group is not allowed to contain itself, even indirectly
• Transforms still accumulate along path from root
  – now paths from root to leaves are identified with scene objects
Implementing a hierarchy

• Object-oriented language is convenient
  – define shapes and groups as derived from single class

```java
abstract class Shape {
    void draw();
}

class Square extends Shape {
    void draw() {
        // draw unit square
    }
}

class Circle extends Shape {
    void draw() {
        // draw unit circle
    }
}
```
Implementing traversal

• Pass a transform down the hierarchy
  – before drawing, concatenate

abstract class Shape {
  void draw(Transform t_c);
}

class Square extends Shape {
  void draw(Transform t_c) {
    // draw t_c * unit square
  }
}

class Circle extends Shape {
  void draw(Transform t_c) {
    // draw t_c * unit circle
  }
}

class Group extends Shape {
  Transform t;
  ShapeList members;
  void draw(Transform t_c) {
    for (m in members) {
      m.draw(t_c * t);
    }
  }
}
Basic Scene Graph operations

• Editing a transformation
  – good to present usable UI

• Getting transform of object in canonical (world) frame
  – traverse path from root to leaf

• Grouping and ungrouping
  – can do these operations without moving anything
    – group: insert identity node
    – ungroup: remove node, push transform to children

• Reparenting
  – move node from one parent to another
  – can do without altering position
Adding more than geometry

- Objects have properties besides shape
  - color, shading parameters
  - approximation parameters (e.g. precision of subdividing curved surfaces into triangles)
  - behavior in response to user input
  - ...

- Setting properties for entire groups is useful
  - paint entire window green

- Many systems include some kind of property nodes
  - in traversal they are read as, e.g., “set current color”
Scene Graph variations

• Where transforms go
  – in every node
  – on edges
  – in group nodes only
  – in special Transform nodes

• Tree vs. DAG

• Nodes for cameras and lights?
Spatial Hierarchies

• Broad classification:
  – Spatial hierarchies
    • Grids
    • Octrees
    • Kd-trees, BSP trees
  – Object hierarchies
    • Bounding volume hierarchies
    • Spatial kd-trees
Spatial hierarchies: grids

• Regular subdivision of space into cells
  – Cells almost always cubes
  – Each object is referenced in each cell it overlaps
  – Nested grids also possible
Spatial hierarchies: kd-trees

- Binary tree of space subdivisions
  - Each is axis-aligned plane
Traversing a kd-tree: recursive
- Start at root node
- For current node:
  - If inner node (for ray tracing):
    - Find intersection of ray with plane
    - If ray intersects both children, recurse on near side, then far side
    - Otherwise, recurse on side it intersects
  - If leaf node:
    - Intersect with all object. If hit, terminate.
Kd-tree traversal

• Simple and fast implementation
  – In practice: using stack, not recursion
  – Very quick intersection test (couple FLOPS + tests)

• Overall: logarithmic complexity for each ray or intersection test
Object hierarchies: BVHs

• Different approach:
  subdivide objects, not space
  – Hierarchical clustering of objects
  – Each cluster represented by bounding volume
  – Binary tree
    • Each parent node fully contains children
Bounding volumes

• Practically anything can be bounding volume
  – Just need ray intersection method
• Typical choices:
  – Spheres
  – Axis-aligned bounding boxes (AABBs)
  – Oriented bounding boxes (OBBs)
  – k-DOPs
• Trade-off between intersection speed and how closely the BV encloses the geometry
BVH traversal

• Recursive algorithm:
  – Start with root node
  – For current node (ray tracing):
    • Does ray intersect node’s BV? If no, return
    • Is inner node?
      – Yes, recurse on children
    • Is leaf node?
      – Intersect with object(s) in node, store intersection results
• Widely used for view frustum culling or collision checking
Choosing a structure

• There is no ‘best’ acceleration structure
  – All have pros and cons
• Grid:
  + fast construction
  - bad for high local detail (teapot/stadium)
Choosing a structure

• There is no ‘best’ acceleration structure
  – All have pros and cons
• kd-tree:
  + fast traversal
  - expensive build, only static scenes
Choosing a structure

• There is no ‘best’ acceleration structure
  – All have pros and cons

• BVH:
  + can be updated for dynamic scenes
  - traversal more expensive than kd-tree