Graphics & Visualization

Chapter 9

Scene Management

Introduction

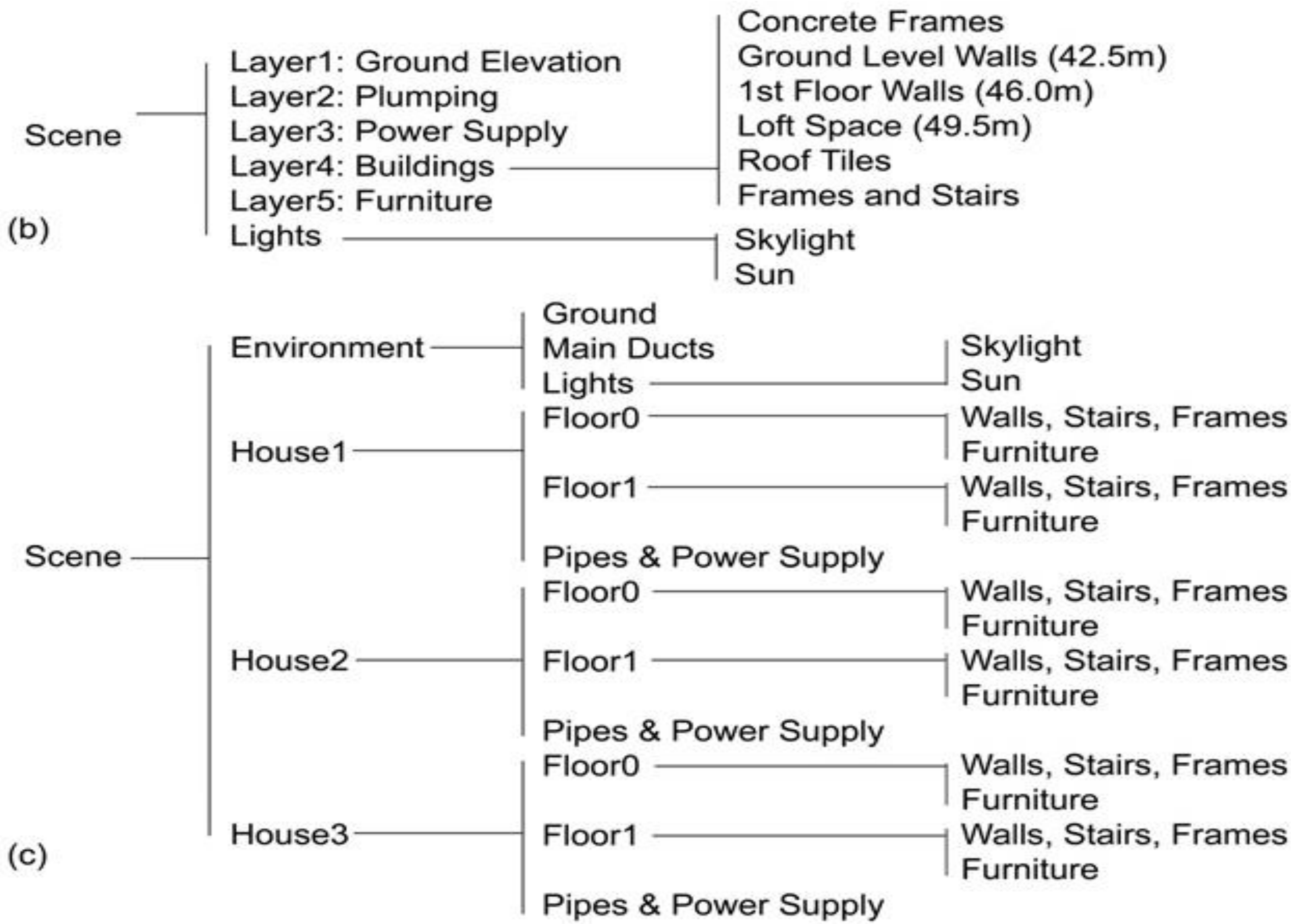
- Scene management:
  - Primitives of a scene are gathered in spatially coherent clusters
  - Applied recursively
- Why is scene management useful?
  - Hierarchy implies efficiency
    - Batch removal in frustum culling
    - Batch transformations
    - Easy management as memory objects (dynamic loading, caching, etc.)
- Virtual world
  - Designed in ontological terms; entities grouped according to logical relationships, vs
- Spatial partitioning (Chapter 5)
  - Data organized in spatially coherent manner

Introduction (2)

- Spatially coherent hierarchies provide significant performance → invisible geometry can be culled early at a high hierarchy level
- Ontological hierarchies are not necessarily optimized for spatial partitioning
- For real-time graphics applications, the common practice is to build scenes as *ontological hierarchies with spatial-coherency priority*
- However, excessive decomposition of a scene can also slow down rendering:
  - The application spend too much time in the traversal of the scene hierarchy
  - In hardware-accelerated graphics, partitioning can lead to poor geometry streams and frequent state changes

Example

Example (2)



Architectural View: no spatial coherence

Spatially coherent scene partitioning

Scene Graphs

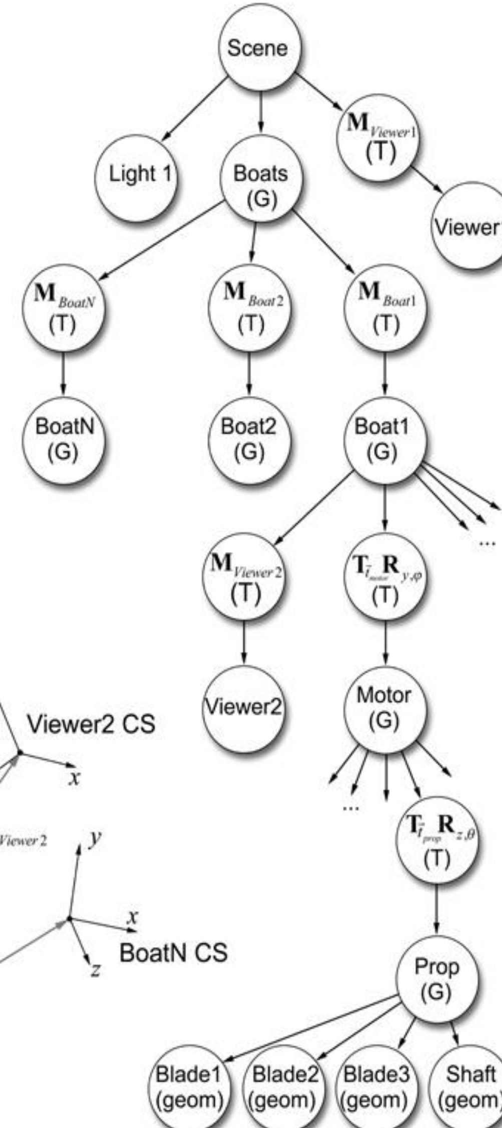
- *Scene graph (SG)*: a hierarchy of geometric elements that are related in an ontological manner *and* are spatially dependent
- It consists of nodes that can represent:
  - Geometric elements (static or animated)
  - Aggregations of nodes
  - Transformations
  - Conditional selections
  - Other renderable entities (e.g. sounds)
  - Pure simulation task nodes
  - Other scene graphs
- SG is a directed, non-cyclic graph of nodes, whose arcs define geometrical or functional dependence of a child node to its parent
- The nodes encapsulate all the functionality required to define a behavior → they adhere to the *object – oriented programming model*

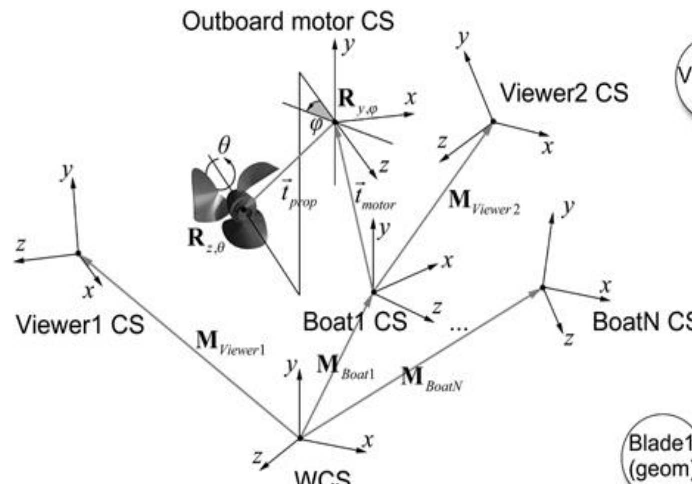
Scene Graphs (2)

- Root node is the abstract scene node and provides a single entry traversal point
- An operation performed on a node affects all of its children
  - Modeling of complex environments and their animation becomes easy
  - A means for construction of self-contained and reusable elements

Example

- What is the apparent motion of the speedboat propeller relative to the person on the dock or to the captain?





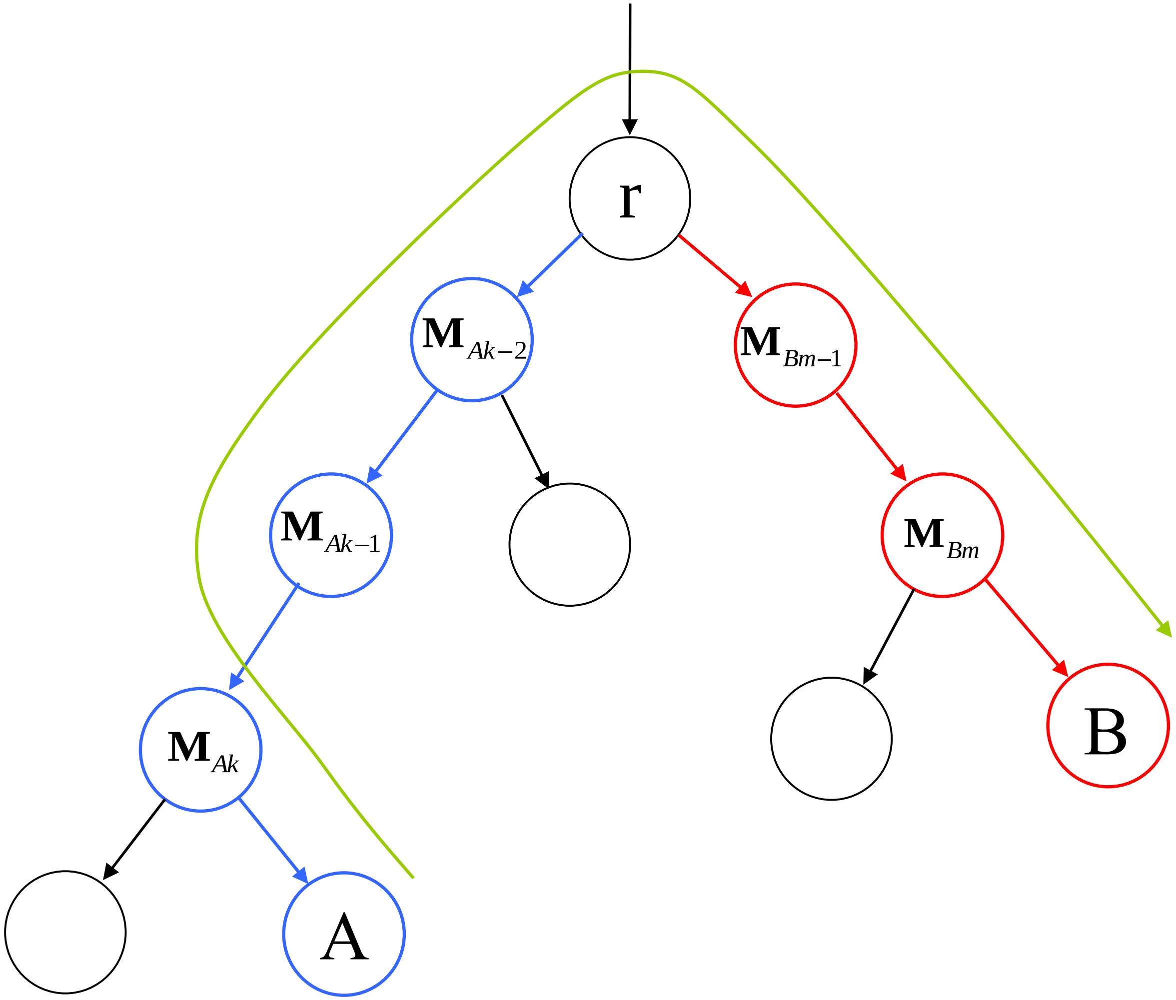
Node Relations

- We can express one node of the scene graph (target) relative to another (reference node) by a change of reference frame according to the intermediate transformations:
  - Perform an upward traversal of the SG from the target to the common parent
  - Descend to the reference node by inversely applying transformations
- Transformation of a node at level k on a branch A relative to another node at level m on a branch B with a common root at level r :

$$\mathbf{M}_{A\to B} = \prod_{j=m}^{r+1} \mathbf{M}_{Bj}^{-1} \prod_{i=r+1}^{k} \mathbf{M}_{Ai}$$

- A direct consequence of the duality between object transformations and axis transformations (change of reference frame)

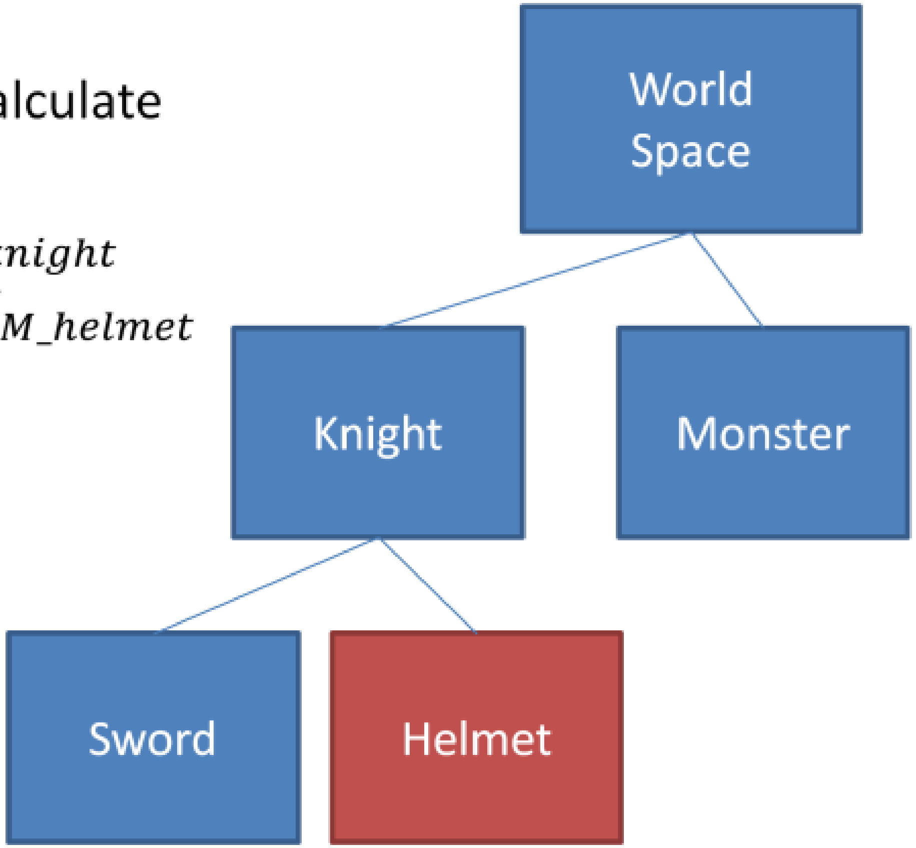
Node Relations (2)



Example: MVP for a sub-object

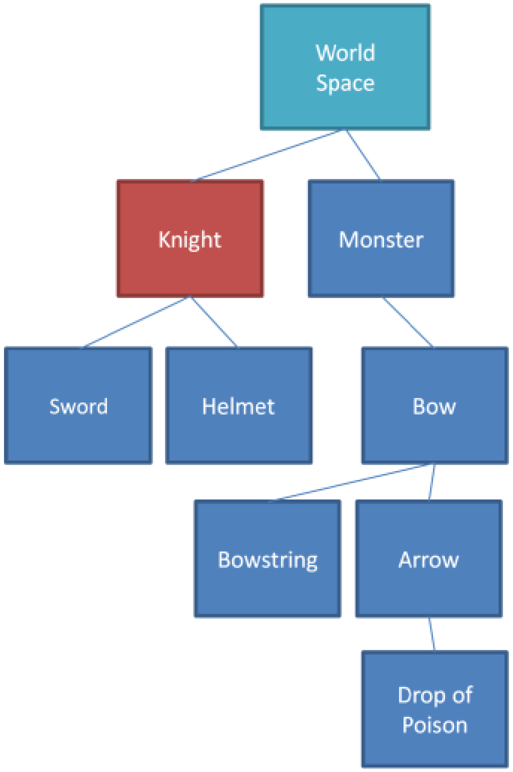
- A knight with a sword and a monster scene
  - To draw the helmet, we wish to calculate

$$M_{M\_helmet} = M_{Head-offset} * M_{M\_knight}$$
$$M_{MVP\_helmet} = M_{Proj} * M_{View} * M_{M\_helmet}$$



Example: Push & Pop

- Run through an SG is depth-first traversal




```
Iterate over all children in worldspace.
- Enter Knight node.
  - Calculate model_knight. Push it.
  - Draw knight using projection * view * model_knight.
  - Iterate over all children in the knight.
    - Enter Sword Node.
      - Calculate model_sword (sword * parent node). Push it.
      - Draw sword model using projection * view * model_sword.
      - Iterate over all children in the sword node. There were none
    - Pop sword matrix.
    - Enter Helmet Node.
      - Calculate model_helmet (helmet * parent node). Push it.
      - Draw helmet model using projection * view * model_helmet.
      - Iterate over all children in the helmetnode. There were none.
    - Pop helmet matrix.
-   Pop knight matrix.
- Enter Monster Node. Etc
```