#5: Models & Scenes

CSE167: Computer Graphics

Instructor: Ronen Barzel

UCSD, Winter 2006

Outline For Today

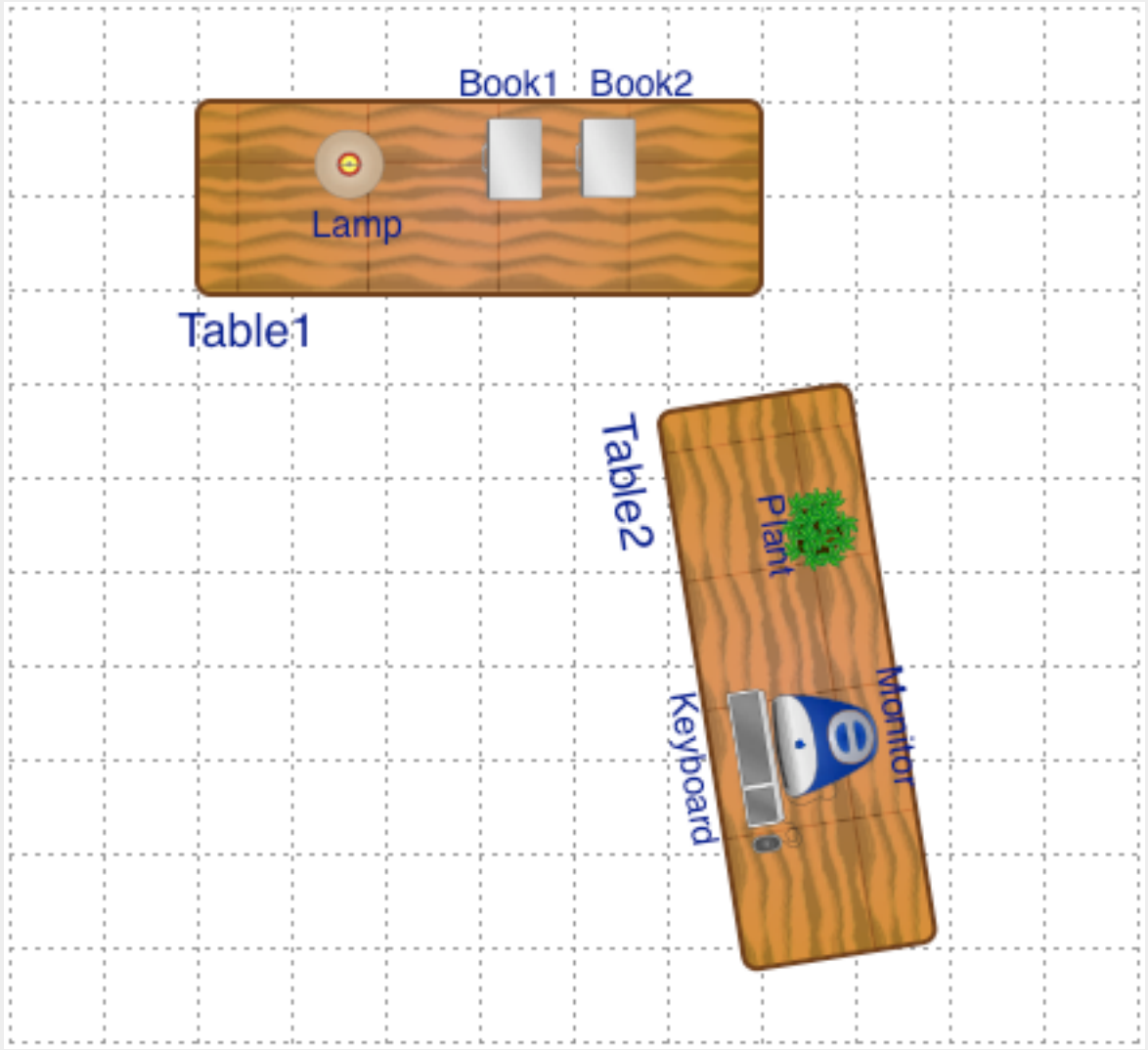
- *Scene Graphs*
- Shapes
- Tessellation

Modeling by writing a program

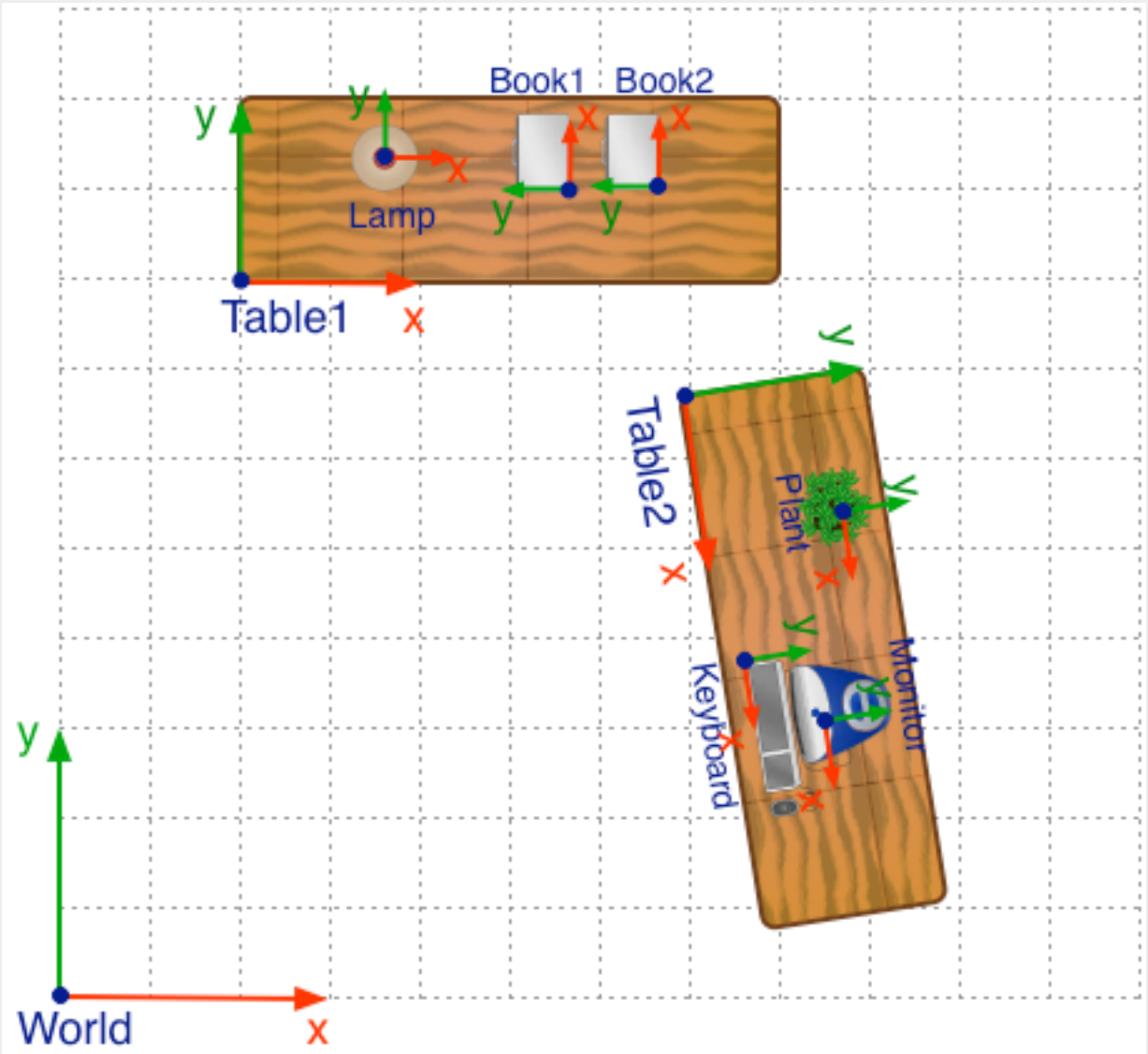
- First two projects: Scene hard-coded in the model
- The scene exists only in the drawScene() method
- Advantages:
  - Simple,
  - Direct
- Problems
  - Code gets complex
  - Special-purpose, hard to change
  - Special-purpose, hard to make many variants
  - Can't easily examine or manipulate models
    - Can only "draw"

Sample Scene

Schematic Diagram (Top View)

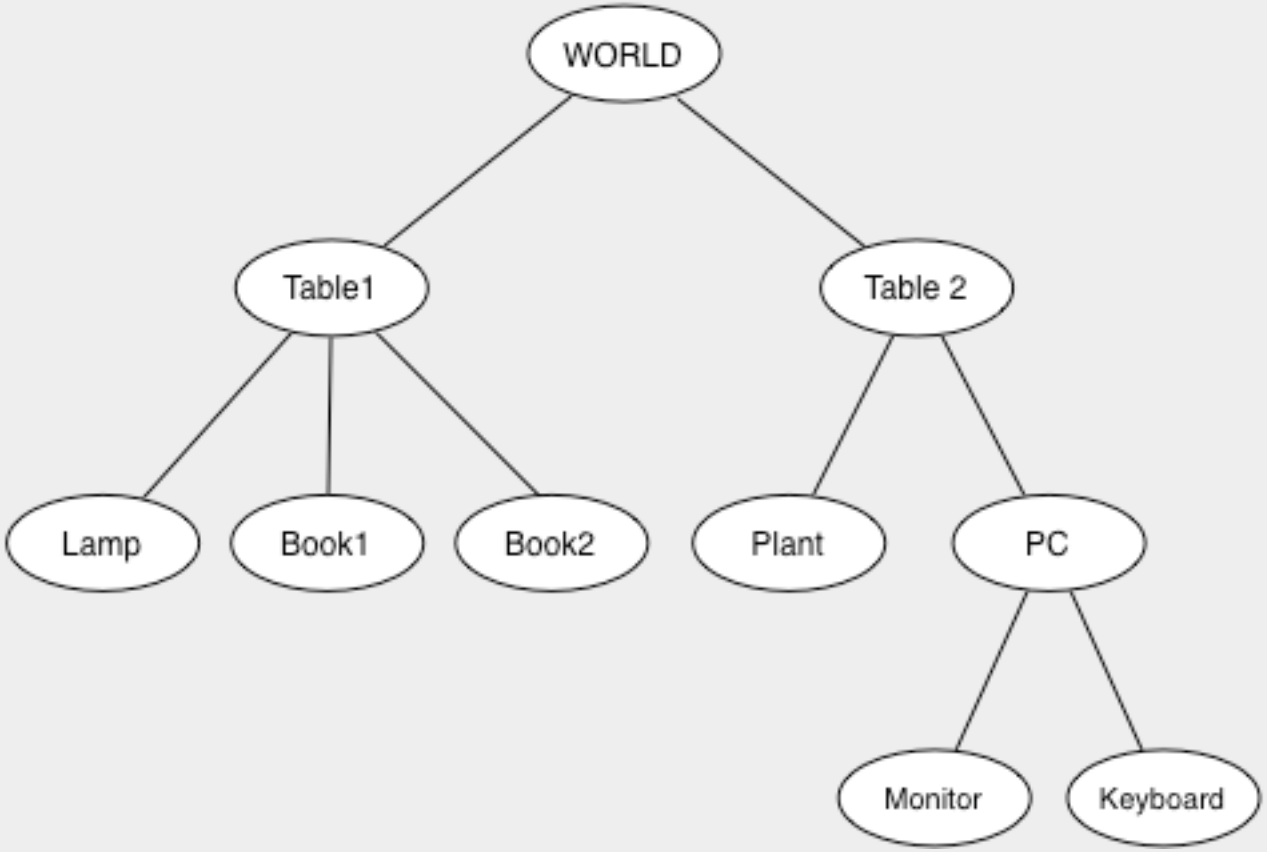


Top view with Coordinates



Hierarchical Transforms

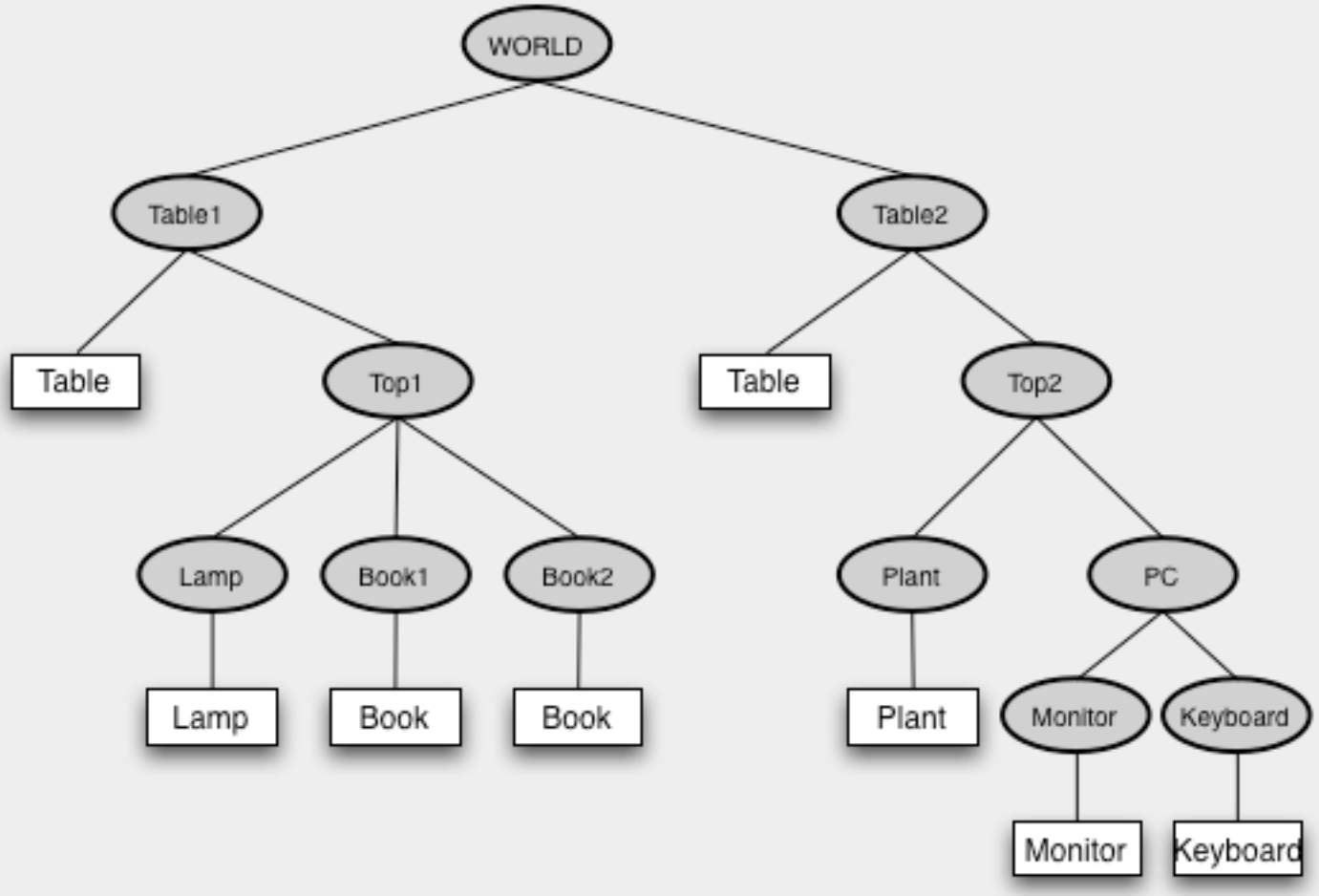
- Last week, introduced hierarchical transforms
- Scene hierarchy:



Data structure for hierarchical scene

- Want:
  - Collection of individual models/objects
  - Organized in groups
  - Related via hierarchical transformations
- Use a tree structure
- Each node:
  - Has associated local coordinates
  - Can define a shape to draw in local coordinates
  - Can have children that inherit its local coordinates
- Typically, different classes of nodes:
  - “Transform nodes” that affect the local coordinates
  - “Shape nodes” that define shapes

Scene Tree


WORLD
Table1
Table2
Table
Top1
Table
Top2
Lamp
Book1
Book2
Plant
PC
Lamp
Book
Book
Plant
Monitor
Keyboard
Monitor
Keyboard

Node base class

- A Node base class might support:
  - getLocalTransform() -- matrix puts node's frame in parent's coordinates
  - getGeometry() -- description of geometry in this node (later today)
  - getChild(i) -- access child nodes
    - addChild(), deleteChild() -- modify the scene
- Subclasses for different kinds of transforms, shapes, etc.
- Note: many designs possible
  - Concepts are the same, details differ
  - Optimize for: speed (games), memory (large-scale visualization), editing flexibility (modeling systems), rendering flexibility (production systems), …
  - In our case: optimize for pedagogy & projects

Node base class

```
class Node {
   // data
   Matrix localTransform;
   Geometry *geometry;
   Node *children[N];
   int numChildren;

    // methods:
   getLocalTransform() { return localTransform; }
   getGeometry() { return geom; }
   getChild(i) { return children[i]; }
   addChild(Node *c) { children[numChildren++] = c; }
}
```

Draw by traversing the tree

```
draw(Node node) {
    PushCTM();
    Transform(node.getLocalTransform());
    drawGeometry(node.getGeometry());
    for (i=0; i<node.numChildren; ++i) {
       draw(node.child[i]);
    }
    PopCTM();
  }
```

- Effect is same hierarchical transformation as last week

Modify the scene

- Change tree structure
  - Add nodes
  - Delete nodes
  - Rearrange nodes
- Change tree contents
  - Change transform matrix
  - Change shape geometry data
- Define subclasses for different kinds of nodes
  - Subclass has parameters specific to its function
  - Changing parameter causes base info to update

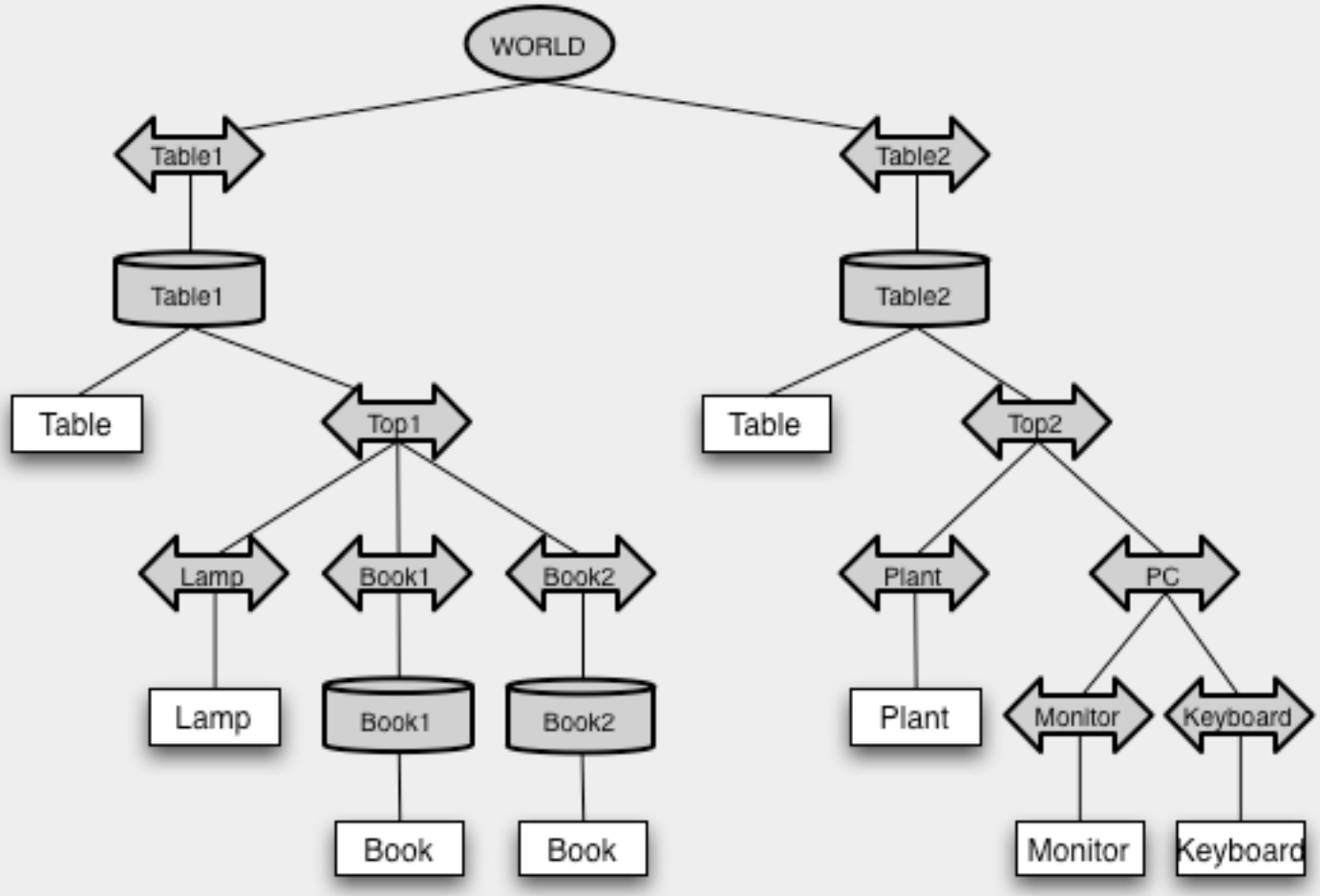
Example: Translation Node

```
class Translation(Transformation) {
   private:
      float x,y,z;
      void update() {
         localTransfom.MakeTranslation(x,y,z);
      }

   public:
     void setTranslation(float tx, float ty, float tz) {
        x = tx; y = ty; z = tz;
        update();
     }
     void setX(float tx) { x = tx; update(); }
     void setY(float ty) { y = ty; update(); }
     void setZ(float tz) { z = tz; update(); }
}
```

Example: Rotation Node

```
class Rotation(Transformation) {
   private:
      Vector3 axis;
      float angle;
      void update() {
         localTransfom.MakeRotateAxisAngle(axis,angle);
      }

   public:
     void setAxis(Vector3 v) {
        axis = v;
        axis.Normalize();
        update();
     }
     void setAngle(float a) {
        angle = a;
        localTransfom.MakeRotateAxisAngle(axis,angle);
     }
   }
```

More detailed scene graph



Building this scene

```
WORLD = new Node();
table1Trans = new Translation(…);  WORLD.addChild(table1Trans);
table1Rot = newRotation(…); table1Trans.addChild(table1Rot);
table1 = makeTable(); table1Rot.addChild(table1);
top1Trans = new Translation(…); table1Rot.addChild(top1Trans);

lampTrans = new Translation(…); top1Trans.addChild(lampTrans);
lamp = makeLamp(); lampTrans.addChild(lamp);

book1Trans = new Translation(…);  top1Trans.addChild(book1Trans);
book1Rot = newRotation(…); book1Trans.addChild(book1Rot);
book1 = makebook(); book1Rot.addChild(book1);

book2Trans = new Translation(…);  top1Trans.addChild(book2Trans);
book2Rot = newRotation(…); book2Trans.addChild(book2Rot);
book2 = makebook(); book2Rot.addChild(book1);

table2Trans = new Translation(…);  WORLD.addChild(table2Trans);
table2Rot = newRotation(…); table2Trans.addChild(table2Rot);
table2 = makeTable(); table2Rot.addChild(table2);
top2Trans = new Translation(…); table2Rot.addChild(top2Trans);
…
```

- Still building the scene hardwired in the program
  - But now can more easily manipulate it…

Change scene

- Change a transform in the tree:
  - `table1Rot.setAngle(23);`
  - Table rotates, everything on the table moves with it
- Allows easy animation
  - Build scene once at start of program
  - Update parameters to draw each frame
  - e.g. Solar system:

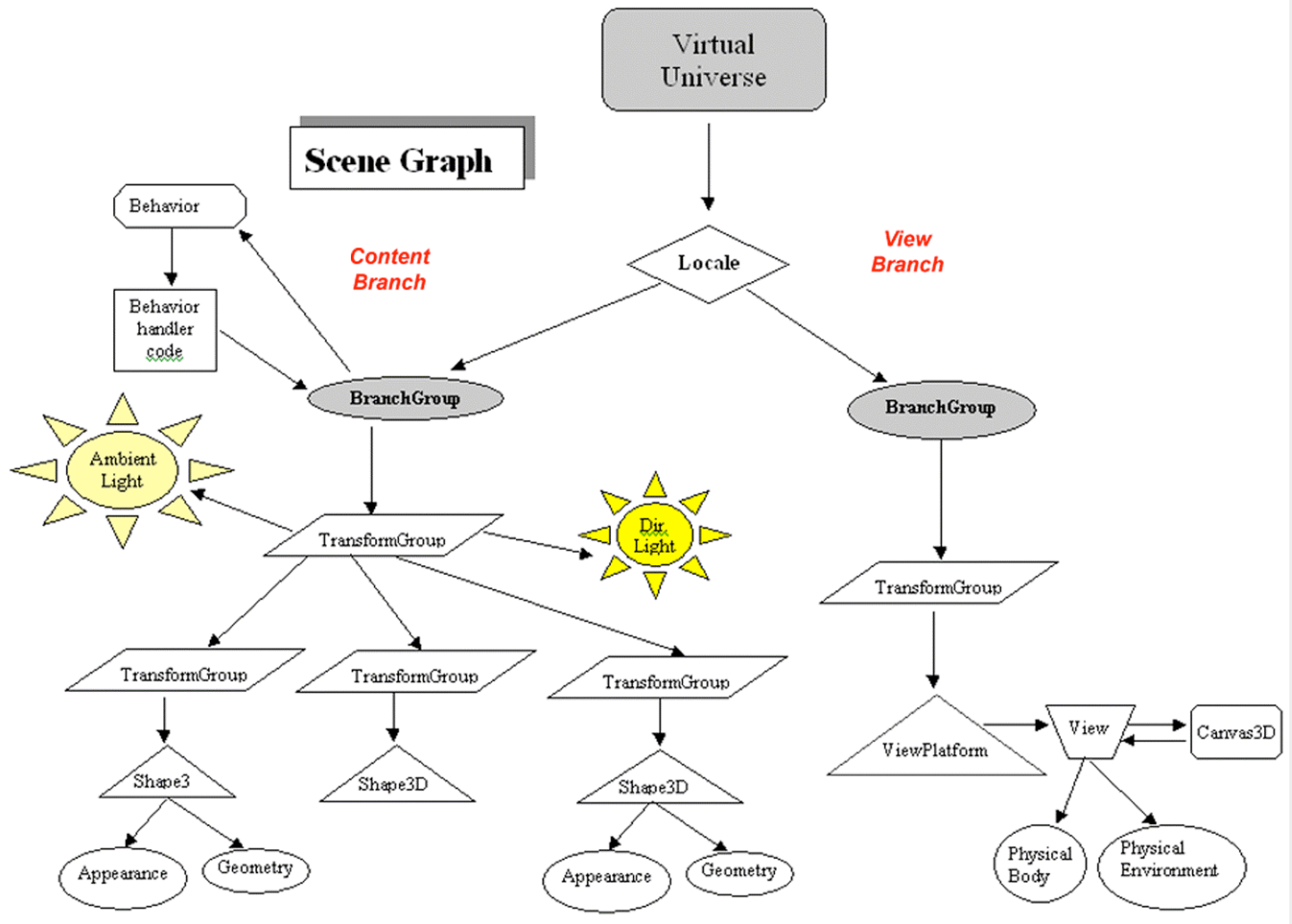
```
drawScene() {
    sunSpin.setAngle(g_Rotation);
    earthSpin.setAngle(3*g_Rotation);
    earthOrbit.setAngle(2*g_Rotation);
    moonOrbit.setAngle(8*g_Rotation);
    draw(WORLD);
  }
```

- Allows interactive model manipulation tools
  - e.g. button to add a book
    - Create subtree with transforms and book shape
    - Insert as child of table

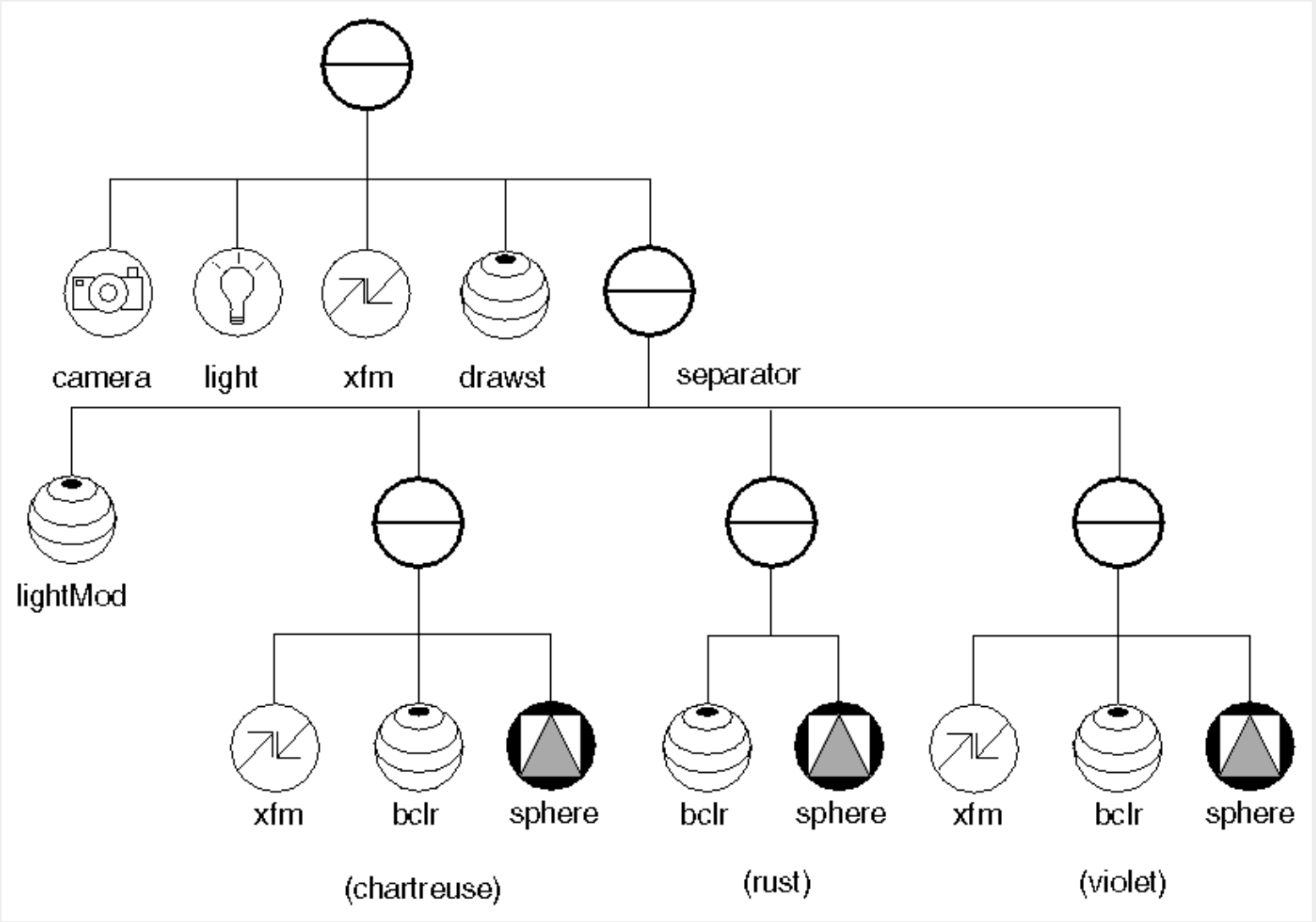
Not just transform nodes

- Shape nodes
  - Contain geometry:
    - cube, sphere (later today)
    - curved surfaces (next week)
    - Etc…
- Can have nodes that control structure
  - Switch/Select: parameters choose whether or which children to enable
  - Group nodes that encapsulate subtrees
  - Etc…
- Can have nodes that define other properties:
  - Color
  - Material
  - Lights
  - Camera
  - Etc…
- Again, different details for different designs

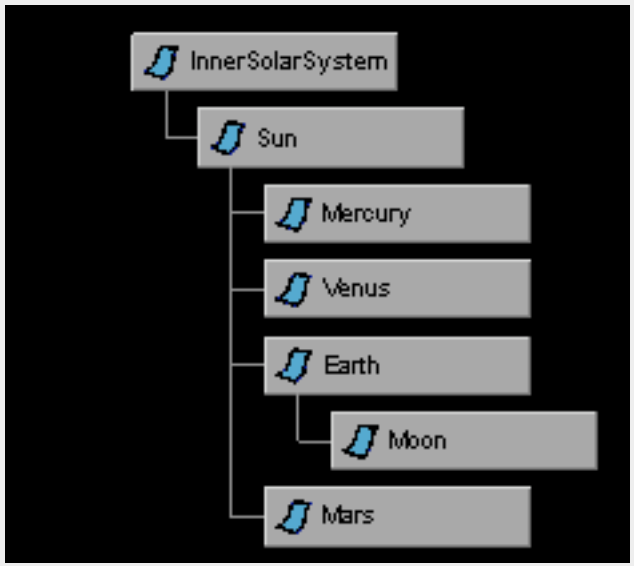
Java3D Scene Graph

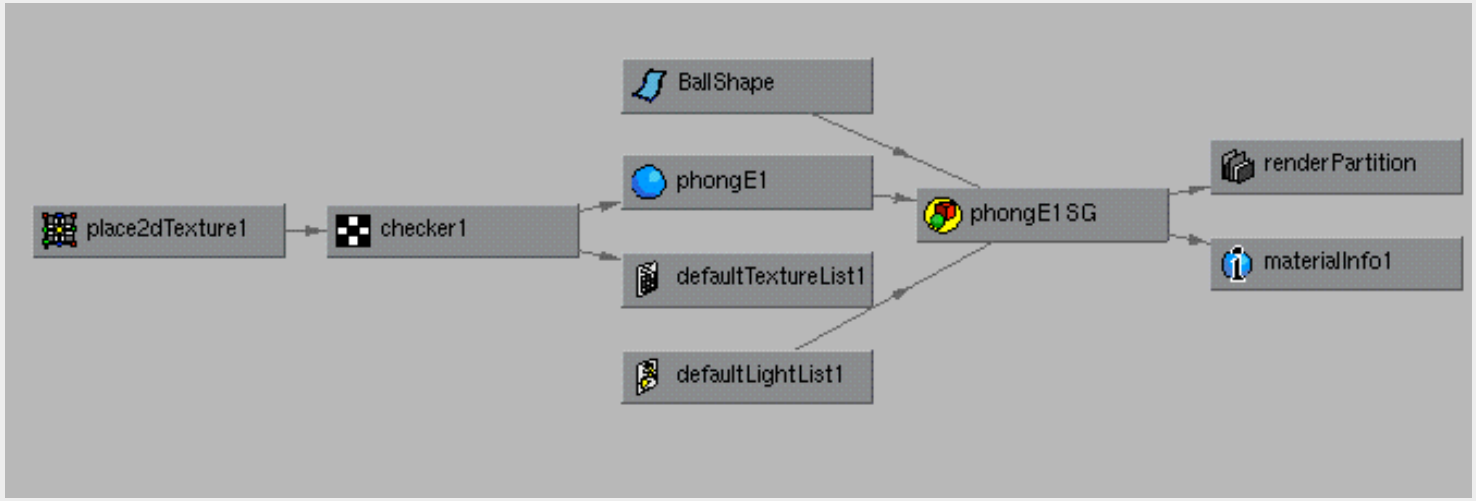


OpenInventor Scene Graph



Maya “Hypergraph”





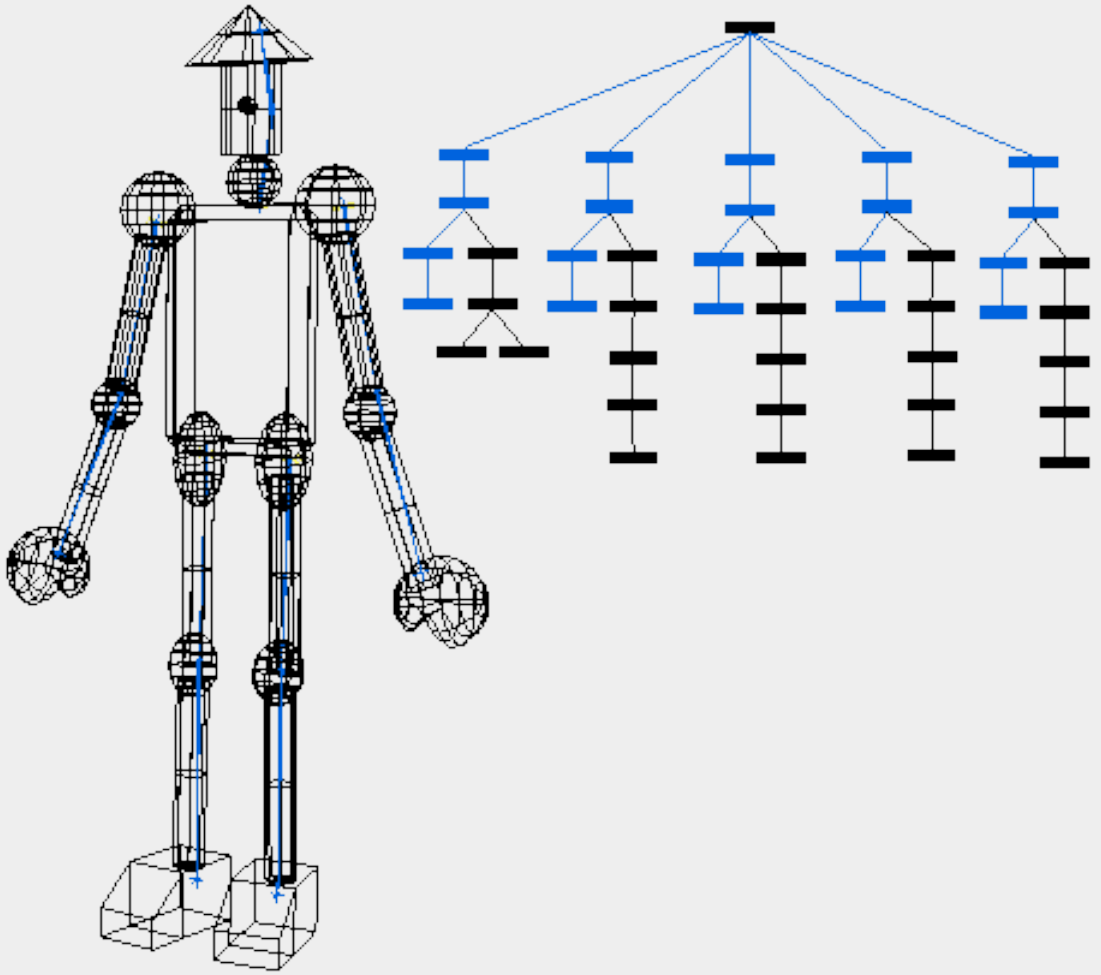
Scene vs. Model

- No real difference between a scene and a model
  - A scene is typically a collection of “models” (or “objects”)
  - Each model may be built from “parts”
- Use the scene graph structure
  - Scene typically includes cameras, lights, etc. in the graph; Model typically doesn’t (but can)

Parameteric models

- Parameters for:
  - Relationship between parts
  - Shape of individual parts
- Hierarchical relationship between parts
- Modeling robots
  - separate rigid parts
  - Parameters for joint angles
  - Hierarchy:
    - Rooted at pelvis: Move pelvis, whole body moves
    - Neck & Head: subtree; move neck and head, or just move head
    - Arms: Shoulder, Elbow, Wrist joints
    - Legs: Hips, Knee, Ankle joints
  - This model idiom is known as: an *Articulated figure*
  - Often talk about *degrees of freedom* (DOFs)
    - Total number of float parameters in the model

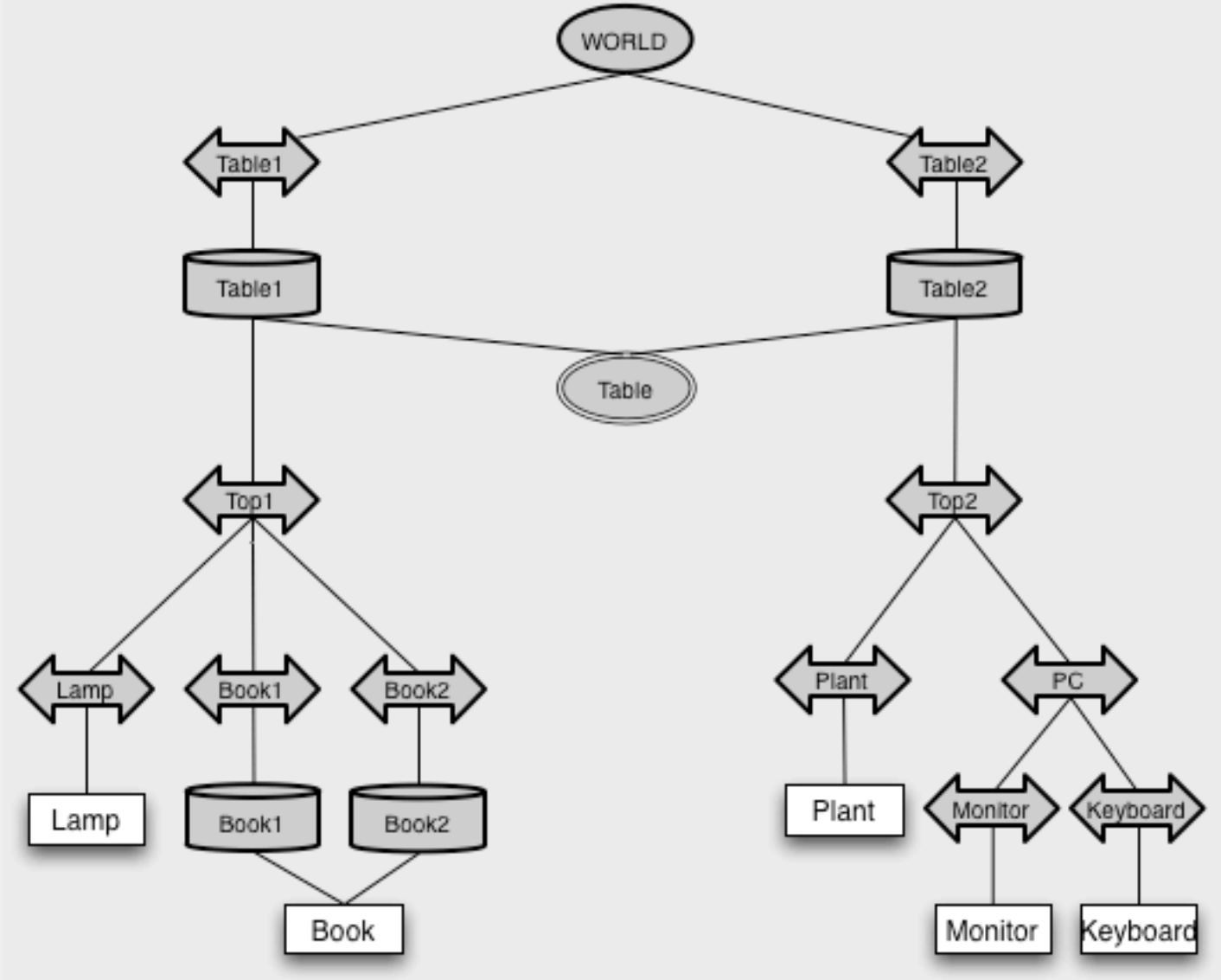
Robot



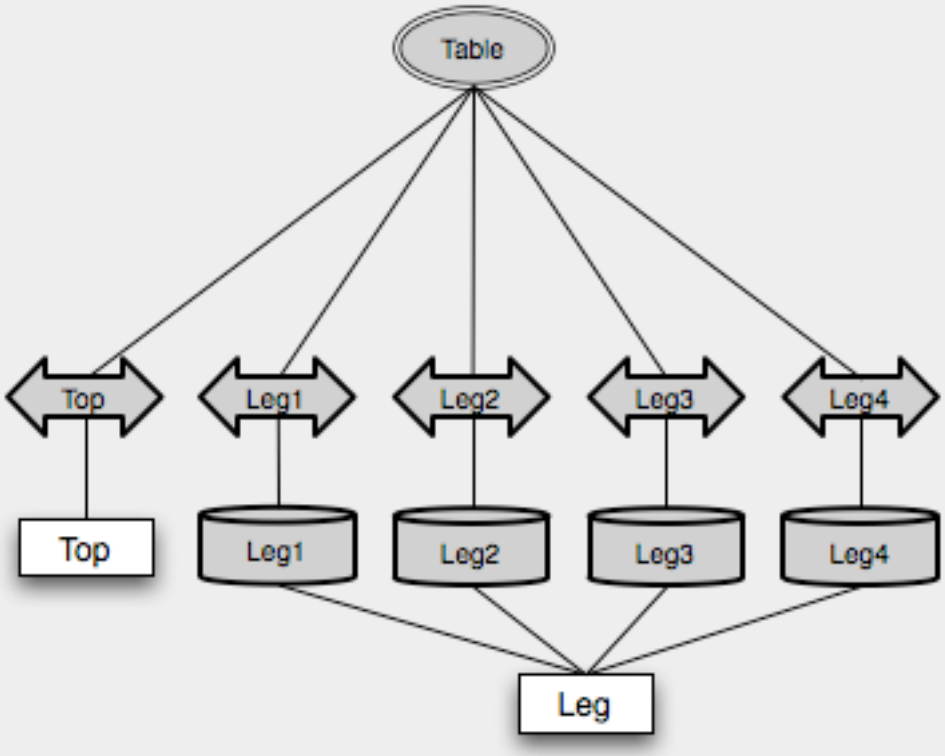
Screen *Graph*, not Tree

- Repetition:
  - A scene might have many copies of a model
  - A model might use several copies of a part
- *Multiple Instantiation*
  - One copy of the node or subtree
  - Inserted as a child of many parents
  - A directed acyclic graph (DAG), not a tree
  - Traversal will draw object each time, with different coordinates
- Saves memory
  - Can save time also, depending on cacheing/optimization
- Change parameter once, affects all instances
  - This can be good or bad, depending on what you want
  - Some scene graph designs let other properties inherit from parent

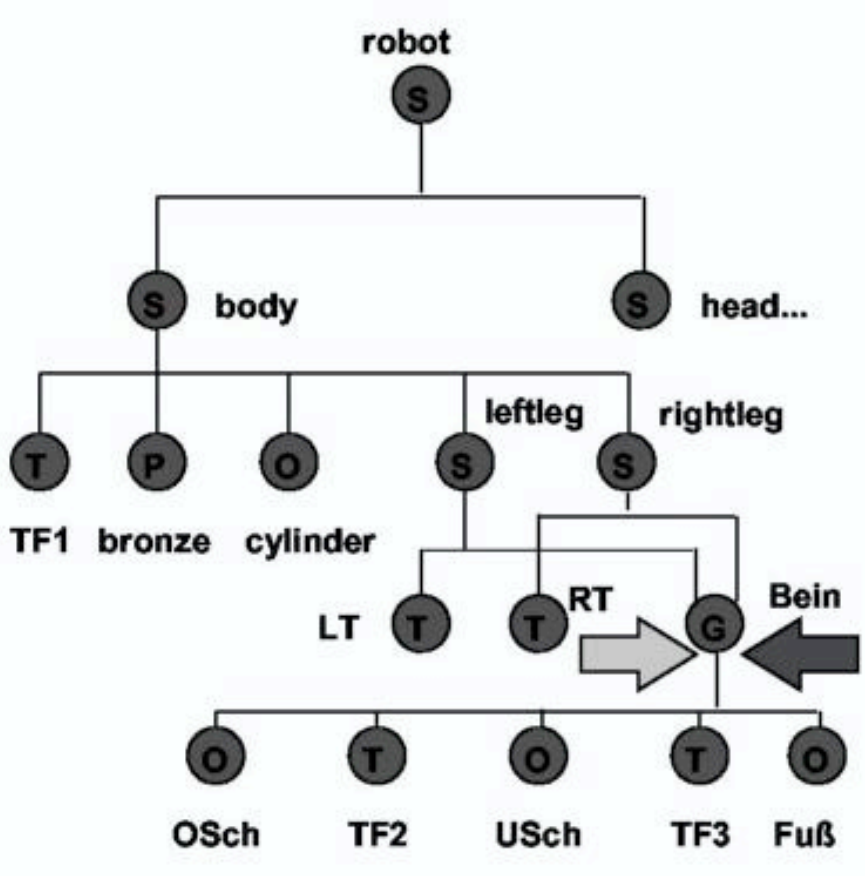
Instantiation - scene

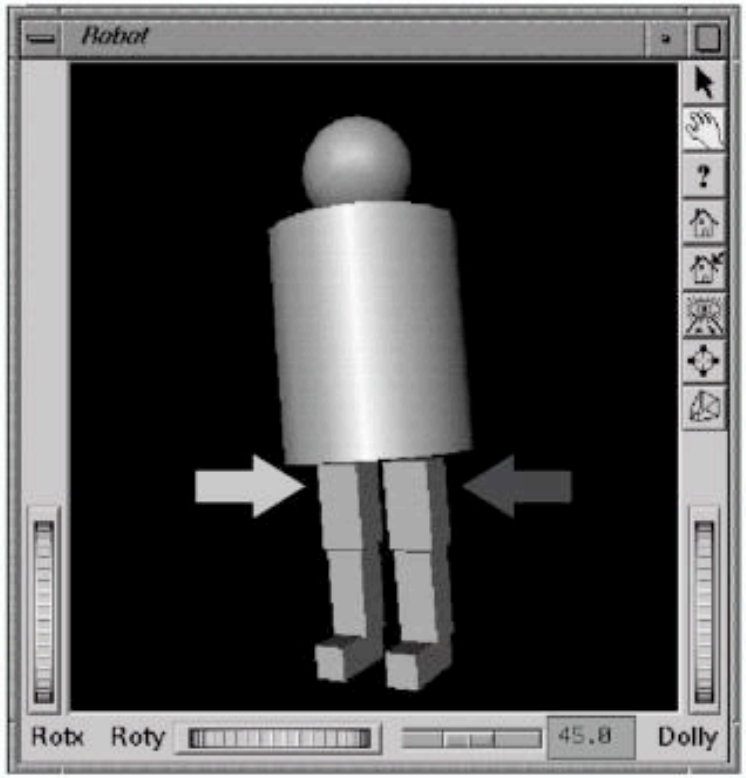


Instantiation - model parts



Instantiation (OpenInventor)

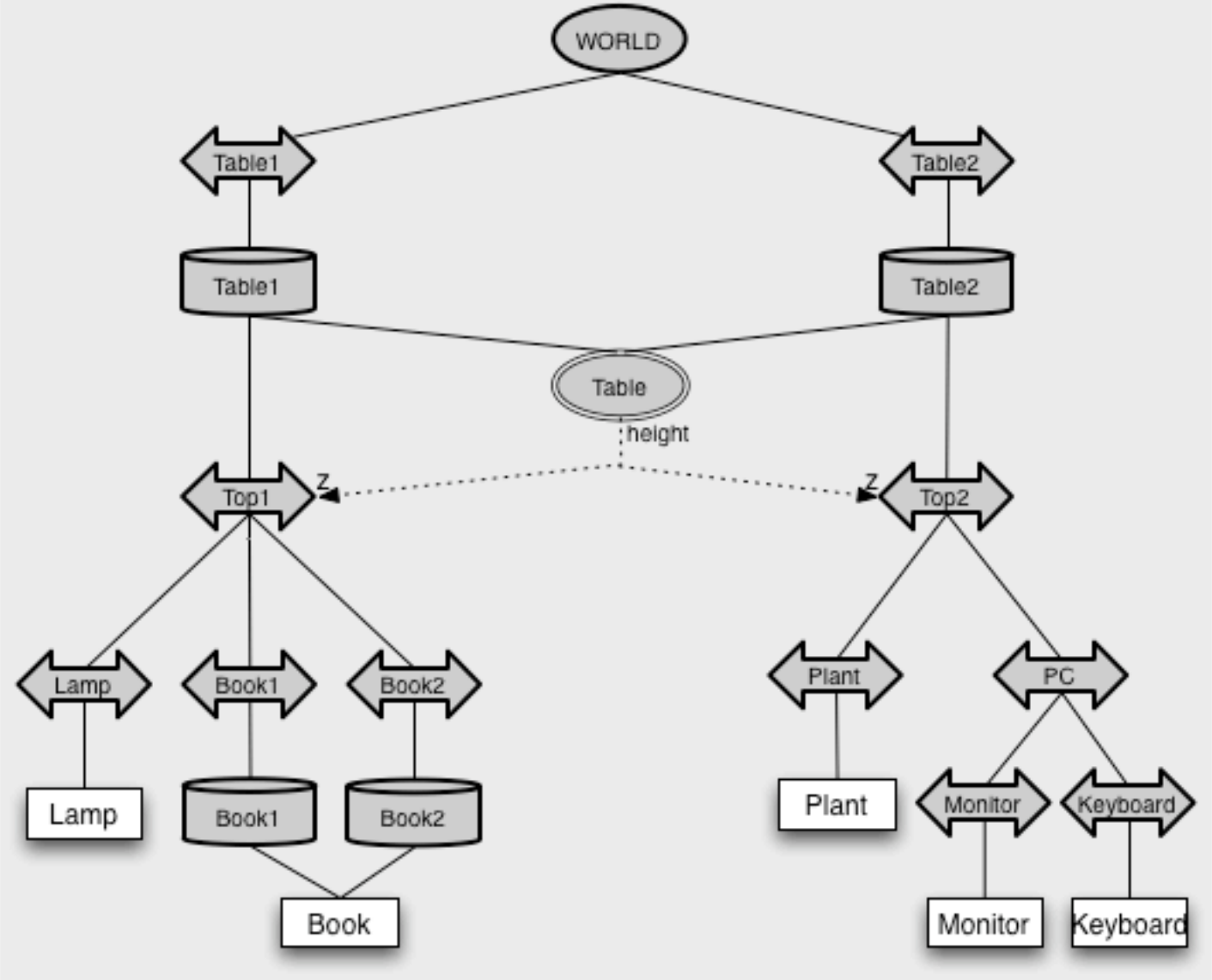




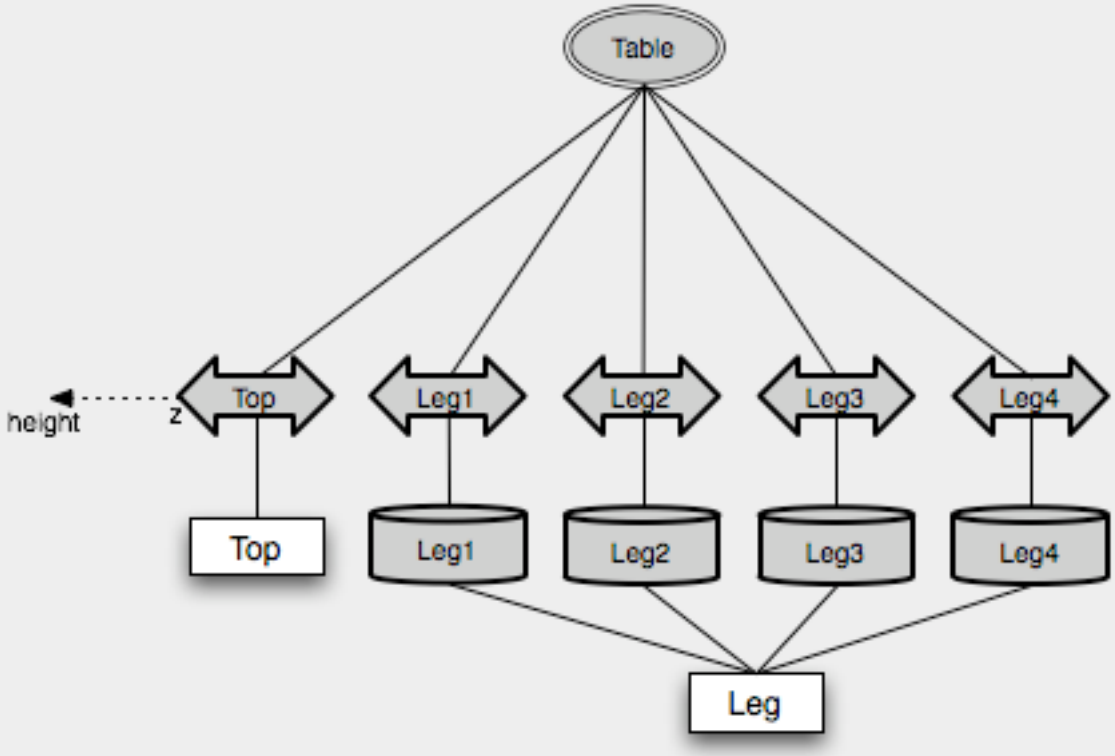
Fancier things to do with scene graphs

- Skeletons, skin, deformations
  - Robot-like jointed rigid skeleton
  - Shape nodes that put surface across multiple joint nodes
  - Nodes that change shape of geometry
- Computations:
  - Properties of one node used to define values for other nodes
  - Sometimes can include mathematical expressions
  - Examples:
    - Elbow bend angle -> bicep bulge
    - Our scene has translation to put objects on table…
      - But how much should that translation be?
      - What if the table changes?

Linked parameters



Linked parameters



Other things to do with scene graphs

- Names/paths
  - Unique name to access any node in the graph
  - e.g. “WORLD/table1Trans/table1Rot/top1Trans/lampTrans”
- Compute Model-to-world transform
  - Walk from node through parents to root, multiplying local transforms
- Bounding box or sphere
  - Quick summary of extent of object
  - Useful for culling (next class)
  - Compute hierarchically:
    - Bounding box is smallest box that encloses all children’s boxes
- Collision/contact calculation
- Picking
  - Click with cursor on screen, determine which node was selected
- Edit: build interactive modeling systems