CSE 167:
Introduction to Computer Graphics
Lecture #9: Advanced Textures

Jürgen P. Schulze, Ph.D.
University of California, San Diego
Fall Quarter 2010

Announcements

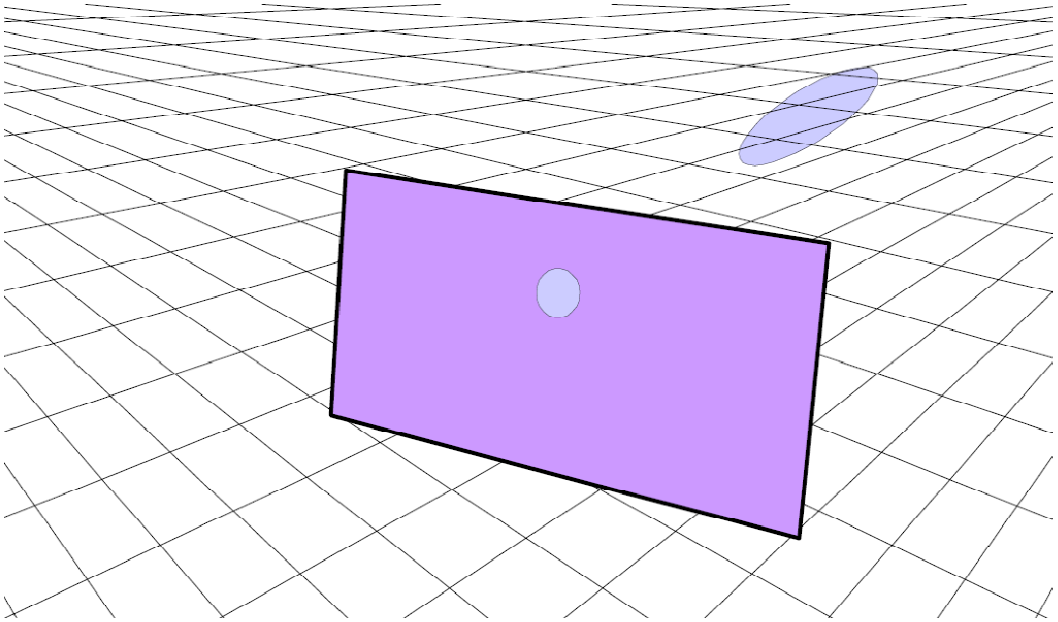
- Homework assignment #4 due Friday, Oct 29
- Office hours this week as usual

Lecture Overview

- Texturing
  - Anisotropic Texture Filtering
- Scene Graphs & Hierarchies
  - Introduction
  - Data structures

Mipmapping Limitations

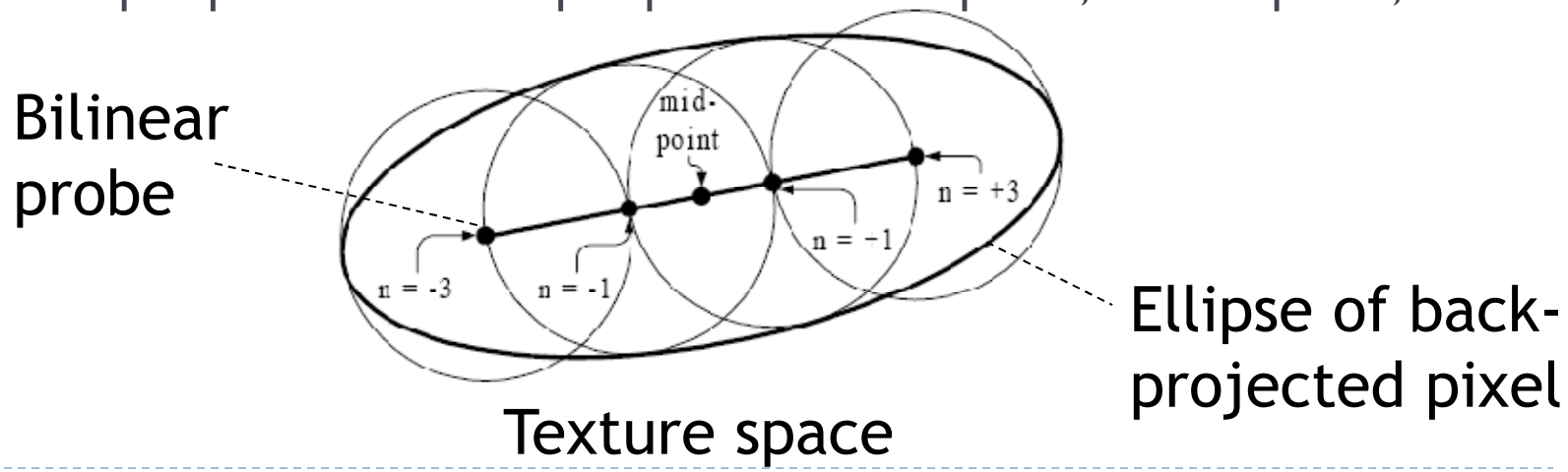
- Mipmap texels always represent square areas
- Pixel area is not always square in texture space
- Mipmapping tries to balance between aliasing effects and a fuzzy image



A circular area in the image plane
can be generated by an ellipse in object space

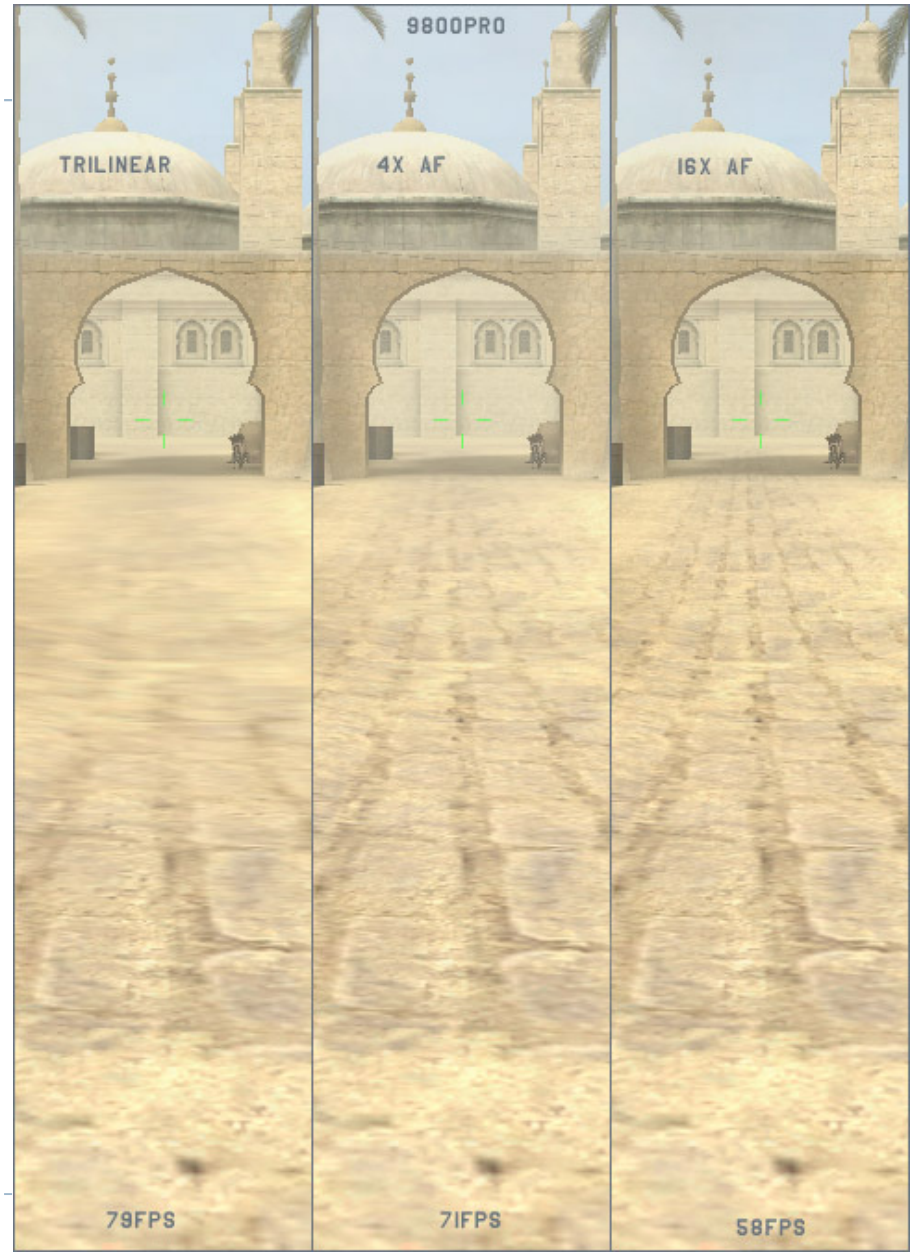
Anisotropic Texture Filtering

- Average texture over elliptical area
  - Higher quality than trilinear mip-mapping
  - More expensive
- Anisotropic filtering in hardware
  - Take several bilinear probes approximating the ellipse
  - Reduces rendering performance on current GPUs
  - Pre-calculates non-square mipmap textures: e.g., in addition to a 256x256 pixel mipmap it will store mipmaps of 256x128 pixels, 64x256 pixels, etc.



Example #1

Source: http://www.garry.tv



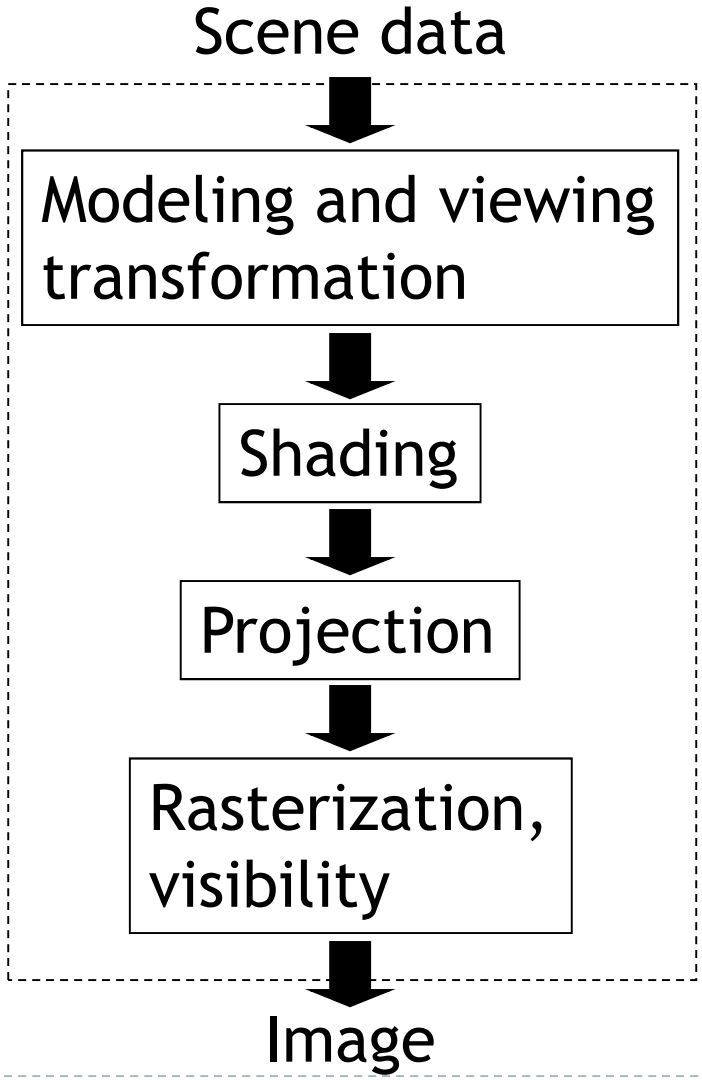
Example #2

Source: http://www.tomshardware.com/reviews/ati,819-5.html

Lecture Overview

- Texturing
  - Anisotropic Texture Filtering
- Scene Graphs & Hierarchies
  - Introduction
  - Data structures

Rendering Pipeline



System Architecture

Interactive Applications

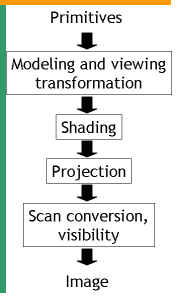
- Games, virtual reality, visualization

Rendering Engine, Scene Graph API

- Implement functionality commonly required in applications
- Back-ends for different low-level APIs

Low-level graphics API

- Interface to graphics hardware



System Architecture

Interactive Applications

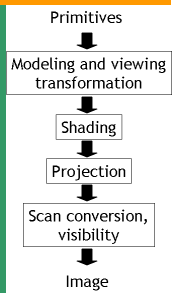
- Thousands

Rendering Engine, Scene Graph API

- No broadly accepted standards
- OpenSceneGraph, OpenSG, NVSG, Java3D, Ogre

Low-level graphics API

- Highly standardized: OpenGL, Direct3D



Scene Graph APIs

- APIs focus on different clients/applications
- Java3D (https://java3d.dev.java.net/)
  - Simple, easy to use, web-based applications
- OpenSceneGraph (www.openscenegraph.org)
  - Scientific visualization, virtual reality, GIS (geographic information systems)
- NVSG (http://developer.nvidia.com/object/scenix-home.html)
  - Optimized for Nvidia graphics cards
  - Up-to-date shader support (Cg 2.2)
- Ogre3D (http://www.ogre3d.org/)
  - Games, high-performance rendering

Common Functionality

- Resource management
  - Content I/O (geometry, textures, materials, animation sequences)
  - Memory management
- High-level scene representation
  - Scene graph
- Rendering
  - Efficiency

Lecture Overview

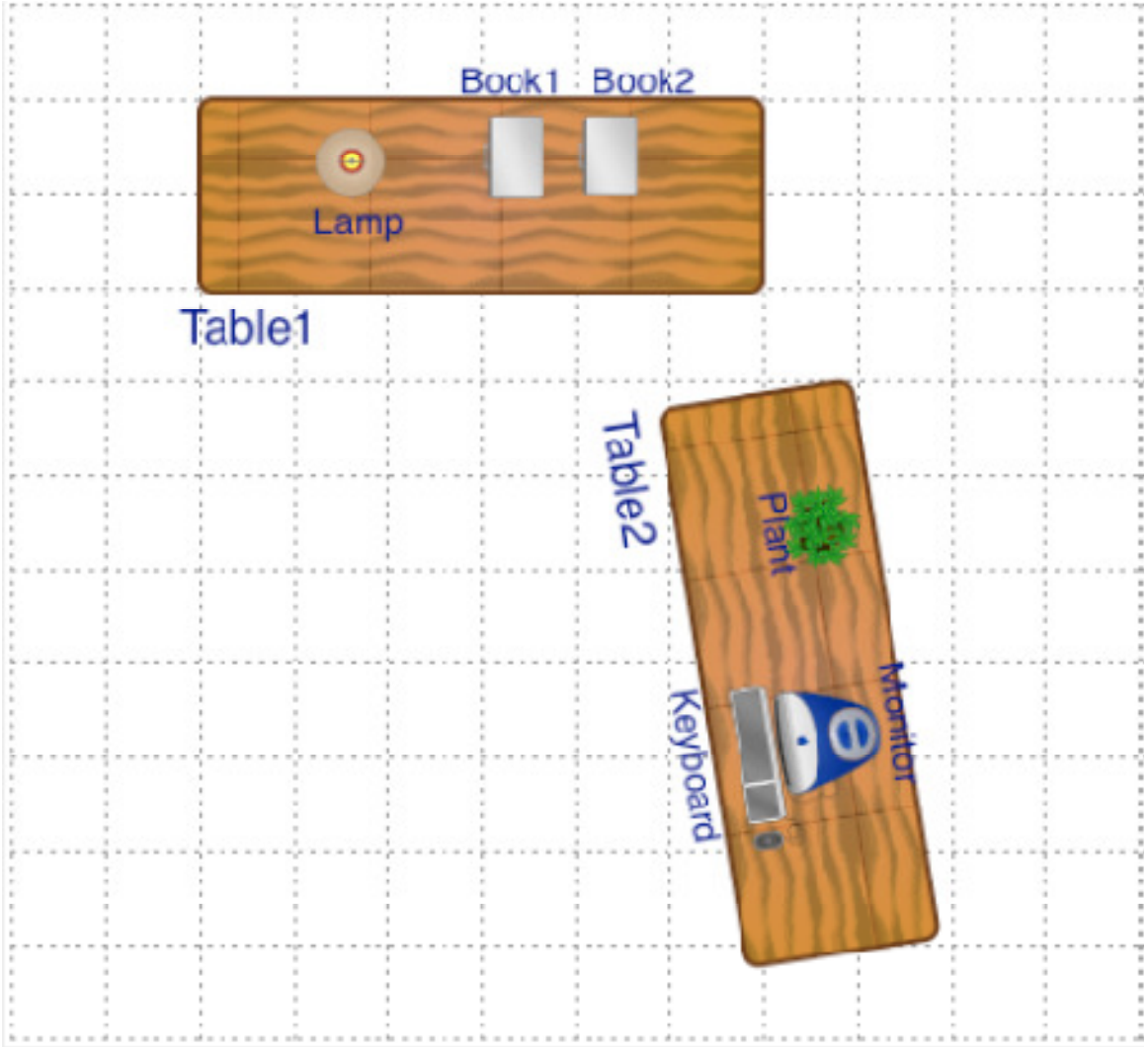
- Texturing
  - Anisotropic Texture Filtering
- Scene Graphs & Hierarchies
  - Introduction
  - Data structures

Scene Graphs

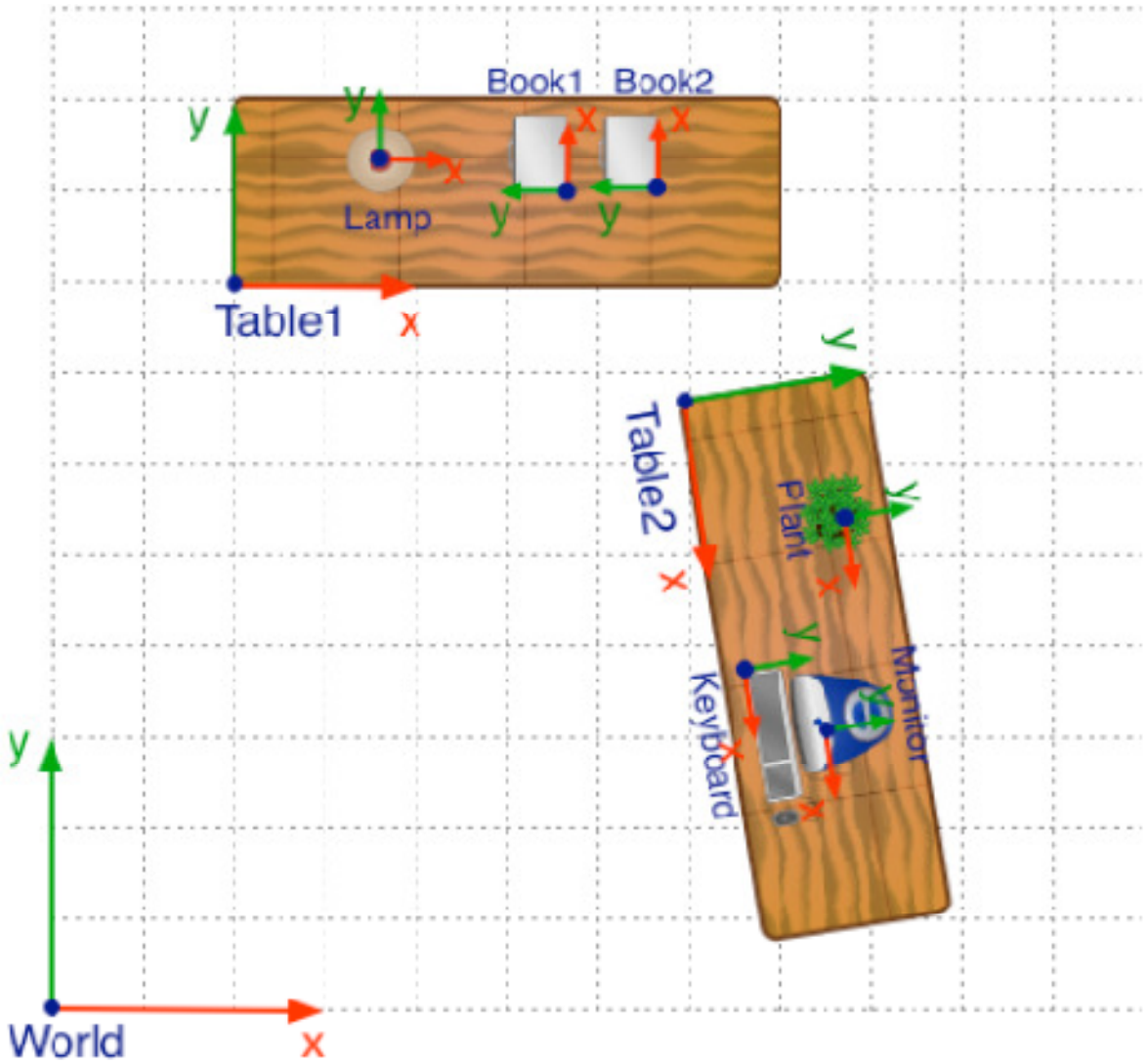
- Data structure for intuitive construction of 3D scenes
- So far, our GLUT-based projects store a linear list of objects
- This approach does not scale to large numbers of objects in complex, dynamic scenes

Sample Scene

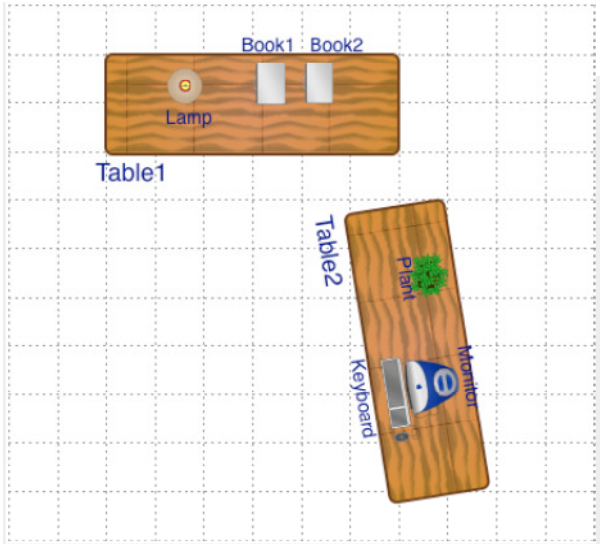
Top View

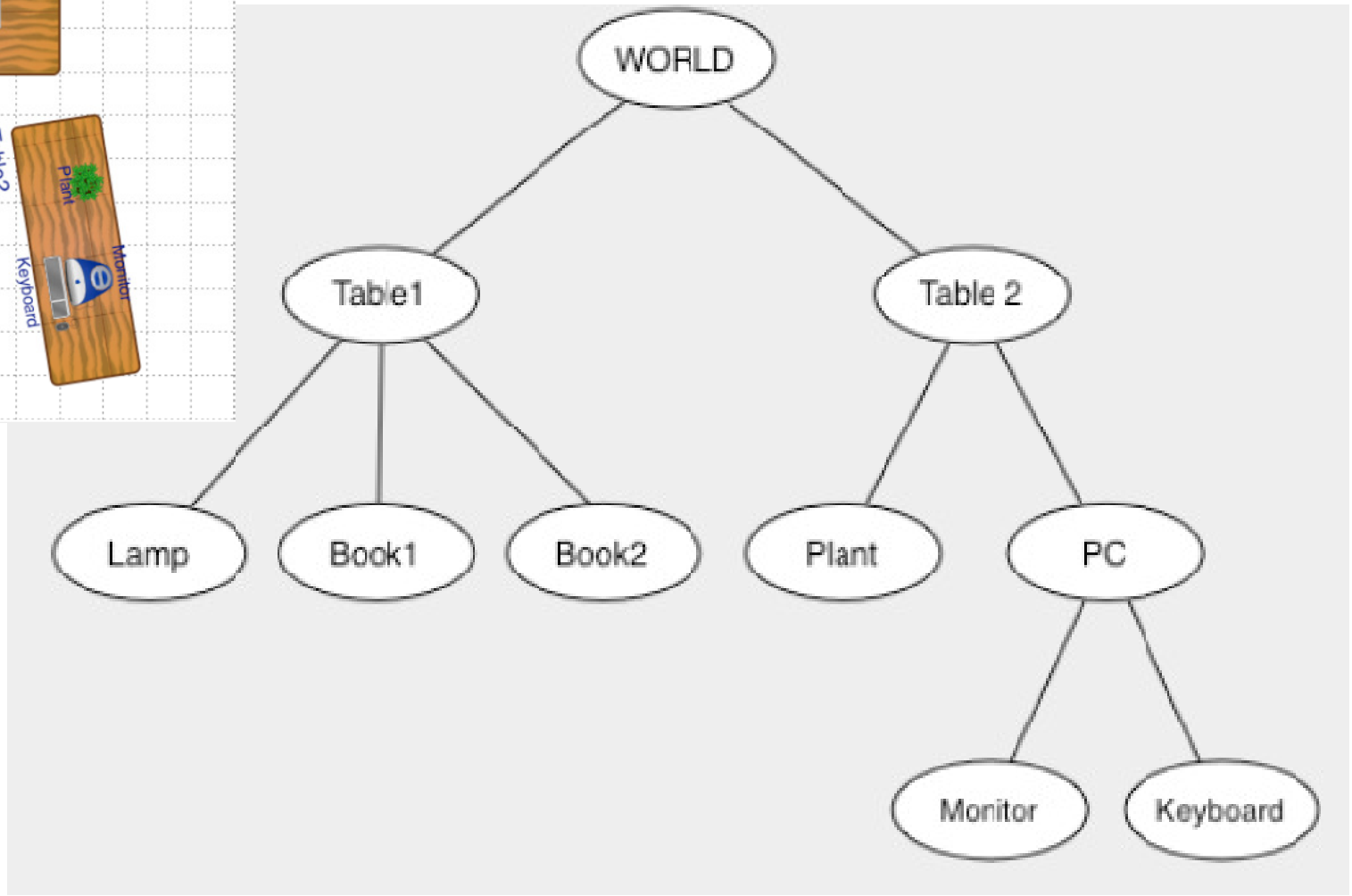


Top View With Coordinate Systems



Hierarchical Organization





Data Structure

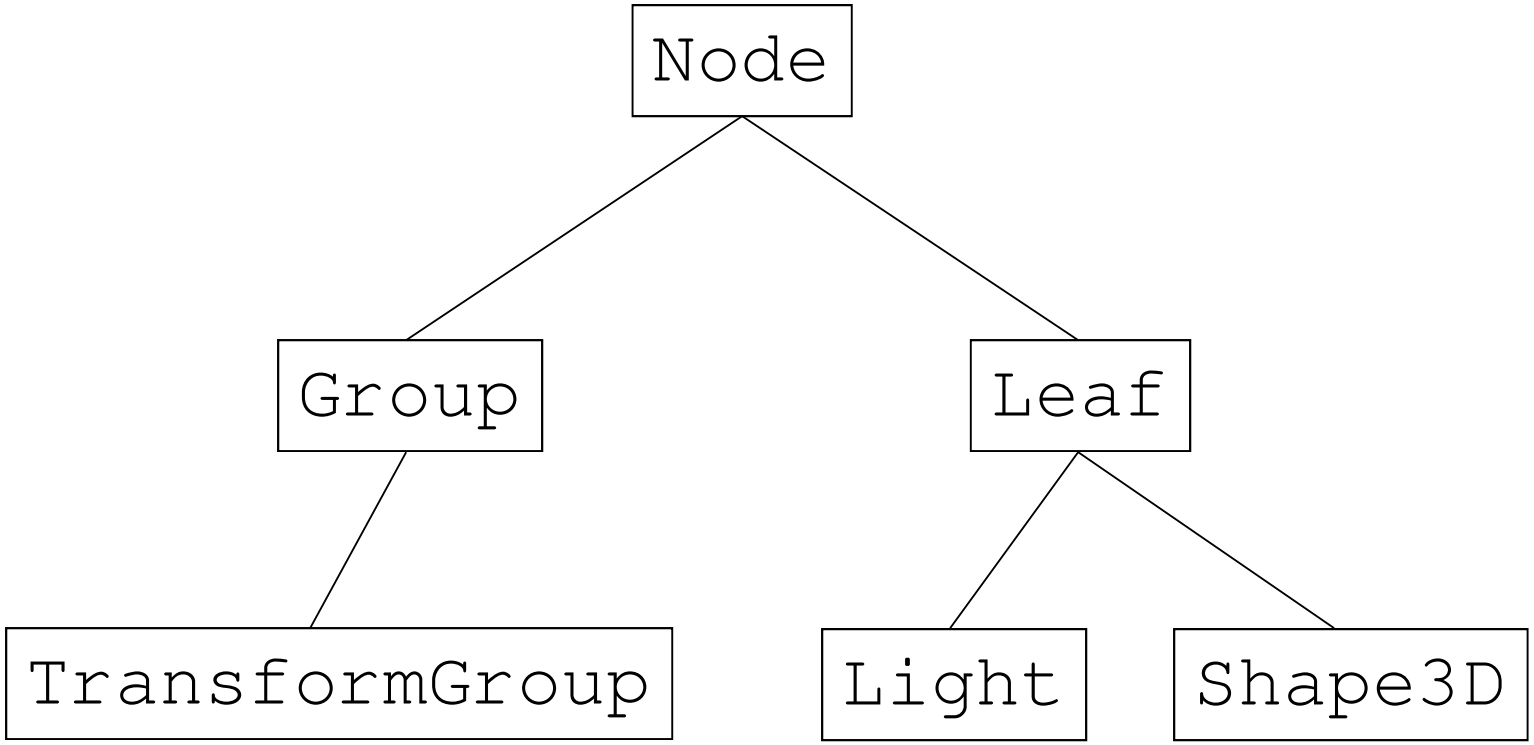
- Requirements
  - Collection of individual models/objects
  - Organized in groups
  - Related via hierarchical transformations
- Use a tree structure
- Nodes have associated local coordinates
- Different types of nodes
  - Geometry
  - Transformations
  - Lights
  - etc.

Class Hierarchy

- Many designs possible
- Concepts are the same, details differ
- Design driven by intended application
  - Games
    - optimized for speed
  - Large-scale visualization
    - Optimized for memory requirements
  - Modeling system
    - Optimized for editing flexibility

Class Hierarchy

- Inspired by Java3D



Class Hierarchy

`Node`

- Access to local-to-world coordinate transform

`Group`

- List of children
- Get, add, remove child

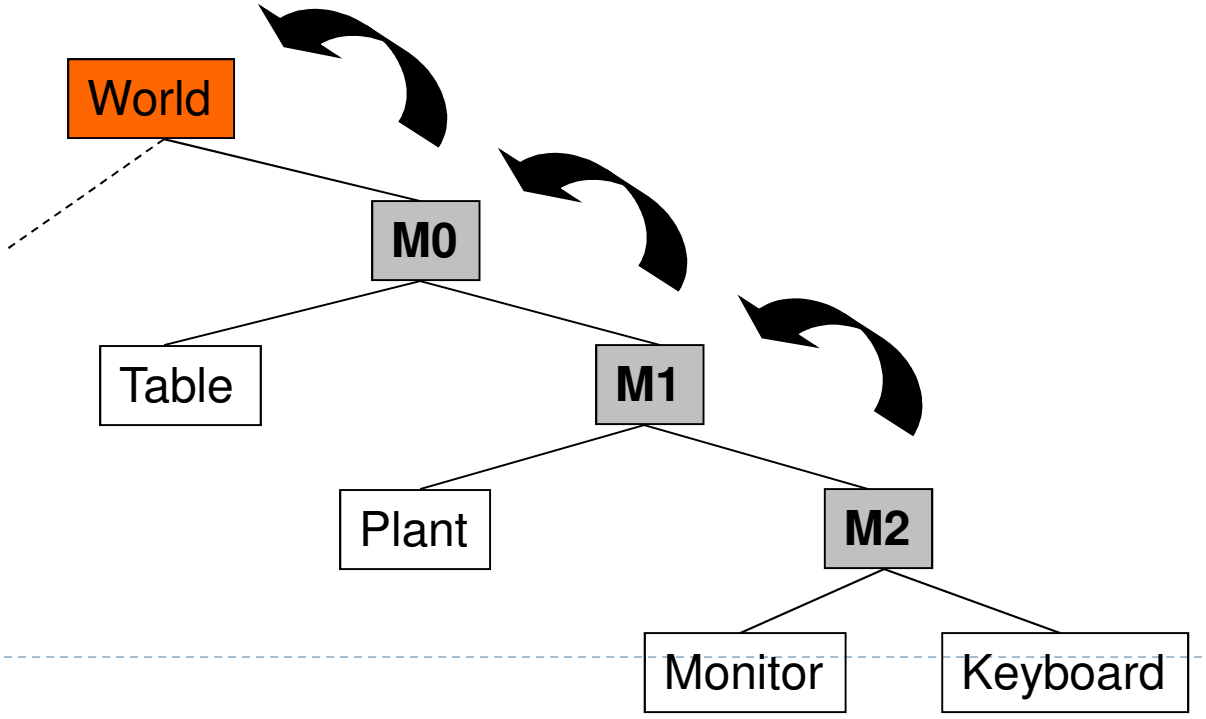
`Leaf`

- Node with no children

Class Hierarchy

`TransformGroup`

- Stores additional transformation $\mathbf{M}$
- Transformation applies to subtree below node
- Monitor-to-world transform $\mathbf{M}_0\mathbf{M}_1\mathbf{M}_2$



Class Hierarchy

Subclasses of `Leaf`

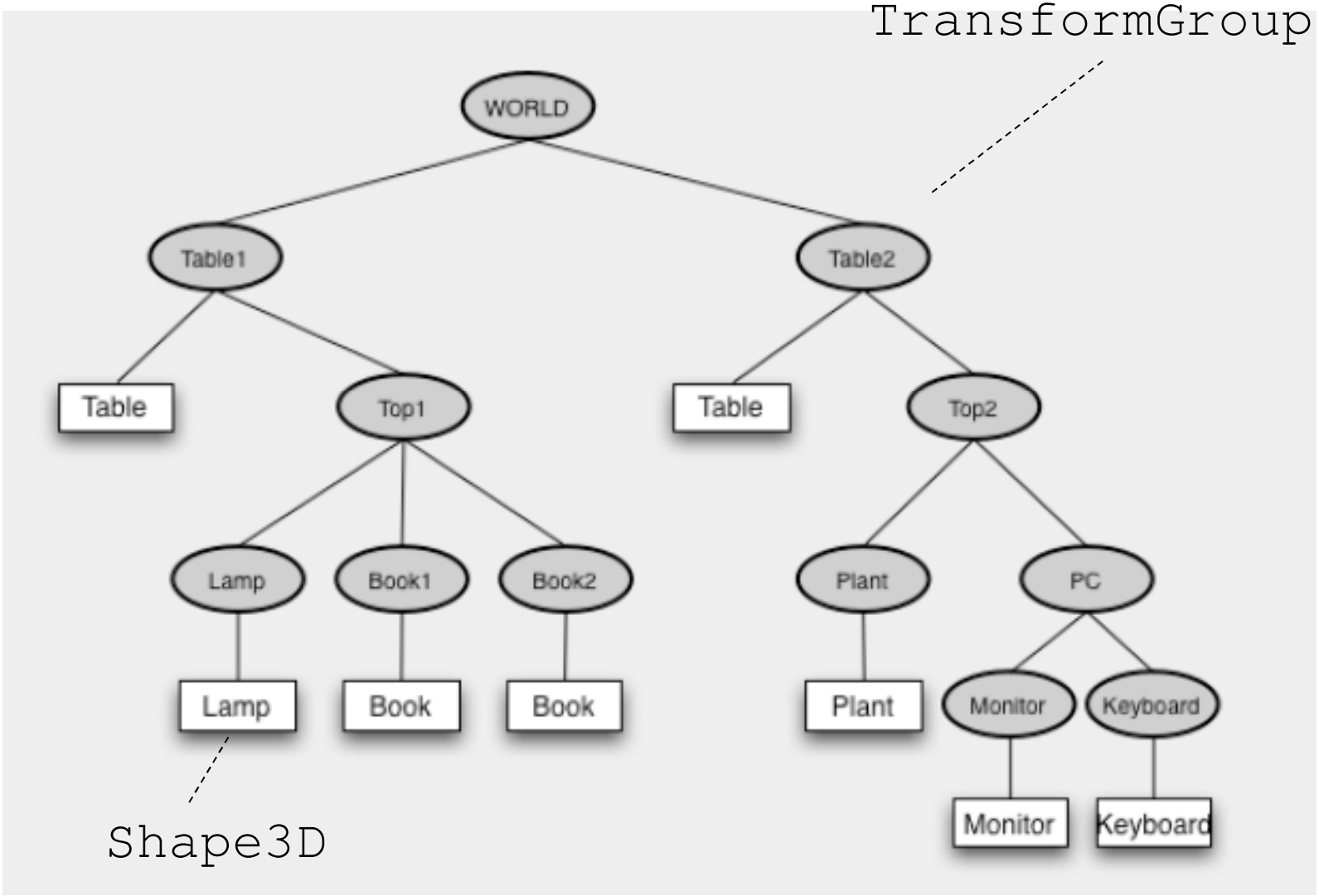
`Light`

- Stores light sources

`Shape3D`

- References a geometric object, material

Scene Graph for Sample Scene



Source Code for Sample Scene

```
WORLD = new Group();
table1Trafo = new TransformGroup(…);
  WORLD.addChild(table1Trafo);
table1 = makeTable(); table1Trafo.addChild(table1);
top1Trafo = new TransformGroup(…);
  table1Trafo.addChild(top1Trafo);

lampTrafo = new TransformGroup(…); top1Trafo.addChild(lampTrafo);
lamp = makeLamp(); lampTrafo.addChild(lamp);

book1Trafo = new TransformGroup(…);
  top1Trafo.addChild(book1Trafo);
book1 = makeBook(); book1Trafo.addChild(book1);
```

- More convenient to construct hierarchical scenes than using linear list of objects
- Easier to manipulate

Modifying the Scene

- Change tree structure
  - Add, delete, rearrange nodes
- Change node parameters
  - Transformation matrices
  - Shape of geometry data
  - Materials
- Define specific subclasses
  - Animation, triggered by timer events

Modifying the Scene

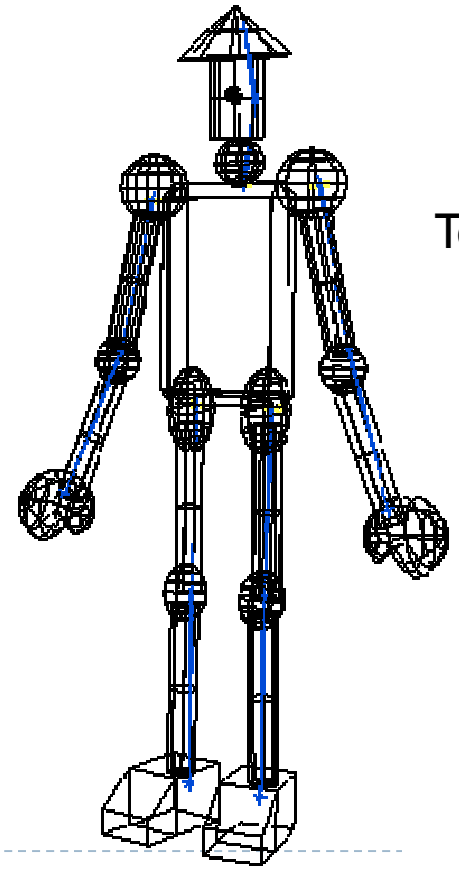
- Change a transform in the tree

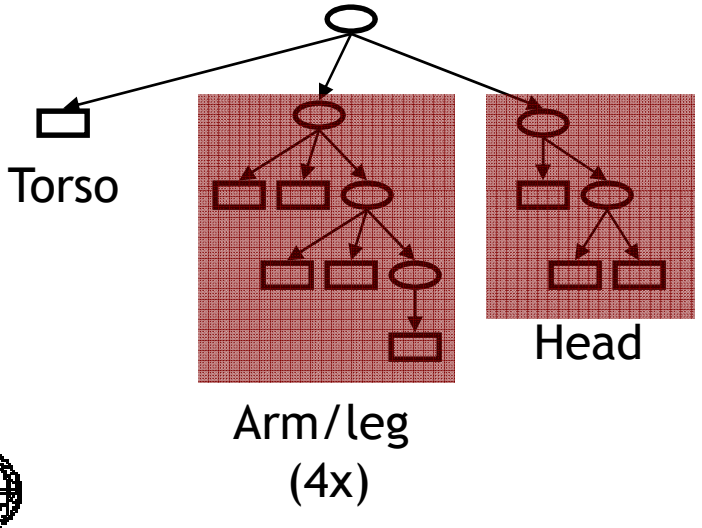
```
table1Trafo.setRotationZ(23);
```

- 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
- Allows interactive model manipulation tools
  - Add objects relative to parent objects
  - E.g., book on table

Articulated Character

- Separate rigid parts
- Joint angles define transformation matrices
- Hierarchy
  - Rooted at torso
  - Neck, head subtree
  - Arms subtree
  - Legs subtree





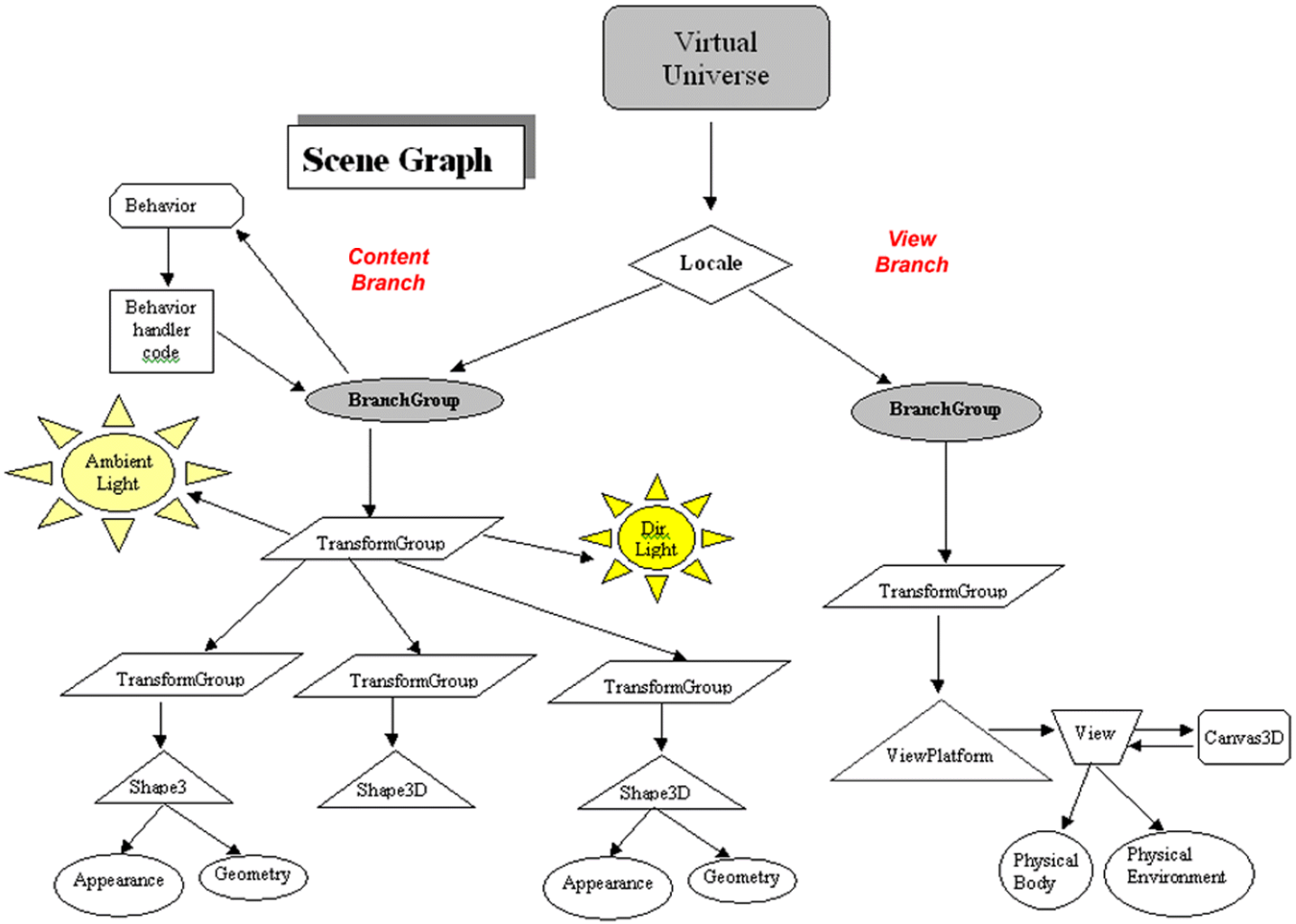
Parameteric Models

- Parameters for
  - Relationship between parts (e.g., joint angles)
  - Shape of individual parts (e.g., length of limbs)
- Hierarchical relationship between parts
- *Degrees of freedom* (DOFs)
  - Total number of float parameters in the model

More Node Types

- Shape nodes
  - Cube, sphere, curved surface, etc…
- Nodes that control structure
  - Switch/Select: parameters choose whether or which children to enable, etc…
- Nodes that define other properties
  - Camera
- Other, application domain dependent nodes:
  - Video node
  - Terrain node
  - Dynamic object node with trajectory, etc.

Java3D Scene Graph



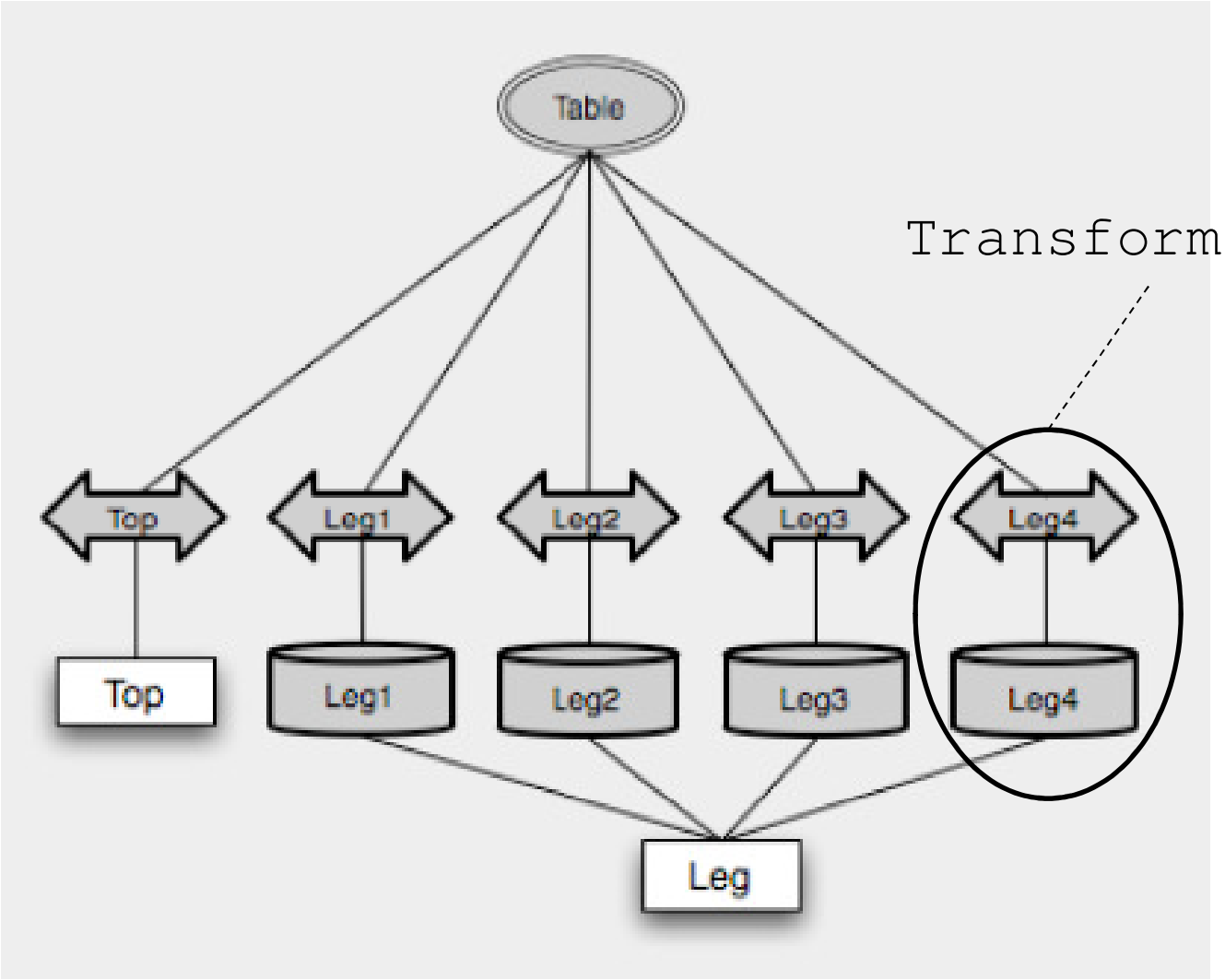
Graph Definitions

- Wikipedia:
  - "A **graph** is an abstract representation of a set of objects where some pairs of the objects are connected by links."
  - "A **tree** is a graph in which any two vertices are connected by exactly one simple path."
  - "A **directed graph** differs from an undirected graph, in that the latter is defined in terms of unordered pairs of vertices (edges)."
  - "A **directed acyclic graph** (commonly abbreviated to DAG), is a directed graph with no directed cycles"

Scene *Graph*, Not Tree

- A scene may have many copies of a model
- A model might use several copies of a part
- Multiple Instantiation:
  - One copy of node or subtree in memory
  - Reference (pointer) inserted as child of many parents
- Not the same as instantiation in C++ terminology
- A directed acyclic graph (DAG), not a tree
- Object appears in scene multiple times, with different coordinates

Instantiation



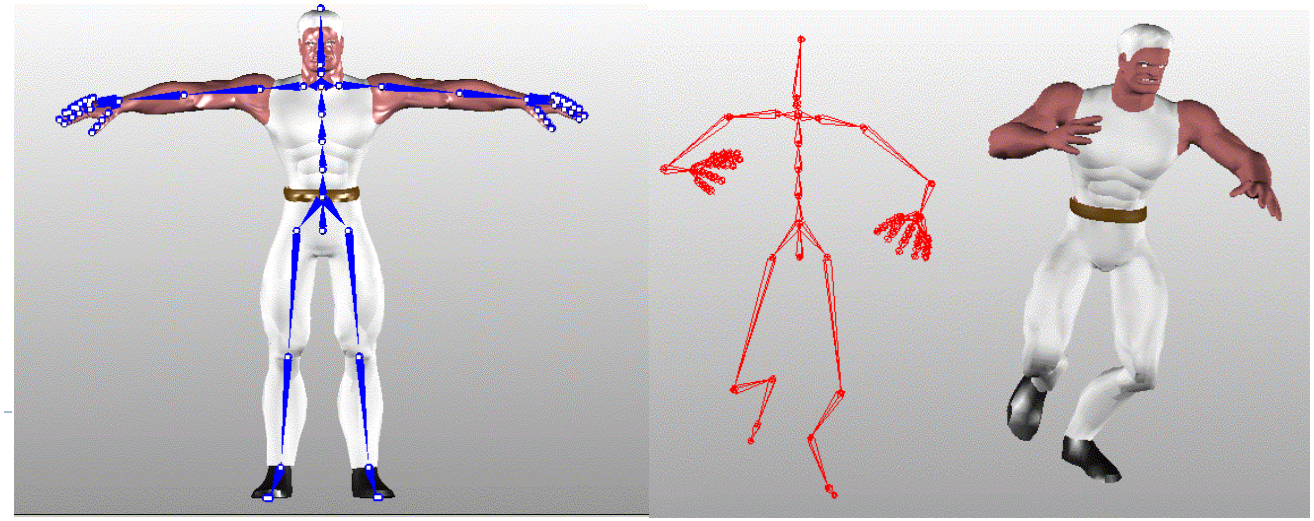
Scene Graph, Not Tree

- Saves memory
- May save time, depending on caching/optimization
- Change parameter once, affects all instances
  - Can be good or bad, depending on what you want
  - Some scene graph designs let other properties inherit from parent

More Complex Operations

Articulated character

- Shape nodes that compute surface across multiple joint nodes
- Nodes that change shape of geometry
- Very popular in games



Basic Rendering

- Traverse the tree recursively

```
TransformGroup::draw(Matrix4 C) {
  C_new = C*M;   // M is a class member
  for all children
    draw(C_new);
}

Shape3D::draw(Matrix4 C) {
  setModelView(C);
  setMaterial(myMaterial);
  render(myObject);
}
```

Basic Rendering

- Traverse the tree recursively

```
TransformGroup::draw(Matrix4 C) {
  C_new = C*M;   // M is a class member
  for all children
    draw(C_new);
}

Shape3D::draw(Matrix4 C) {
  setModelView(C);
  setMaterial(myMaterial);
  render(myObject);
}
```

Initiate rendering with

```
world->draw(IDENTITY);
```

Next Lecture

- Scene Graphs & Hierarchies
  - Performance Optimization
- Curves