

CSE 167:
Introduction to Computer Graphics
Lecture #9: Scene Graphs

Jürgen P. Schulze, Ph.D.
University of California, San Diego
Spring Quarter 2015

Announcements

- ▶ Project 4 due tomorrow at 1pm
- ▶ Midterm results

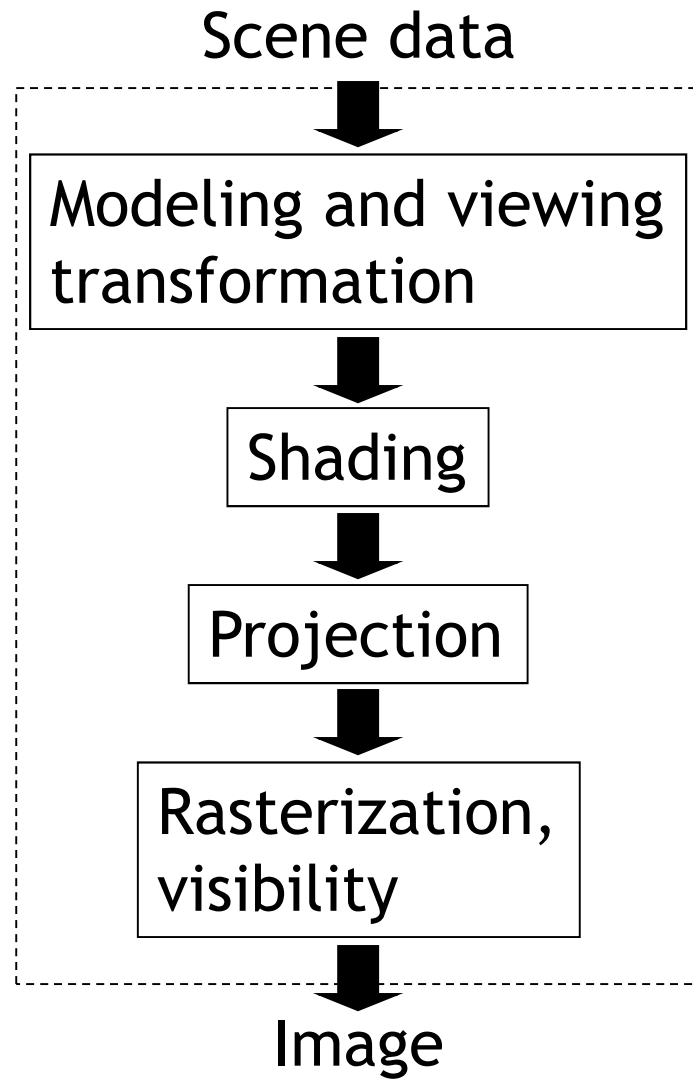
Midterm Statistics

Maximum Score	80
Average score	62.4
Highest score	80
Lowest score	34
70-80	14
60-70	26
50-60	10
40-50	7
30-40	1
# Exams submitted	58

Lecture Overview

- ▶ Scene Graphs & Hierarchies
 - ▶ Introduction
 - ▶ Data structures
- ▶ Performance Optimization
 - ▶ Level-of-detail techniques
 - ▶ View Frustum Culling

Rendering Pipeline



Graphics System Architecture

Interactive Applications

- ▶ Games, scientific visualization, virtual reality

Rendering Engine, Scene Graph API

- ▶ Implement functionality commonly required in applications
- ▶ Back-ends for different low-level APIs
- ▶ No broadly accepted standards
- ▶ Examples: OpenSceneGraph, NVSG, Java3D, Ogre

Low-level graphics API

- ▶ Interface to graphics hardware
- ▶ Highly standardized: OpenGL, Direct3D

Scene Graph APIs

- ▶ APIs focus on different types of applications
- ▶ OpenSceneGraph (www.openscenegraph.org)
 - ▶ Scientific visualization, virtual reality, GIS (geographic information systems)
- ▶ NVIDIA SceniX (<https://developer.nvidia.com/scenix>)
 - ▶ Optimized for shader support
 - ▶ Support for interactive ray tracing
- ▶ Java3D (<http://java3d.java.net>)
 - ▶ Simple, easy to use, web-based applications
- ▶ Ogre3D (<http://www.ogre3d.org/>)
 - ▶ Games, high-performance rendering

Commonly Offered Functionality

- ▶ **Resource management**
 - ▶ Content I/O (geometry, textures, materials, animation sequences)
 - ▶ Memory management
- ▶ **High-level scene representation**
 - ▶ Graph data structure
- ▶ **Rendering**
 - ▶ Optimized for efficiency (e.g., minimize OpenGL state changes)

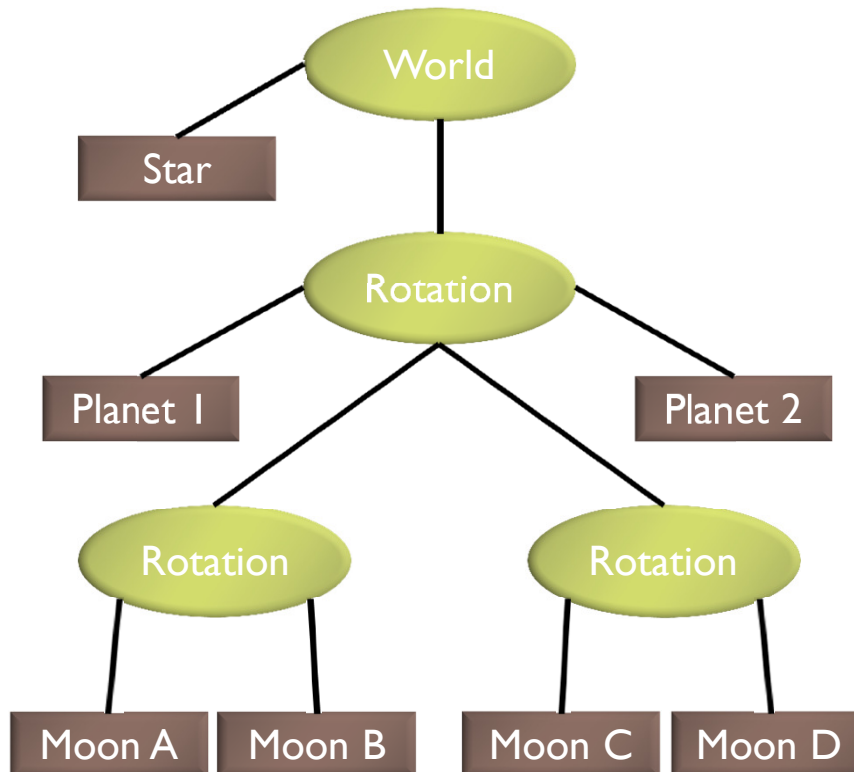
Lecture Overview

- ▶ Scene Graphs & Hierarchies
 - ▶ Introduction
 - ▶ Data structures
- ▶ Performance Optimization
 - ▶ Level-of-detail techniques
 - ▶ View Frustum Culling

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

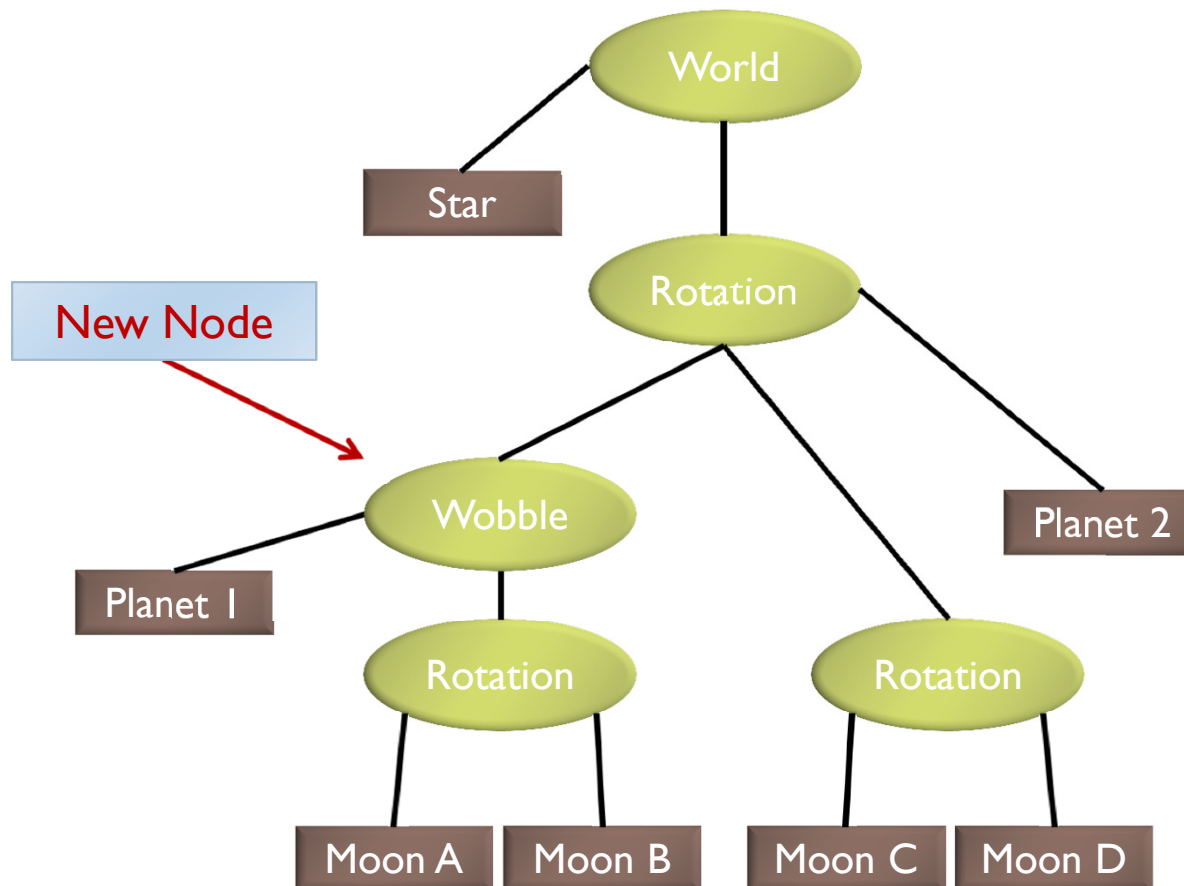
Solar System



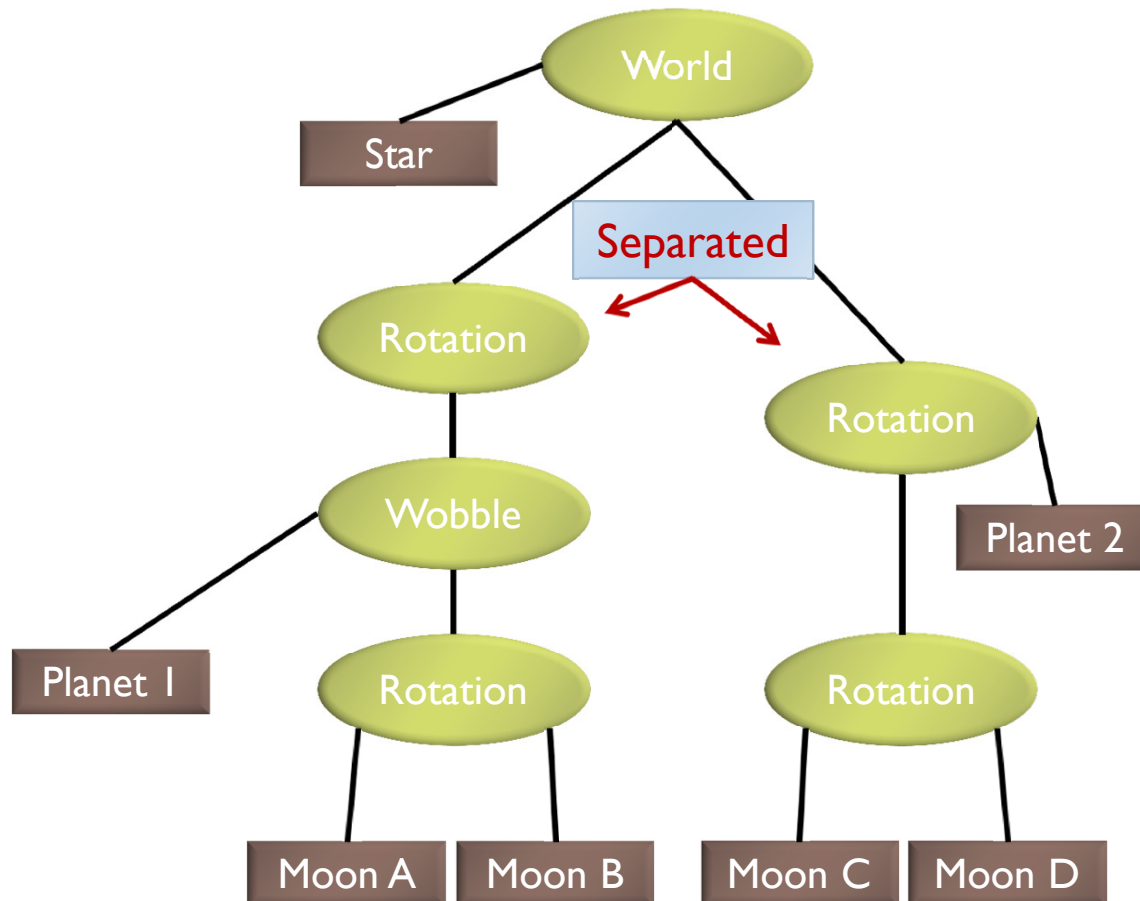
- Draw the star
- Save the current matrix
- - Apply a rotation
 - Draw Planet One
 - Save the current matrix
- - Apply a second rotation
 - Draw Moon A
 - Draw Moon B
- Reset the matrix we saved
- Draw Planet two
- Save the current matrix
- - Apply a rotation
 - Draw Moon C
 - Draw Moon D
- Reset the matrix we saved
- Reset the matrix we saved

Example from <http://www.gamedev.net>

Solar System with Wobble



Planets rotating at different speeds



- Draw the Star
- Save the current matrix
- • Apply a rotation
 - • Save the current matrix
 - • Apply a wobble
 - Draw Planet 1
 - • Save the current matrix
 - • Apply a rotation
 - • Draw Moon A
 - Draw Moon B
 - Reset the Matrix
- Reset the matrix
- Reset the matrix
- Save the current matrix
- • Apply a rotation
 - • Draw Planet 2
 - Save the current matrix
 - • Apply a rotation
 - Draw Moon C
 - Draw Moon D
 - Reset the current matrix
 - Reset the current matrix
- Reset the current matrix

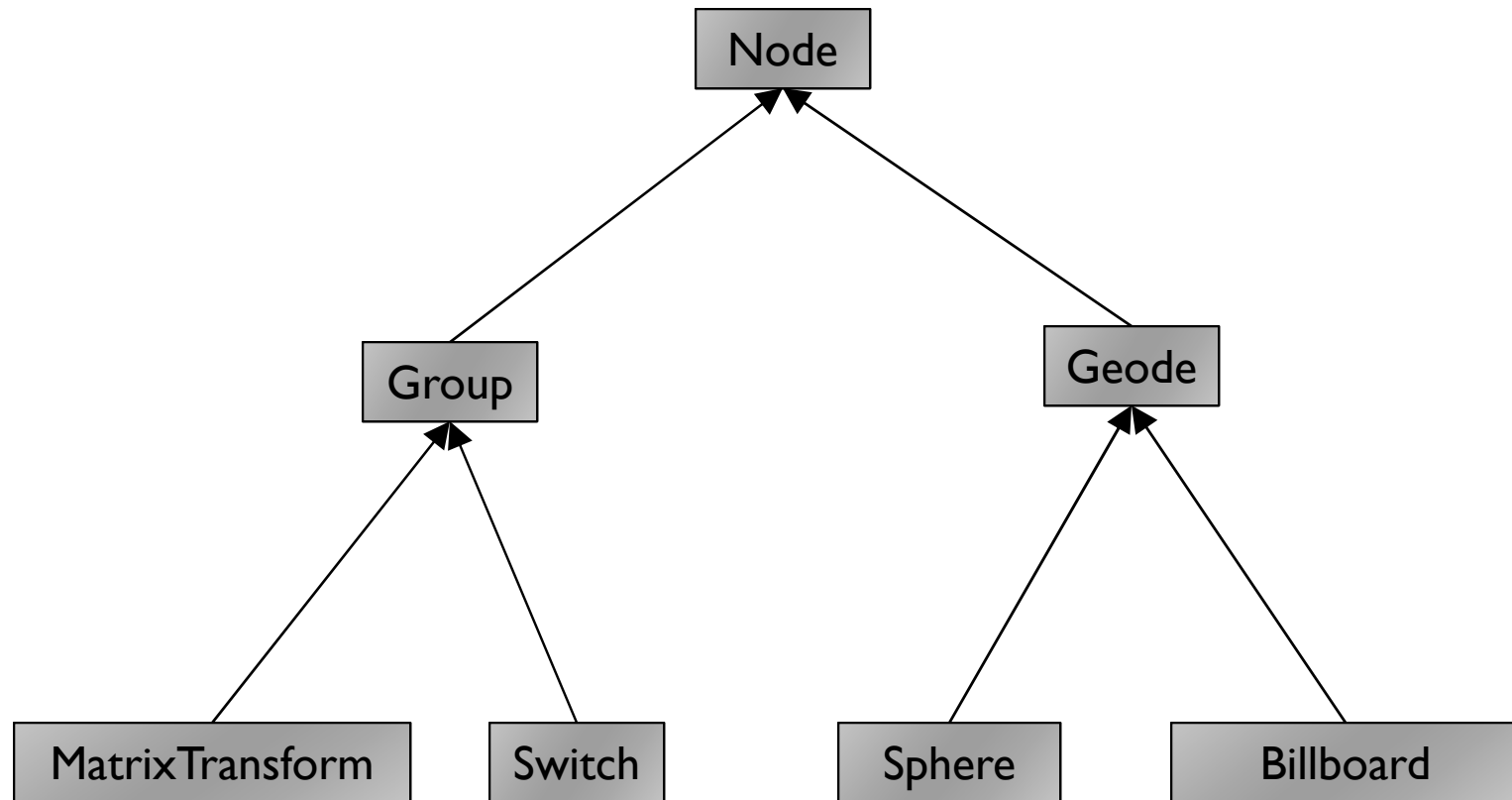
Data Structure

- ▶ **Requirements**
 - ▶ Collection of separable geometry models
 - ▶ Organized in groups
 - ▶ Related via hierarchical transformations
- ▶ **Use a tree structure**
- ▶ **Nodes have associated local coordinates**
- ▶ **Different types of nodes**
 - ▶ Geometry
 - ▶ Transformations
 - ▶ Lights
 - ▶ Many more

Class Hierarchy

- ▶ Many designs possible
- ▶ Design driven by intended application
 - ▶ Games
 - ▶ Optimized for speed
 - ▶ Large-scale visualization
 - ▶ Optimized for memory requirements
 - ▶ Modeling system
 - ▶ Optimized for editing flexibility

Sample Class Hierarchy



Inspired by OpenSceneGraph

Class Hierarchy

Node

- ▶ Common base class for all node types
- ▶ Stores node name, pointer to parent, bounding box

Group

- ▶ Stores list of children

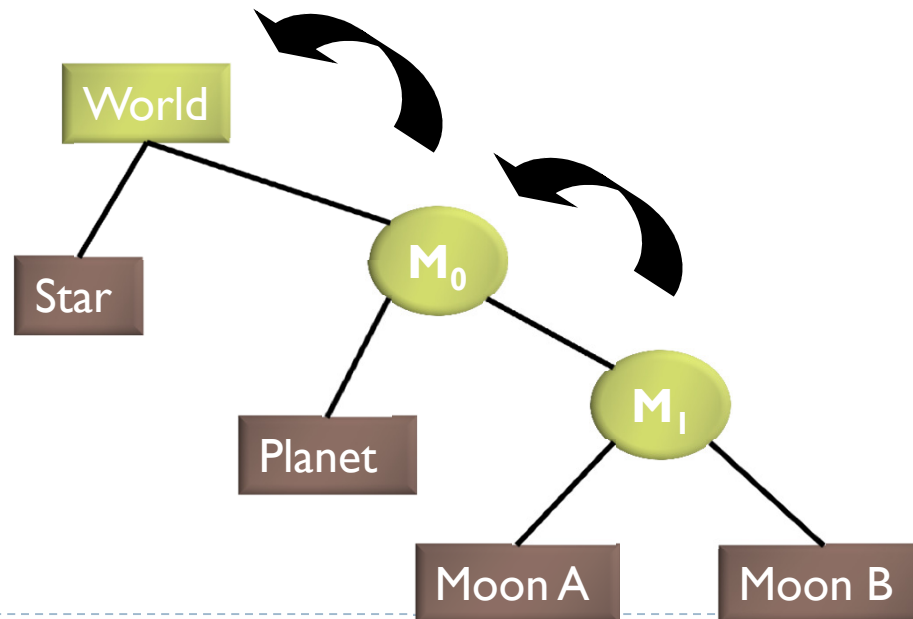
Geode

- ▶ **Geometry Node**
- ▶ Knows how to render a specific piece of geometry

Class Hierarchy

MatrixTransform

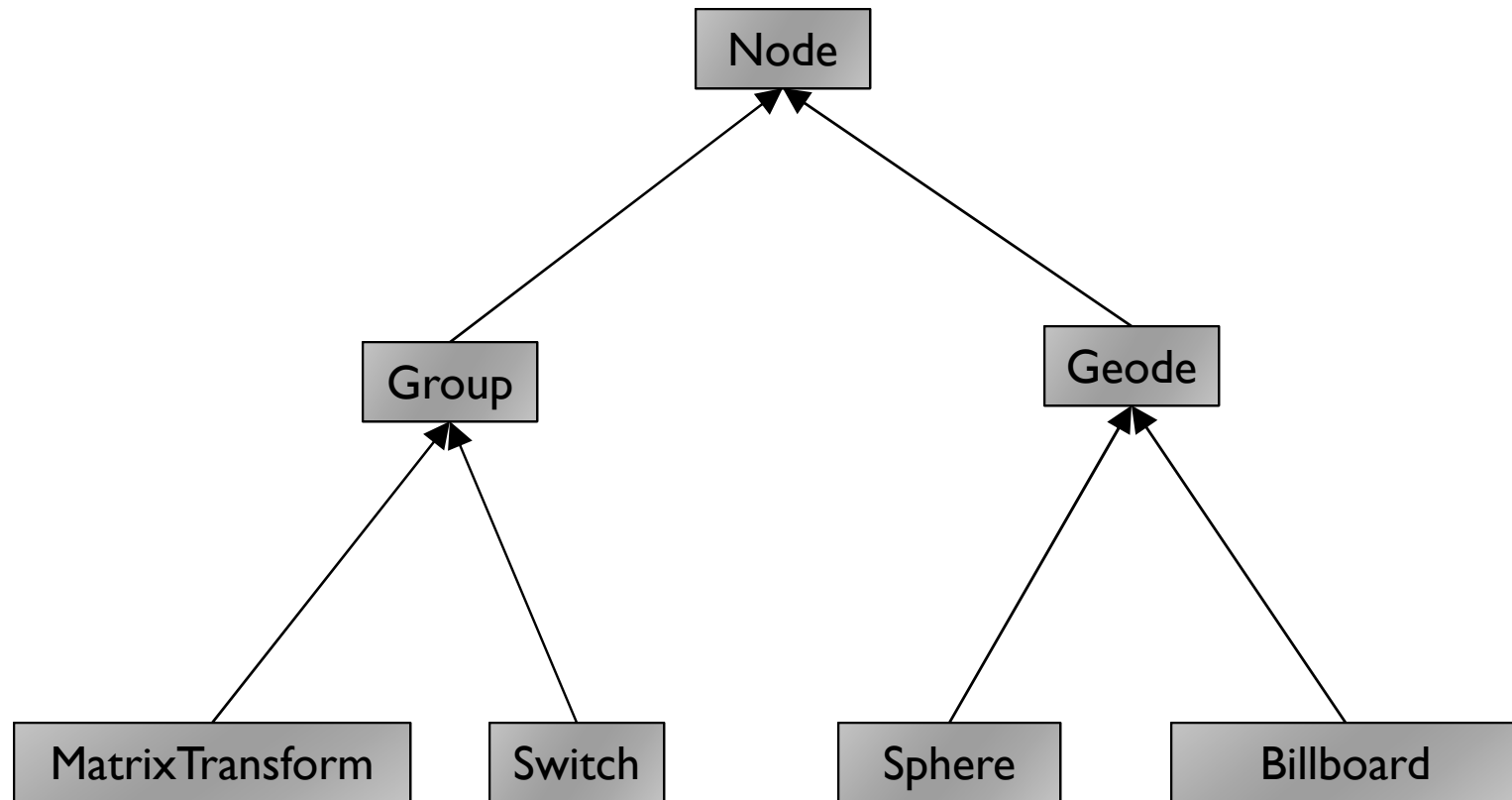
- ▶ Derived from Group
- ▶ Stores additional transformation **M**
- ▶ Transformation applies to sub-tree below node
- ▶ Monitor-to-world transformation **M₀M₁**



Lecture Overview

- ▶ Scene Graphs & Hierarchies
 - ▶ Introduction
 - ▶ Data structures
- ▶ Performance Optimization
 - ▶ Level-of-detail techniques
 - ▶ View Frustum Culling

Sample Class Hierarchy



Inspired by OpenSceneGraph

Class Hierarchy

Node

- ▶ Common base class for all node types
- ▶ Stores node name, pointer to parent, bounding box

Group

- ▶ Stores list of children

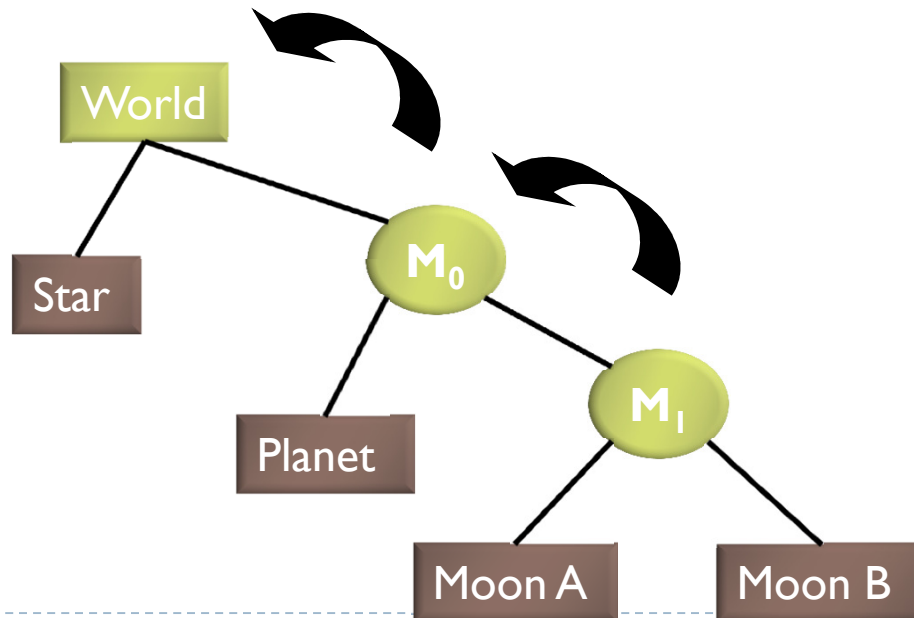
Geode

- ▶ **Geometry Node**
- ▶ Knows how to render a specific piece of geometry

Class Hierarchy

MatrixTransform

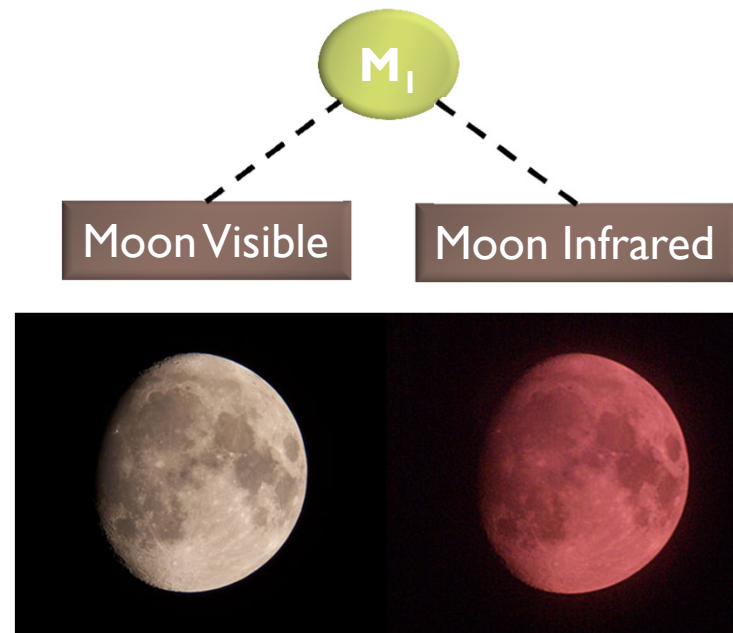
- ▶ Derived from Group
- ▶ Stores additional transformation **M**
- ▶ Transformation applies to sub-tree below node
- ▶ Monitor-to-world transformation **M₀M₁**



Class Hierarchy

Switch

- ▶ Derived from Group node
- ▶ Allows hiding (not rendering) all or subsets of its child nodes
- ▶ Can be used for state changes of geometry, or “key frame” animation



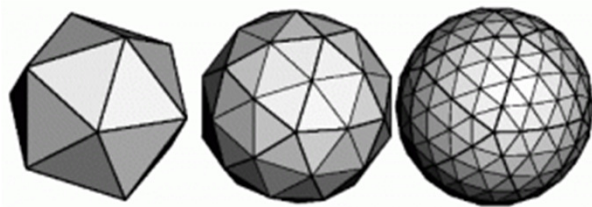
Class Hierarchy

Sphere

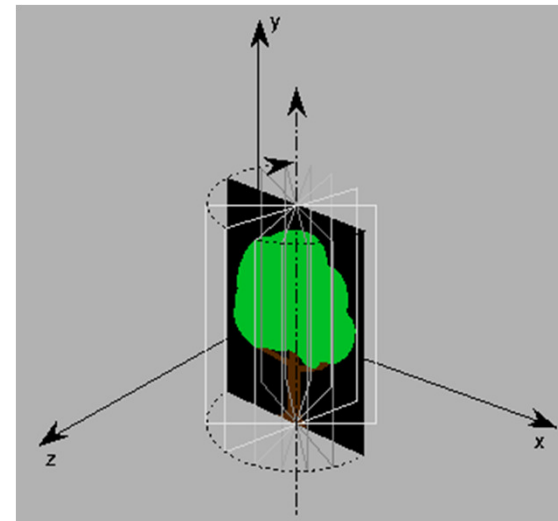
- ▶ Derived from Geode
- ▶ Pre-defined geometry with parameters, e.g., for tessellation level, solid/wireframe, etc.

Billboard

- ▶ Special geometry node to display an image always facing the viewer

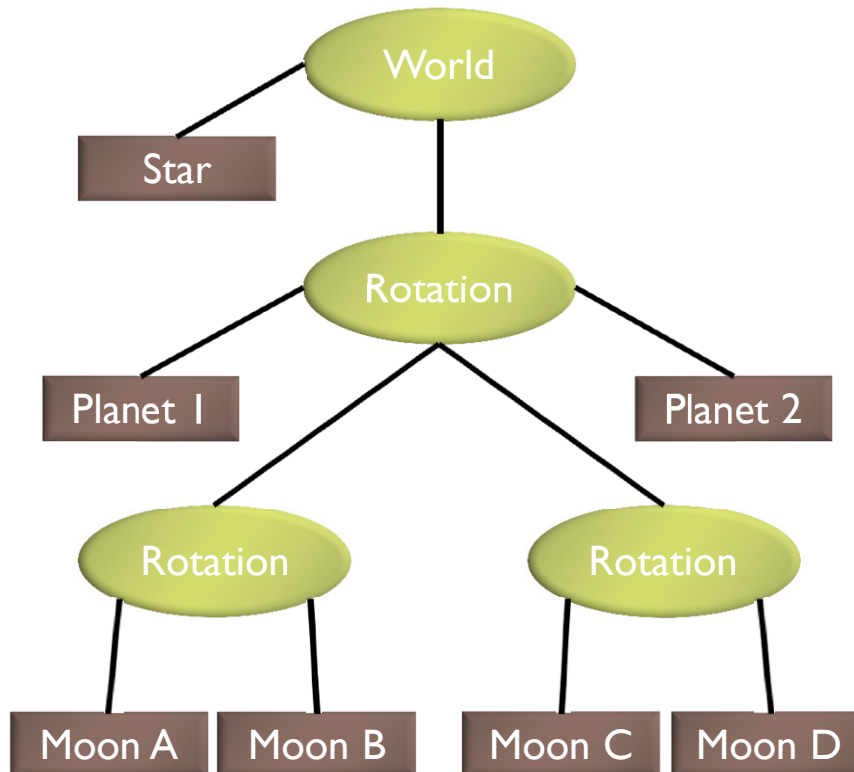


Sphere at different tessellation levels



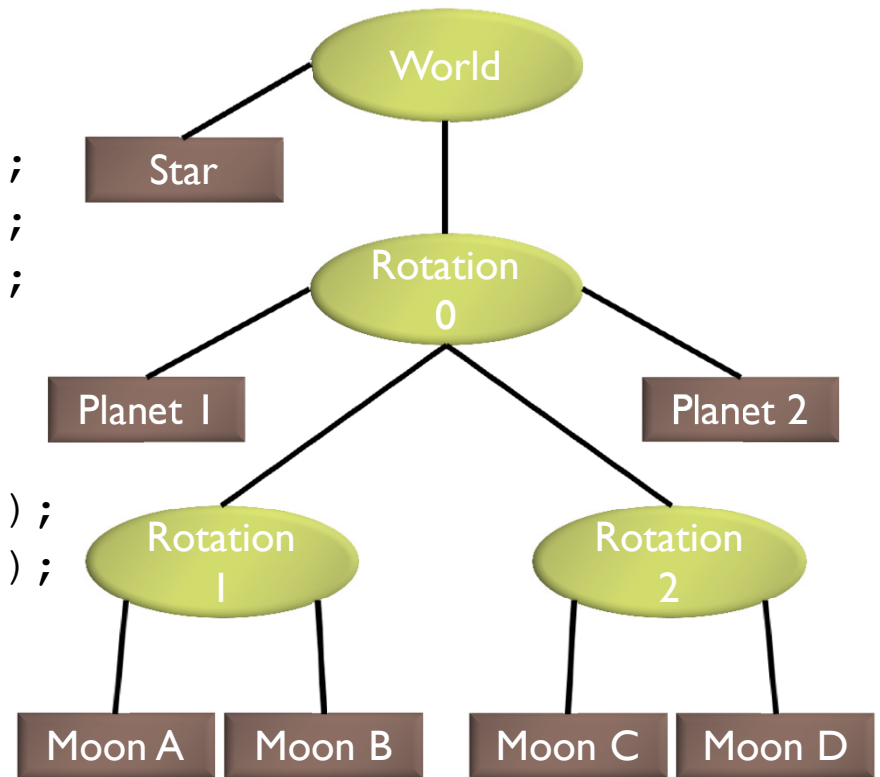
Billboarded Tree

Solar System



Source Code for Solar System

```
world = new Group();
world.addChild(new Star());
rotation0 = new MatrixTransform(...);
rotation1 = new MatrixTransform(...);
rotation2 = new MatrixTransform(...);
world.addChild(rotation0);
rotation0.addChild(rotation1);
rotation0.addChild(rotation2);
rotation0.addChild(new Planet("1"));
rotation0.addChild(new Planet("2"));
rotation1.addChild(new Moon("A"));
rotation1.addChild(new Moon("B"));
rotation2.addChild(new Moon("C"));
rotation2.addChild(new Moon("D"));
```



Basic Rendering

► Traverse the tree recursively

```
Group::draw(Matrix4 C)
{
    for all children
        draw(C);
}
```

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

```
Geode::draw(Matrix4 C)
{
    setModelView(C);
    render(myObject);
}
```

Initiate rendering with
`world->draw(IDENTITY);`

Modifying the Scene

- ▶ Change tree structure
 - ▶ Add, delete, rearrange nodes
- ▶ Change node parameters
 - ▶ Transformation matrices
 - ▶ Shape of geometry data
 - ▶ Materials
- ▶ Create new node subclasses
 - ▶ Animation, triggered by timer events
 - ▶ Dynamic “helicopter-mounted” camera
 - ▶ Light source
- ▶ Create application dependent nodes
 - ▶ Video node
 - ▶ Web browser node
 - ▶ Video conferencing node
 - ▶ Terrain rendering node

Benefits of a Scene Graph

- ▶ Can speed up rendering by efficiently using low-level API
 - ▶ Avoid state changes in rendering pipeline
 - ▶ Render objects with similar properties in batches (geometry, shaders, materials)
- ▶ Change parameter once to affect all instances of an object
- ▶ Abstraction from low level graphics API
 - ▶ Easier to write code
 - ▶ Code is more compact
- ▶ Can display complex objects with simple APIs
 - ▶ Example: osgEarth class provides scene graph node which renders a Google Earth-style planet surface