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

Jürgen P. Schulze, Ph.D.
University of California, San Diego
Spring Quarter 2016

Announcements

- Project 2 due tomorrow at 2pm
- Midterm next Tuesday

HP Summer Internship

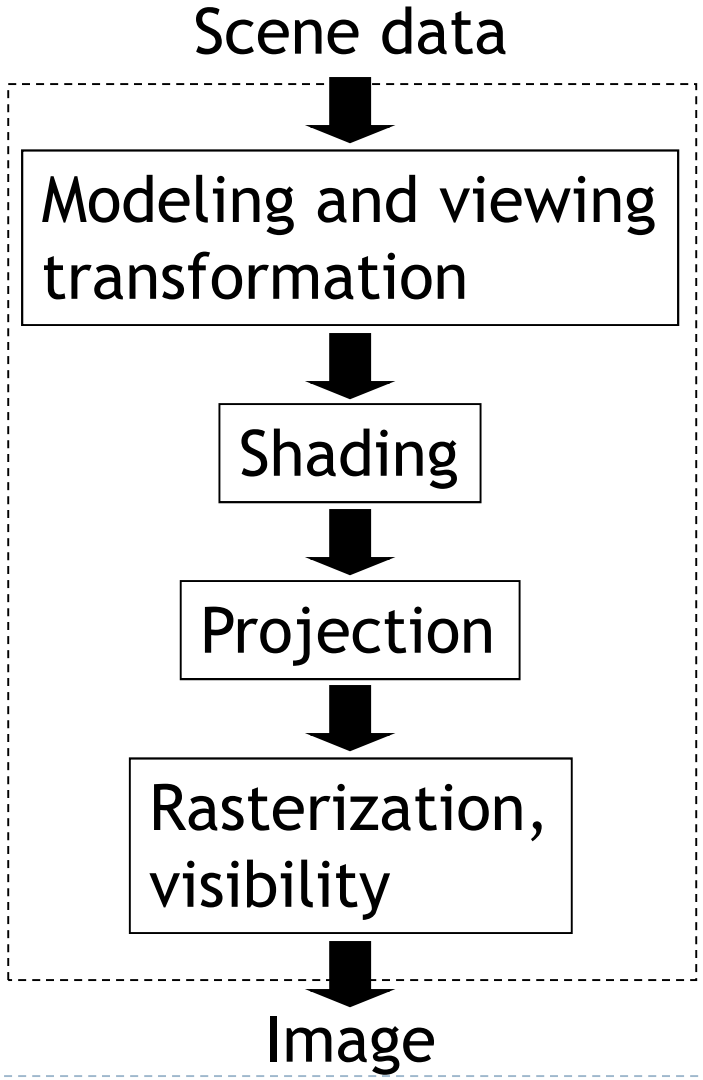
- *Calling all UCSD students who are interested in a computer science internship! San Diego can be a tough place to gain computer science experience as a student compared to other cities which is why I am so excited to present this positon to your sharp students! (There are 10 open spots for the right candidates)*
- *The position is with HP and will be for the duration of the upcoming summer. Here are a few details about the exciting opportunity.*
- **Company:** HP
  Position Title: Refresh Support Technician
  Contract/Perm: 3-4 month contract
  Pay Rate: $13-15/hr based on experience
  Interview Process: Hire off of a resume
  Work Address: Rancho Bernardo location
  Top Skills: Refresh windows 7 & 8.1 experience
- Must have:
- Windows refresh 7 or 8.1 experience
- Minimum Vocational/Diploma/Associate Degree (technical field)
- Equivalent with 1-2 years of working experience in related fields, or Degree holder with no or less than 1 year relevant working experience.
- This is a competitive position and will move quickly. If you have any students who might be interested please have them contact me to be considered for this role. My direct line is **858 568 7582.**
- **Curtis Stitts**
- *Technical Recruiter*
- **THE SELECT GROUP**
- CStitts@selectgroup.com | Web Site
- 9339 Genesee Avenue, Ste. 320 | San Diego, CA 92121

UCSD

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)

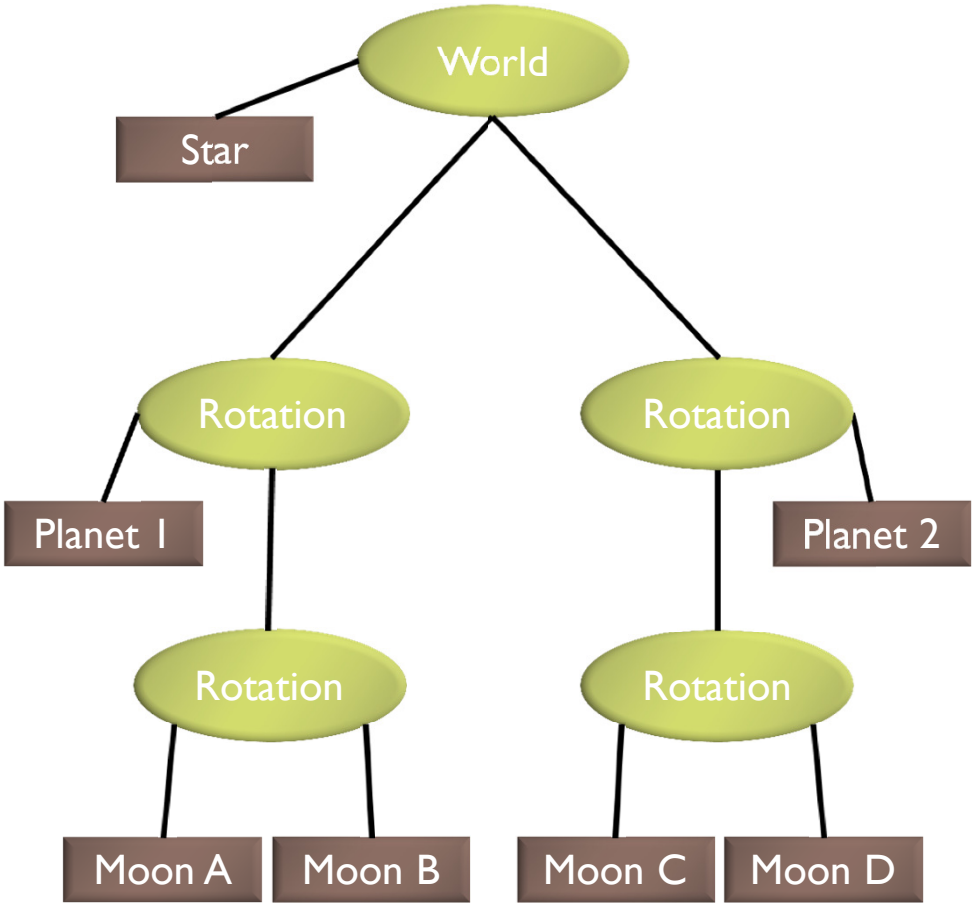
Lecture Overview

- 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 GLFW-based projects store a linear list of objects
- This approach does not scale to large numbers of objects in complex, dynamic scenes

Example: Solar System



Source: http://www.gamedev.net

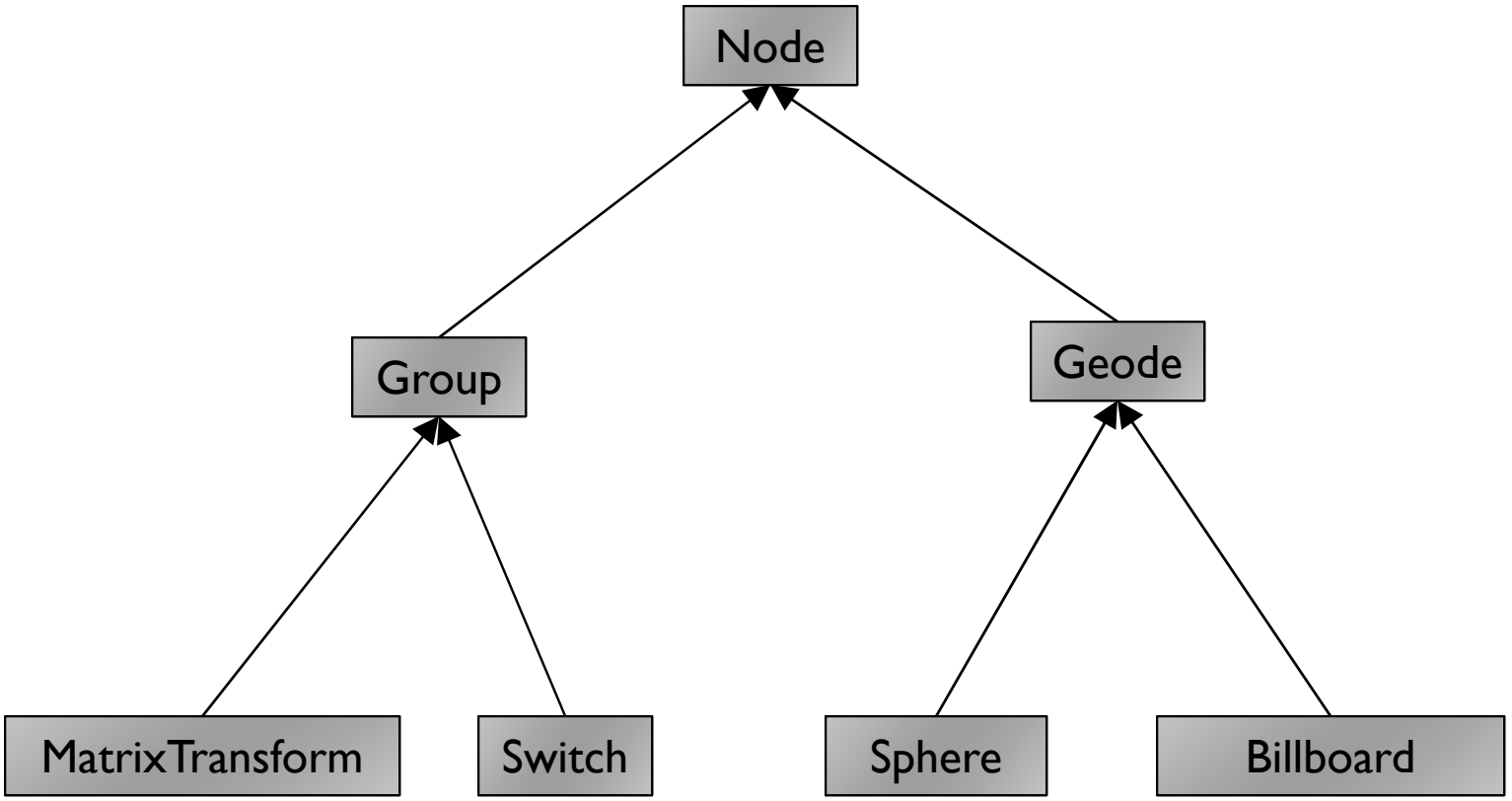
Data Structure

- 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

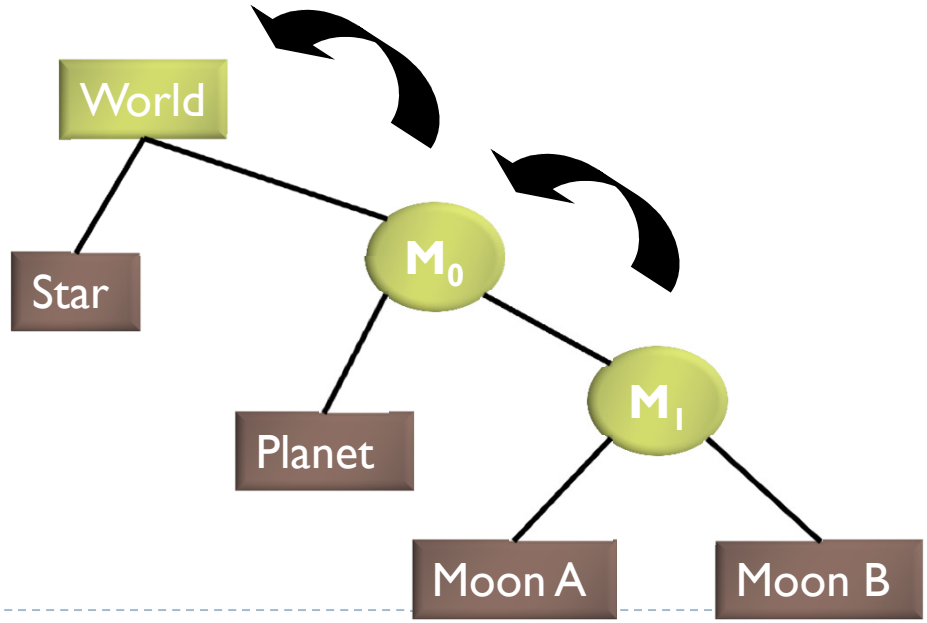
`Geode`

- 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 $\mathbf{M_0 M_1}$

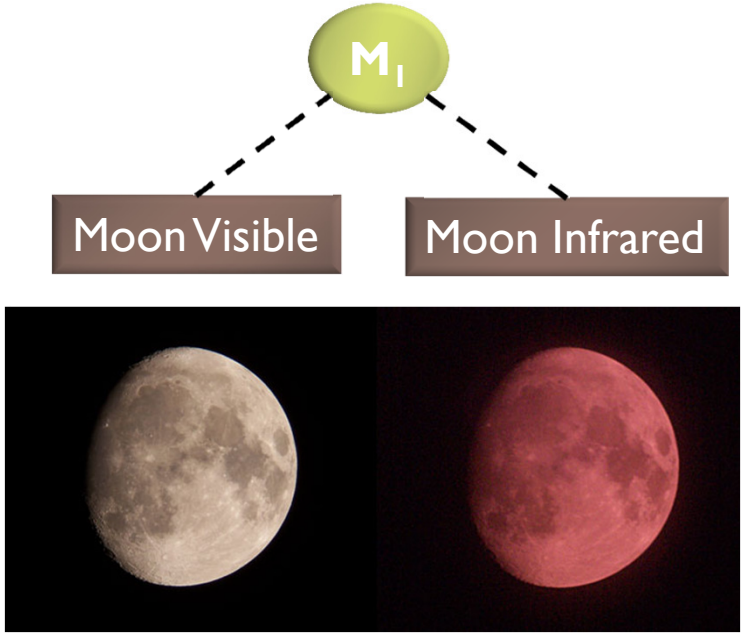


UCSD

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



UCSD

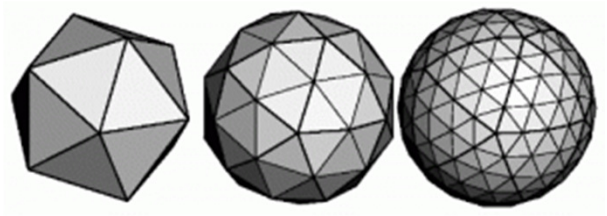
Class Hierarchy

`Sphere`

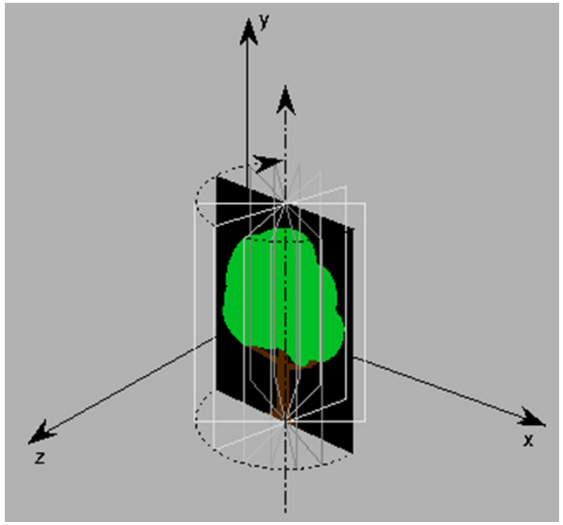
- Derived from Geode
- Pre-defined geometry with parameters, e.g., for tesselation level, solid/wireframe, etc.

`Billboard`

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



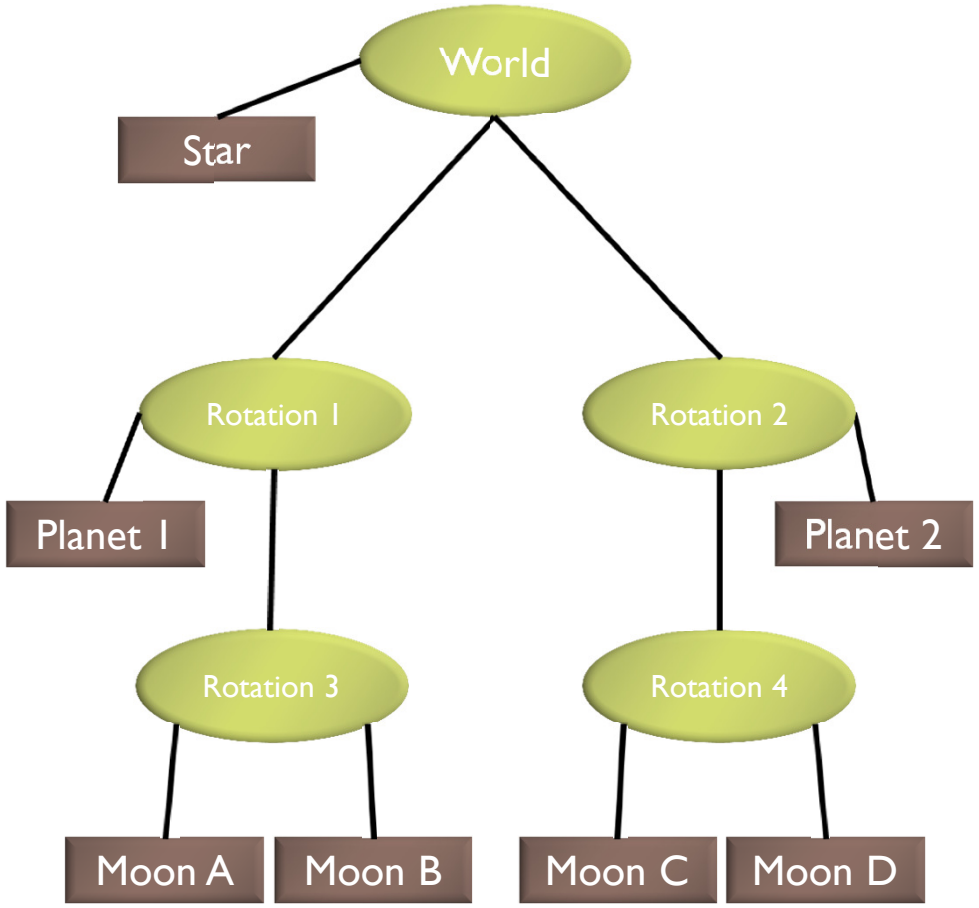
Sphere at different tessellation levels



Billboarded Tree

Source Code for Solar System

```
world = new Group();
world.addChild(new Star());
rotation1 = new MatrixTransform(…);
rotation2 = new MatrixTransform(…);
rotation3 = new MatrixTransform(…);
rotation4 = new MatrixTransform(…);
world.addChild(rotation1);
world.addChild(rotation2);
rotation1.addChild(rotation3);
rotation2.addChild(rotation4);
rotation1.addChild(new Planet("1"));
rotation2.addChild(new Planet("2"));
rotation3.addChild(new Moon("A"));
rotation3.addChild(new Moon("B"));
rotation4.addChild(new Moon("C"));
rotation4.addChild(new Moon("D"));
```



UCSD

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)
{
  setModelViewMatrix(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