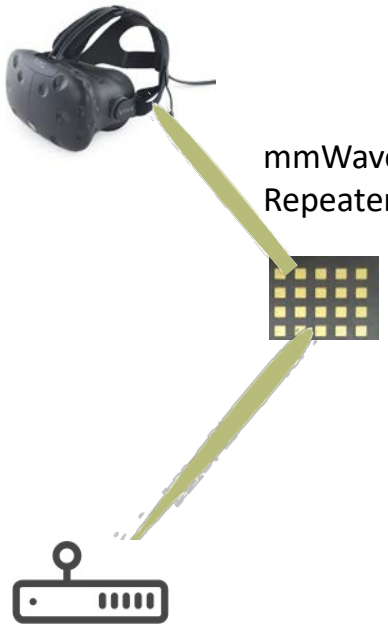
CSE 167:
Introduction to Computer Graphics
Lecture #9: Scene Graph

Jürgen P. Schulze, Ph.D.
University of California, San Diego
Fall Quarter 2017

Announcements

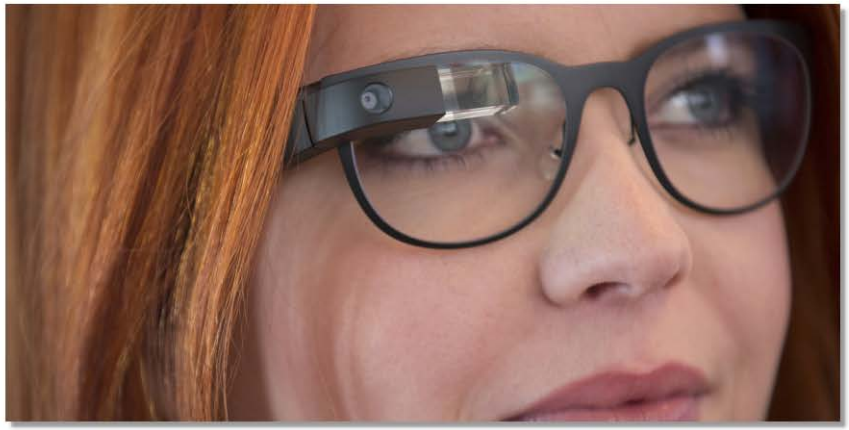
- Project 2 due tomorrow at 2pm
  - Grading in basement labs B260 and B270
  - Will use separate sign up lists on whiteboards in each room
- Midterm #1 next Tuesday in class
- Next Friday: late grading for project 2

Wireless VR – Professor Dinesh Bharadia



Wireless Virtual Reality

Can we untether the VR experience? What we need:



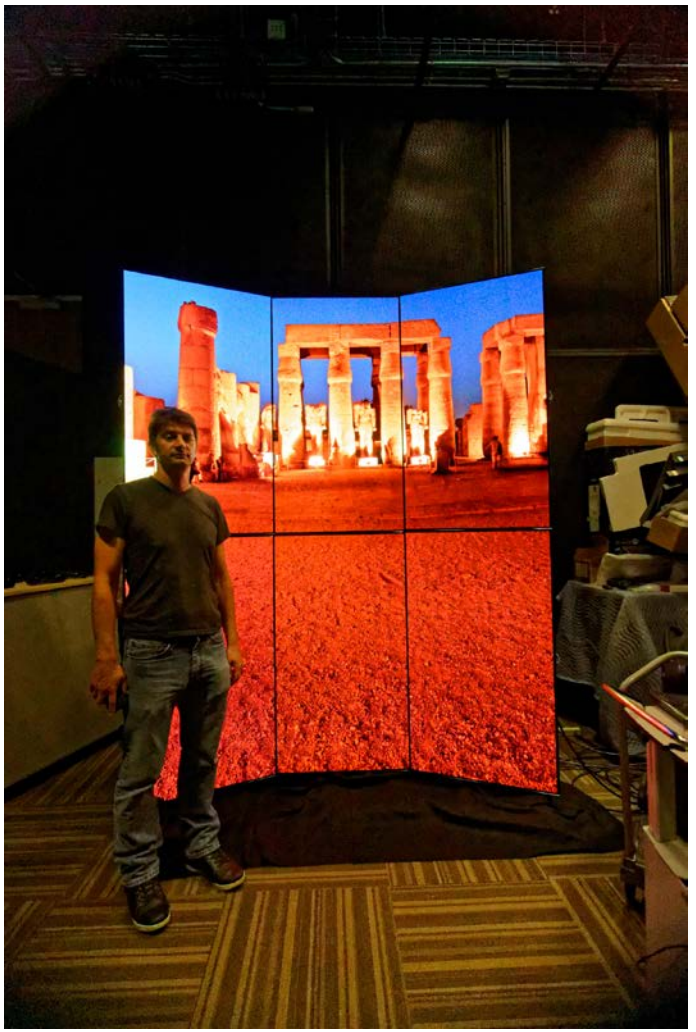
- Off-load compute and sensing to existing network infrastructure
- Fine-grained user context and sensing (head position, direction, speed) provided by the network
- Requires < 20ms RTT latency including compute, high bandwidth, move with user and rich video content

Can the wireless network become the platform for VR / AR?

UCSD

Unity Programmers Wanted

- Need Unity programmer(s) to improve UI on CAVE Kiosk at Geisel library
- Collaboration with Archaeology Department
- Windows 10 PC with 3 Nvidia GTX 1080 GPUs to drive 6 4k displays in 3D stereo
- Interaction with Xbox controller
- Starts immediately
- Can do for CSE 198 Independent Study credit



Ivy Film Festival

My son, Adam Hersko-RonaTas, currently a student at Brown University, is chief coordinator of the New Media section of the Ivy Film Festival, an annual event for student artists.
http://ivyfilmfestival.org/

He is soliciting new media artwork (eg. **360° films, VR games, AR applications**, projection/sound installations, etc.) with a strong narrative component. He wants to invite submissions from students at UCSD and asked me to help.

Who should he contact at UCSD? Where should he send the call?

Akos Rona-Tas
Professor of Sociology
University of California, San Diego

Lecture Overview

- Scene Graphs & Hierarchies
  - Introduction
  - Data structures

Graphics System Architecture

Interactive Applications

- Video 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, SceniX, Torque, Ogre

Low-level graphics API

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

Scene Graph APIs

- OpenSceneGraph (www.openscenegraph.org)
  - For scientific visualization, virtual reality, GIS (geographic information systems)
- NVIDIA SceniX
  - Optimized for shader support
  - Support for interactive ray tracing
  - http://www.nvidia.com/object/scenix-home.html
- Torque 3D
  - Open source game engine
  - For Windows and browser-based games
  - http://www.garagegames.com/products/torque-3d
- Ogre3D
  - Open source rendering engine
  - For Windows, Linux, OSX, Android, iOS, Javascript
  - http://www.ogre3d.org/

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)

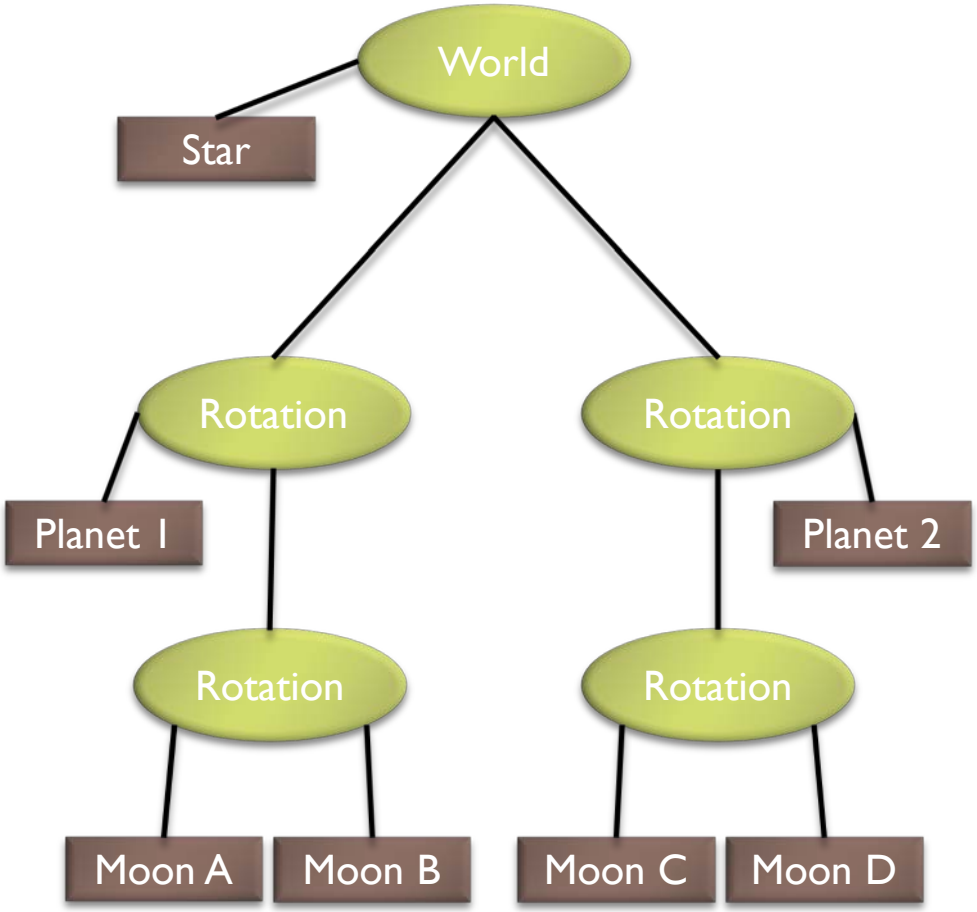
Lecture Overview

- Scene Graphs & Hierarchies
  - Introduction
  - Data structures

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

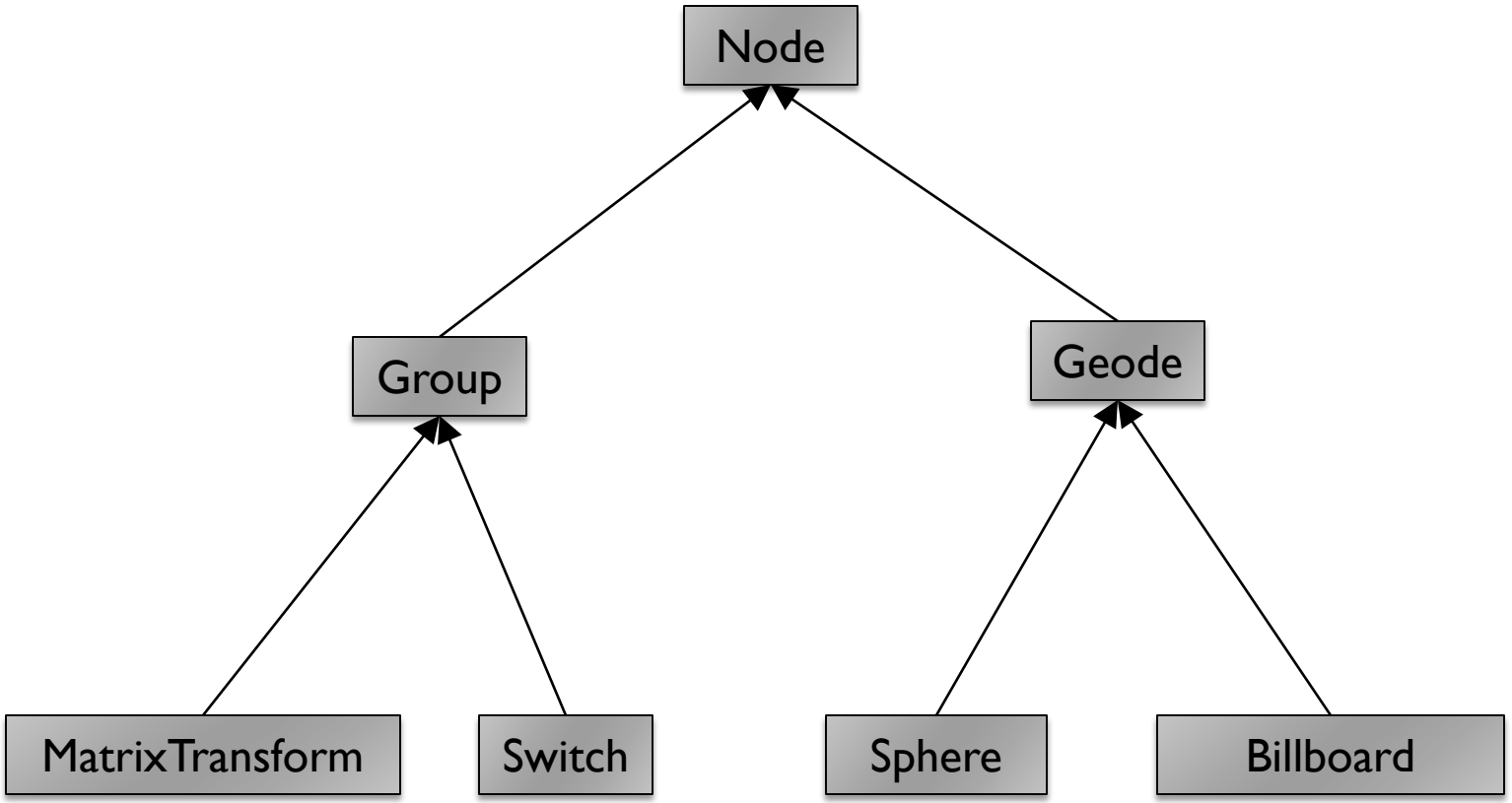
Data Structure

- 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

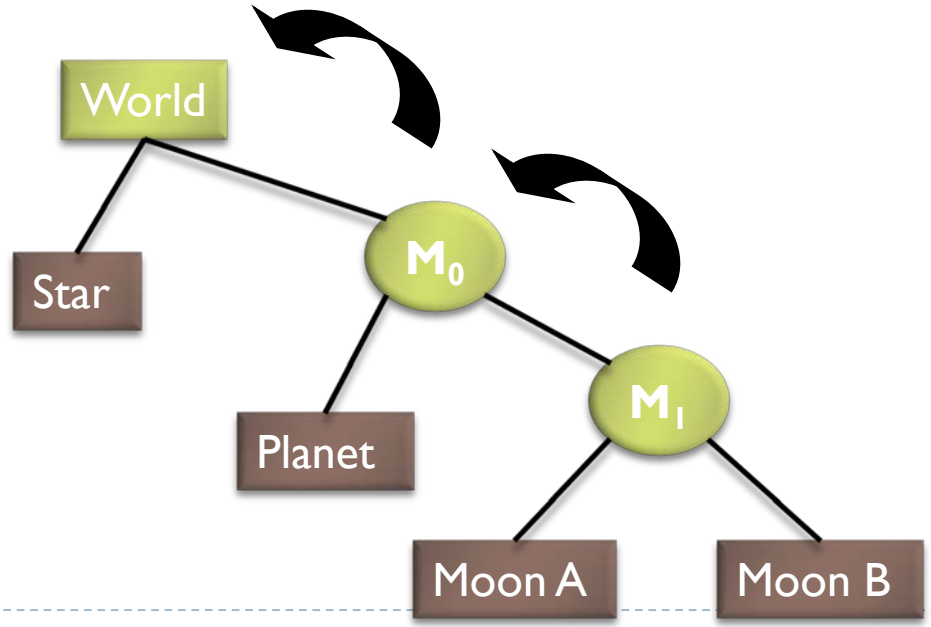
`Geode`

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

Class Hierarchy

`MatrixTransform`

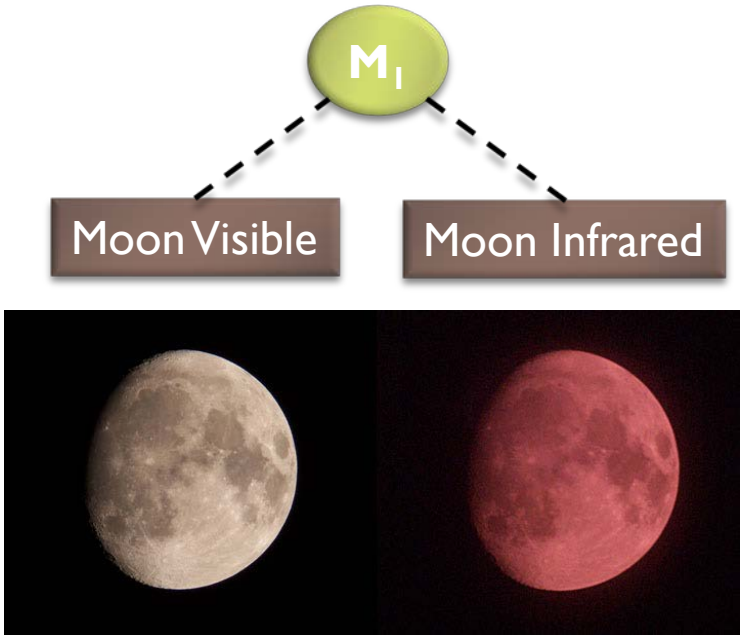
- Derived from Group
- Stores additional transformation **M**
- Transformation applies to sub-tree below node
- Monitor-to-world transformation $\mathbf{M_0 M_1}$



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



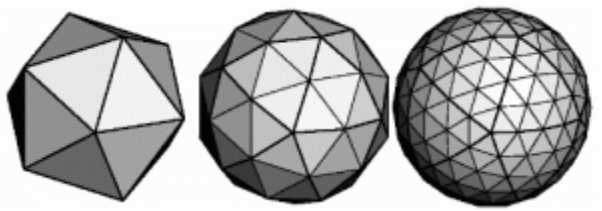
Class Hierarchy

`Sphere`

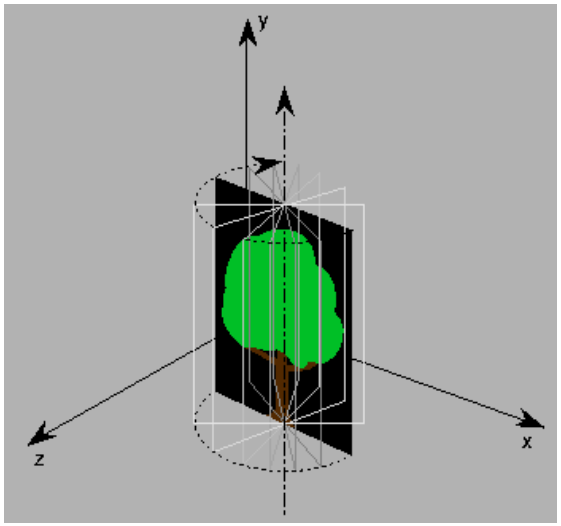
- Derived from Geode
- Pre-defined geometry with parameters, e.g., for tesselation level, solid/wireframe, etc.

`Billboard`

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



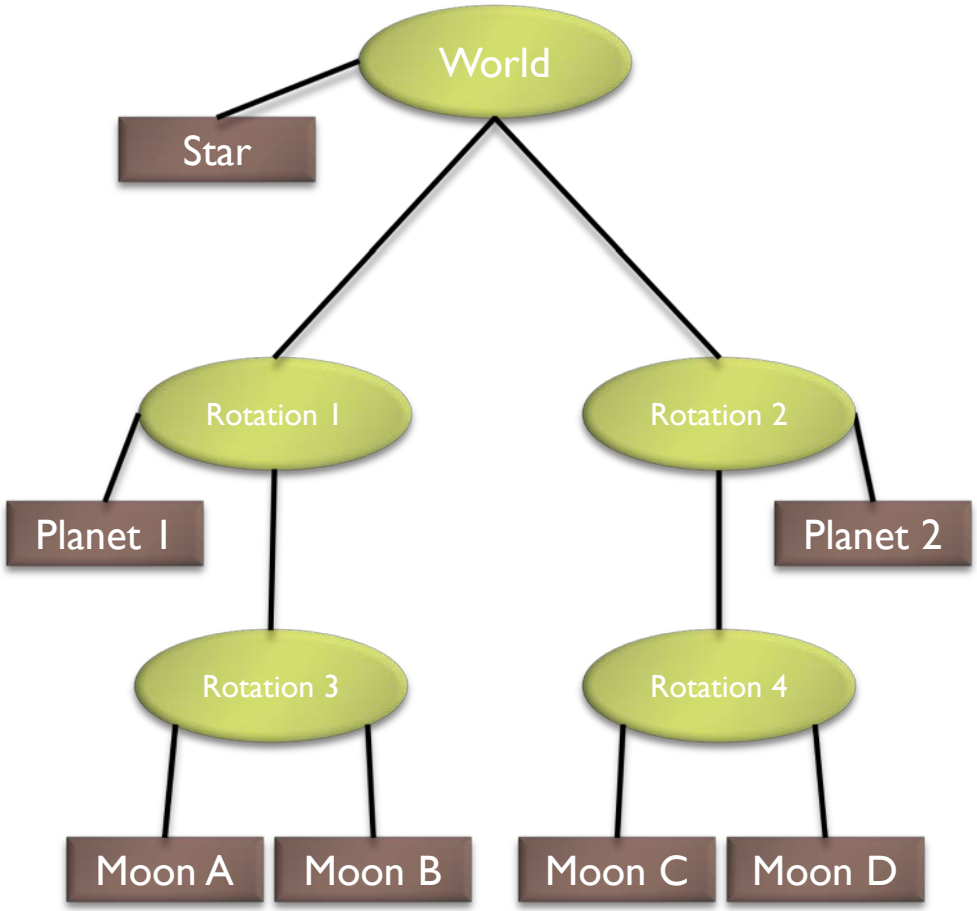
Sphere at different tessellation levels



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"));
```



Basic Rendering

- Traverse the tree recursively

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

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

Geode::draw(Matrix4 M)
{
  setModelMatrix(M);
  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