CSE 167:
Introduction to Computer Graphics
Lecture #6: Scene Graph

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

Fall Quarter 2019

Announcements

- Thursday: midterm exam #1
  - In-class
  - Closed book, no cheat sheets
  - Allowed: pen, pencil, eraser, ruler, triangle ruler, protractor, scratch paper
- Friday: late grading for homework project 2
  - Upload code to Canvas by 2pm
  - Demonstrate in CSE basement labs
- Next Monday: discussion homework project 3
- Next Tuesday: lecture given by TA
- Next Friday: homework 3 due at 2pm
- Today: homework project 3 introduction

Lecture Overview

- Scene Graphs & Hierarchies
  - Introduction
  - Data structures

Graphics System Architecture

Interactive Applications

- Video games, scientific visualization, CAD modeling

Rendering Engine, Scene Graph API

- Implement functionality commonly required in applications
- Back-ends for different low-level APIs
- No broadly accepted standards
- OpenSceneGraph, Nvidia SceniX, Torque3D, Ogre3D

Low-level graphics API

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

Commonly Offered Functionality

- High-level scene representation
  - Graph data structure
- Resource management
  - File loaders for geometry, textures, materials, animation sequences
  - Memory management
    - CPU <-> GPU memory
    - HDD <-> CPU memory
- Rendering
  - Optimized for efficiency (e.g., minimize OpenGL state changes)

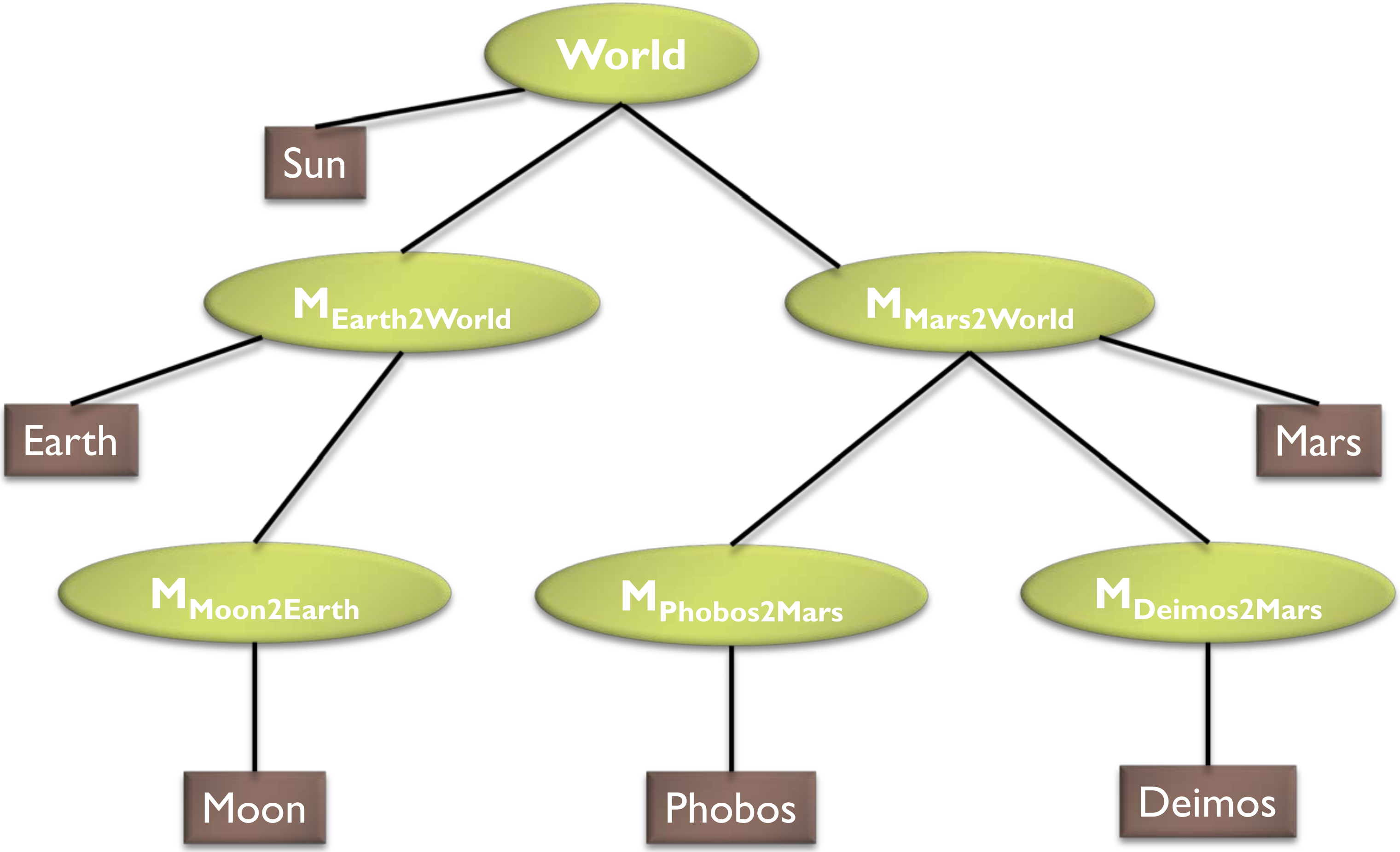
Lecture Overview

- 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
  - Does not scale to large numbers of objects in complex dynamic scenes

Example: Scene Graph for Solar System



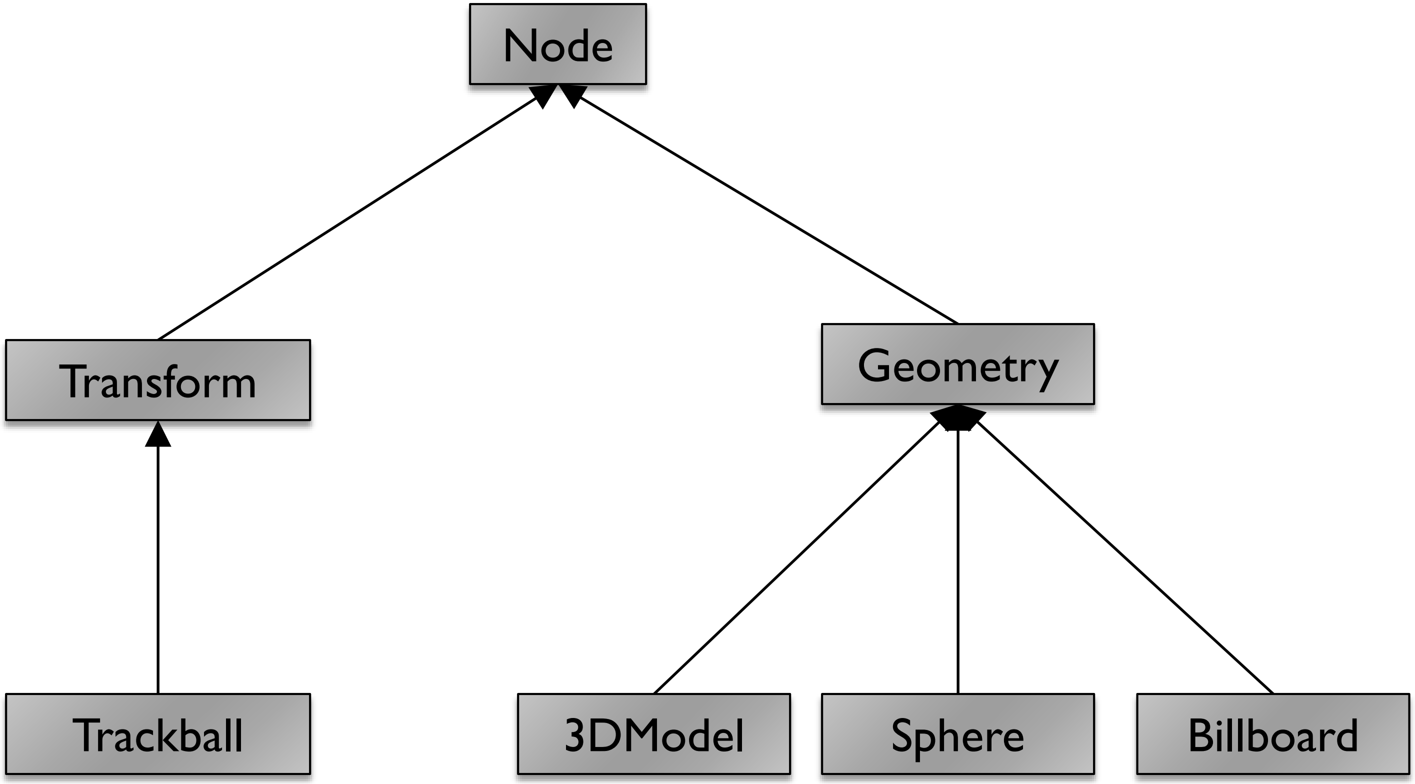
Data Structure

- 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



Class Hierarchy

`Node`

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

`Geometry`

Geometry

- sets the modelview matrix to the current C matrix
- has a class method which draws its associated geometry

`Transform`

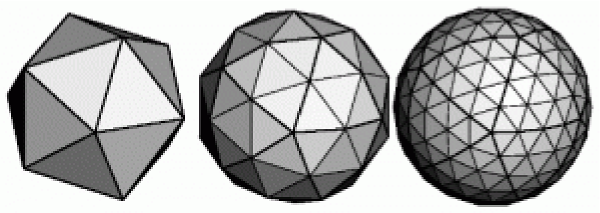
Transform

- Stores list of children
- Stores 4x4 matrix for affine transformation

Class Hierarchy

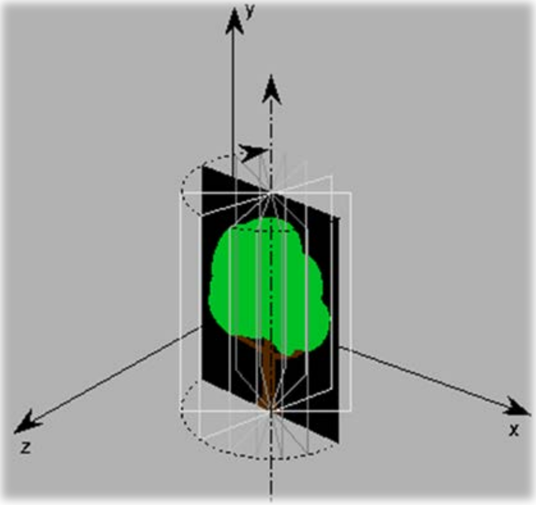
`Sphere`

- Derived from Geometry node
- Pre-defined geometry with parameters, e.g., for tesselation level (number of triangles), solid/wireframe, etc.



`Billboard`

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



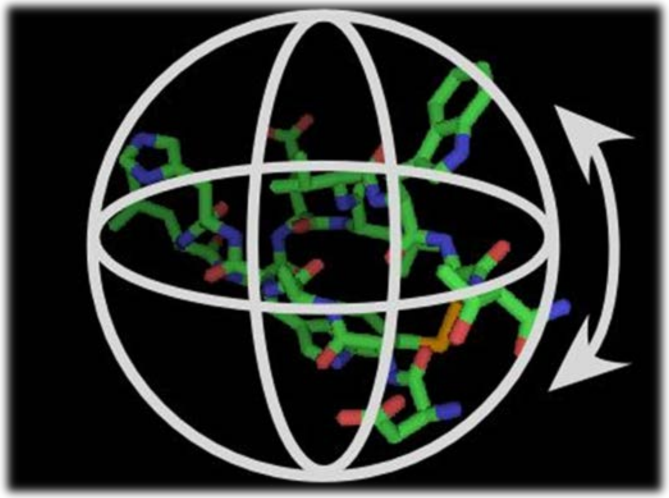
Class Hierarchy

`3DModel`

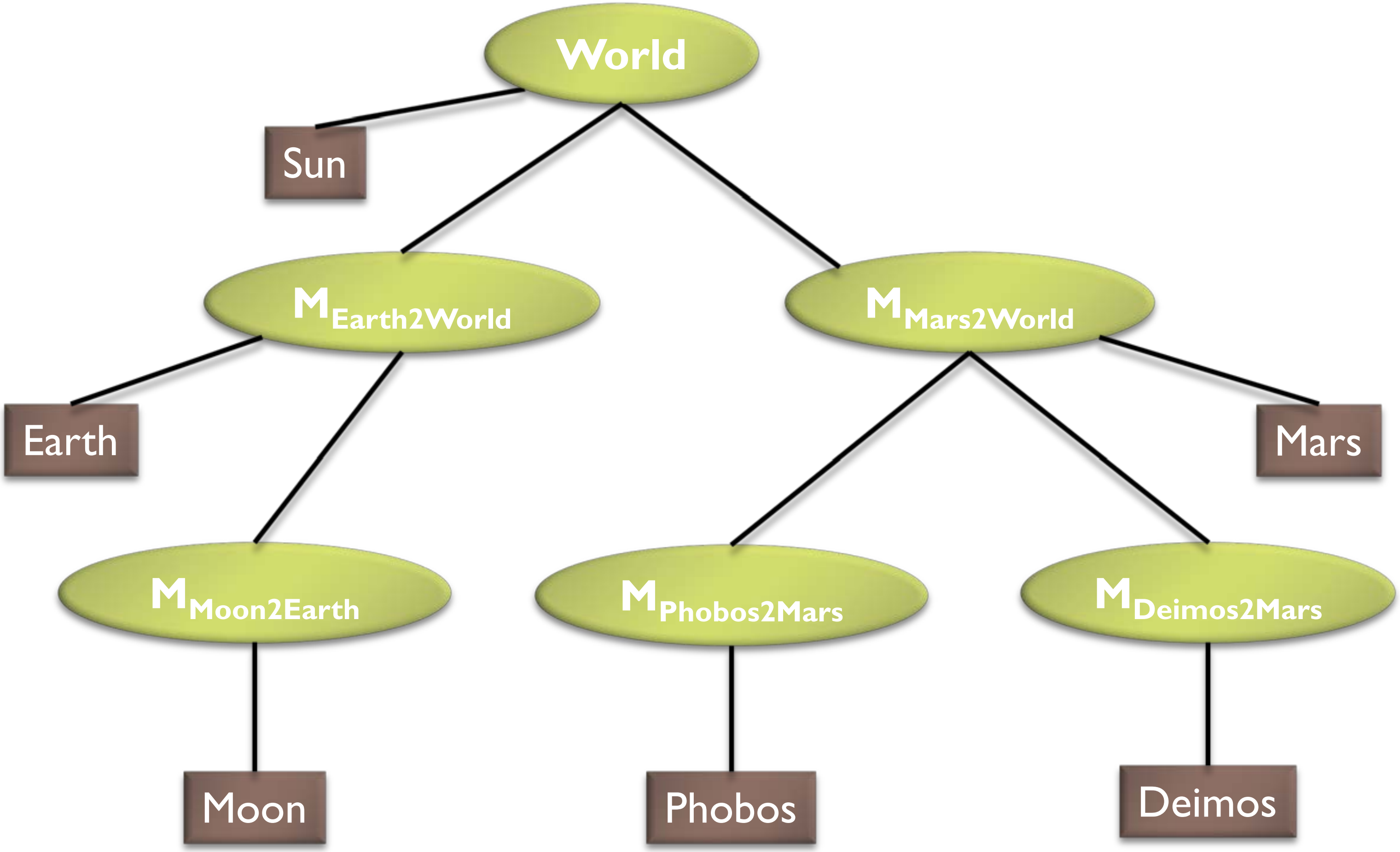
- Takes file name to load 3D model file

`Trackball`

- Creates the matrix transformation based on a virtual trackball controlled with the mouse



Scene Graph for Solar System



Building the Solar System

```
// create sun:
world = new Transform();
world.addChild(new Model(“Sun.obj”));

// create planets:
earth2world = new Transform(…);
mars2world = new Transform(…);
earth2world.addChild(new Model(“Earth.obj”));
mars2world.addChild(new Model(“Mars.obj”));
world.addChild(earth2world);
world.addChild(mars2world);

// create moons:
moon2earth = new Transform(…);
phobos2mars = new Transform(…);
deimos2mars = new Transform(…);
moon2earth.addChild(new Model(“Moon.obj”));
phobos2mars.addChild(new Model(“Phobos.obj”));
deimos2mars.addChild(new Model(“Deimos.obj”));
earth2world.addChild(moon2earth);
mars2world.addChild(phobos2mars);
mars2world.addChild(deimos2mars);
```

Transformation Calculations

- moon2world = moon2earth * earth2world;
- phobos2world = phobos2mars * mars2world;
- deimos2world = deimos2mars * mars2world;

Scene Rendering

- Recursive draw calls

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

Geometry::draw(Matrix4 M)
{
  setModelMatrix(M);
  render(myObject);
}
```

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

Ideas for Scene Graph Nodes

- 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 drone-style camera
  - Light source
- Provide complex functionality as nodes
  - Video node
  - Elevator node
  - Terrain rendering node