CSE 167:
Introduction to Computer Graphics
Lecture 10: Scene Graph

Jürgen P. Schulze, Ph.D.
University of California, San Diego
Fall Quarter 2013

Midterm

- Midterm has been graded
  - A score of 90 in the exam will count as a grade of 100
- Please return midterm after review if you want to discuss with me later
  - Otherwise feel free to keep it

# Submissions	**112**
Average score	**60.0**
Median score	**61.0**
Highest score	**89.5**
Lowest score	**9.5**

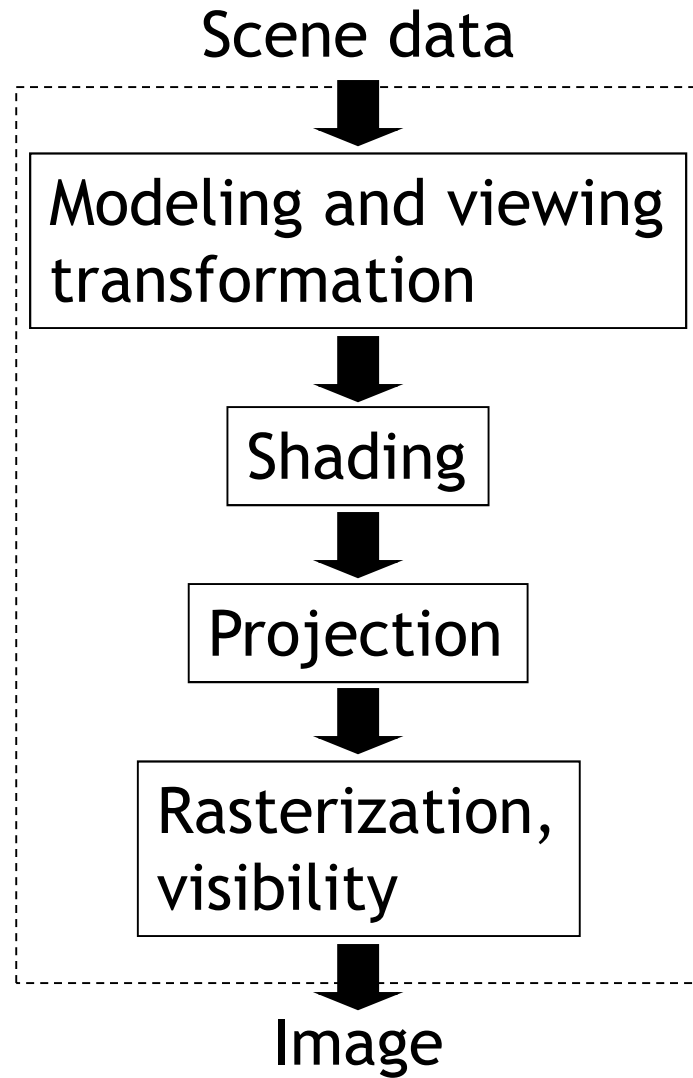# Announcements

- Homework #4:
  Glee web site has been down:
  Matteo put files on Dropbox link: see course forums
- Homework #5 discussion on Monday, Nov 4

Lecture Overview

- Scene Graphs & Hierarchies
  - Introduction
  - Data structures

- Performance Optimization
  - Level-of-detail techniques
  - View Frustum Culling

Rendering Pipeline

Scene data

Modeling and viewing transformation

Shading

Projection

Rasterization, visibility

Image

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
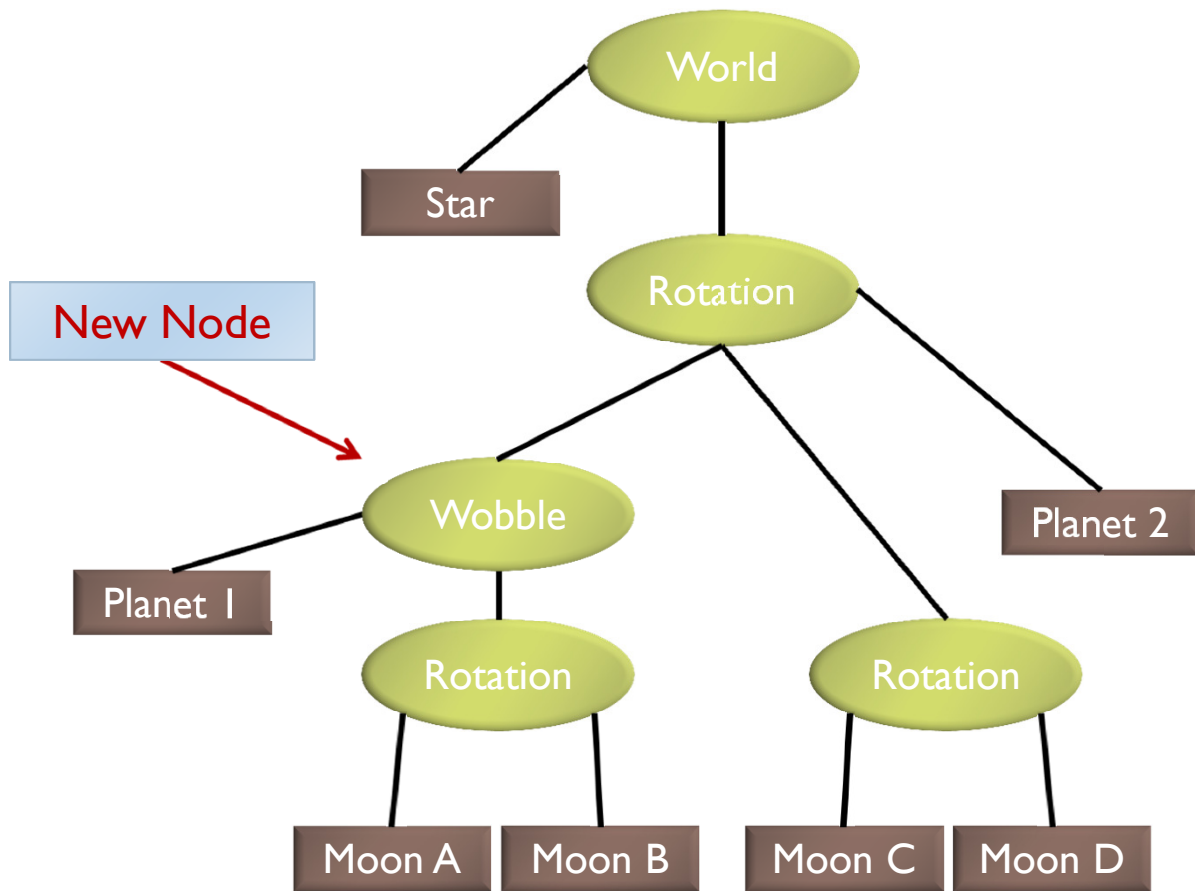
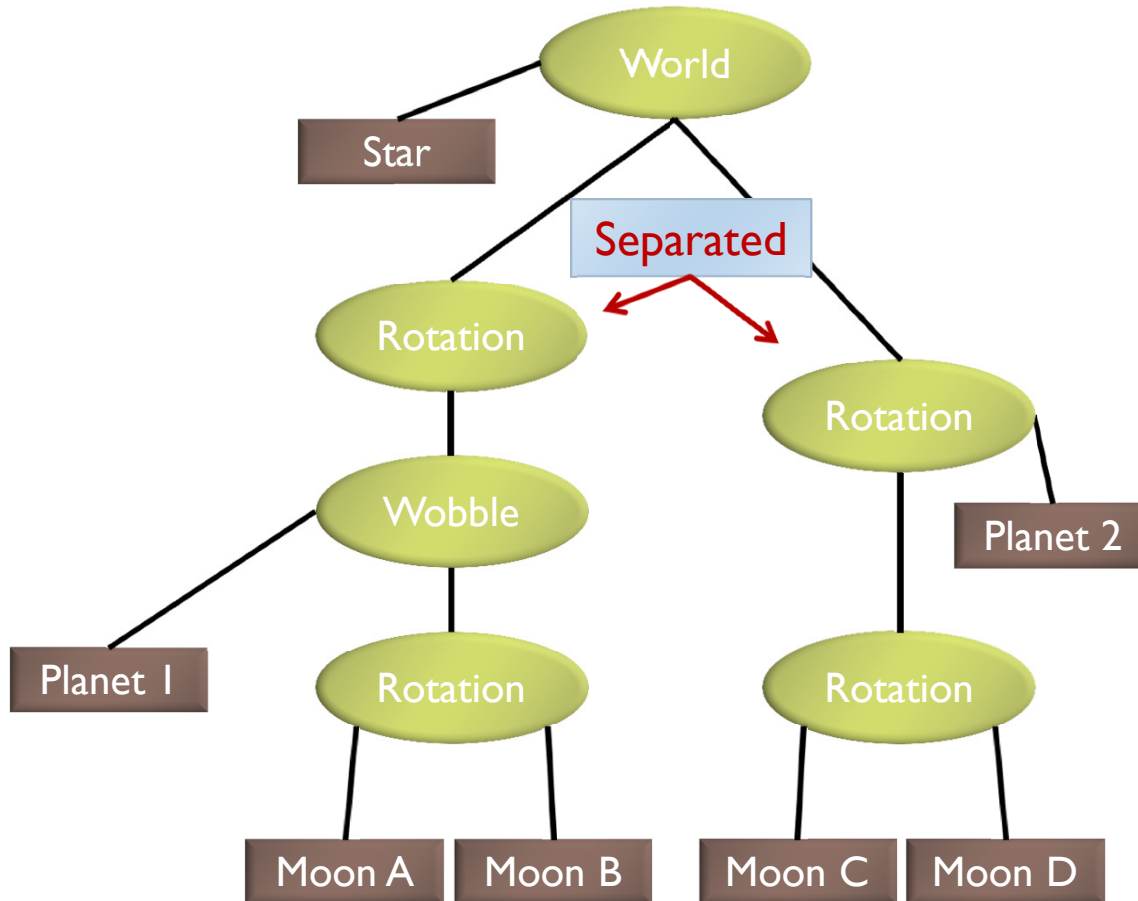 → Homework Assignment #1 – Animated Objects

Solar System
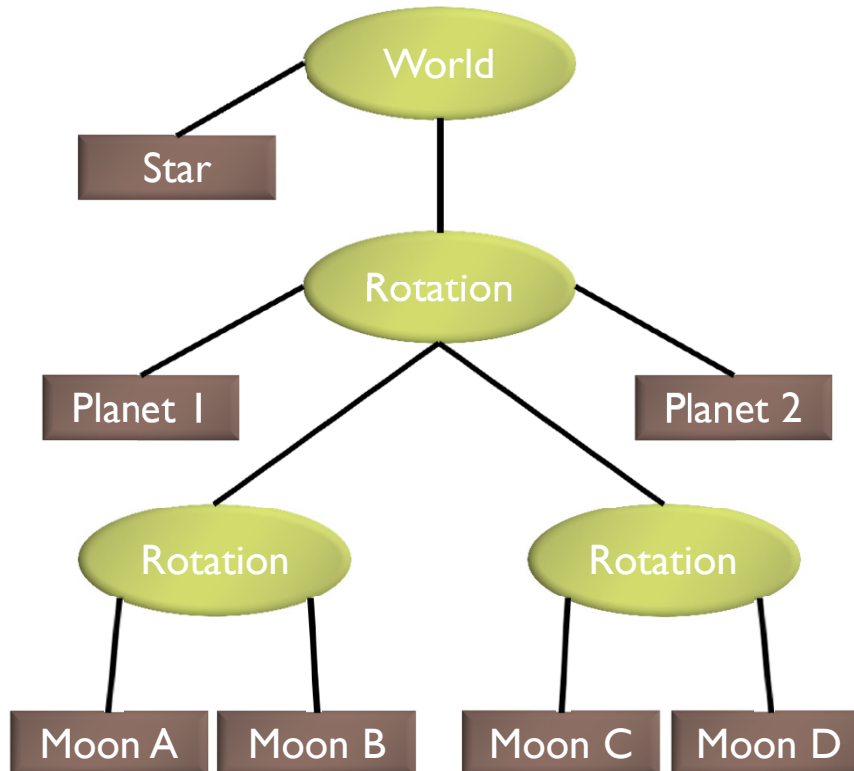
- Draw the star
- Save the current matrix

- - 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

- 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
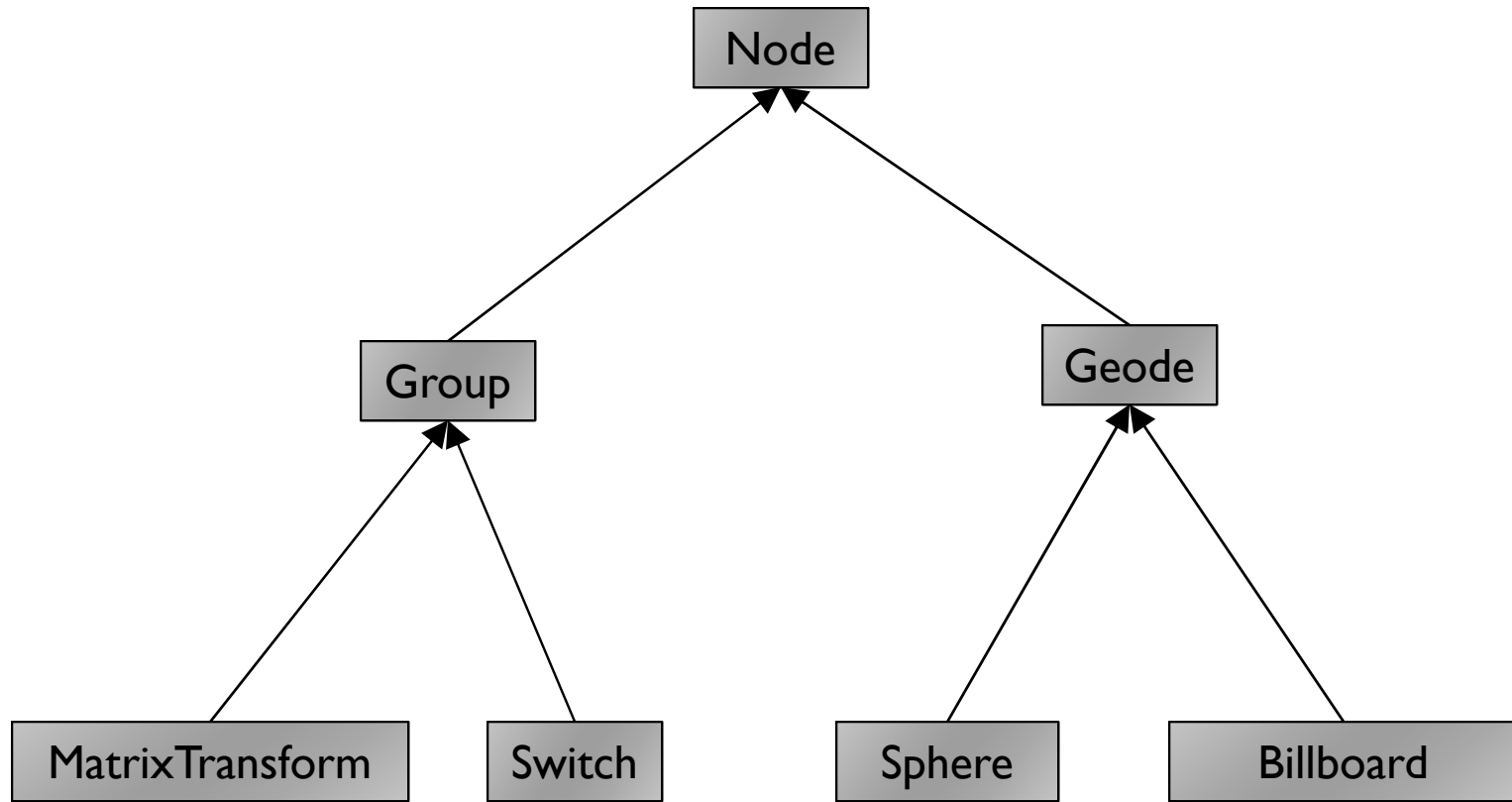
Inspired by OpenSceneGraph

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 Geode

▸ Stores additional transformation **M**

▸ Transformation applies to sub-tree below node

▸ Monitor-to-world transformation $M_0 M_1$
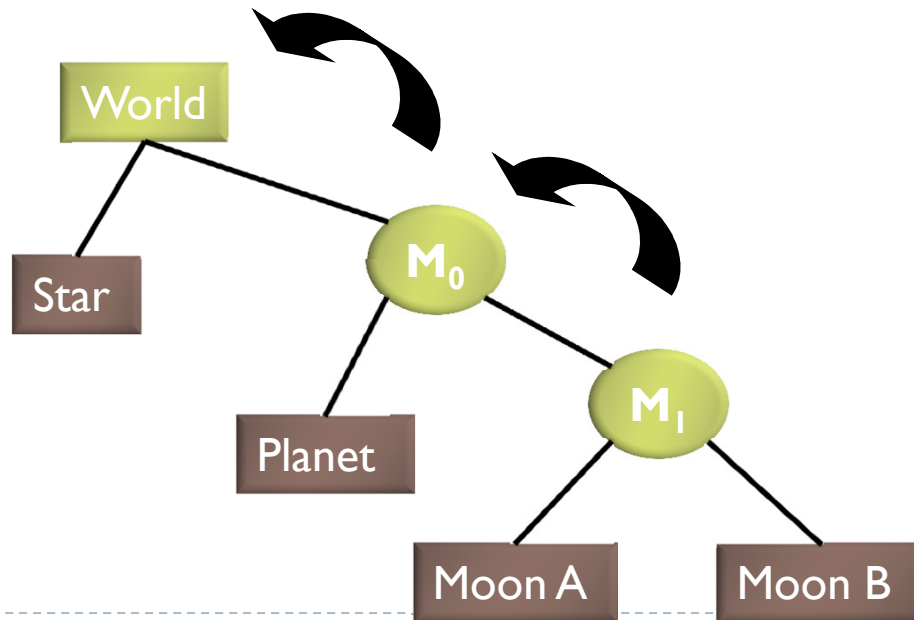
Class Hierarchy

`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
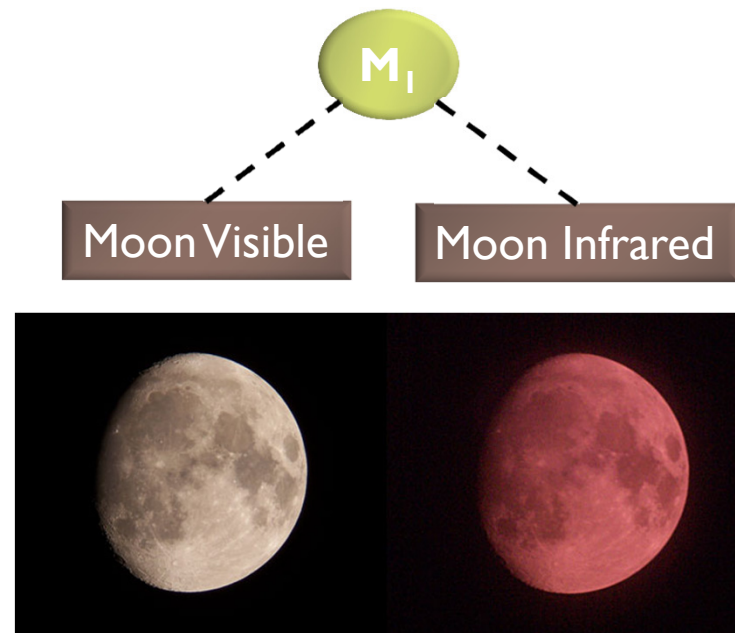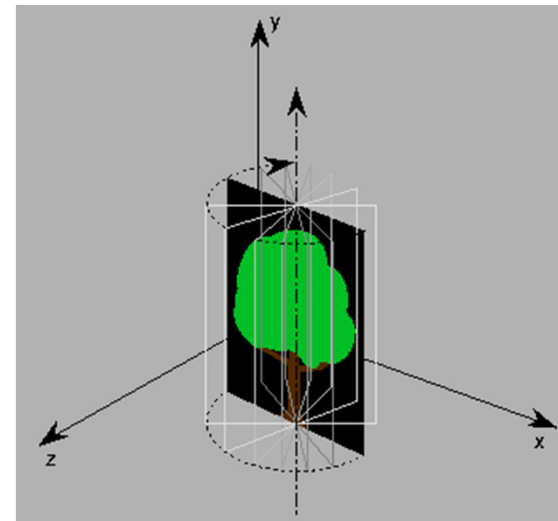
Class Hierarchy

`Sphere`

▸ Derived from Geode

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

`Billboard`

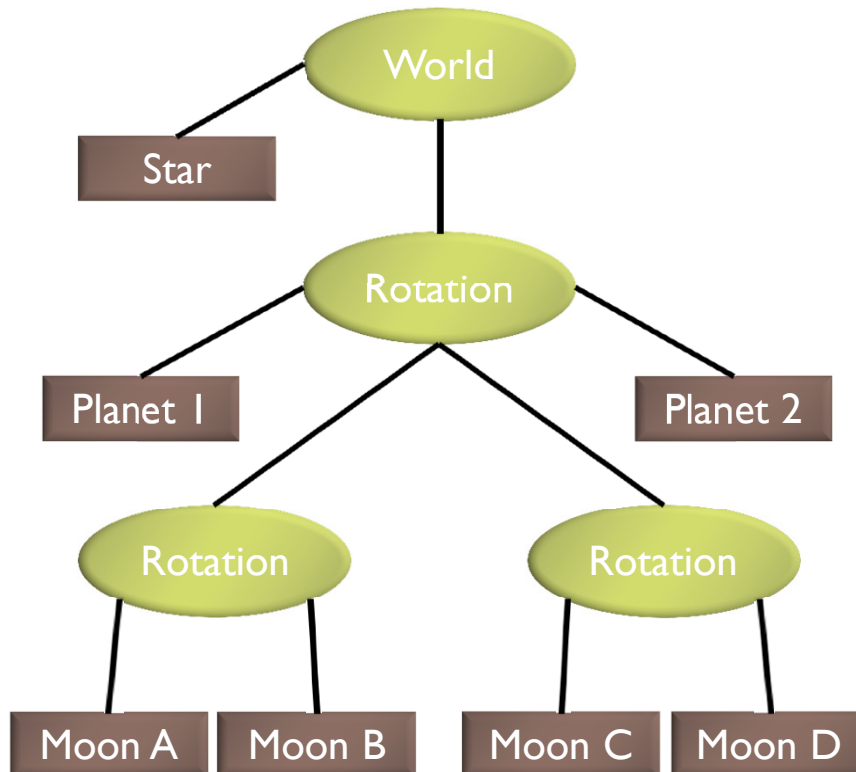▸ Special geometry node to display an image always facing the viewer
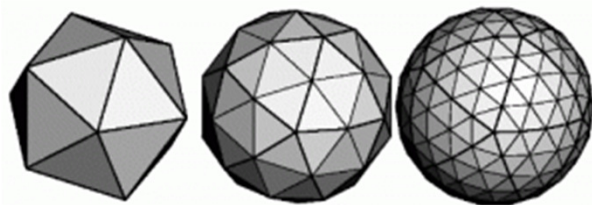
Sphere at different tessellation levels

Billboarded Tree

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"));
```

World

Star

Rotation 0

Planet 1   Planet 2

Rotation 1   Rotation 2

Moon A   Moon B   Moon C   Moon D
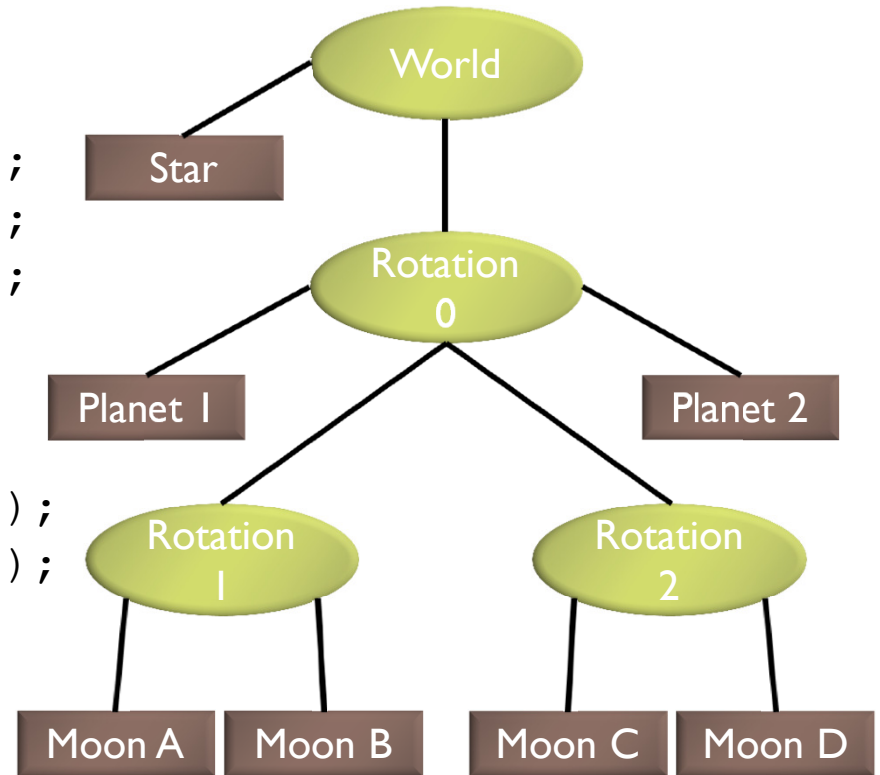
Basic Rendering

▶ **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

Lecture Overview

- Scene Graphs & Hierarchies

  - Introduction

  - Data structures

- Performance Optimization

  - Level-of-detail techniques

  - View Frustum Culling

Level-of-Detail Techniques

▶ **Don't draw objects smaller than a threshold**

  ▸ Small feature culling

  ▸ Popping artifacts

▶ **Replace 3D objects by 2D impostors**

  ▸ Textured planes representing the objects

Impostor generation

▶ **Adapt triangle count to projected size**
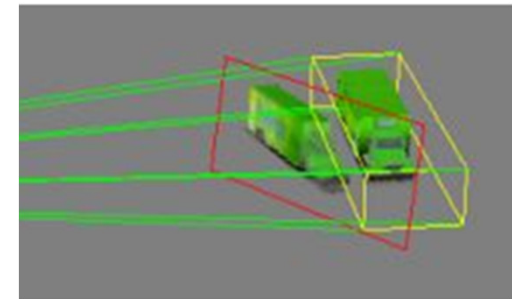
Original vs. impostor

Size dependent mesh reduction
(Data: Stanford Armadillo)

View Frustum Culling

- Frustum defined by 6 planes
- Each plane divides space into "outside", "inside"
- Check each object against each plane
  - Outside, inside, intersecting
- If "outside" all planes
  - Outside the frustum
- If "inside" all planes
  - Inside the frustum
- Else partly inside and partly out
- Efficiency

PFIS_FALSE

PFIS_ALL_IN

PFIS_TRUE
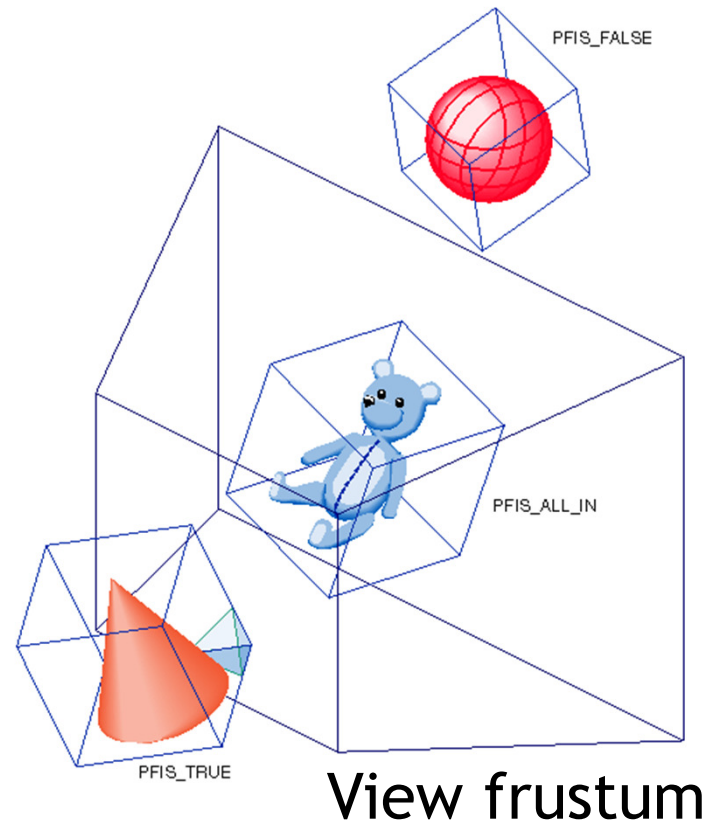
View frustum

Bounding Volumes

▶ **Simple shape that completely encloses an object**

▶ **Generally a box or sphere**

▶ **We use spheres**

  ▶ Easiest to work with

  ▶ But hard to calculate tight fits

▶ **Intersect bounding volume with view frustum instead of each primitive**
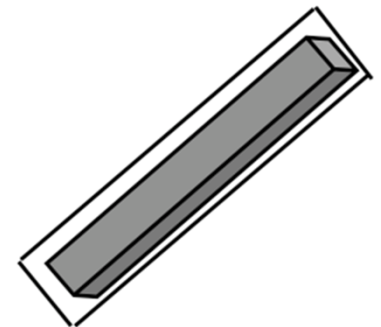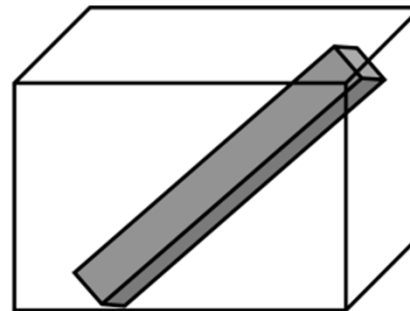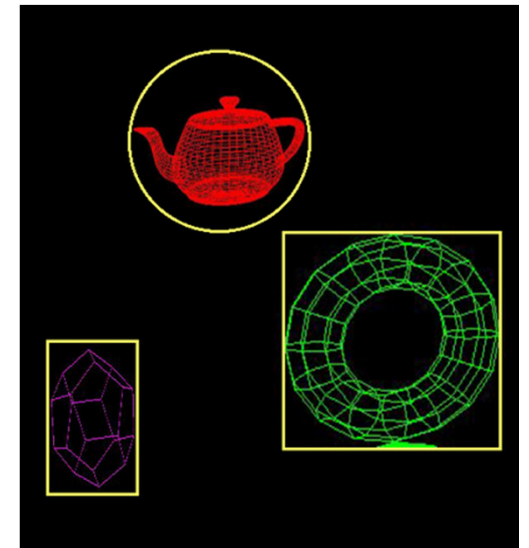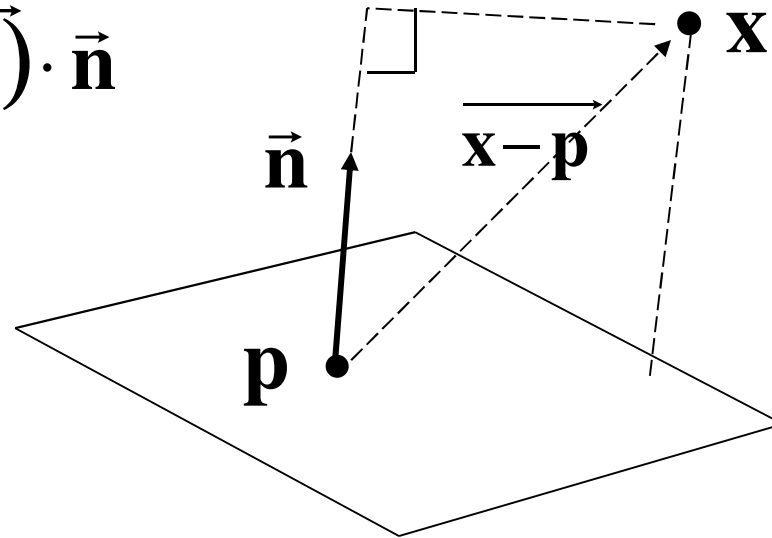
Distance to Plane

▸ A plane is described by a point **p** on the plane and a unit normal **n**

▸ Find the (perpendicular) distance from point **x** to the plane

• **x**

$$\vec{n}$$

p •
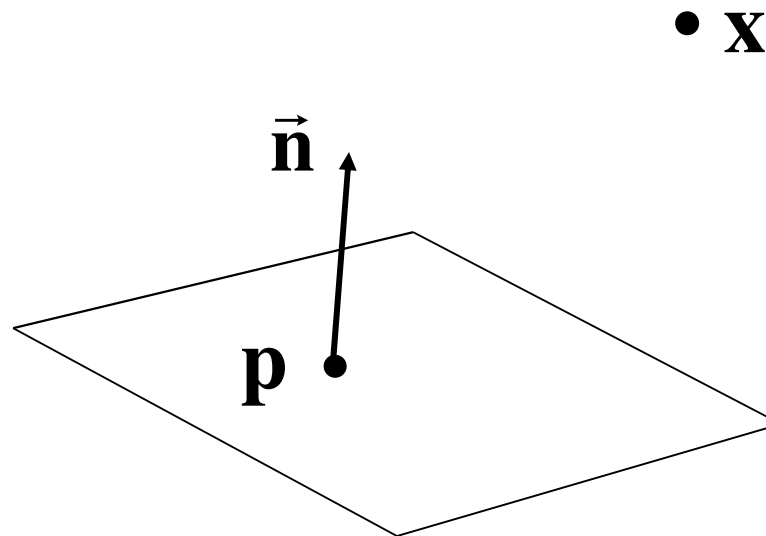
Distance to Plane

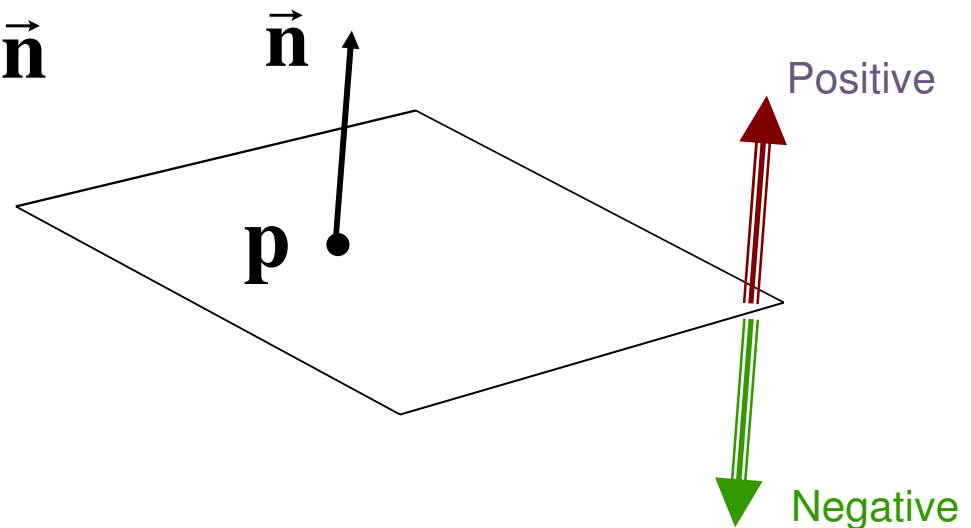▸ The distance is the length of the projection of **x-p** onto **n**

$$dist = \overrightarrow{(\mathbf{x} - \mathbf{p})} \cdot \vec{\mathbf{n}}$$

Distance to Plane

▸ The distance has a sign

  ▸ positive on the side of the plane the normal points to

  ▸ negative on the opposite side

  ▸ zero exactly on the plane

▸ Divides 3D space into two infinite half-spaces

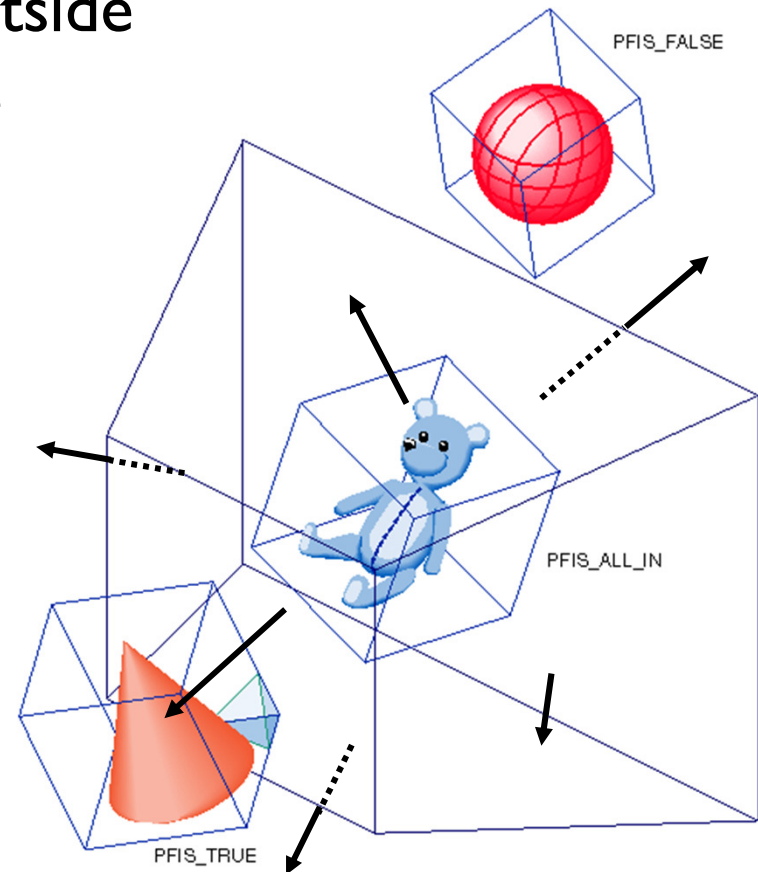$$dist(\mathbf{x}) = \overrightarrow{(\mathbf{x} - \mathbf{p})} \cdot \vec{\mathbf{n}}$$

$\vec{\mathbf{n}}$

Positive

$\mathbf{p}$

Negative

Distance to Plane

- Simplification

$$dist(\mathbf{x}) = (\mathbf{x} - \mathbf{p}) \cdot \mathbf{n}$$
$$= \mathbf{x} \cdot \mathbf{n} - \mathbf{p} \cdot \mathbf{n}$$
$$dist(\mathbf{x}) = \mathbf{x} \cdot \mathbf{n} - d, \quad d = \mathbf{pn}$$

- d is independent of $\mathbf{x}$
- d is distance from the origin to the plane
- We can represent a plane with just d and $\mathbf{n}$

Frustum With Signed Planes

- Normal of each plane points outside
  - "outside" means positive distance
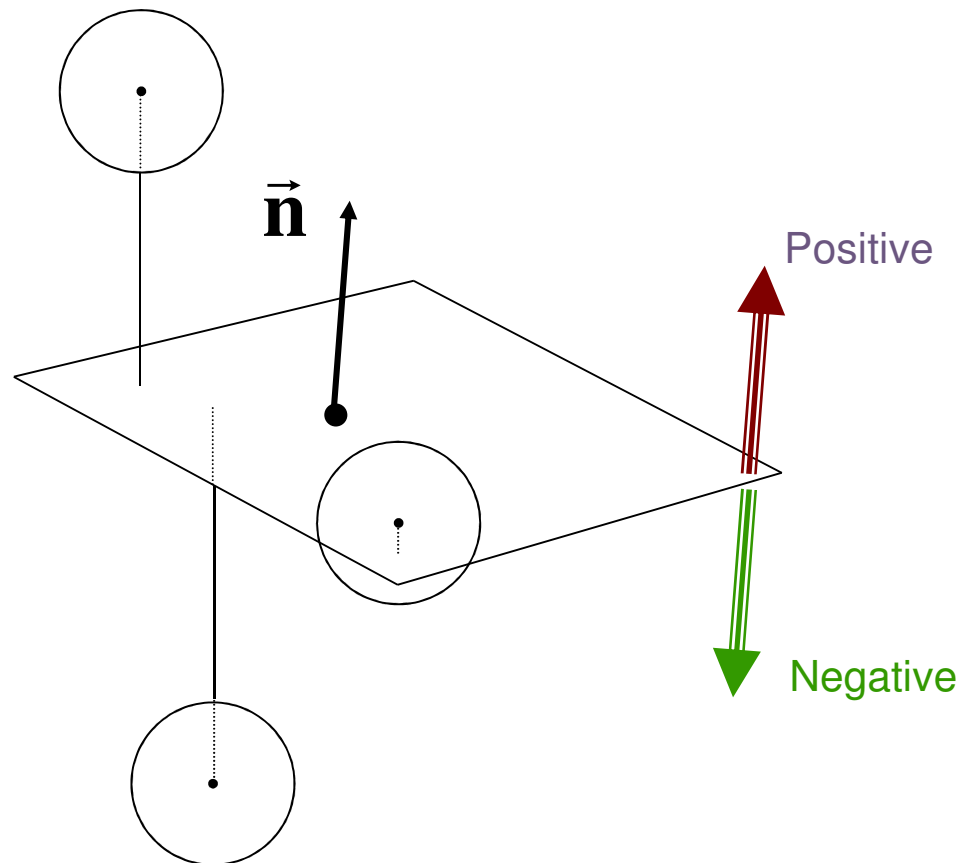 - "inside" means negative distance

Test Sphere and Plane

▸ For sphere with radius r and origin $\mathbf{x}$, test the distance to the origin, and see if it is beyond the radius

▸ Three cases:

  ▸ *dist($\mathbf{x}$)>r*

    ▸ completely above

  ▸ *dist($\mathbf{x}$)<-r*

    ▸ completely below

  ▸ *-r<dist($\mathbf{x}$)<r*

    ▸ intersects

$\vec{\mathbf{n}}$

Positive

Negative

Culling Summary

- Precompute the normal $\mathbf{n}$ and value d for each of the six planes.
- Given a sphere with center $\mathbf{x}$ and radius r
- For each plane:
  - if $dist(\mathbf{x}) > r$: sphere is outside! (no need to continue loop)
  - add 1 to count if $dist(\mathbf{x}) < -r$
- If we made it through the loop, check the count:
  - if the count is 6, the sphere is completely inside
  - otherwise the sphere intersects the frustum
 - *(can use a flag instead of a count)*

Culling Groups of Objects

▸ Want to be able to cull the whole group quickly

▸ But if the group is partly in and partly out, want to be able to cull individual objects
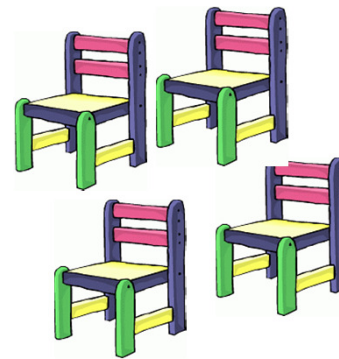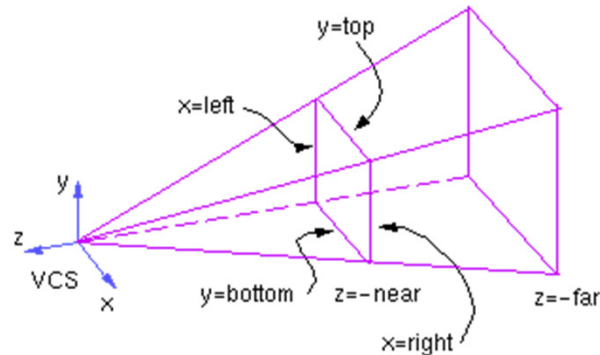
Hierarchical Bounding Volumes

▸ Given hierarchy of objects

▸ Bounding volume of each node encloses the bounding volumes of all its children

▸ Start by testing the outermost bounding volume

　▸ If it is entirely outside, don't draw the group at all

　▸ If it is entirely inside, draw the whole group
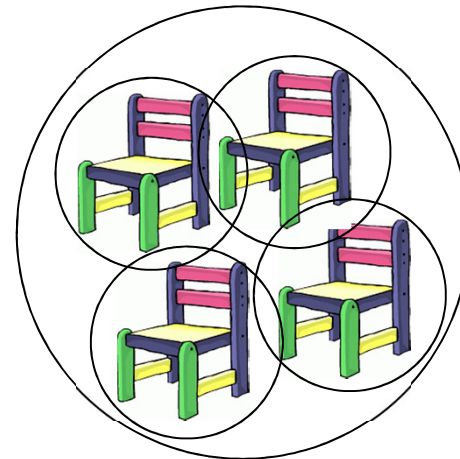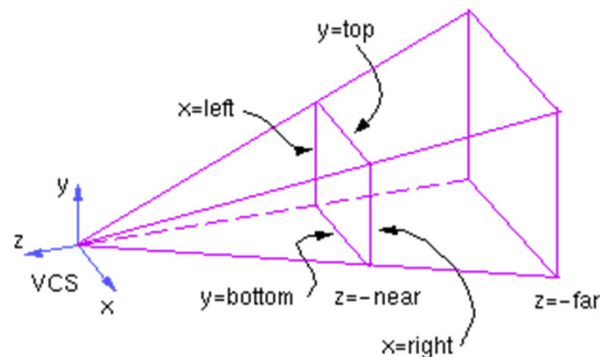
Hierarchical Culling

▶ **If the bounding volume is partly inside and partly outside**

   ▶ Test each child's bounding volume individually

   ▶ If the child is in, draw it; if it's out cull it; if it's partly in and partly out, recurse.

   ▶ If recursion reaches a leaf node, draw it normally
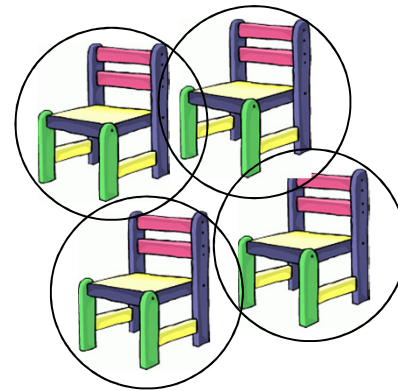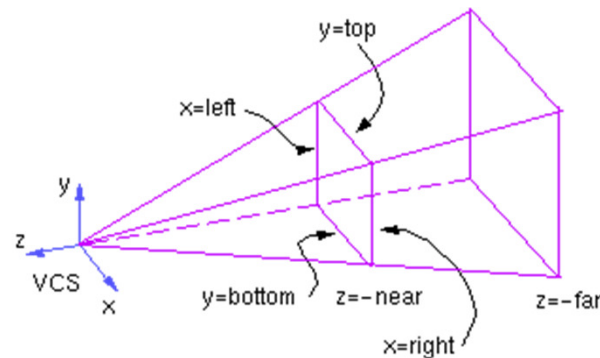
Hierarchical Culling: Octree

▸ Octrees are the three-dimensional analog of quadtrees.

▸ An octree is a tree data structure in which each node has exactly eight children.

▸ Most often used to partition a 3D space by recursively subdividing it into eight octants.
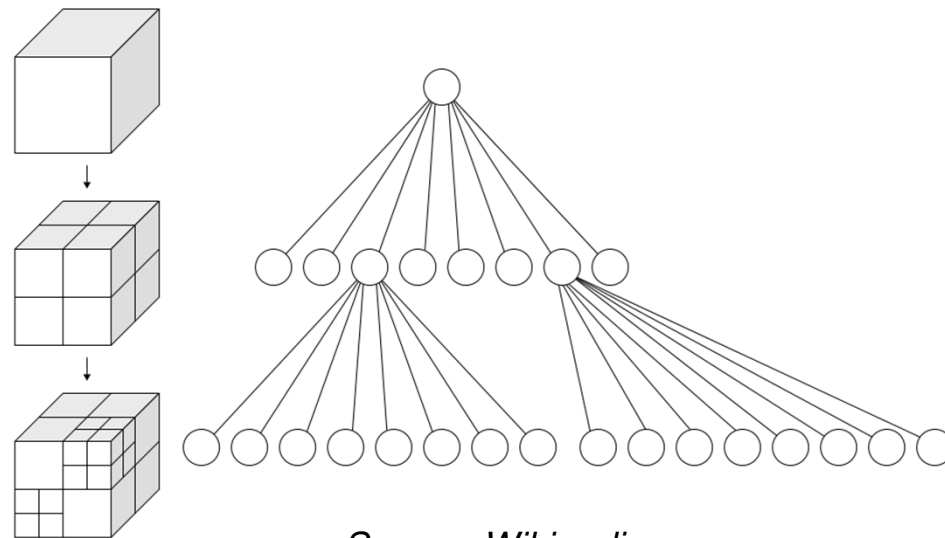
Source: Wikipedia

Video

- **An OpenGL Demo - Frustum Culling with Octree**
  - http://www.youtube.com/watch?v=H-SsvZZv1sw