Announcements

- Midterm exam this Thursday, Oct 27
- Midterm tutorial today 3:45pm-5pm
  - At Atkinson Hall, room 4004
- Homework assignment #4 due Friday, Oct 28, grading in lab 260 starts as usual at 1:30pm
- Texturing topic will continue next week
  Today: Scene Graph
Lecture Overview

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

Scene data

Modeling and viewing transformation

Shading

Projection

Rasterization, visibility

Image
# Graphics System Architecture

## Interactive Applications
- Games, virtual reality, visualization

## Rendering Engine, Scene Graph API
- Implement functionality commonly required in applications
- Back-ends for different low-level APIs
- No broadly accepted standards
- Examples: OpenSceneGraph, OpenSG, NVSG, Java3D, Ogre

## Low-level graphics API
- Interface to graphics hardware
- Highly standardized: OpenGL, Direct3D
Scene Graph APIs

- APIs focus on different clients/applications
- **Java3D** ([https://java3d.dev.java.net/](https://java3d.dev.java.net/))
  - Simple, easy to use, web-based applications
- **OpenSceneGraph** ([www.openscenegraph.org](http://www.openscenegraph.org))
  - Scientific visualization, virtual reality, GIS (geographic information systems)
  - Optimized for Nvidia graphics cards
  - Up-to-date shader support (Cg 2.2)
- **Ogre3D** ([http://www.ogre3d.org/](http://www.ogre3d.org/))
  - Games, high-performance rendering
Common 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
  - Occlusion Culling
  - 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 – Exercise 3 (Animated Objects)
Solar System

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

- Node
  - Group
    - MatrixTransform
    - Switch
  - Geode
    - Sphere
    - osgEarth
Class Hierarchy

Node

- **Access to local-to-world coordinate transformation**

Group

- **Stores list of children**
- **Get, add, remove children**

Geode

- **Geometry node, does not have children**
Class Hierarchy

MatrixTransform

- Stores additional transformation $M$
- Transformation applies to sub-tree below node
- Monitor-to-world transformation $M_0M_1$
Class Hierarchy

Switch

- Stores list of children
- Only one child is visible at a time
- Used to step through children (“key frame” animation), or to show/hide a sub graph
Class Hierarchy

Sphere

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

osgEarth

- Special geometry node providing Google Earth-like functionality

Sphere at different tessellation levels
Source Code for Solar System

```java
world = new Group();
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(1));
rotation1.addChild(new Moon(2));
rotation2.addChild(new Moon(3));
rotation2.addChild(new Moon(4));
```
Basic Rendering

- Traverse the tree recursively

```cpp
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
```cpp
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
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
- Can provide powerful visual objects with simple APIs
  - Complex node types, e.g., planet visualization node
Lecture Overview

- Scene Graphs & Hierarchies
  - Introduction
  - Data structures
- **Performance Optimization**
  - Level-of-detail techniques
  - Occlusion Culling
  - View Frustum Culling
Level-of-Detail Techniques

- Don’t draw objects smaller than a threshold
  - Popping artifacts

- Replace objects by impostors
  - Textured planes representing the objects

- Adapt triangle count to projected size

Size dependent mesh reduction
Occlusion Culling

- Discard objects hidden behind other objects
- **Cell-based occlusion culling**
  - Divide scene into cells
  - Determine *potentially visible set* (PVS) for each cell
  - Discard all cells not in PVS
- **Two main variants**
  - Precomputation using binary space partitioning (BSP) trees
  - Portal algorithms
- **Specialized algorithms for different types of geometry**
  - Indoor scenes
  - Terrain
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
Bounding Volumes

- Simple shape that completely encloses an object
- Generally a box or sphere
- We use spheres
  - Easiest to work with
  - Though hard to get tight fits
- Intersect bounding volume with view frustum, instead of full geometry
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
Distance to Plane

- The distance is the length of the projection of $\vec{x} - \vec{p}$ onto $\vec{n}$

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

\[ dist(x) = (x - p) \cdot \mathbf{n} \]

\( \mathbf{n} \)

\( p \)

Positive

Negative
Distance to Plane

- **Simplification**

\[
dist(x) = (x - p) \cdot n
\]
\[
= x \cdot n - p \cdot n
\]
\[
dist(x) = x \cdot n - d, \quad d = pn
\]

- \(d\) is independent of \(x\)
- \(d\) is distance from the origin to the plane
- We can represent a plane with just \(d\) and \(n\)
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 $x$, test the distance to the origin, and see if it is beyond the radius

- Three cases:
  - $dist(x) > r$
    - completely above
  - $dist(x) < -r$
    - completely below
  - $-r < dist(x) < r$
    - intersects
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
Next Lecture

- Thursday, October 27: Midterm Exam