Today
Scene graphs & hierarchies
- Introduction
- Scene graph data structures
- Rendering scene graphs

So far: rendering pipeline

System architecture

Low-level graphics API
- Interface to graphics hardware

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

Low-level graphics API
- Interface to graphics hardware
### System architecture

<table>
<thead>
<tr>
<th>Interactive applications</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Thousands</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Rendering engine, scene graph API</th>
</tr>
</thead>
<tbody>
<tr>
<td>• No broadly accepted standards</td>
</tr>
<tr>
<td>• Java3D, Ogre3D, OpenSceneGraph, RE167</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Low-level graphics API</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Highly standardized</td>
</tr>
<tr>
<td>• OpenGL, Direct3D</td>
</tr>
</tbody>
</table>

### Scene graph APIs

- APIs focus on different clients/applications
- Java3D (https://java3d.dev.java.net/)
  - Simple, easy to use, web-based applications
- OpenSceneGraph (www.openscenegraph.com)
  - Scientific visualization, virtual reality, GIS (geographic information systems)
- Ogre3D (http://www.ogre3d.org/)
  - Games
- RE167
  - Under development...

### Common functionality

- Resource management
  - Content I/O (geometry, textures, materials, animation sequences)
  - Memory management
- High level scene representation
  - Scene graph
- Rendering
  - Efficiency

### Today

- Scene graphs & hierarchies
  - Introduction
  - Scene graph data structures
  - Rendering scene graphs

### Scene graphs

- Data structure for intuitive construction of 3D scenes
- So far, RE167 just stores a linear list of objects
- Ideas for improvement?
**Data structure**

- Requirements
  - Collection of individual models/objects
  - Organized in groups
  - Related via hierarchical transformations
- Use a tree structure
- Nodes have associated local coordinates
- Different types of nodes
  - Geometry
  - Transformations
  - Lights
  - ...

**Class hierarchy**

- Many designs possible
- Concepts are the same, details differ
- Design driven by intended application
- Games
  - Optimize for speed
- Large-scale visualization
  - Optimize for memory requirements
- Modeling system
  - Optimize for editing flexibility
### Class hierarchy

- Inspired by Java3D

- Node
  - Group
  - Leaf

  - TransformGroup
  - Light
  - Shape3D

- Access to local-to-world coordinate transform
- List of children
- Get, add, remove child
- Leaf
  - Node with no children

### Class hierarchy

- TransformGroup
  - Stores additional transformation $M$
  - Transformation applies to subtree below node
  - Keyboard-to-world transform $M_0 M_1 M_2$

### Subclasses of Leaf

- Light
  - Stores light sources

- Shape3D
  - References a geometric object, material

### Scene graph for sample scene

```java
class TransformGroup {
    public TransformGroup() {
        // Constructor code here
    }

    // Additional methods...
}
```

- Building sample scene

```java
WORLD = new Group();
tableTrafo = new TransformGroup(); WORLD.addChild(tableTrafo);
table1 = makeTable(); table1Trafo.addChild(table1);
top1Trafo = new TransformGroup(); table1Trafo.addChild(top1Trafo);
lampTrafo = new TransformGroup(); top1Trafo.addChild(lampTrafo);
lamp = makeLamp(); lampTrafo.addChild(lamp);

... // More code...
```

- More convenient to construct than using linear list of objects
- Easier to manipulate
Modifying the scene

- Change tree structure
  - Add, delete, rearrange nodes
- Change node parameters
  - Transformation matrices
  - Shape of geometry data
  - Materials
- Define specific subclasses
  - Animation, triggered by timer events...

Modifying the scene

- Change a transform in the tree
  table1Trafo.setRotationZ(23);
- Table rotates, everything on the table moves with it
- Allows easy animation
  - Build scene once at start of program
  - Update parameters to draw each frame
- Allows interactive model manipulation tools
  - Add objects relative to parent objects
  - E.g., book on table

Articulated character

- Separate rigid parts
- Joint angles define transformation matrices
- Hierarchy
  - Rooted at pelvis
  - Neck, head subtree
  - Arms subtree
  - Legs subtree

Parameteric models

- Parameters for
  - Relationship between parts (e.g., joint angles)
  - Shape of individual parts (e.g., length of limbs)
- Hierarchical relationship between parts
- Degrees of freedom (DOFs)
  - Total number of float parameters in the model

Questions?

More node types

- Shape nodes
  - Cube, sphere, curved surface, etc...
- Nodes that control structure
  - Switch/Select: parameters choose whether or which children to enable, etc...
- Nodes that define other properties
  - Camera
- Again, different details for different designs
Java3D scene graph

OpenInventor scene graph

Screen graph, not tree

- A scene may have many copies of a model
- A model might use several copies of a part
- **Multiple Instantiation**
  - One copy of node or subtree in memory
  - Reference (pointer) inserted as child of many parents
- Not the same as instantiation in C++ terminology
- A directed acyclic graph (DAG), not a tree
- Object appears in scene multiple times, with different coordinates

Instantiation

Scene graph, not tree

- Saves memory
- May save time, depending on caching/optimization
- Change parameter once, affects all instances
  - Can be good or bad, depending on what you want
  - Some scene graph designs let other properties inherit from parent

Fancier operations

Given articulated character, i.e., skeleton, compute skin

- Shape nodes that compute surface across multiple joint nodes
- Nodes that change shape of geometry
- Extremely popular in games
- More details in CSE169
Questions?  

Today  
Scene graphs & hierarchies  
• Introduction  
• Scene graph data structure  
• Rendering scene graphs

Basic rendering  
• Traverse the tree recursively

    TransformGroup::draw(Matrix4 C) {  
        C_new = C*M;  // M is a class member  
        for all children  
            draw(C_new);  
    }  

    Shape3D::draw(Matrix4 C) {  
        setModelView(C);  
        setMaterial(myMaterial);  
        render(myObject);  
    }

Performance optimization  
• Culling  
  • Quickly discard invisible parts of the scene  
• Level-of-detail techniques  
  • Use lower quality for distant (small) objects  
• Scene graph compilation  
  • Efficient use of low-level API  
  • Avoid state changes in rendering pipeline  
  • Render objects with similar properties (geometry, shaders, materials) in batches

    Initiate rendering with  
        world->draw(IDENTITY);  

    Focus on culling today

    Focus on culling today
Level-of-detail techniques

• Don’t draw objects smaller than a threshold
  - Popping artifacts
• Replace objects by impostors
  - Textured planes representing the objects

Culling

• View frustum culling
  - Discard objects outside view frustum
• Occlusion culling
  - Discard objects that are within view frustum, but hidden behind other objects
• Essential for interactive performance with large scenes

Occlusion culling

• 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 $\hat{n}$
- Find the (perpendicular) distance from point $x$ to the plane

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

Distance to plane
- The distance has a sign
  - positive on the side of the plane the normal points to
  - negative on the opposite side
  - 0 exactly on the plane
- Divides all of space into two infinite half-spaces

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

Distance to plane
- Simplification
  \[ \text{dist}(x) = (x - p) \cdot n = x \cdot n - p \cdot n \]
  \[ \text{dist}(x) = x \cdot n - d, \quad d = p \cdot n \]
- $d$ is independent of $p$
- $d$ is distance from the origin to the plane
- We can represent a plane with just $d$ and $\hat{n}$

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’s beyond the radius
- Three cases
  - $\text{dist}(x) > r$
    - completely above
  - $\text{dist}(x) < -r$
    - completely below
  - $-r < \text{dist}(x) < r$
    - intersects
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 \( \text{dist}(\mathbf{x}) > r \): sphere is outside! (no need to continue loop)
  - add 1 to count if \( \text{dist}(\mathbf{x}) < -r \)
- 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)

Questions?

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’s entirely out, don’t draw the group at all
  - If it’s entirely in, 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 time

- Curves