CSE167: Introduction to Computer Graphics

Lecture #11

Jürgen Schulze
University of California, San Diego
Fall 2008
Review

• Mipmapping
Sampling Texture Maps

- How to map the texture area seen through the pixel window to a single pixel value?
Sampling Texture Maps

- When texture mapping it is rare that the screen-space sampling density matches the sampling density of the texture.

64x64 pixels

Original Texture

Magnification for Display

Minification for Display

for which we must use a reconstruction filter
Linear Interpolation

- Tell OpenGL to use a tent filter instead of a box filter.
- Magnification looks better, but blurry
  - (texture is under-sampled for this resolution)
Spatial Filtering

• Remove the high frequencies which cause artifacts in texture minification.

• Compute a spatial integration over the extent of the pixel

• This is equivalent to convolving the texture with a filter kernel centered at the sample (i.e., pixel center)!

• Expensive to do during rasterization, but an approximation it can be precomputed
MIP Mapping

- Construct a pyramid of images that are pre-filtered and re-sampled at 1/2, 1/4, 1/8, etc., of the original image's sampling rate.

- During rasterization we compute the index of the decimated image that is sampled at a rate closest to the density of our desired sampling rate.

- MIP stands for *multum in parvo* which means *many in a small place*.
MIP Mapping Example

• Thin lines may become disconnected / disappear
Examples of Aliasing

Texture Errors

- Point sampling
- Mipmaps & linear interpolation
Storing MIP Maps

- Can be stored compactly
- Illustrates the 1/3 overhead of maintaining the MIP map

10-level mip map

Memory format of a mip map
**Anisotropic MIP-Mapping**

- What happens when the surface is tilted?

![Nearest Neighbor vs MIP Mapped (Bi-Linear) Comparison](image)
Anisotropic MIP-Mapping

- Square MIP-map area is a bad approximation
Anisotropic MIP-Mapping

- We can use different mipmaps for the 2 directions.
- Additional extensions can handle non-axis-aligned views.

Today

Scene graphs & hierarchies

• Introduction

• Scene graph data structures
So far: rendering pipeline

Scene data

- Modeling and viewing transformation
  - Shading
    - Projection
      - Rasterization, visibility

Image
System architecture

Low-level graphics API

- Interface to graphics hardware
System architecture

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

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

Interactive applications
- Thousands

Rendering engine, scene graph API
- No broadly accepted standards
- Java3D, Ogre3D, OpenSceneGraph, RE167

Low-level graphics API
- Highly standardized
- OpenGL, Direct3D
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, high-performance rendering
- 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
Typical functionality (example: Ogre)
Today

Scene graphs & hierarchies

• Introduction

• Scene graph data structures
Scene graphs

- Data structure for intuitive construction of 3D scenes
- So far, RE167 just stores a linear list of objects
- Ideas for improvement?
Top view
Top view with coordinates
Hierarchical organization
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
    - optimized for speed
  - Large-scale visualization
    - Optimized for memory requirements
  - Modeling system
    - Optimized for editing flexibility
Class hierarchy

- Inspired by Java3D

```
Node
  ├── Group
  │   └── TransformGroup
  └── Leaf
      ├── Light
      └── Shape3D
```
Class hierarchy

Node

- Access to local-to-world coordinate transform

Group

- 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
- *Monitor-to-world transform* \( M_0 M_1 M_2 \)
Class hierarchy

Subclasses of Leaf

Light
- Stores light sources

Shape3D
- References a geometric object, material
Scene graph for sample scene

TransformGroup

Shape3D
Building sample scene

WORLD = new Group();
table1Trafo = new TransformGroup(...); WORLD.addChild(table1Trafo);
table1 = makeTable(); table1Trafo.addChild(table1);
top1Trafo = new TransformGroup(...); table1Trafo.addChild(top1Trafo);

lampTrafo = new TransformGroup(...); top1Trafo.addChild(lampTrafo);
lamp = makeLamp(); lampTrafo.addChild(lamp);

book1Trafo = new TransformGroup(...); top1Trafo.addChild(book1Trafo);
book1 = makeBook(); book1Trafo.addChild(book1);

...  

- More convenient to construct scenes 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
  \[
  \text{table1Trafo.setRotationZ}(23);
  \]
- 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**
Scene 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

TransformGroup
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
More complex 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?
Next time

- More on scene graphs
- Curves