Scene Graphs: Rendering
Lab 3b: Shader

William H. Hsu
Department of Computing and Information Sciences, KSU

Public mirror web site: http://www.kddresearch.org/Courses/CIS636
Instructor home page: http://www.cis.ksu.edu/~bhsu

Readings:
Next class: §5.3 – 5.5, Eberly 2e, CGA handout
Lecture Outline

- Reading for Last Class: §5.1 – 5.2, Eberly 2e
- Reading for Today: §4.4 – 4.7, Eberly 2e
- Reading for Next Class: §5.3 – 5.5, Eberly 2e, CGA handout
- Last Time: Introduction to Animation
  - Definition, overview, brief history
  - Principles of traditional animation
  - Keyframe animation, inbetweening (interpolation)
  - Articulated figures (preliminaries of character modeling)
  - Dynamics vs. kinematics, forward vs. inverse
- Today: Scene Graph Rendering
  - State: transforms, bounding volumes, render state, animation state
  - Managing renderer and animation state
  - Rendering: object-oriented message passing overview
- Next Class: Special Effects (SFX), Skinning, Morphing
- Coming Up: More Videos (Lectures 19 & 20)
### Where We Are

<table>
<thead>
<tr>
<th>Lecture</th>
<th>Topic</th>
<th>Primary Source(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Course Overview</td>
<td>Chapter 1, Eberty 2nd</td>
</tr>
<tr>
<td>1</td>
<td>CG Basics: Transformation Matrices; Lab 0</td>
<td>Sections (s) 2.1, 2.2</td>
</tr>
<tr>
<td>2</td>
<td>Viewing 1: Overview, Projections</td>
<td>§ 2.3 – 2.4, 2.8</td>
</tr>
<tr>
<td>3</td>
<td>Viewing 2: Viewing Transformation</td>
<td>§ 2.3 esp. 2.3.4; VFXH slides</td>
</tr>
<tr>
<td>4</td>
<td>Lab 1a: Flash &amp; OpenGL Basics</td>
<td>Ch. 2, 16, <em>Angel Primer</em></td>
</tr>
<tr>
<td>5</td>
<td>Viewing 3: Graphics Pipeline</td>
<td>§ 2.3 esp. 2.3.7, 2.6, 2.7</td>
</tr>
<tr>
<td>6</td>
<td>Scan Conversion 1: Lines, Midpoint Algorithm</td>
<td>§ 2.3.1, 3.1; VFXH slides</td>
</tr>
<tr>
<td>7</td>
<td>Viewing 4: Clipping &amp; Culling; Lab 1b</td>
<td>§ 2.3.5, 2.4, 3.1.3</td>
</tr>
<tr>
<td>8</td>
<td>Scan Conversion 2: Polygons, Clipping Intro</td>
<td>§ 2.4, 2.6 esp. 2.5.4, 3.1.6</td>
</tr>
<tr>
<td>9</td>
<td>Surface Detail 1: Illumination &amp; Shading</td>
<td>§ 2.5, 2.6.1 – 2.6.2, 4.3.2, 20.2</td>
</tr>
<tr>
<td>10</td>
<td>Lab 2a: DirectX Intro</td>
<td>§ 2.7, <em>Direct3D handout</em></td>
</tr>
<tr>
<td>11</td>
<td>Surface Detail 2: Textures; OpenGL Shading</td>
<td>§ 2.6.3, 20.3 – 20.4, <em>Primer</em></td>
</tr>
<tr>
<td>12</td>
<td>Surface Detail 3: Mappings; OpenGL Textures</td>
<td>§ 20.5 – 20.13</td>
</tr>
<tr>
<td>13</td>
<td>Surface Detail 4: Pixel/Vertex Shader; Lab 2b</td>
<td>§ 3.1</td>
</tr>
<tr>
<td>14</td>
<td>Surface Detail 5: Direct3D Shading; OGLSL</td>
<td>§ 3.2 – 3.4, <em>Direct3D handout</em></td>
</tr>
<tr>
<td>15</td>
<td>Demos 1: CGA, Fun; Scene Graphs: State</td>
<td>§ 4.1 – 4.3, <em>CGA handout</em></td>
</tr>
<tr>
<td>16</td>
<td>Lab 3a: Shading &amp; Transparency</td>
<td>§§ 2.6, 20.1, <em>Primer</em></td>
</tr>
<tr>
<td>17</td>
<td>Animation 1: Basics, Keyframes; HW/Exam</td>
<td>§ 5.1 – 5.2</td>
</tr>
<tr>
<td>18</td>
<td>Exam 1 review; Hour Exam 1 (evening)</td>
<td>Chapters 1 - 4, 20</td>
</tr>
<tr>
<td>19</td>
<td>Demos 2: SFX, Skinning, Morphing</td>
<td>§§ 4.4 – 4.7</td>
</tr>
<tr>
<td>20</td>
<td>Demos 3: Surfaces, B-reps/Volume Graphics</td>
<td>§§ 5.3 – 5.5, <em>CGA handout</em></td>
</tr>
</tbody>
</table>

Lightly-shaded entries denote the due date of a written problem set, heavily-shaded entries, that of a machine problem (programming assignment), blue-shaded entries, that of a paper review, and the green-shaded entry, that of the term project.

Green, blue and red letters denote exam review, exam, and exam solution review dates.
Acknowledgements:
Computer Animation Intro

Jason Lawrence
Assistant Professor
Department of Computer Science
University of Virginia
http://www.cs.virginia.edu/~jdl/

Acknowledgment: slides by Misha Kazhdan, Allison Klein, Tom Funkhouser, Adam Finkelstein and David Dobkin

Thomas A. Funkhouser
Professor
Department of Computer Science
Computer Graphics Group
Princeton University
http://www.cs.princeton.edu/~funk/
Review [1]: 19th Century Animation
Before Motion Pictures

© 2007 Wikipedia, Phenakistoscope

Zoetrope (Praxinoscope)

© 2008 Wikipedia, Thaumatrope

Tarzan © 2000 Disney
http://youtu.be/zc3MnoSS5Hw

Adapted from slides © 2010 J. Lawrence, University of Virginia
Review [2]: Animation, Simulation & Visualization

- What is animation?
  - Make objects change over time according to scripted actions

- What is simulation?
  - Predict how objects change over time according to physical laws

Adapted from slides © 2010 J. Lawrence, University of Virginia
Review [3]:
Principles of Traditional Animation

- Squash and Stretch
- Timing
- Anticipation
- Staging
- Follow Through and Overlapping Action
- Straight Ahead Action and Pose-to-Pose Action
- Slow In and Out
- Arcs
- Exaggeration
- Secondary action
- Appeal

© 2010 J. Lawrence, University of Virginia

SIGGRAPH: http://bit.ly/1DsO44
Review [4]:
Traditional Animation – Anticipation

- The preparation for an action.
  - Muscle contraction prior to extension
  - Bending over to lift a heavy object
  - Luxo’s dad responds to Luxo Jr. off screen before Luxo Jr. appears.

© 2010 J. Lawrence, University of Virginia

Luxo Jr. © 1986 Pixar
http://www.pixar.com/shorts/ljr/
http://youtu.be/qGxoui3IFS0
Review [5]:
Keyframe Animation & Inbetweening

- Interpolate variables describing keyframes to
determine poses for character “in-between”

Lasseter ‘87

© 2010 J. Lawrence, University of Virginia
Review [6]:
Linear Interpolation aka Lerping

- Inbetweening:
  - Linear interpolation - usually not enough continuity

H&B Figure 16.16

© 2010 J. Lawrence, University of Virginia
Review [7]: Articulated Figures

- Character poses described by set of rigid bodies connected by “joints”

Scene Graph

© 2010 J. Lawrence, University of Virginia
Review [8]: Character Modeling

- Well-suited for humanoid characters

Rose et al. ’96

© 2010 J. Lawrence, University of Virginia
Review [9]: Bones & Joints

- Articulated figure:

  - Hip
  - Upper leg
    - Knee
    - Lower leg
      - Ankle
      - Foot
      - Upper leg (hip rot)
        - Hip rotate
        - Lower leg (knee rot)
          - Hip rotate + knee rot
          - Foot (ankle rot)
Scene Graph Traversal

a 3d scene...

...and its scene graph

Object:
- Torus A
  - Mesh:
    - Torus
  - Material:
    - Brick
  - Shader:
    - Water Fx

Object:
- Torus B
  - Mesh:
    - Torus
  - Material:
    - Blue Water

Object:
- Cube
  - Mesh:
    - Chamfer Box
  - Material:
    - Yellow Water

© 2002 – 2005 Virtools
http://bit.ly/eM1gz8
Scene Graph Rendering

Performer © 1997 D. Pape
http://www.evl.uic.edu/pape/talks/VS97/pf/
Acknowledgements:

Scene Graphs – Eberly 1e

David H. Eberly
Chief Technology Officer
Geometric Tools, LLC
http://www.geometrictools.com

3D Game Engine Design © 2000 D. H. Eberly
Review:
What Information is in Scene Graphs?

- Transforms
- Bounding Volumes
- Render State
- Animation State

Figure 4.1 A simple tree with one grouping node.
Review: Kinds of Transforms

- **Local**
  - Translation, rotation, scaling, shearing
  - All within parent’s coordinate system

  \[
  \begin{pmatrix}
  M' & P' \\
  0 & 1
  \end{pmatrix} = \begin{pmatrix}
  M & P \\
  0 & 1
  \end{pmatrix}
  \]  

  Using this compressed notation, the product of two homogenous matrices is

  \[
  \begin{pmatrix}
  M_1 & \mathbf{P}_1 \\
  0 & 1
  \end{pmatrix} \begin{pmatrix}
  M_2 & \mathbf{P}_2 \\
  0 & 1
  \end{pmatrix} = \begin{pmatrix}
  M_1 M_2 & M_1 \mathbf{P}_2 + \mathbf{P}_1 \\
  0 & 1
  \end{pmatrix}
  \]  

  and the product of a homogenous matrix with a homogenous vector \([\mathbf{v}]^T\) is

  \[
  \begin{pmatrix}
  M & \mathbf{v} \\
  0 & 1
  \end{pmatrix} \begin{pmatrix}
  \mathbf{v} \\
  1
  \end{pmatrix} = \begin{pmatrix}
  M \mathbf{v} + \mathbf{v} \\
  1
  \end{pmatrix}.
  \]  

- **World: Position Child C With Respect to Parent P (Depends on Local)**

  \[
  \begin{pmatrix}
  M_{\text{world}}^C & \mathbf{P}_{\text{world}}^C \\
  0 & 1
  \end{pmatrix} = \begin{pmatrix}
  M_{\text{local}}^C & \mathbf{P}_{\text{local}}^C \\
  0 & 1
  \end{pmatrix} \begin{pmatrix}
  M_{\text{world}}^C & \mathbf{P}_{\text{world}}^C \\
  0 & 1
  \end{pmatrix}
  \]

  \[
  = \begin{pmatrix}
  M_{\text{world}}^C M_{\text{local}}^C & M_{\text{world}}^C \mathbf{P}_{\text{local}}^C + \mathbf{P}_{\text{world}}^C \\
  0 & 1
  \end{pmatrix}.
  \]

- **Both Together Part of Modelview Transformation**

Adapted from *3D Game Engine Design* © 2000 D. H. Eberly

Traversing Scene Graph: World Transform of Node

The world transform of the root node in the scene graph is just its local transform. The world position of a node $N_k$ in a path $N_0 \cdots N_k$, where $N_0$ is the root node, is generated recursively by the above definition as

$$\begin{bmatrix} M_{\text{world}}^{(N_k)} \\ T_{\text{world}}^{(N_k)} \end{bmatrix} = \begin{bmatrix} M_{\text{local}}^{(N_0)} \\ T_{\text{local}}^{(N_0)} \end{bmatrix} \cdots \begin{bmatrix} M_{\text{local}}^{(N_k)} \\ T_{\text{local}}^{(N_k)} \end{bmatrix}.$$
Bounding Volume Hierarchies (BVHs)

- Root: entire scene
- Interior node: rectangle (volume in general) enclosing other nodes
- Leaves: primitive objects
- Often axis-aligned (e.g., axis-aligned bounding box aka AABB)

Used

- Visible surface determination (VSD) – especially occlusion culling
- Other intersection testing: collisions, ray tracing
Bounding Volumes [2]: Types Covered in Eberly

- Spheres
- Oriented Boxes *aka* Oriented Bounding Boxes (OBBs)
- Capsules
- Lozenges
- Cylinders
- Ellipsoids

Adapted from 3D Game Engine Design © 2000 D. H. Eberly
Renderer State

- Can Capture Render Information Hierarchically
- Example
  - Suppose subtree has all leaf nodes that want textures alpha blended
  - Can tag root of subtree with “alpha blend all”
  - Alternatively: tag every leaf
- How Traversal Works: State Accumulation
  - Root-to-leaf traversal accumulates state to draw geometry
  - Renderer checks whether state change is needed before leaf drawn
- Efficiency Considerations
  - Minimize state changes
  - Reason: memory copy (e.g., system to video memory) takes time

Adapted from 3D Game Engine Design © 2000 D. H. Eberly
Animation State

- Can Capture Animation Information Hierarchically
- Example
  - Consider articulated figure from last lecture
  - Let each node represent joint of character model
    - Neck
    - Shoulder
    - Elbow
    - Wrist
    - Knee
- Procedural Transformation
- How It Works: Controllers
  - Each node has controller function/method
  - Manages quantity that changes over time (e.g., angle)

Adapted from 3D Game Engine Design © 2000 D. H. Eberly

© 2002 D. M. Murillo
Updating Scene Graphs

- Need to Merge Bounding Volumes (Boxes, Lozenges, Capsules)
- Update Geometric State: UpdateGS

```cpp
void Spatial::UpdateGS(float time, bool initiator)
{
    UpdateWorldData(time);
    UpdateWorldBound();
    if (initiator)
        PropagateBoundToRoot();
}
```

- UpdateWorldData: Virtual Function, Controls Downward Pass
- UpdateWorldBound: Also Virtual, Controls Upward Pass
- PropagateBoundToRoot: Not Virtual, Simple Recursive Call
  - parent.UpdateWorldBound()
  - parent.PropagateBoundToRoot()
### Rendering Scene Graphs [1]: View Frustum Culling

- **By Spheres vs. By Oriented Boxes**

  ![Examples of culled and unculled objects.](image1)

  ![Examples of culled and unculled objects.](image2)

- **Pseudocode**

  ```c
  bool CullSpherePlane (Sphere sphere, Plane plane)
  {
      return Dot(plane.N, sphere.C) - plane.d < -sphere.r;
  }

  bool CullBoxPlane (Box box, Plane plane)
  {
      r = box.0[0]*Dot(plane.N, box.A0) +
          box.1[0]*Dot(plane.N, box.A1) +
          box.2[0]*Dot(plane.N, box.A2);
      return Dot(plane.N, box.C) - plane.d < -r;
  }
  ```

- **Can Also Cull by: Lozenges, Cylinders, Ellipsoids**

Adapted from 3D Game Engine Design © 2000 D. H. Eberly
Rendering Scene Graphs [2]: Message Passing

- **Main Draw Method**
  
  ```
  void Renderer::Draw (Spatial scene)
  {
    scene.OnDraw(thisRenderer);
  }
  ```
  
  - Spatial::OnDraw(Renderer renderer)
  - **Calls virtual function** Draw(renderer)
  - **Passed down**
    - Geometry::Draw(Renderer renderer)
    - Node::Draw(Renderer renderer) **Calls** child.onDraw(renderer)

  **Derived Classes of Geometry**
  - TriMesh::Draw(Renderer renderer)

  **Similarly for other derived classes**

Adapted from *3D Game Engine Design* © 2000 D. H. Eberly
Summary

- Reading for Last Class: §5.1 – 5.2, Eberly 2e
- Reading for Today: §4.4 – 4.7, Eberly 2e
- Reading for Next Class:
- Last Time: Introduction to Animation
  - Definition, overview, brief history, principles
  - Keyframes, interpolation, articulated figures for character modeling
  - Dynamics vs. kinematics, forward vs. inverse
- Today: Scene Graph Rendering
  - State: transforms, bounding volumes, render state, animation state
  - Updating: merging bounding volumes
  - View frustum culling
  - Rendering: object-oriented message passing overview
- Next Class: Special Effects (SFX), Skinning, Morphing; More Videos
Terminology

- Shading and Transparency in OpenGL: Alpha, Painter’s, z-buffering
- **Animation** – Modeling Change Over Time According to Known Actions
- Keyframe Animation
  - **Keyframe**
  - **Interpolation**
  - **Character model**
- **State** in Scene Graphs
  - **Transforms** – local & global TRS to orient parts of model
  - **Bounding volumes** – spheres, boxes, capsules, lozenges, ellipsoids
  - **Renderer state** – lighting, shading/textures/alpha
  - **Animation state** – TRS transformations (especially R), controllers
- **Traversal**: Moving through Data Structure, Calling Methods