Lecture 15 of 41

Scene Graphs: State
Videos 1: CGA Shorts, Demos

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:
Today: §4.1 – 4.3, Eberly 2e; Computer-Generated Animation handout
List of videos (trailers, shorts, etc.): http://bit.ly/l2e2qq
List of software demos: http://bit.ly/gDWqUb
A Long Ray’s Journey into Light: http://youtu.be/b_UgzLBFz4Y
Wikipedia, Scene Graph: http://en.wikipedia.org/wiki/Scene_graph
Lecture Outline

- Reading for Last Class: §3.2 – 3.4, Eberly 2e; Direct3D handout
- Reading for Today: §4.1 – 4.3, Eberly 2e; CGA handout
- Reading for Next Class: §2.6, 20.1, Eberly 2e; OpenGL primer material
- Last Time: Shader Languages – OGLSL & Direct3D
  - OpenGL Shading Language (OGLSL or GLSL) – main topic
  - High-Level Shading Language (HLSL) & Direct3D
  - Tutorials from K. Ditchburn based on Direct3D 10, HLSL
  - More on pixel shading on Toymaker site: http://bit.ly/gBScYK
  - Pixar’s RenderMan – preview
- Today: Scene Graphs; Computer-Generated Animation Demos, Videos
  - Scene graphs and state – main topic
  - State of CGA: videos and discussion
  - Demos to download
    - Adobe Maya: http://students.autodesk.com
# 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. Eberly 2nd</td>
</tr>
<tr>
<td>1</td>
<td>CG Basics: Transformation Matrices; Lab 0</td>
<td>Sections (§) 2.1, 2.2</td>
</tr>
<tr>
<td>2</td>
<td>Viewing 1: Overview, Projections</td>
<td>§ 2.2.3 – 2.2.4, 2.8</td>
</tr>
<tr>
<td>3</td>
<td>Viewing 2: Viewing Transformation</td>
<td>§ 2.3 esp. 2.3.4; PVFH 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>§ 2.5.1, 3.1, PVFH 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>§ 2.4.2.5 esp. 2.5.4, 3.16</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: Direct3D / DirectX Intro</td>
<td>§ 2.7, Direct3D handout</td>
</tr>
<tr>
<td>11</td>
<td>Surface Detail 2: Textures; OpenGL Shading</td>
<td>§ 2.6.3, 20.3 – 20.4, Primer</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 Shad.; Lab 2b</td>
<td>§ 3.1</td>
</tr>
<tr>
<td>14</td>
<td>Surface Detail 5: Direct3D Shading; CAA SL</td>
<td>§ 3.2 – 3.4, Direct3D handout</td>
</tr>
<tr>
<td>15</td>
<td>Demos 1: CAA, Fun; Scene Graphs: State</td>
<td>§ 4.1 – 4.3, CAA handout</td>
</tr>
<tr>
<td>16</td>
<td>Lab 3a: Shading &amp; Transparency</td>
<td>§ 2.6, 20.1, Primer</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>Scene Graphs: Rendering; Lab 3b: Shader</td>
<td>§ 4.4 – 4.7</td>
</tr>
<tr>
<td>20</td>
<td>Demos 2: SFX; Skinning, Morphing</td>
<td>§ 5.3 – 5.5, CAA handout</td>
</tr>
<tr>
<td>21</td>
<td>Demos 3: Surfaces, B-reps/Volume Graphics</td>
<td>§ 10.4, 12.7, Mesh handout</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.
Review [1]:
Simple OGLSL Vertex & Pixel Shaders

Vertex Shader

```c
void main(void)
{
    vec4 a = gl_Vertex;
    a.x = a.x * 0.5;
    a.y = a.y * 0.5;
    gl_Position = gl_ModelViewProjectionMatrix * a;
}
```

Q: What does this do?
A: Incoming x and y components are scaled with a factor 0.5
Scaled vertex is transformed with concatenated modelview and projection matrix.

Fragment Shader

```c
void main (void)
{
    gl_FragColor = vec4 (0.0, 1.0, 0.0, 1.0);
}
```

Q: What does this do?
A: Makes everything green!
Review [2]:
OGLSL Loading, Compiling, Linking

Loading Programs without `aShaderManager` class

Step 1: Declare shader programs and shader objects

Step 2: Load, add and compile/link programs in `AppInit()`

Reserve memory and initialize objects
- `myShader = new aShaderObject;`
- `myVertexShader = new aVertexShader;`
- `myFragmentShader = new aFragmentShader;`

Load:
- `myVertexShader->load("simple.vert");`
- `myFragmentShader->load("simple.frag");`

Compile:
- `myVertexShader->compile();`
- `myFragmentShader->compile();`

Add (compiled) programs to the object and link it:
- `myShader->addShader(myVertexShader);`
- `myShader->addShader(myFragmentShader);`
- `myShader->link();`
Review [3]:
OGLSL Shader Application

Loading Programs without aShaderManager class (Step 3 of 3)

Step 3: Use shader (see rest of Tutorials, #2 – 10!)

myShader->begin();
... {draw something with GL} ...  
myShader->end();

Adapted from material © 2003 – 2005 M. Christen, ClockworkCoders.com
Defining `TVertex` data structure: position/normal/texture tuple:

```c
struct TVertex
{
    D3DXVECTOR3 position;
    D3DXVECTOR3 Normal;
    D3DXVECTOR3 Tex;
};
```

Vertex declaration to describe this structure:

```c
const D3DVERTEXELEMENT9 dec[4] =
{
    {0, 0, D3DDECLTYPE_FLOAT3, D3DDECLMETHOD_DEFAULT, D3DDECLUSAGE_POSITION, 0},
    {0, 12, D3DDECLTYPE_FLOAT3, D3DDECLMETHOD_DEFAULT, D3DDECLUSAGE_NORMAL, 0},
    {0, 24, D3DDECLTYPE_FLOAT2, D3DDECLMETHOD_DEFAULT, D3DDECLUSAGE_TEXCOORD, 0},
    D3DDECL_END()
};
```

Each line corresponds to one of the elements in `TVertex`. The data in each line is:

- WORD Stream; WORD Offset; BYTE Type; BYTE Method; BYTE Usage; BYTE UsageIndex

We need to tell Direct3D about our vertex declaration using the following call:

```c
IDirect3DVertexDeclaration9 m_vertexDeclaration;
gDevice->CreateVertexDeclaration(dec, &m_vertexDeclaration);
```

To render:

```c
gDevice->SetStreamSource(0, m_vb, 0, sizeof(TVertex));
gDevice->SetVertexDeclaration(m_vertexDeclaration);
```

Adapted from Toymaker © 2004 – 2010 K. Ditchburn, Teesside University
http://bit.ly/g8Q8hC
Scene Graphs: State

- **Scene Graph**: General Data Structure used in CG
  - Used to: compute visibility, set up rendering pipeline
  - **Nodes**
    - Leaves: primitive components
    - Interior: assembly operations, modelview transformations
    - Root(s): scene or major objects
- **Scene Graph Traversal**: Initial Step – Drives Rendering

Images © 2007 A. Bar-Zeev

Aesthetics: Non-Photorealistic Shading, Aliasing

- **Non-Photorealistic Rendering**: Aimed at Achieving Natural Aesthetic
  - *Cartoon shaders*: use sharp gradient (thresholded)
  - *Pencil shaders*: blurring, stippling
- **CGA and Realism**
  - Term from signal processing
  - Two sampled signals indistinguishable from (aliases of) one another
  - Examples: jaggies, Moiré vibration
  - Anti-aliasing: operations to prevent such effects
- **Temporal Aliasing**
  - Similar effect in animation
  - Small artifact can be much more jarring!
  - Example: think of flecks in traditional film reels
Next Time: Lab 3
OpenGL Shading & Transparency

- Set Up Point Light Sources
  - Directional light given by "position" vector
    ```
    GLfloat light_position[] = {1.0, 1.0, -1.0, 0.0};
    glLightfv(GL_LIGHT0, GL_POSITION, light_position);
    ```
  - Point source given by "position" point
    ```
    GLfloat light_position[] = {1.0, 1.0, -1.0, 1.0};
    glLightfv(GL_LIGHT0, GL_POSITION, light_position);
    ```

- Set Up Materials, Turn Lights On
  ```
  GLfloat mat_specular[]= {0.0, 0.0, 0.0, 1.0};
  GLfloat mat_diffuse[]= {0.8, 0.8, 0.4, 1.0};
  GLfloat mat_ambient[]= {0.8, 0.6, 0.4, 1.0};
  GLfloat mat_shininess=20.0;
  
  glMaterialf(GL_FRONT, GL_SPECULAR, mat_specular);
  glMaterialf(GL_FRONT, GL_AMBIENT, mat_ambient);
  glMaterialf(GL_FRONT, GL_DIFFUSE, mat_diffuse);
  glMaterialf(GL_FRONT, GL_SHININESS, mat_shininess);
  ```

- Start Drawing (glBegin ... glEnd)

Frank Pfenning
Professor of Computer Science
School of Computer Science
Carnegie Mellon University
http://www.cs.cmu.edu/~fp/

See also: OpenGL: A Primer, 3e (Angel)

Adapted from slides © 2003 F. Pfenning, Carnegie Mellon University
http://bit.ly/g1J2nj
Preview:
Painter’s Algorithm

© 2004 – 2009 Wikipedia, Painter’s Algorithm
Preview: Z-buffering

A simple three-dimensional scene

Z-buffer representation

Trailers:
Video Games

Crysis 2 © 2011 Electronic Arts
http://youtu.be/j4mOQhWSXYQ

Starcraft II: Wings of Liberty © 2010 Blizzard
http://youtu.be/rgyL08hktkw

Rage & id Tech 5 © 2011 id Software
http://youtu.be/vzZPjYPDBvY

Unreal Engine 3 © 2004-2011 Epic/Valve
http://youtu.be/MGf0oGGQnQ

Inspired by slides © 2002 – 2003 van Dam et al., Brown University
Videos: CG Feature Films & Shorts

Monsters Inc.  © 2001 Disney/Pixar
http://youtu.be/cyOQo0zL4S0

Kung-Fu Panda  © 2008 DreamWorks Animation SKG

Happy Feet  © 2006 Warner Brothers

Toy Story 3  © 2010 Disney/Pixar
http://youtu.be/JcpWXaA2qeg

Shrek Forever After  © 2010 DreamWorks Animation SKG
http://youtu.be/u7__TG7swq0

Luxo Jr.  © 1986 Pixar Animation Studios
http://youtu.be/L_oL_27KqgU

Wall-E  © 2008 Disney/Pixar

Tron: Legacy  © 2010 Walt Disney Pictures
http://youtu.be/plwXwVJ3BY

Happy Feet

Toy Story 3

Shrek Forever After

Luxo Jr.
Summary

- Last Time: Shader Languages – OGLSL & Direct3D
  - **OpenGL Shading Language** (OGLSL or GLSL) – main topic
  - **High-Level Shading Language** (HLSL) & Direct3D
- Today: Scene Graphs; **Computer-Generated Animation** Demos, Videos
  - Scene graphs and state – main topic
  - State of CGA: videos
  - Issues
    - Photorealism and non-photorealistic rendering (NPR)
    - Making most of hardware
    - Role of animators (see CNBC’s Pixar Story, [http://bit.ly/gShkXL](http://bit.ly/gShkXL))
  - Techniques showcased
    - Multipass texturing
    - Alpha compositing/blending
    - Portals and binary space partitioning
  - Demos to download: Maya, LightWave
Terminology

- **Scene Graph**: General Data Structure used in CG
  - Used to: compute visibility, set up rendering pipeline
  - Actual graph: general graph, forest, or rooted tree
- **Scene Graph Traversal**: Initial Step – Drives Rendering
- **Features of Scene Graphs**
  - Spatial partitioning: e.g., using bounding volume hierarchies
  - Leaves: primitive components
  - Interior nodes: assembly operations, modelview transformations
  - Root(s): scene or major objects
- **Non-Photorealistic Rendering**: Aimed at Achieving Natural Aesthetic
  - Cartoon shaders: use sharp gradient (thresholded)
  - Pencil shaders: blurring, stippling
- **CGA and Realism**
  - Aliasing & anti-aliasing
  - Temporal aliasing & temporal anti-aliasing