Lecture 15 of 41

Scene Graphs: State
Videos 1: CGA Shorts, Demos

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

KSOL course pages: http://bit.ly/hGvXlH / http://bit.ly/eVizrE
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/i2e2gg**
List of software demos: **http://bit.ly/gDWqUb**
A Long Ray's Journey into Light: **http://youtu.be/b_UqzLBFz4Y**
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**
    - **NewTek Lightwave: http://www.newtek.com/lightwave/**

Where We Are

Lecture	Topic	Primary Source(s)
0	Course Overview	Chapter 1, Eberly 2e
1	**CG Basics: Transformation Matrices; Lab 0**	**Sections (§) 2.1, 2.2**
2	Viewing 1: Overview, Projections	§ 2.2.3 – 2.2.4, 2.8
3	Viewing 2: Viewing Transformation	§ 2.3 esp. 2.3.4; FVFH slides
4	**Lab 1a: Flash & OpenGL Basics**	**Ch. 2, 16[1], Angel *Primer***
5	Viewing 3: Graphics Pipeline	§ 2.3 esp. 2.3.7; 2.6, 2.7
6	Scan Conversion 1: Lines, Midpoint Algorithm	§ 2.5.1, 3.1; FVFH slides
7	**Viewing 4: Clipping & Culling; Lab 1b**	**§ 2.3.5, 2.4, 3.1.3**
8	Scan Conversion 2: Polygons, Clipping Intro	§ 2.4, 2.5 esp. 2.5.4, 3.1.6
9	Surface Detail 1: Illumination & Shading	§ 2.5, 2.6.1 – 2.6.2, 4.3.2, 20.2
10	**Lab 2a: Direct3D / DirectX Intro**	**§ 2.7, Direct3D handout**
11	Surface Detail 2: Textures; OpenGL Shading	§ 2.6.3, 20.3 – 20.4, *Primer*
12	Surface Detail 3: Mappings; OpenGL Textures	§ 20.5 – 20.13
13	**Surface Detail 4: Pixel/Vertex Shad.; Lab 2b**	**§ 3.1**
14	Surface Detail 5: Direct3D Shading; OGLSL	§ 3.2 – 3.4, Direct3D handout
15	Demos 1: CGA, Fun; Scene Graphs: State	§ 4.1 – 4.3, **CGA handout**
16	**Lab 3a: Shading & Transparency**	**§ 2.6, 20.1, *Primer***
17	**Animation 1: Basics, Keyframes; HW/Exam**	**§ 5.1 – 5.2**
	Exam 1 review; Hour Exam 1 (evening)	**Chapters 1 – 4, 20**
18	**Scene Graphs: Rendering; Lab 3b: Shader**	**§ 4.4 – 4.7**
19	**Demos 2: SFX; Skinning, Morphing**	**§ 5.3 – 5.5, CGA handout**
20	Demos 3: Surfaces; B-reps/Volume Graphics	§ 10.4, 12.7, **Mesh handout**

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

```
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

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

Q: What does this do?

A: Makes everything green!

http://bit.ly/et5qOp

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();
```

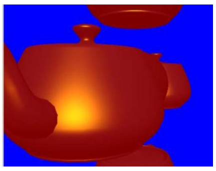

http://bit.ly/et5qOp

Review [3]: OGLSL Shader Application

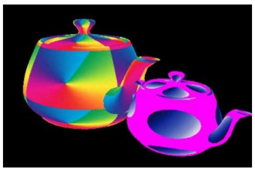
Loading Programs without `aShaderManager` class (Step 3 of 3)

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

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



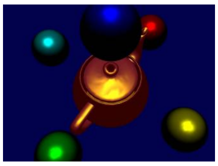
#5 Phong shading in OpenGL
http://bit.ly/i5lFL0



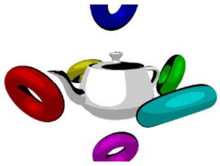
#6 Texture Mapping
http://bit.ly/hRhFQD

#7 Color Keys (Transparency Masks)
http://bit.ly/hUEi62

#8 Multitexturing
http://bit.ly/glaYoY



#9 Procedural Texture (Procedural Materials)
http://bit.ly/ieeER5



#10 Cartoon Shading
http://bit.ly/gR5EH3

Adapted from material © 2003 – 2005 M. Christen, ClockworkCoders.com
http://bit.ly/et5qOp

Review [4]
Vertex Shaders in Direct3D & HLSL

Defining `TVertex` data structure: position/normal/texture tuple:

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

Vertex declaration to describe this structure:

```
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:

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

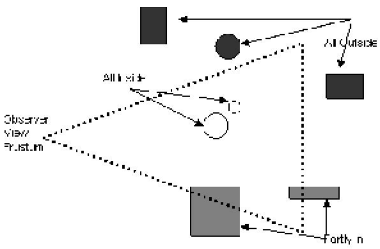
To render:

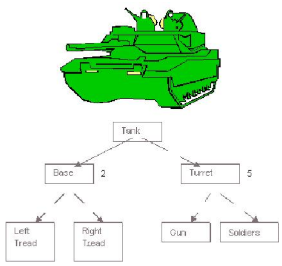
```
gDevice->SetStreamSource( 0, m_vb,0, sizeof(TVertex));
gDevice->SetVertexDeclaration(m_vertexDeclaration);
```

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

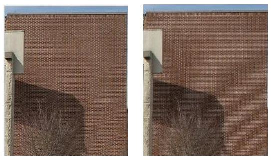




http://bit.ly/gxy9ed

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
- Aliasing (see Wikipedia: http://bit.ly/flkCkr)
  - 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



© 2004 – 2009 Wikipedia, *Aliasing*
http://bit.ly/flkCkr

Next Time: Lab 3
OpenGL Shading & Transparency

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)
http://bit.ly/hVcVWN

- **Set Up Point Light Sources**
  - Directional light given by "position" vector

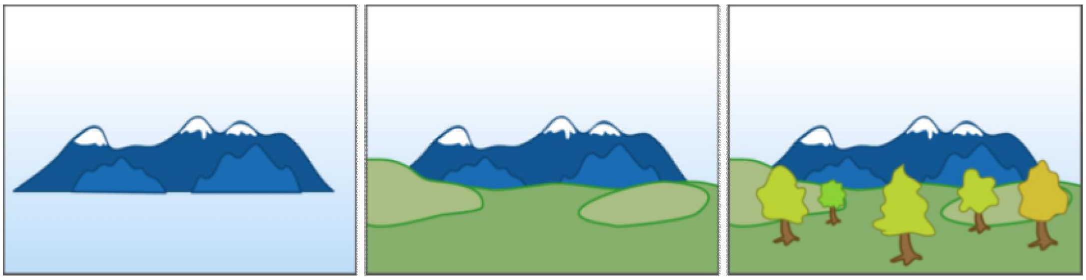
    ```
    GLfloat light_position[] = {-1.0, 1.0, -1.0, 0.0};
    glLightfv(GL_LIGHT0, GL_POSITION, light_position);
    ```
  - Point source given by "position" point

    ```
    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, 0.6, 0.4, 1.0};
  GLfloat mat_ambient[]={0.8, 0.6, 0.4, 1.0};
  GLfloat mat_shininess={20.0};
  glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
  glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient);
  glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
  glMaterialf(GL_FRONT, GL_SHININESS, mat_shininess);

  glShadeModel(GL_SMOOTH); /*enable smooth shading */
  glEnable(GL_LIGHTING); /* enable lighting */
  glEnable(GL_LIGHT0);  /* enable light 0 */
  ```
- **Start Drawing (`glBegin` … `glEnd`)**

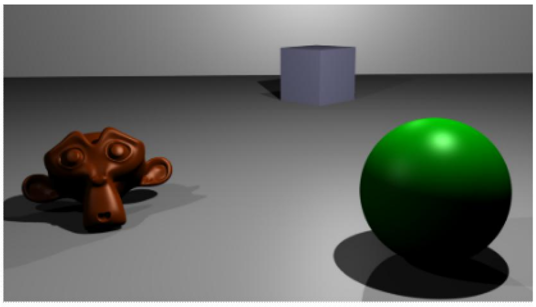
Preview: Painter's Algorithm



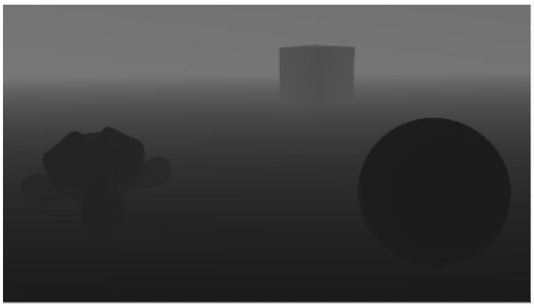
© 2004 – 2009 Wikipedia, *Painter's Algorithm*
http://bit.ly/eeebCN

Preview: Z-buffering

© 2009 Wikipedia, *Z-buffering*
http://bit.ly/gGRFMA



A simple three-dimensional scene



Z-buffer representation

Trailers: Video Games

Crysis 2 © 2011 Electronic Arts
http://youtu.be/j4mOGhWSXYQ

Starcraft II: Wings of Liberty © 2010 Blizzard
http://youtu.be/rgyL08nhtkw

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

Unreal Engine 3 © 2004-2011 Epic/Valve
http://youtu.be/MGf0oGGGQqQ

Inspired by slides © 2002 – 2003 van Dam *et al.*, Brown University
http://bit.ly/fiYmje Reused with permission.

Videos: CG Feature Films & Shorts

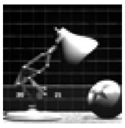
Monsters Inc.
© 2001 Disney/Pixar
http://youtu.be/cvOQeozL4S0

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

Tron: Legacy
© 2010 Walt Disney Pictures
http://youtu.be/plwXwVJZ3BY

Kung-Fu Panda
© 2008 DreamWorks Animation SKG
http://bit.ly/h8krLv

Happy Feet
© 2006 Warner Brothers
http://bit.ly/gTnp2V



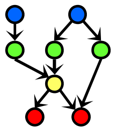
Luxo Jr.
© 1986 Pixar Animation Studios
http://youtu.be/L_oL_27KqgU

Shrek Forever After
© 2010 DreamWorks Animation SKG
http://youtu.be/u7__TG7swg0

Wall-E
© 2008 Disney/Pixar
http://bit.ly/eKDwkk

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)
  - 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