Computer Games

CSCI-GA.3033-010
Spring 2012

Hubertus Franke

( frankeh@cims.nyu.edu )
Thanks

Many thanks to:

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

For allowing the reuse of his excellent course material that he created for the the Eberle book:

Public mirror web site: http://www.kddresearch.org/Courses/CIS636
Acknowledgements

**Jim Foley**  
Professor, College of Computing &  
Stephen Fleming Chair in  
Telecommunications  
Georgia Institute of Technology

**Andy van Dam**  
T. J. Watson University Professor of  
Technology and Education &  
Professor of Computer Science  
Brown University

**Steve Feiner**  
Professor of Computer Science &  
Director, Computer Graphics and User  
Interfaces Laboratory  
Columbia University

**John F. Hughes**  
Associate Professor of Computer Science  
Brown University

---

James D. Foley  
Georgia Tech  

Andries van Dam  
Brown University  

Steven K. Feiner  
Columbia University  

John F. Hughes  
Brown University  
OpenGL
Fixed Function Pipeline

- OpenGL FFP Diagram (for v1.5)

OpenGL 1.5 Fixed Function Pipeline (see OpenGL Reference Manual)
Lecture 4

Lights and Textures
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
Review [4]:
OpenGL Shading (Overview)

- **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)**
  ```
  glBegin(...);
  glEnd();
  ```

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

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

Adapted from slides © 2003 F. Pfenning, Carnegie Mellon University
15 Transparency, Translucency, and Blending

15.010 What is the difference between transparent, translucent, and blended primitives?

A transparent physical material shows objects behind it as unobscured and doesn't reflect light off its surface. Clear glass is a nearly transparent material. Although glass allows most light to pass through unobscured, in reality it also reflects some light. A perfectly transparent material is completely invisible.

A translucent physical material shows objects behind it, but those objects are obscured by the translucent material. In addition, a translucent material reflects some of the light that hits it, making the material visible. Physical examples of translucent materials include sheer cloth, thin plastic, and smoked glass.

Transparent and translucent are often used synonymously. Materials that are neither transparent nor translucent are opaque.

Blending is OpenGL's mechanism for combining color already in the framebuffer with the color of the incoming primitive. The result of this combination is then stored back in the framebuffer. Blending is frequently used to simulate translucent physical materials. One example is rendering the smoked glass windshield of a car. The driver and interior are still visible, but they are obscured by the dark color of the smoked glass.
15.020 How can I achieve a transparent effect?

OpenGL doesn't support a direct interface for rendering translucent (partially opaque) primitives. However, you can create a transparency effect with the blend feature and carefully ordering your primitive data. You might also consider using screen door transparency.

An OpenGL application typically enables blending as follows:

```c
glEnable (GL_BLEND);
glBlendFunc (GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
```

After blending is enabled, as shown above, the incoming primitive color is blended with the color already stored in the framebuffer. `glBlendFunc()` controls how this blending occurs. The typical use described above modifies the incoming color by its associated alpha value and modifies the destination color by one minus the incoming alpha value. The sum of these two colors is then written back into the framebuffer.

The primitive's opacity is specified using `glColor4*()`. RGB specifies the color, and the alpha parameter specifies the opacity.

When using depth buffering in an application, you need to be careful about the order in which you render primitives. Fully opaque primitives need to be rendered first, followed by partially opaque primitives in back-to-front order. If you don't render primitives in this order, the primitives, which would otherwise be visible through a partially opaque primitive, might lose the depth test entirely.
Transparency in OpenGL [3]: Screen Door

15.030 How can I create screen door transparency?

This is accomplished by specifying a polygon stipple pattern with `glPolygonStipple()` and by rendering the transparent primitive with polygon stippling enabled (`glEnable(GL_POLYGON_STIPPLE)`). The number of bits set in the stipple pattern determine the amount of translucency and opacity; setting more bits result in a more opaque object, and setting fewer bits results in a more translucent object. Screen door transparency is sometimes preferable to blending, because it's order independent (primitives don't need to be rendered in back-to-front order).

Technical University of Vienna, IEEE Vis 2004
Transparency in OpenGL [4]:
Glass

15.040 How can I render glass with OpenGL?

This question is difficult to answer, because what looks like glass to one person might not to another. What follows is a general algorithm to get you started.

First render all opaque objects in your scene. Disable lighting, enable blending, and render your glass geometry with a small alpha value. This should result in a faint rendering of your object in the framebuffer. (Note: You may need to sort your glass geometry, so it's rendered in back to front Z order.)

Now, you need to add the specular highlight. Set your ambient and diffuse material colors to black, and your specular material and light colors to white. Enable lighting. Set glDepthFunc(GL_EQUAL), then render your glass object a second time.

Goon Creative, Maya Transparency Tutorial
15.050 Do I need to render my primitives from back to front for correct rendering of translucent primitives to occur?

If your hardware supports destination alpha, you can experiment with different glBlendFunc() settings that use destination alpha. However, this won't solve all the problems with depth buffered translucent surfaces. The only sure way to achieve visually correct results is to sort and render your primitives from back to front.
15.070 If I draw a translucent primitive and draw another primitive behind it, I expect the second primitive to show through the first, but it's not there?

Is depth buffering enabled?

If you're drawing a polygon that's behind another polygon, and depth test is enabled, then the new polygon will typically lose the depth test, and no blending will occur. On the other hand, if you've disabled depth test, the new polygon will be blended with the existing polygon, regardless of whether it's behind or in front of it.
15.080 How can I make part of my texture maps transparent or translucent?

It depends on the effect you're trying to achieve.

If you want blending to occur after the texture has been applied, then use the OpenGL blending feature. Try this:

```c
glEnable (GL_BLEND);
glBlendFunc (GL_ONE, GL_ONE);
```

You might want to use the alpha values that result from texture mapping in the blend function. If so, (GL_SRC_ALPHA,GL_ONE_MINUS_SRC_ALPHA) is always a good function to start with.

However, if you want blending to occur when the primitive is texture mapped (i.e., you want parts of the texture map to allow the underlying color of the primitive to show through), then don't use OpenGL blending. Instead, you'd use glTexEnv(), and set the texture environment mode to GL_BLEND. In this case, you'd want to leave the texture environment color to its default value of (0,0,0,0).