Textures and Transparency (Java3D)

10-19-05
What is a Texture?

- Usually a 2D array of pixels (can also be 1D, 3D, or 4D).
- Textures may have multiple layers to support mipmapping, a LOD feature.
- In Java3D, a Texture owns a ImageComponent, which is compatible with the java.awt.image.BufferedImage and java.awt.image.RenderedImage.
- Easiest method of loading a Texture is to use the Java3D TextureLoader class.
Textures and Appearance

- Textures can be added to an Appearance in several ways:
  - As individual textures (1 texture).
  - As multiple texture levels (n textures).
  - As cubic environment maps (6 textures).

- Not supported in Java3D (yet):
  - Dynamic shadow textures (requires render-to-texture).
  - Custom use of textures by vertex and pixel shader programs (e.g. bump maps, specular maps).
Individual Textures

1. Create Texture object:
   String n = "../data/ground1.gif";
   Texture t = new TextureLoader(n).getTexture();

2. Add Texture to Appearance:
   appearance.setTexture(t);

3. Add TextureAttributes:
   TextureAttributes ta = new TextureAttributes();
   appearance.setTextureAttributes(ta);
Multiple Texture Levels

1. Create a Texture (t) object.
2. Create a TextureAttributes (ta) object.
3. Create a TexCoordGeneration (tcg) object (optional – for cubic reflection maps).
4. Create a TextureUnitState object:
   TextureUnitState tus = new TextureUnitState(t, ta, tcg);
5. Add TextureUnitState to Appearance:
   appearance.setTextureUnitState(0, tus);
6. Repeat.
Cubic Environment Maps

- Supported by the TextureCubeMap object in Java3D (which extends Texture).
- Useful when you are using a SkyBox and want an object to appear as though it is reflecting the environment.
- Uses six ImageComponent2D objects to represent the environment.
- Add images (im[]) to the cube map (tcm) using:

```java
tcm.setImage(0, TextureCubeMap.POSITIVE_X, im[0]);
```
TextureAttributes

- Texture blending options are determined by the TextureAttributes object.
- Several blending modes are supported; two of the most useful modes are:
  - DECAL
    - Replace the current object colors (per pixel) with the texture in the current unit. Uses alpha channel to blend.
  - MODULATE
    - Multiply the current object colors (per pixel) with the texture in the current unit.
TextureAttributes (cont.)

- Set the texture mode for a TextureAttributes (ta) object using:
  `ta.setTextureMode(TextureAttributes.MODULATE);`
- Additional modes allow further customization:
  - **BLEND**
    - Blends with a constant color.
  - **COMBINE**
    - Uses additional parameters to fine-tune blending.
DECAL Example
MODULATE Example
How does transparency work?

- All opaque objects are rendered first.
- All transparent objects are then rendered in order starting with the object furthest from the viewer.
- When a transparent object is rasterized, pixels are linearly interpolated with the pixels already in the frame buffer according to transparency.
TransparencyAttributes

- Transparency is handled using a Transparency object.
- Several Transparency modes are available, BLENDED being the most common (and useful).
- Set up transparency using:
  ```java
  TransparencyAttributes ta = new TransparencyAttributes();
ta.setTransparencyMode(TransparencyAttributes.BLENDED);
ta.setTransparency(0.5f);
  ```
TransparencyAttributes (cont.)

- The transparency attribute in TransparencyAttributes is used when no alpha channel is available (from vertex or texture color).
- Source and destination blend functions can be specified to further tune transparency.
- Screen door mode is also available (though it doesn’t look very good).
Transparency