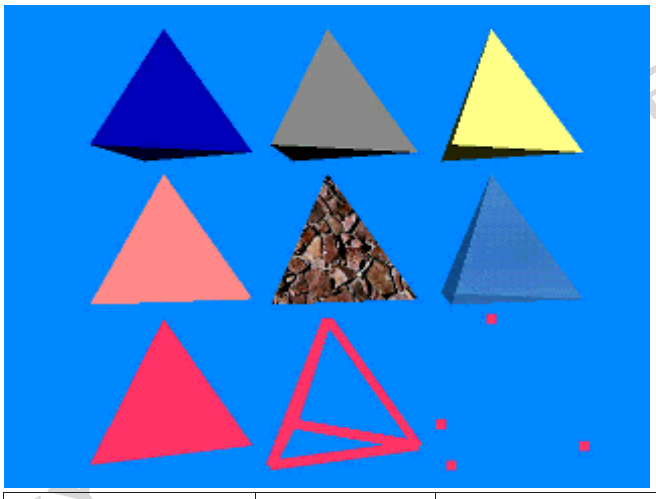
MATERIAL & TRANSPARENCY IN APPEARANCE

1. The `Appearance` class and its components provide the mechanism for appearance control.
   - Appearance control determines how Java 3D renders `Geometry`
     - Color
     - Transparency
     - Shading model
     - Line thickness
     - And lots more

- Appearance example



Diffuse	Specular	Diffuse&Specular
Shaded	Textured	Transparent
Unlit polygons	Unlit lines	Unlit points

2. Appearance attributes are grouped into several node components:
   - Color and transparency control
     - `Material`
     - `ColoringAttributes`
     - `TransparencyAttributes`
   - Rendering control
     - `PointAttributes`
     - `LineAttributes`
     - `PolygonAttributes`
     - `RenderingAttributes`

   - Texture control
     - `Texture`
     - `TextureAttributes`
     - `TexCoordGeneration`

3. Using material attributes
   - `Material` controls:
     - Ambient, emissive, diffuse, and specular color
     - Shininess factor
   - Use materials when a shape *is* shaded
     - Most scene shapes
     - Overrides `ColoringAttributes` intrinsic color (when lighting is enabled)

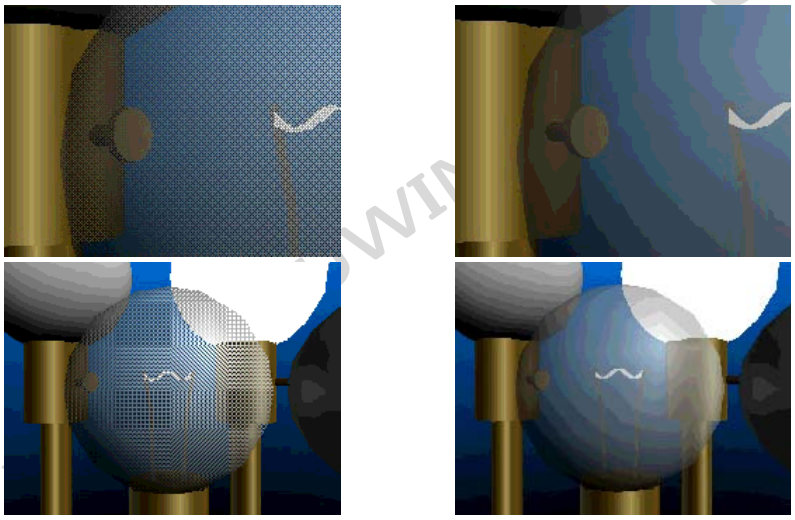
4. Using material colors

- *Diffuse color* sets the main shading color, giving a dull, matte finish (upper-left)
- *Specular color* and *shininess factor* make a shape appear shiny (lower-right)
- *Emissive color* makes a shape appear to glow (upper-right)
- Defaults include white diffuse and specular colors, a black emissive color, (0.2,0.2,0.2) ambient color, shininess of 64.0, and lighting enabled.

5. `TransparencyAttributes` controls:

- Transparency range is 0.0 (opaque) to 1.0 (invisible)
  - By default, transparency amount is 0.0 (opaque) with a `FASTEST` transparency mode

- Transparency mode includes: `SCREEN_DOOR`, `BLENDED`, `NONE`, `FASTEST` (default), and `NICEST`



`SCREEN_DOOR` **`BLENDED`**

  - The `FASTEST` and `NICEST` transparency modes automatically select the fastest and highest quality modes available

6. Interpolators associated with appearance control

- All interpolators use a target into which to write new values
  - A `ColorInterpolator` uses a `Material` target
  - A `TransparencyInterpolator` uses a `TransparencyAttributes` target

THE APPEARANCE OF TEXTURES

1. The appearance of textures
   - Texture image colors can replace, modulate, or blend with shape color
     - Different *texture modes* are useful for different effects
     - Some are faster to draw than others
   - Different texture images can be used at different distances between the shape and the user
     - Use lower resolution images for distant shapes
     - This is known as *Mip-mapping*
2. Combining texture and shape colors
   - A texture image may contain:

     - A red-green-blue color at each pixel
     - A transparency, or *alpha* value at each pixel
   - Alpha blending is a linear blending from one value to another as alpha goes from 0.0 to 1.0:

$$Value = (1.0 - alpha) * Value0 + alpha * Value1$$

3. The *Texture mode* in `TextureAttributes` controls how texture pixels affect shape color
   - Different texture modes
     - `REPLACE` Texture color completely replaces the shapes material color
     - `DECAL` Texture color is blended as a decal on top of the shapes material color

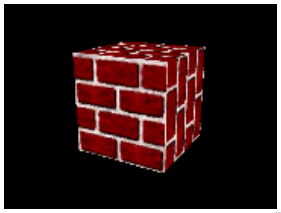
- `MODULATE` Texture color modulates (filters) the shapes material color
- `BLEND` Texture color blends the shapes material color with an arbitrary *blend color*

• Resulting appearance

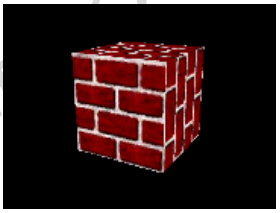
Mode	Result color	Result transparency
`REPLACE`	T_{rgb}	T_a
`DECAL`	$S_{rgb} * (1 - T_a) + T_{rgb} * T_a$	S_a
`MODULATE`	$S_{rgb} * T_{rgb}$	$S_a * T_a$
`BLEND`	$S_{rgb} * (1 - T_{rgb}) + B_{rgb} * T_{rgb}$	$S_a * T_a$

- T_{rgb} is the texture pixel color
- T_a is the texture pixel alpha
- S_{rgb} is the color of the shape being texture mapped

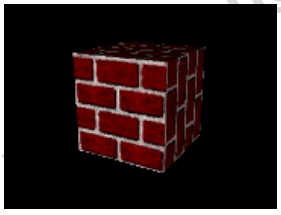
- S_a is the alpha of the shape being texture mapped
- B_{rgb} is the shape blend color
- B_a is the shape blend alpha



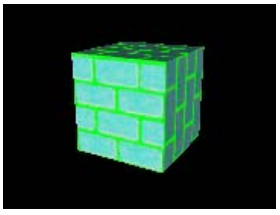
REPLACE



DECAL



ODULATE with white



BLEND with green

4. Typical use:
   - Use `REPLACE` for emissive textures
     - Glowing "neon" textures
     - Textures where lighting is painted in
   - Use `MODULATE` on a white shape for shaded textures
     - Most textured shaded surfaces
   - Use `BLEND` on a colored shape for colorized textures
     - Colorizing a grayscale woodgrain, marble, etc.
5. Texture mode example code
   - Create `TextureAttributes`

     ```
     TextureAttributes myTA = new TextureAttributes( );
     ```
   - Set the texture mode to `MODULATE`


     ```
     myTA.setTextureMode( Texture.MODULATE );
     ```
   - Set the texture attributes on an `Appearance`

     ```
     Appearance myAppear = new Appearance( );
     myAppear.setTextureAttributes( myTA );
     ```
6. *Mip-mapping* is an anti-aliasing technique that uses different texture versions (levels) at different distances from the user
   - There can be any number of *levels*
     - Level 0 is the base image used when the user is close
   - Mip-maps can be computed automatically from a base image:
     - Use a mip-mapping mode of `BASE_LEVEL`

- Or you can specify each image level explicitly:
  - Use a mip-mapping mode of `MULTI_LEVEL_MIPMAP`
- A *Minification filter* controls texture interpolation when a scene pixel maps to multiple texture pixels (texels)
  - `FASTEST` uses fastest method
  - `NICEST` uses best looking method
  - `BASE_LEVEL_POINT` uses nearest texel in level 0 map
  - `BASE_LEVEL_LINEAR` bilinearly interpolates 4 nearest texels in level 0 map
  - `MULTI_LEVEL_POINT` uses nearest texel in mip-mapped maps
  - `MULTI_LEVEL_LINEAR` bilinearly interpolates 4 nearest texels in mip-mapped maps

- A *Magnification filter* controls how a texture is interpolated when a scene pixel maps to less than one texel
  - `FASTEST` uses fastest method
  - `NICESET` uses best looking method
  - `BASE_LEVEL_POINT` uses nearest texel in level 0 map
  - `BASE_LEVEL_LINEAR` bilinearly interpolates 4 nearest texels in level 0 map

BASE_LEVEL_POINT
No interpolation

BASE_LEVEL_LINEAR
Linear interpolation of 4 nearest neighbors

7. Texture filter example code

- Load a texture image

```
TextureLoader myLoader = new TextureLoader("brick.jpg");
ImageComponent2D myImage = myLoader.getImage( );
```

- Create a Texture2D using the image, and turn it on

```
Texture2D myTex = new Texture2D( );
myTex.setImage( 0, myImage );
myTex.setEnable( true );
```

- Set the filtering types

```
myTex.setMagFilter( Texture.BASE_LEVEL_POINT );
myTex.setMinFilter( Texture.BASE_LEVEL_POINT );
```

- Create an Appearance and set the texture in it

```
Appearance myAppear = new Appearance( );
```



```
myAppear.setTexture( myTex );
```