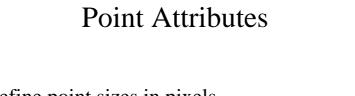
# Shape3D - Appearances

Appearance objects can refer to several different Node Component subclasses called appearance attribute objects

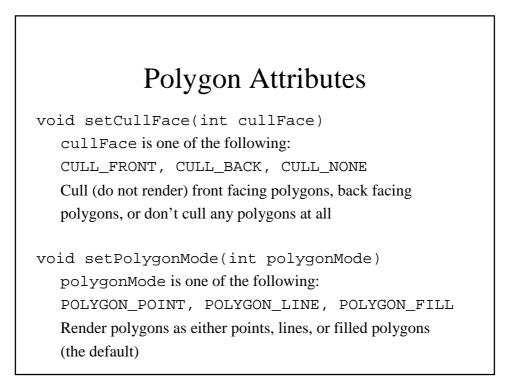
- PointAttributes
- LineAttributes
- PolygonAttributes
- ColoringAttributes
- TransparencyAttributes
- RenderingAttributes
- Material
- TextureAttributes
- Texture
- TexCoordGeneration

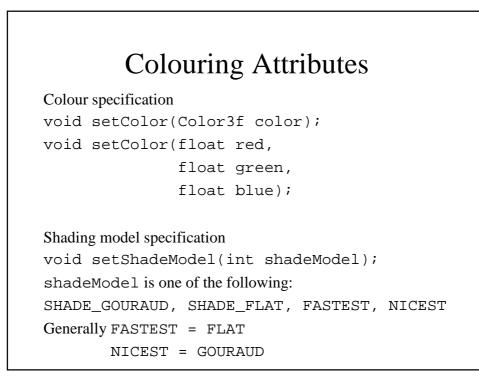


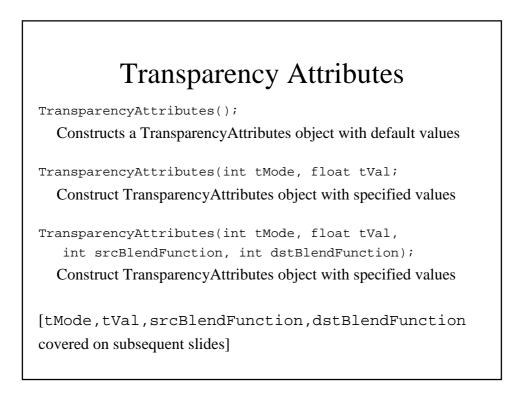
To define point sizes in pixels void setPointSize(float pointSize)

To Enable/disable point anti-aliasing void setPointAntialiasingEnable (boolean state) Only relevant if pointSize > 1 pixel

#### Line Attributes void setLinePattern(int linePattern) PATTERN\_SOLID solid lines (no pattern). This is the default. PATTERN\_DASH dashed lines; ideally, a repeating pattern of 8 pixels on and 8 pixels off. PATTERN\_DOT dotted lines; ideally, a repeating pattern of 1 pixel on and 7 pixels off. PATTERN\_DASH\_DOT dashed-dotted lines; ideally, a repeating pattern of 7 pixels on, 4 pixels off, 1 pixel on, and 4 pixels off.

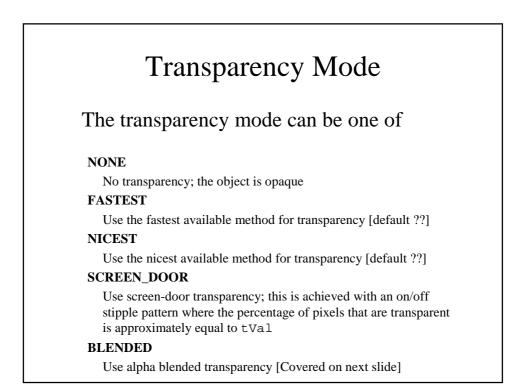






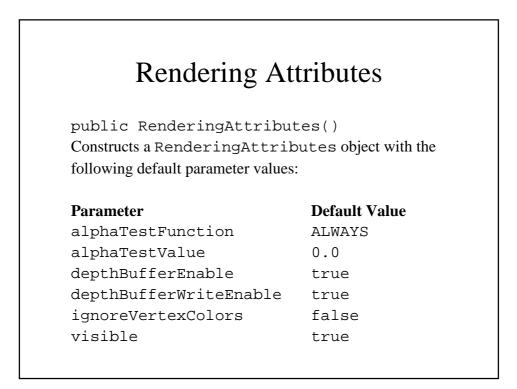
#### Transparency Value

- Note that the *transparency value* (tVal) is the *opposite* of the *alpha value* used in the AlphaComposite class of Java 2D
- tVal lies in the range [0.0, 1.0]
  - 0.0 is fully opaque
    - Equivalent to Java 2D alpha of 1.0
  - 1.0 is fully transparent
    - Equivalent to Java 2D alpha of 0.0
- Quantity (1-tVal) still used in blend equations of alpha blend functions in Java 3D [See later]



## **Blend Equations**

- Transparency mode BLENDED
- The blend equation is specified by a srcBlendFunction and a dstBlendFunction (cf AlphaComposite class in Java 2D)
- Blend equation form:  $Blend = f_s * SrcColour + f_d * DstColour$
- Default source blend function is BLEND\_SRC\_ALPHA
- Default destination blend function is BLEND\_ONE\_MINUS\_SRC\_ALPHA
- Specifiable blend functions are:
  - BLEND\_ZERO the blend function is  $f_i = 0$  i  $\varepsilon$  {s,d} BLEND\_ONE - the blend function is  $f_i = 1$ BLEND\_SRC\_ALPHA - the blend function is  $f_i = alphasrc$ BLEND\_ONE\_MINUS\_SRC\_ALPHA - the blend function is  $f_i = 1 - alphasrc$ where alphasrc = 1 - tVal of the source



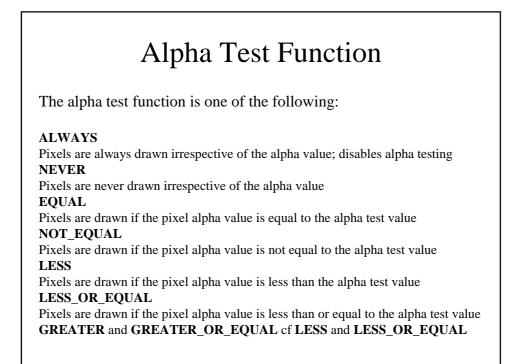
# Pixel Rendering Operations

- RenderingAttributes controls two different per-pixel rendering operations
  - Alpha test
    - setAlphaTestValue()
       setAlphaTestFunction()

determine whether and how the alpha test function is used

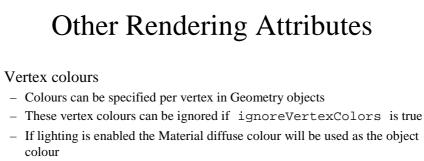
- Depth buffer test
  - setDepthBufferEnable()
     setDepthBufferWriteEnable()

determine whether and how the depth buffer is used for hidden surface removal



## Depth Buffer

- The **depth buffer** is the collection of depth values for rendered pixels
  - It is used to determine the visibility or occlusion of pixels as they are rendered
  - It is used differently when rendering opaque and transparent objects
  - As transparent objects do not occlude opaque objects they do not normally update the depth buffer
- The Depth buffer can be enabled or disabled for this RenderingAttributes component object
  - Disabling the depth buffer ensures that an object is always visible, regardless of any occlusion that would normally have occurred
- The setDepthBufferWriteEnable() method enables or disables writing the depth buffer for this object
- By default both the buffer and DepthBufferWrite are enabled



- Otherwise, if lighting is disabled, the ColoringAttributes colour is used
- The default value is false
- Visibility
  - Visual objects are made invisible using the Visibility flag
  - When the Visibility flag is false, visual objects are not rendered
  - The flag is set with the setVisible() method
  - By default, the Visibility flag is true

#### Material

The Material object defines the appearance of an object under illumination. If the Material object in an Appearance object is null, lighting is disabled for all nodes that use that Appearance

```
Material();
```

constructs and initialises a Material object using default parameters
Material(Color3f ambientColor,

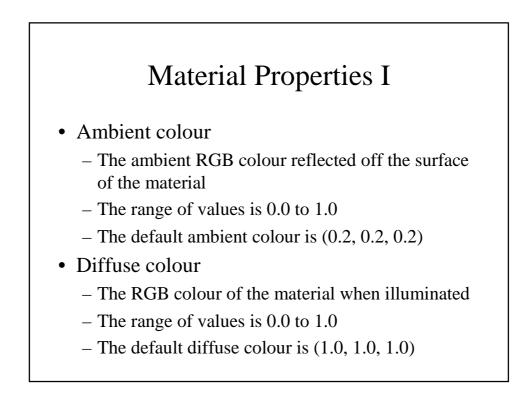
Color3f emissiveColor,

```
Color3f diffuseColor,
```

```
Color3f specularColor,
```

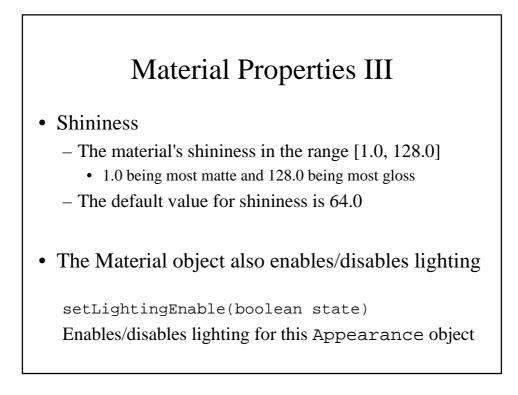
float shininess);

constructs and initialises a new Material object with given parameters



### Material Properties II

- Specular colour
  - The RGB specular colour of the material
    - Highlights
  - The range of values is 0.0 to 1.0
  - The default specular colour is (1.0, 1.0, 1.0)
- Emissive colour
  - The RGB colour of the light the material emits
  - The range of values is 0.0 to 1.0
  - The default emissive colour is (0.0, 0.0, 0.0)



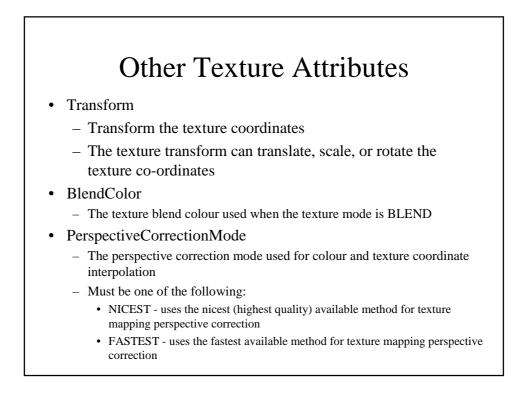
#### **Texture Attributes**

TextureAttributes() Constructs a TextureAttributes object with default parameters

```
TextureAttributes(int textureMode,
   Transform3D transform, Color4f textureBlendColor,
   int perspectiveCorrectionMode)
Constructs a TextureAttributes object with the specified values
```

The TextureAttributes object defines attributes that apply to texture mapping according to a textureMode which is one of the following -

MODULATE - modulates the object colour with the texture colour DECAL - applies the texture colour to the object as a decal BLEND - blends the texture blend colour with the object colour REPLACE - replaces the object colour with the texture colour



## Texture

Texture() Constructs a Texture object with default parameters

Texture(int mipMapMode, int format, int width, int height) Constructs an empty Texture object with specified mipMapMode, format (RGBA normally), width and height

# MipMap Mode

The Mipmap mode specifies how many levels (images) form the texture map -

#### BASE\_LEVEL

Indicates that this Texture object only has a base-level image If multiple levels are needed they will be implicitly computed

#### MULTI\_LEVEL\_MIPMAP

Indicates that this Texture object has multiple images One for each mipmap level If this mode is used images for all levels must be provided

#### TexCoordGeneration

- Java 3D can automatically generate the texture coordinates needed for texture mapping onto contours.
- The TexCoordGeneration attributes specify functions for automatically generating texture coordinates

TexCoordGeneration(int genMode, int format, Vector4f planeS, Vector4f planeT, Vector4f planeR) Constructs a TexCoordGeneration object with the specified genMode, format, and S, T, and R coordinate plane equations.

