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`

Point Attributes

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.
- **`PATTERN_USER_DEFINED`** lines with a user-defined line pattern.

Polygon Attributes

`void setCullFace(int cullFace)`

`cullFace` is one of the following:

`CULL_FRONT, CULL_BACK, CULL_NONE`

Cull (do not render) front facing polygons, back facing polygons, or don't cull any polygons at all

`void setPolygonMode(int polygonMode)`

`polygonMode` is one of the following:

`POLYGON_POINT, POLYGON_LINE, POLYGON_FILL`

Render polygons as either points, lines, or filled polygons (the default)

Colouring Attributes

Colour specification

```
void setColor(Color3f color);
void setColor(float red,
              float green,
              float blue);
```

Shading model specification

```
void setShadeModel(int shadeModel);
```

`shadeModel` is one of the following:

`SHADE_GOURAUD, SHADE_FLAT, FASTEST, NICEST`

Generally `FASTEST = FLAT`

`NICEST = GOURAUD`

Transparency Attributes

```
TransparencyAttributes();
```

Constructs a TransparencyAttributes object with default values

```
TransparencyAttributes(int tMode, float tVal;
```

Construct TransparencyAttributes object with specified values

```
TransparencyAttributes(int tMode, float tVal,
   int srcBlendFunction, int dstBlendFunction);
```

Construct TransparencyAttributes object with specified values

[`tMode,tVal,srcBlendFunction,dstBlendFunction` covered on subsequent slides]

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]

Transparency Mode

The transparency mode can be one of

NONE
No transparency; the object is opaque

FASTEST
Use the fastest available method for transparency [default ??]

NICEST
Use the nicest available method for transparency [default ??]

SCREEN_DOOR
Use screen-door transparency; this is achieved with an on/off stipple pattern where the percentage of pixels that are transparent is approximately equal to `tVal`

BLENDED
Use alpha blended transparency [Covered on next slide]

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 $f_i = 0$ $\quad i \varepsilon \{s,d\}$

  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

Rendering Attributes

`public RenderingAttributes()`
Constructs a `RenderingAttributes` object with the following default parameter values:

Parameter	**Default Value**
`alphaTestFunction`	`ALWAYS`
`alphaTestValue`	`0.0`
`depthBufferEnable`	`true`
`depthBufferWriteEnable`	`true`
`ignoreVertexColors`	`false`
`visible`	`true`

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

Alpha Test Function

The alpha test function is one of the following:

ALWAYS
Pixels are always drawn irrespective of the alpha value; disables alpha testing
NEVER
Pixels are never drawn irrespective of the alpha value
EQUAL
Pixels are drawn if the pixel alpha value is equal to the alpha test value
NOT_EQUAL
Pixels are drawn if the pixel alpha value is not equal to the alpha test value
LESS
Pixels are drawn if the pixel alpha value is less than the alpha test value
LESS_OR_EQUAL
Pixels are drawn if the pixel alpha value is less than or equal to the alpha test value
GREATER and **GREATER_OR_EQUAL** cf **LESS** and **LESS_OR_EQUAL**

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

Other Rendering Attributes

- Vertex colours
  - Colours can be specified per vertex in Geometry objects
  - These vertex colours can be ignored if `ignoreVertexColors` is true
  - If lighting is enabled the Material diffuse colour will be used as the object colour
  - 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 I

- Ambient colour
  - The ambient RGB colour reflected off the surface of the material
  - The range of values is 0.0 to 1.0
  - The default ambient colour is (0.2, 0.2, 0.2)
- Diffuse colour
  - The RGB colour of the material when illuminated
  - The range of values is 0.0 to 1.0
  - The default diffuse colour is (1.0, 1.0, 1.0)

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)

Material Properties III

- Shininess
  - The material's shininess in the range [1.0, 128.0]
    - 1.0 being most matte and 128.0 being most gloss
  - The default value for shininess is 64.0
- The Material object also enables/disables lighting

`setLightingEnable(boolean state)`
Enables/disables lighting for this `Appearance` object

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

Other Texture Attributes

- Transform
  - Transform the texture coordinates
  - The texture transform can translate, scale, or rotate the texture co-ordinates
- BlendColor
  - The texture blend colour used when the texture mode is BLEND
- PerspectiveCorrectionMode
  - The perspective correction mode used for colour and texture coordinate interpolation
  - Must be one of the following:
    - NICEST - uses the nicest (highest quality) available method for texture mapping perspective correction
    - FASTEST - uses the fastest available method for texture mapping perspective correction

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.

Texture Format

A texture generation mode defines how the texture coordinates are generated

OBJECT_LINEAR

Texture coordinates are generated as a linear function in object coordinates

EYE_LINEAR

Texture coordinates are generated as a linear function in eye coordinates

SPHERE_MAP

Texture coordinates are generated using spherical reflection mapping in eye coordinates

TexCoordGeneration needs to know whether the texture is in 2D or 3D format

TEXTURE_COORDINATE_2

Generates 2D texture coordinates (S, T)

TEXTURE_COORDINATE_3

Generates 3D texture coordinates (R, S, T)

Plane Equation Definition

Plane equation coefficients define the plane equations used to generate the coordinates in the OBJECT_LINEAR and EYE_LINEAR texture generation modes.
The coefficients define a reference plane in either object coordinates or in eye coordinates, depending on the texture generation mode

The equation coefficients are set by the `setPlaneS()`, `setPlaneT()`, and `setPlaneR()` methods

By default the equation coefficients are

plane S = (1.0, 0.0, 0.0, 0.0)
plane T = (0.0, 1.0, 0.0, 0.0)
plane R = (0.0, 0.0, 0.0, 0.0)

Loading Textures from Files

NB The dimensions of a texture image must be 2^n

```
import com.sun.j3d.utils.image.*;
TextureLoader loader = new
  TextureLoader(”imagefilename”, this);
ImageComponent2D image = loader.getImage();

Texture2D texture = new Texture2D();
texture.setImage(0, image);

Appearance appear = new Appearance();
appear.setTexture(texture);
```

Sharing Node Component Attributes