CMER

Centre for Mobile Education and Research

Java ME & Blackberry APIs for Game Dev

Week III

Overview

- Java 2D API
- Java 3D API
- SVG
- Blackberry APIs

Java 2D API

- Set of classes for advanced 2D graphics and imaging
- Encompasses
  - Line art
  - Text
  - Images
- Provides extensive support for
  - Image composition
  - Alpha channel images

Interfaces and Classes

- Java.awt - Interfaces
  - Composite
    - Defines methods to compose a draw primitive with the underlying graphics area.
  - CompositeContext
    - Defines the encapsulated and optimized environment for a composite operation
  - Paint
    - Defines colors for a draw or fill operation

Interfaces and Classes (Cont.)

- Continued…
  - PaintContext
    - Defines the optimized environment for a pain operation
  - Stroke
    - Generates the Shape that encloses the outline of the Shape to be rendered.

Interfaces and Classes (Cont.)

- Java.awt Classes
  - AffineTransform (java.awt.geom)
    - Represents a 2D affine transform, which performs a linear mapping from 2D coordinants to other 2D coordinants
  - AlphaComposite
    - Implements basic alpha composite rules for shapes, text and images
  - BasicStroke
    - Defines the “pen style” to be applied to the Shape
  - Color
    - Defines a solid color fill for a Shape

Interfaces and Classes (Cont.)

- Continued
  - GradientPaint
    - Defines a linear color gradient fill pattern for a Shape
  - Graphics2D
    - Fundamental class for 2D rendering.
  - TexturePaint
    - Defines a texture or pattern fill for a Shape.

2D Rendering Concepts

- To render a graphic object you must
  - Set up a Graphics2D context then
  - Pass the graphic object to one of the Graphics2D rendering methods

2D Rendering Concepts (Cont.)

- You can modify the state attributes to:
  - Vary the stroke width
  - Change how strokes are joined together
  - Set a clipping path to limit the rendered area
  - Translate, rotate, scale or shear rendered objects
  - Define colors and patterns to fill shapes with
  - Specify how multiple graphics objects should be composed.

Rendering Process

- Rendering process can be broken into 4 steps

1. If the shape is to be stroked, the Stroke attribute in the Graphics2D context is used to generate a new Shape that encompasses the stroked path
2. The coordinates of the Shape’s path are transformed from user space into device space according to the transform attribute in the Graphics2D context

Rendering Process (Cont.)

3. The Shape's path is clipped using the clp attribute in the Graphics2D context
4. The remaining Shape, if any, is filled using the Paint and Composite attributes in the Graphics2D context

Controlling Rendering Quality

- 2D API lets you indicate whether you want objects to be rendered as quickly as possible
- Or quality rendering to be s high was possible
- Your preferences are specified as hints through the RenderingHints attribute in the Graphics2D context

Controlling Rendering Quality (Cont.)

- RenderingHints class supports the following types of hints:
  - Alpha interpolation – can be set to default, quality, or speed
  - Antialiasing – can be set to default, on or off
  - Color Rendering – can be set to default, quality, or speed
  - Dithering – can be set to default, disable or enable

Controlling Rendering Quality (Cont.)

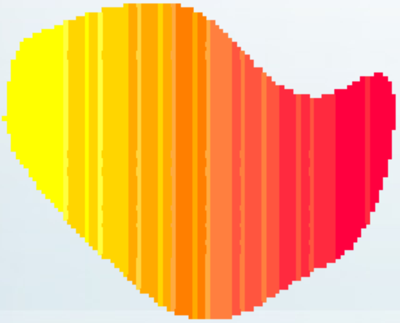
- RenderingHints continued
  - Fractional Metrics – can be set to default on/off
  - Interpolation – can be set to nearest-neighbor, bilinear, or bicubic
  - Rendering – can be set to default, quality, or speed
  - Text antialiasing – can be set to default, on/off

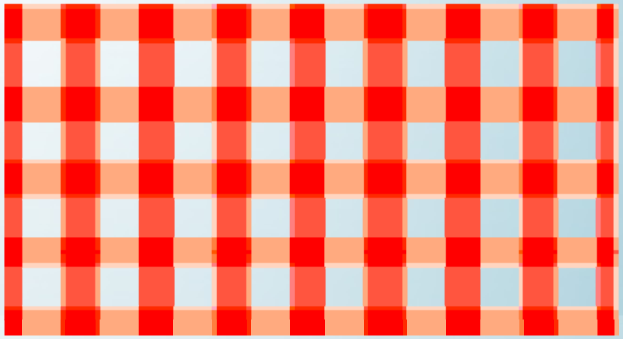
Filling Attributes

- The fill attribute in the Graphics2D context is represented by a Pain object
  - Use setPaint to add Paint to the Graphics2D
- Simple solid color fills can be set with the setColor method. Color is the simplest implementation of the Paint interface

Filling Attributes (Cont.)

- To fill Shapes with more complex paint styles like gradients and textures, use Paint classes:
  - GradientPaint and TexturePaint





- When fill is called to render a shape:
  - Determines what pixels comprise the Shape.
  - Gets the color of each pixel from the Paint object.
  - Converts the color to an appropriate pixel value for the output device.
  - Writes the pixel to that device.

Transformations

- The Graphics2D context contains a transform that is used to transform objects from user space to device space during rendering
- To perform additional transformations, like rotations or scaling, add other transforms to the Graphics2D context
- Simplest transform ability is to call methods like:
  - Rotate - Scale
  - Shear - Translate

Transformations (Cont.)

- Other Abilities include
  - Transparency / Managing Transparency
  - Clipping
  - Specifying Composition Style

Java 3D API

- Is an application programming interface for writing 3-dimensional graphics applications
- Gives high-level constructs for
  - creating and manipulating 3D geometry
  - for constructing the structures used in rendering that geometry
- Part of JavaMedia suite API, making it "write once, run anywhere"

Java 3D API (Cont.)

- It draws the ideas from existing graphics APIs and from new technology.
- Java 3D’s low-level graphics constructs synthesize the best ideas found in low-level APIs such as Direct3D, QuickDraw3D, OpenGL, and XGL
- Java 3D introduces some concepts not commonly considered part of the graphics environment, ex 3D spatial sound

Rendering Modes

- Immediate Mode
  - Raised level of abstraction and accelerates immediate mode rendering on a per-object basis
- Retained Mode
  - Requires an application to construct a scene graph and specify which elements of that scene graph may change during rendering
- Compiled-Retained Mode
  - Like retained mode, additional the application can compile some or all of the subgraphs that make up a complete scene graph

High Performance

- Target Hardware Platforms
  - Aimed at a wide range of 3d-capable hardware and software platforms, from low to high end 3D image generators
  - 3D implementations are expected to provide useful rendering rates on most modern PCs, on midrange PCs near full-speed hardware performance
  - Java 3D is designed to scale as the underlying hardware platforms increase in speed over time.

High Performance (Cont.)

- Layered Implementation
  - One of the more important factors that determines performance is the time it takes to render the visible geometry
  - Java 3D is layered to take advantage of native low-level API that is available on a given system
  - In particular, implementations use Direct3D and OpenGL are available.

Recipe for a Java 3D Program

- An example for the steps to create scene graph elements and link them together

1. Create a Canvas3D and add it
2. Create a BranchGroup as the root of the scene branch graph
3. Construct a Shape3D node with a TransformGroup node above it
4. Attach a RotationInterpolator behavior to the TransformGroup.

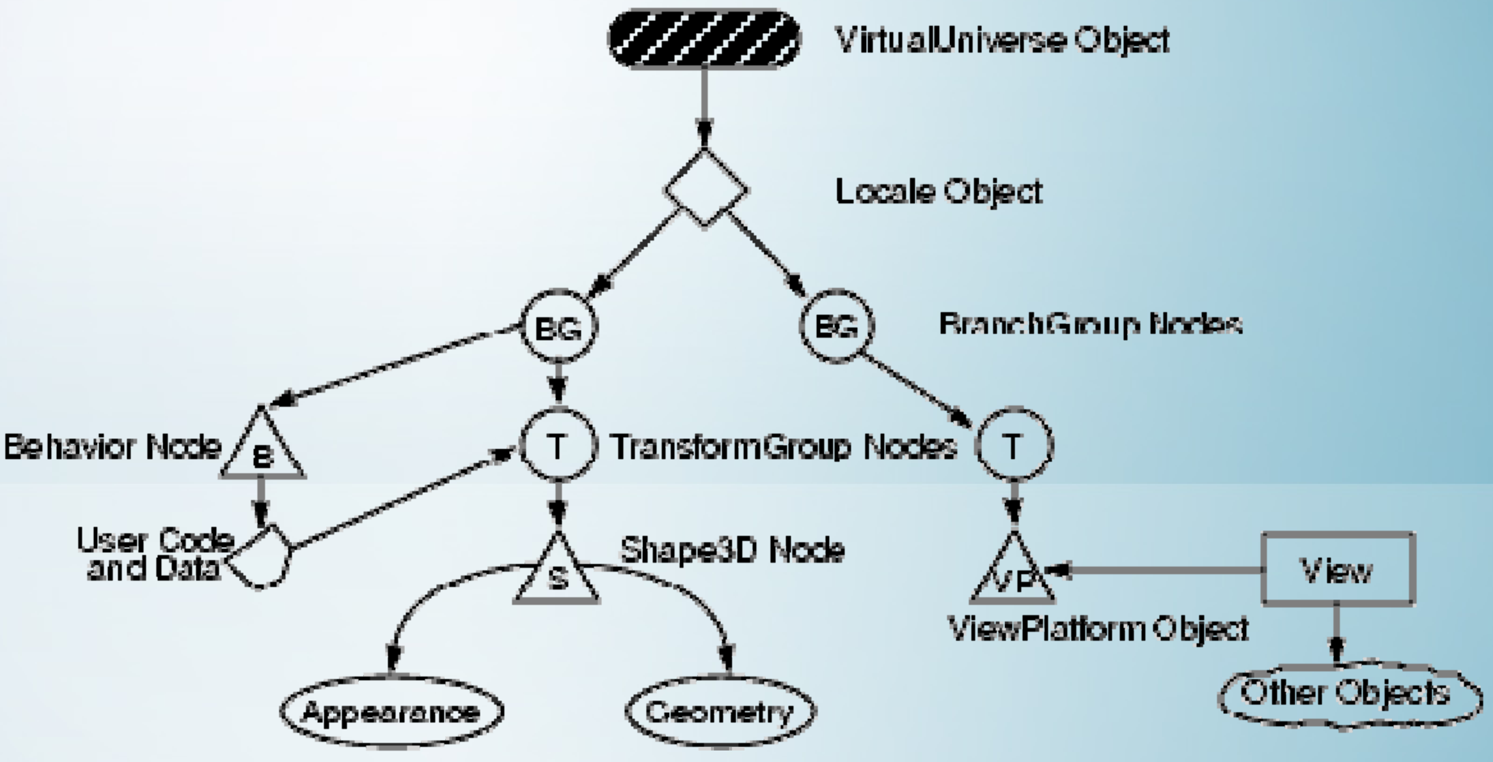
Recipe for a Java 3D Program (Cont.)

5. Call the simple universe utility function to do the following:
   - Establish a virtual universe with a single high-res Locale
   - Create PhysicalBody, PhysicalEnvironment, View, and ViewPlat-form objects
   - Create a BranchGroup s the root of the view platform branch graph
   - Insert the view platform branch graph into the Locale
6. Insert the scene branch graph into the simple universe’s Locale

Java 3D Application Scene Graph

- Below is a sample application
- The scene graph consists of a superstructure components-a VirtualUniverse object, a Locale object and a set of branch graphs.
- Each branch graph is a subgraph that is rooted by a BranchGroup node that is attached to the super structure.

Java 3D Object Hierarchy



SVG

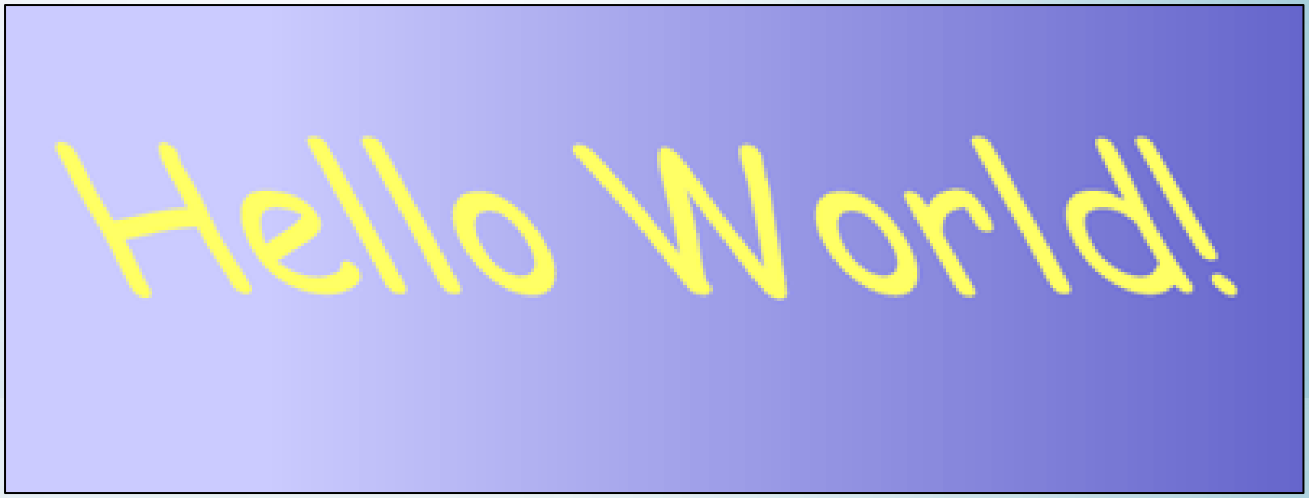
- Scalable Vector Graphics
- SVG is a web format that allows content developers to create two dimensional graphics in a standard way, using XML grammar.
- Several authoring tools already support this format (such as Adobe Illustrator and Corel Draw)

SVG: Example

```
<svg width="640" height="240">
 <title>SVG Hello World! Example</title>
 <defs>
     <linearGradient id="the_gradient"
       gradientUnits="objectBoundingBox"
       x1="0" y1="0"
       x2="1" y2="0">
         <stop offset="0" stop-color="rgb(204,204,255)"/>
         <stop offset="0.2" stop-color="rgb(204,204,255)"/>
         <stop offset="1" stop-color="rgb(102,102,204)"/>
     </linearGradient>
 </defs>
 <g>
     <rect x="0" y="0" width="640" height="480" fill="url(#the_gradient)"/>
     <text x="145" y="140" transform="translate(175,140) scale(4) skewX(30)
     translate(-175,-140)" font-size="24" font-family="ComicSansMS"
     fill="rgb(255,255,102)">Hello World!</text>
 </g>
</svg>
```

SVG: Example

- The code would output an image like this:



Blackberry

- Important Objects used in creation
  - BitmapField
  - ButtonField
  - LabelField

Blackberry - Managers

- The following four classes extend the Manager class:
  - – VerticalFieldManager
  - – HorizontalFieldManager
  - – FlowFieldManager
  - – DialogFieldManager

References

- Java 2D API

  http://java.sun.com/j2se/1.4.2/docs/guide/2d/index.html

- Java 3D API

  http://java.sun.com/javase/technologies/desktop/java3d/forDevelopers/J3D_1_2_API/j3dguide/Intro.doc.html

- Java AVG

  http://java.sun.com/developer/technicalArticles/GUI/svg/