

University of Southern California
School of Engineering
USC Viterbi

Using 2D Animation With SVG: Introduction to JSR 226

Murali Annavaram & Bhaskar
Krishnamachari
Ming Hsieh Department of Electrical
Engineering
USC

Lecture notes based in part on slides created by Vidya Settur,
Thanks also to Mikko H. Lipasti for course suggestions.

Previous Lecture

- In previous class we looked at Tiledlayers, Sprites for generating graphics in gaming
- These API use Raster images (gif, jpeg) and transform them
 - > Raster stores pixel information
 - > Raster images are good for picture like quality on screen
 - > The disadvantage of Raster images is that they take more space – serious disadvantage in mobile domain
 - > Raster images cause distortion in zooming
- How do we address these drawbacks?

Vector Graphics

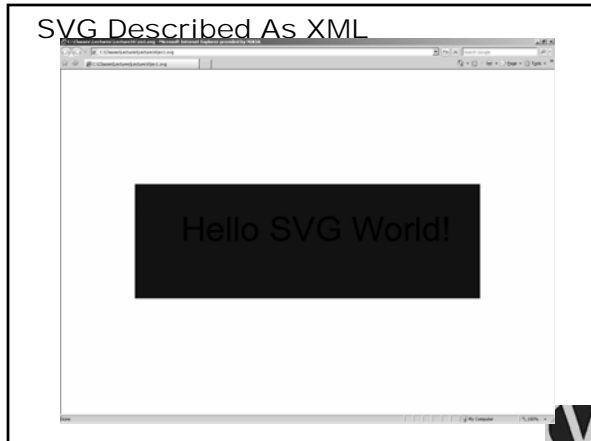
- Vector graphics use mathematical equations to represent basic shapes
 - > Lines, points, curves, rectangles, circles, ovals...
 - > Much smaller file sizes
- Vector graphics do not have a set resolution
 - > Better at zooming without distortion
 - > Independent of resolution – uses the best resolution based on the device screen resolution
- Unfortunately, poor representation of images
 - > Most photo images are not simple shapes
 - > In raster graphics representation of images is a collection of pixels.

Scalable Vector Graphics

- XML-based graphics standard from W3C.
- Allows three types of visual entities:
 - > Graphics shapes
 - > Text
 - > Bitmap graphics like PNGs embedded
- Simple Example of SVG

```
<?xml version="1.0"?>
<svg xmlns="http://www.w3.org/2000/svg" version="1.1"
baseProfile="tiny" viewBox="0 0 30 30">
  <desc>Example SVG file</desc>
  <rect x="0" y="10" width="30" height="10" fill="blue"/>
  <text x="4" y="15" style="fill:#000000;stroke:none;font-
family:Arial;font-size:3">Hello SVG World!</text> </svg>
```

SVG Described As XML



SVG Rendering Model on Mobile Phones

- Uses hierarchical model to render the XML on display
 - > The XML file is parsed depth first and painted in that order
 - > The most shallow element is the top layer
 - > Old layers can be obscured by new layers, unless we set the transparency levels...
- I am not a graphics person, so for more information read on your own☺
 - > More on SVG: <http://www.w3.org/TR/SVGMobile12/struct.html>
 - > More on Vector Graphics: http://en.wikipedia.org/wiki/Vector_graphics
 - > SVG Creation Tools: www.tinyline.com



Vector Graphics + Mobile Devices : JSR 226

- JSR 226 provides API for rendering and manipulating vector graphics on CLDC+MIDP 2.0 platforms
- Supports a subset of SVG specification called SVG tiny
 - > More: <http://www.w3.org/TR/SVGMobile/>
- Supports API for rendering images and manipulating the XML component of the image



Vector Graphics Classes

- One package
 - > javax.microedition.m2g
- Supports 5 classes
 - > ScalableGraphics (the fundamental class for rendering SVG images)
 - > ScalableImage (a class that represent images in vector format)
 - > SVGAnimator (a class that's used for animations)
 - > SVGImage (a class the represents an SVG Tiny 1.1 image)



Rendering SVG

- Create an image object using ScalableImage.createImage() API
 - > image = (SVGImage) ScalableImage.createImage(getClass().getResourceAsStream("image.svg"), new CustomResourceHandler());
- Choice 1: Create a new canvas from the image
 - > Canvas canvas = new M2GCanvas(image);

```

public M2GCanvas( ScalableImage inImage ) {
scalableImage = inImage;
// create the scalable graphics instance
scalableGraphics = ScalableGraphics.createInstance();
scalableGraphics.render(image)
}

```



Animating SVGs

- Choice 2: Instead of creating an image you can create an animation object from SVG

```

// create an animator to load the content
SVGAnimator anmtr = SVGAnimator.createAnimator( img);
// add our custom event listener
anmtr.setEventListener(new MyListener(anmtr) );

```
- CreateAnimator creates an animation player
 - > Three states
 - Stop : Player initialized (no events handled)
 - Play : Receives user inputs and invokes handler
 - Paused
- Primary advantage is event handling (user key press) is done through callbacks
 - > Makes animation based on user input very simple (demo)



Receiving User Inputs

- SVGEventListener receives all user inputs from the SVGAnimator canvas
- Write a key event handler that handles all input keys

```

int gameAction = canvas.getGameAction( keyCode );
switch ( gameAction )
{
case Canvas.UP:
scale( 0.10f );
}

```



Manipulating Images

- Get a pointer to the SVG XML file and use several supported transformations


```
Document document = svgImage.getDocument();
SVGSVGElement root =
(SVGSVGElement) document.getDocumentElement();
root.setCurrentRotate(root.getCurrentRotate() + delta);
```

Review

- Using SVG is more efficient than raster images on mobile devices
- JSR 226 provides basic API for manipulating SVG
 - > Create a new image
 - > Create a new animation from the image
 - > Register a listener
 - > Handle user inputs in the listener

Project #4

- Relatively simple assignment
 - > Create an SVG image
 - > Load the SVG images
 - > Handle user inputs and make transformations to SVG
- <http://www.forum.nokia.com/info/sw.nokia.com/id/5305ba6b-e943-42cb-8bff-83d5960a9df4.html>

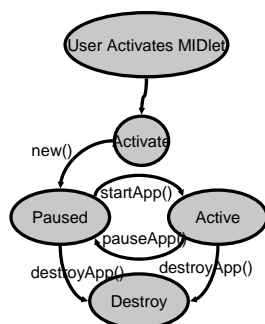
Push Registry : Wake Up Only When You Need To ☺

Murali Annavaram & Bhaskar Krishnamachari
Ming Hsieh Department of Electrical Engineering
USC

Lecture notes based in part on slides created by Vidya Setlur, Thanks also to Mikko H. Lipasti for course suggestions.

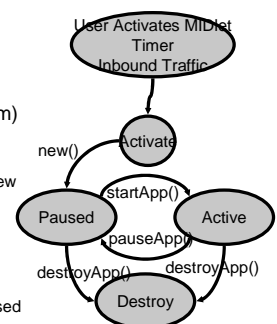
Motivation for Push Registry

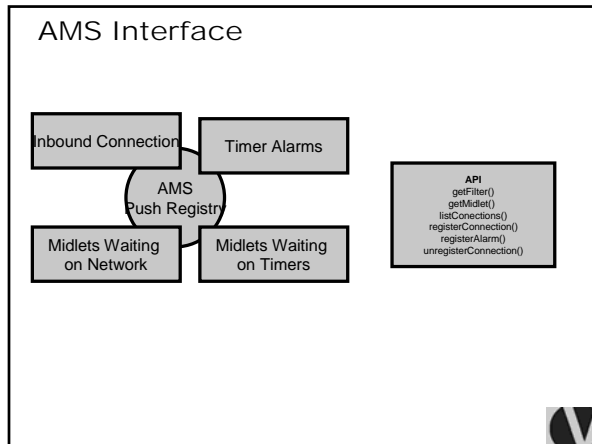
- MIDlet life cycle as we know it
 - > User has to explicitly start the app
- What if we want to execute some code only an incoming SMS message from your boss
 - > Boss, I am too busy; working at the beach on my surfboard!
- One option: Let MIDlet run and continuously poll for incoming SMS
 - > Power inefficient
 - > Compute hungry
- Better option:
 - > PUSH REGISTRY!
 - Act on information as it become available



Push Registry Basics

- Activate MIDlet on
 - > Inbound network activity: SMS, Incoming traffic
 - > Timer based: Act every 10 minutes
- Push Registry handled by AMS (Application Management System)
 - > AMS handles all the application life cycle management already
 - > Add Push Registry API to handle new activation modes
 - > Maintain a list of applications and what they are waiting on
 - > Keeps track of time and network activity
 - > Activate appropriate application based on the events





- ### SMS Recvd While Processing Previous
- When MIDlet is not active
 - > AMS monitors and activates MIDlets
 - When MIDlet is already active
 - > MIDlet's responsibility to handle all timers and inbound network data
 - > In essence, AMS washes off its hand if MIDlet is active

- ### Inbound Network Connections
- HTTP is really not a push connection!
 - > Why?
 - UDP/TCP Sockets/SMS are inbound network connections
 - > Application specifies the protocol, but check for exceptions for code portability since not all protocols may be supported by a device
 - MIDlet activated on incoming data
 - > MIDlet must open the connection and receive data
 - > AMS may not buffer the data (except for SMS)

- ### Push Registration Types
- Static Registration
 - > Happens during MIDlet installation
 - > Registration must use unique connection address
 - > Specified in MIDlet JAD file
 - On netbeans, go to MIDlet → Properties → Application Descriptor → Push Registry → ADD
 - MIDlet-Push-<n>: <ConnectionURL>, <MIDletClassName>, <AllowedSender>
 - MIDlet-Push-<n> : Attribute name
 - Connection URL: string that identifies the inbound endpoint to register (e.g. socket://:5000)
 - MIDlet name should be a fully qualified name
 - Allowed Sender is the filter to restrict sender
 - Dynamic Registration
 - > Register dynamic connections and timer alarms at runtime, using the PushRegistry API
 - > Timer based events can only be registered dynamically

Registering MIDlet API

```

try {
    // Open the connection.
    ServerSocketConnection ssc = (ServerSocketConnection)Connector.open(url);
    // Register the connection
    PushRegistry.registerConnection(url, midletClassName, filter);
    // Now wait for inbound network activity.
    SocketConnection sc = (SocketConnection)ssc.acceptAndOpen();
    // Read data from inbound connection.
    InputStream is = sc.openInputStream();
    // ..... read from the input stream.

    // Here process the inbound data.
    // .....
}
  
```

- ### Unregister Connections
- Remember once you register connection it is there forever
 - > Unless you explicitly unregister
 - Unregister the connection when MIDlet is done with whatever it is processing (or however long you need to wait for a timer)
 - > status = PushRegistry.unregisterConnection(url);

Checking Push Registry State

- Use `listConnections(false)` to list all connections
 - `connections = PushRegistry.listConnections(false)`
- Use `getMIDlet(str)` to get MIDlet waiting on connection
 - `PushRegistry.getMIDlet(connections[i]);`
- Use `getFilter(str)` to get filter information
 - `PushRegistry.getFilter(connections[i]);`



Review

- Push Registry is a useful feature for automatically triggering applications
- Push Registry saves power by relieving the application from having to poll continuously
- Push Registry features can be used in conjunction with any applications that you are developing in this class
 - Although there is no specific project requirement try it out!

