Animating Thousands of Graphics with ArcGIS Runtime SDK for Java

Mark Baird and Vijay Gandhi
Overview

- Runtime architecture
- SDK basics: graphics, features, graphics layers
- Demo 1: Rendering graphics
- Demo 2: Adding graphics
- Demo 3: Moving graphics
- Top tips!
Runtime architecture

- Java SE
- Android
- JNI (Java Native Interface)
- C++ Runtime Core
- Platform Graphics Hardware

SDKs
- Platform-specific

Interop layer
- Native Code
- Cross-platform

C++
Graphics and features

<table>
<thead>
<tr>
<th>Graphic</th>
<th>GeodatabaseFeature</th>
</tr>
</thead>
<tbody>
<tr>
<td>stored in memory on the client</td>
<td>stored in a dataset (geodatabase)</td>
</tr>
<tr>
<td>displayed in a GraphicsLayer</td>
<td>displayed in a FeatureLayer</td>
</tr>
<tr>
<td>can have mixed geometry type layer</td>
<td>one geometry type per layer</td>
</tr>
</tbody>
</table>

For animating moving objects, use Graphic
Graphics layer basics

- **API class is** `GraphicsLayer`
- A graphics layer contains graphics (you guessed it!)
- **Static** and **Dynamic** rendering modes
- Graphic class is immutable: so don’t hold references to Graphic objects
- Update / move / remove graphics using methods on `GraphicsLayer`
- Work with graphics via the layer using their unique ID

```java
id = addGraphic(Graphic)
graphic = getGraphic(id)
...
updateGraphic(id, Graphic)
updateGraphic(id, Symbol)
updateGraphic(id, Geometry)
...
removeGraphic(id)
setGraphicVisible(id, visible)
...```
Rendering graphics

Mark Baird
## Static vs Dynamic - summary

<table>
<thead>
<tr>
<th>Static</th>
<th>Dynamic</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ Volume of graphics has little impact on frame render time (scales well)</td>
<td>- Volume of graphics has direct impact on (GPU) resources</td>
</tr>
<tr>
<td>- Rendering graphic updates is CPU / system memory intensive</td>
<td>+ Individual graphics changes can be efficiently applied directly to GPU state</td>
</tr>
<tr>
<td>Use for static graphics, complex geometries</td>
<td>Good in most cases, especially moving objects</td>
</tr>
</tbody>
</table>

**Java SE:** use Dynamic mode unless you see performance issues
Adding graphics

Vijay Gandhi
Renderer vs Symbol

Symbol (no renderer): Time to add 10k graphics: 17s

Renderer set: Time to add 10k graphics: 0.122s

Memory:

~360,000K

~180,000K
Moving graphics
Mark Baird and Vijay Gandhi
Top tips

- **Move point graphics using `movePointGraphic`** – optimized in Runtime Core
  
  ```java
  graphicsLayer.movePointGraphic(id, newPointLocation);
  ```

- **Use a renderer on the graphics layer rather than setting individual symbols on graphics**
  
  ```java
  graphicsLayer.setRenderer(myRenderer);
  ```

- **Use bulk `addGraphics` method when adding many graphics at the same time**
  
  ```java
  graphicsLayer.addGraphics(Graphic[] graphics);
  ```
More tips for performance

• Split dynamic data from static data
  - Reduce load on dynamic rendering pipeline

• Keep different geometry types on different layers
  - And split polygon fills from polygon outlines if possible

• Use multiple graphics layers
  - 5 layers of 100k point graphics will scale and perform better than 500k on one layer

• Set scale thresholds on the layer
  - only display the relevant subset of graphics
Questions?
## ArcGIS Runtime SDK sessions Thursday

<table>
<thead>
<tr>
<th>Session Name</th>
<th>Time</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>ArcGIS Runtime SDKs: The Road Ahead</td>
<td>1:30pm – 2:45pm</td>
<td>Room 07 A/B</td>
</tr>
</tbody>
</table>
Thank you...

• Please fill out the session survey:

**Offering ID: 1493**

Online – [www.esri.com/ucsessionsurveys](http://www.esri.com/ucsessionsurveys)
Paper – pick up and put in drop box