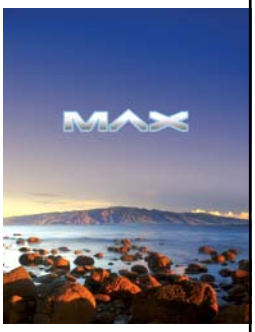
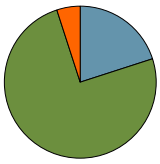
Tips and Tricks for Delivering More Responsive Flex Applications

David George
Adobe Systems

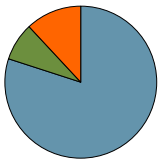


Big Picture

Rendering-intensive tasks (effects, scrolling, resizing)



Other tasks (startup, navigation, data manipulation)



ActionScript
Rendering
Other

Critical areas:

Object creation **Measurement/Layout** **Rendering**

Overview

- Introduction
- Object creation
- Measurement/layout
- Rendering
- Miscellaneous topics

Optimizing ActionScript: Object Creation

Demo

- Demo: rendering speed
- Demo: script execution

The Birth of an Object

- Create instance of ActionScript class
- Assign initial property values

  `<mx:TextArea text="Hi" width="100"/>`
- Wire objects together
  - Add new object to display list
  - Event handlers: `<mx:Button click="goNext()"/>`
  - Data binding: `<mx:Label text="{city}"/>`
  - Effect listeners: `<mx:Label showEffect="{fade}"/>`

Solution #1: Deferred Creation

- Delay object creation until the object becomes visible
- Is baked into Accordion, TabNavigator, and ViewStack
- Can be added to custom containers, but subclassing ViewStack is easier

Optimizing ActionScript: Measurement/Layout

Solution #2: Ordered Creation

- During startup, stagger creation of objects
- Improves *perceived* startup time
- Demo

```
<mx:Application>

  <mx:Panel width="250" height="100" creationPolicy="queued" />
    <mx:Label text="One" />
  </mx:Panel>

  <mx:Panel width="250" height="100" creationPolicy="queued" />
    <mx:Label text="Two" />
  </mx:Panel>

</mx:Application>
```


Measurement/Layout: Definition

The process of assigning a position and size to every component

```
<mx:Application>
  <mx:HBox>
    <mx:Button label="1"/>
    <mx:Button label="2"/>
  </mx:HBox>
  <mx:TextArea width="100%" height="100%" text="Text"/>
</mx:Application>
```

Solution #3: Use <mx:Repeater> Carefully

- Don't allow <mx:Repeater> to create elements that are clipped
- Bad:

```
<mx:VBox>
  <mx:Repeater id="r" dataProvider="{arr}">
    <mx:Image source="r.currentItem.url"/>
  </mx:Repeater>
</mx:VBox>
```

- Good:

```
<mx:List dataProvider="{arr}">
  <mx:itemRenderer>
    <mx:Component>
      <mx:Image source="{dataObject.url}"/>
    </mx:Component>
  </mx:itemRenderer>
</mx:List>
```

- Caveat: Repeater scrolls more smoothly

Measurement/Layout: Description

```
<mx:Application>
    <mx:HBox>
      <mx:Button label="1"/>
      <mx:Button label="2"/>
    </mx:HBox>
    <mx:TextArea width="100%" height="100%" text="Text"/>
</mx:Application>
```

- Measurement Phase: traverse tree from bottom up
  - Buttons and TextArea compute measured sizes
  - HBox computes its measured size
  - Application computes its measured size
- Layout Phase: traverse tree from top down
  - Application sets sizes and positions of HBox and TextArea
  - HBox sets sizes and positions of Buttons
- O(n) algorithm, n = number of objects

Solution #1: Reduce Container Nesting

- Try to use HBox and VBox instead of Grid
- Avoid nesting a VBox inside a Panel or Application
- The root of an MXML component doesn't need to be a Container
- Use Canvas with constraints
- Warning sign: a Container with a single child

Solution #2: Avoid Redundant Measurement/Layout

- Scenario: Flickr app issues 25 image requests
  - When image data arrives, corresponding Image object resizes
  - For each image, whole screen does measurement/layout
- Scenario: Dashboard app creates 6 portal windows
  - Each portal issues web service request
  - For each web service response, portal window's size changes
- Solutions:
  - Delay requests until creationComplete (after incremental layout)
  - Limit geometry changes when responses arrives
  - Stagger requests or queue responses

Optimizing Rendering

Redraw Regions

- If an object's properties are changed, its bounding box is a "redraw region"
- Visualize using "show redraw region"
  - Debug player only
- Objects that overlap the redraw region are redrawn

cacheAsBitmap Protects Innocent Bystanders

- If cacheAsBitmap is true **then** the object and its children are rendered into an offscreen bitmap
- If an object overlaps a redraw region **and** the object is unchanged **then** the cached bitmap is used
- Example: a Move effect

cacheAsBitmap is a Double-Edged Sword

- Objects with cached bitmaps are more expensive to change
- Examples when cacheAsBitmap hurts performance
  - Resize effect
  - Resizing the browser window'
- Suggestion: cache bitmaps only for short periods of time

Factors that Affect Rendering Speed

- Size of redraw region
  - Suggestion: refactor UI
- Cached bitmaps (can help or hurt)
- Total number of vectors in the redraw region
  - Suggestion: simplify geometry
  - Suggestion: use Resize.hideChildren and hide children during state transition
- Mixture of device text and vector graphics
- Clip masks
- Filters (e.g.: DropShadow)
- Other background processing
  - Suggestion: Effect.suspendBackgroundProcessing

Miscellaneous Topics

Reducing Memory Usage

- Discard unused UI
  - myViewStack.removeChild(childView);
  - childView.removeEventListener("click", clickHandler) **or** use weak references
- Clear references to unused data
  - myProperty = null;
  - myWebService.getAddressBook.clearResult();
- Use memory profiling tools
  - MAX talk: "End-to-End Debugging of Flex Applications"

Large Number of UI Screens

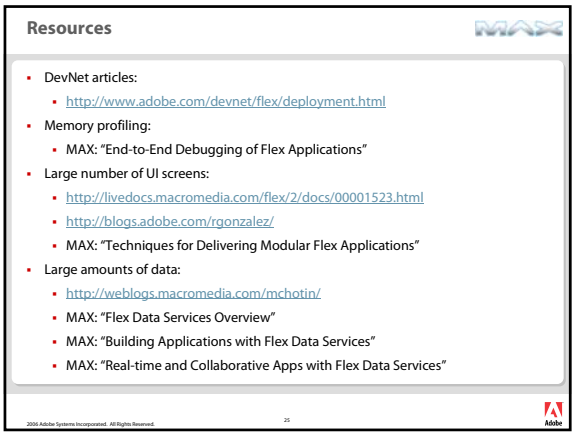
- Main application loads subordinate SWFs using <mx:Loader>
  - Example: Flex Component Explorer http://www.adobe.com/devnet/flex/samples/code_explorer/
- Main application loads subordinate SWFs using "modules"
  - MAX talk: "Techniques for Delivering Modular Flex Applications"
  - Roger's blog: http://blogs.adobe.com/rgonzalez/
- Subordinate SWFs share code using RSLs
  - Docs: http://livedocs.macromedia.com/flex/2/docs/00001523.html

Large Amounts of Data

- Implement your own paging scheme
  - Matt's blog: http://weblogs.macromedia.com/mchotin/
- Use new data services architecture
  - MAX talk: Flex Data Services Overview
  - MAX talk: Building Applications with Flex Data Services
  - MAX talk: Real-time and Collaborative Apps with Flex Data Services

Setting Styles

- Changing a rule set is most expensive
  - StyleManager.styles.Button.setStyle("color", 0xFF0000)
- For inline styles, expense is proportional to the number of objects affected
  - Example: myVBox.setStyle("color", 0xFF0000)
  - Exception: setStyle is cheap during object creation
- If a value will change at runtime, initialize it at authoring time
  - <mx:Style> Button { color: #000000 } </mx:Style>
  - <mx:VBox id="myVBox" color="0x000000">


Resources
DevNet articles:
http://www.adobe.com/devnet/flex/deployment.html
Memory profiling:
MAX: "End-to-End Debugging of Flex Applications"
Large number of UI screens:
http://livedocs.macromedia.com/flex/2/docs/00001523.html
http://blogs.adobe.com/rgonzalez/
MAX: "Techniques for Delivering Modular Flex Applications"
Large amounts of data:
http://weblogs.macromedia.com/mchotin/
MAX: "Flex Data Services Overview"
MAX: "Building Applications with Flex Data Services"
MAX: "Real-time and Collaborative Apps with Flex Data Services"
2006 Adobe Systems Incorporated. All Rights Reserved.
25
Adobe


Better by Adobe™
2006 Adobe Systems Incorporated. All Rights Reserved.
26
Adobe