Performance Optimization of Tizen WebKit: Memory and Graphics

Seojin Kim
Samsung Electronics
Contents

• Introduction
• Rendering Architecture
• Performance Optimization
• Memory Optimization
• Conclusions
Introduction

- Resource Loading
- Parsing
- Layout
- JS Execution

- Render Tree
- Render Layer Tree
- Graphics Layer Tree

- Painting
- Backing Store

- Drawing (Compositing)

- Evas

Legend:
- Write
- Read

html, css, image, js, ...
Overall Rendering Architecture

• WebKit2-EFL
  • UIProcess and WebProcess

• Coordinated graphics architecture
  • WebProcess painting to tile back buffer
  • UIProcess compositing tiles into display

• Benefits
  • User responsiveness: nothing blocks UIProcess from processing user event
Performance
Basic Rendering Flow

1. Paint Web contents

   - WebCore
   - Java Script Core

2. Update tiles and composite

   - Coordinated Compositing
   - Layer
     - Tiled BackingStore
   - TextureMapper

Browser

- Title Bar
- Evas Object
- URL Bar
- Evas Object
- Tool Bar
- Evas Object

WebView - UIProcess

- Evas Object

WebProcess

- CoreIPC

UI FW

- Evas
- OpenGL ES / EGL

Shared Memory

- Backing Store: Tile/Layer Surfaces

Evas

- Title Bar
- URL Bar
- Tool Bar

Window Buffer

- Browser Indicator

Frame Buffer
Optimized Rendering Flow

1. Paint Web contents

2. Composite

- No tile updating by UIProcess
- No intermediate buffer for WebView Evas object
Optimization Render Flow (1/2)

- **SharedPlatformSurfaceTizen**
  - Tile/GraphicsSurface back buffer
  - Locate in GPU memory
  - Both UIProcess and WebProcess access GPU memory directly
  - DDK support is necessary: TBM (tizen buffer manager) and EGL extension

Power consumption and drawing speed
- Zero-copy realized: no texture uploading in UIProcess

Memory consumption
- Restriction: allocation time is long → Introduce pool for recycling
Optimization Render Flow (2/2)

- TextureMapper with Evas GL direct rendering
  - No separate Evas object surface for WebView
  - Compositing destination is window surface itself

Power consumption & Drawing speed
- Reduce compositing count

Memory consumption
- No separate memory space for WebView Evas object
Support Other Graphics Feature

Different Paint Paths:
- Contents layer
- 2D Canvas layer
- WebGL layer

1. Paint Web contents

2. Composite
Accelerated 2D Canvas

• Paint with Cairo-GLES
• Optimize for 2D graphics game
  • Texture Atlas

CanvasPerf Score
2.38
@ Tizen Reference Target + Tizen 2.1

Coordinated Compositing

Content Layer
TiledBackingStore

2D Canvas Layer
GraphicsSurface

Cairo
Pixman
GLES

GPU Memory

Backing Store: Tile/Layer Surfaces
WebGL

- **Directly use GLES**
- **Triple buffering for WebGL layer surface**
  - Pipelining of 3 stages:
  - Painting by WebProcess
  - Issuing commands for compositing by UIProcess
  - Actual compositing by GPU
  → Fully utilize GPU

WebGL Google Aquarium
35.87 fps
@ Tizen Reference Target + Tizen 2.1

![Coordinated Compositing Diagram](image)
Memory
Memory Optimization

Objectives

Ensure that foreground application runs smoothly

- **Foreground: WebView is visible**
  - Active state
  - Ensure certain performance quality with optimal memory usage

- **Background: WebView is invisible**
  - Usually in suspended state
  - Yield as much memory as possible
Foreground Web App Memory Optimization

- **Memory Optimization Parameters:**
  - Tile cover area size
  - Tile size
  - JS engine garbage collection period
  - Resource cache size

<table>
<thead>
<tr>
<th>Contents Characteristics</th>
<th>Major Memory Consumer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dynamic user experience (such as CSS animation, contents layering)</td>
<td>Tile, JS Heap</td>
</tr>
<tr>
<td>Image resource-heavy</td>
<td>Resource Cache</td>
</tr>
<tr>
<td>Large content size</td>
<td>Tile</td>
</tr>
</tbody>
</table>

Memory Usage Breakdown for a web content with very complex UX

- Tile 34%
- JS Heap 21%
- Resource Cache 14%
- Code 7%
- Others 24%
Performance against Memory Trade-off

- **Tile cover area size**
  - Larger means better scrolling responsiveness
  - Smaller means better memory usage

- **Tile size**
  - Larger means better painting speed
  - Smaller means better memory usage
Background Web App Memory Optimization

- **Release memory**
  - Suspend tile painting and purge backing store
  - Utilize disk cache for JS heap and resource cache (to-be)

※ code memory is shared among all web app instances
Conclusions
Conclusions

• Graphics performance optimization
  • Reduce memory operations
  • Different paint path per contents’ characteristics

• Memory optimization
  • Performance-Memory trade-off
  • Release as much memory as possible in the suspended state
Appendix
Coordinated Graphics
Tizen Coordinated Graphics

UI Framework
- OpenGL/ES

Coordinated Compositing
- WebLayerTreeRendererTizen
- LayerTreeCoordinatorProxy
- TextureMapper
- TextureMapperGL
- TextureMapperLayer
- PlatformSurface
- TexturePoolTizen
- PlatformSurfaceTexture
- LayerBackingStoreTizen
- LayerBackingStoreTileTizen

Coordinated Backing Store
- TiledBackingStore
- TiledBackingStoreRemoteTileTizen
- SharedPlatformSurfaceTizen
- PlatformSurfacePoolTizen

WebCore
- RenderLayer
- WebGraphicsLayer
- LayerTreeCoordinator
- LayerTreeCoordinatorProxy

WebProcess
- Cairo

WebKit2-EFL

GPU Memory
- Tile’s backbuffer

PlatformSurface

TexturePoolTizen

LayerBackingStoreTileTizen

Coordinated Tile