TP N°4 : Amélioration des performances de l'interface utilisateur Android

Abdelkader Ouared

---

1 Introduction

Durant 16 millisecondes, Android rafraîchit l'écran du mobile. Ce rafraîchissement dépend des événements tels que la suppression d'un composant ou le changement de visibilité (Visible/Invisible).

2 Cycle de mise à jour de l'UI

Lorsque l'UI change, Android effectue trois opérations principales :

Cycle UI Android

1. **Re-calculate (Espace)** : recalcul des dimensions et positions des composants.
2. **Re-layout (Positionnement)** : repositionnement des composants.
3. **Re-draw (Dessin)** : rendu graphique des composants.

3 Implémentation de l'algorithme de rafraîchissement

Pour information, l'implémentation peut être réalisée en `C`, `C++`, `Rust` ou d'autres langages supportés sur **Android**. Le développeur a également une part de responsabilité pour optimiser et améliorer le rendu de l'interface utilisateur.

3.1 Threads et événements

- **UI Thread** : thread principal responsable des opérations de l'interface utilisateur.
- **RenderThread** : thread chargé du rendu graphique.
- Les événements prenant plus de 16 ms sont colorés en rouge, car ils dépassent le délai correspondant à 60 FPS (frames per second, images par seconde).

3.2 Options pour les développeurs

Pour configurer votre appareil mobile afin de visualiser les mises à jour et les contours des composants, accédez aux options suivantes :

- `Settings > Developer options > Advanced`
- **Show surface updates** : mettre en surbrillance les zones rafraîchies.
- **Show Layout Bounds** : afficher les contours des composants.

4 Optimisation des performances

Pour améliorer les performances, il est conseillé de diminuer le temps consommé par les trois opérations principales :

Opérations à optimiser

- **Re-calculate (Espace)**
- **Re-layout (Positionnement)**
- **Re-draw (Redessiner)**

4.1 Bonnes pratiques

- Éviter le **nesting LinearLayout** trop profond.
- Utiliser **ConstraintLayout** au lieu de LinearLayout.
- Tester sur différents appareils mobiles.
- Consulter **Firebase Monitoring Service (SAS)** pour le suivi des performances.
- Utiliser le **multithreading** :
  - Main Thread : uniquement pour les trois opérations principales.
  - Background Thread : pour les opérations longues.
- Attention à l'erreur `NetworkOnMainThreadException` due aux opérations réseau sur le thread principal.
- Utiliser **Explicit Invalidate** pour rafraîchir uniquement ce qui est nécessaire.
- Supprimer les arrière-plans inutiles dans les layouts.
- Réduire la transparence si possible.
- Optimiser les animations.

5 Travail demandé

1. Réaliser l'UI suivante :
   - Interface de compteur de points divisée en deux colonnes : **Team A** à gauche et **Team B** à droite.
   - Chaque colonne contient :
     - Un score "0" affiché en grand.
     - Trois boutons : **+3 POINTS**, **+2 POINTS**, **FREE THROW**.
   - En bas, au centre, un bouton **RESET**.
   - Un titre **Court Counter** est affiché en haut de l'écran.

   Voir la vidéo : `https://github.com/ouared14/Mobile-Programming-2025/blob/main/TP%20N%C2%B04.mp4`
2. Comparer **LinearLayout** vs **ConstraintLayout**.
3. Gérer la rotation de l'écran.
4. Appliquer d'autres optimisations vues précédemment.
5. Utiliser **Android Monitoring Tool** pour tester les performances.

Rédigez un rapport présentant vos observations sur l'optimisation de l'interface utilisateur. Analysez comment l'utilisation des outils de monitoring, du multithreading et des bonnes pratiques de layout contribue à réduire le temps de calcul, le repositionnement et le redessin des composants afin d'assurer un rendu fluide à 60 FPS sur Android.