

# Utiliser Intel® INDE GPA pour améliorer les performances de votre jeu Android

Par M. TRAPPER

Date de publication : 18 janvier 2016

Vous pouvez commenter le contenu de ce tutoriel sur le forum Jeu : [Commentez](#)

I - Introduction.....	3
II - Remerciements.....	3
III - Organisation du tutoriel.....	3
IV - Prérequis.....	4
V - Application d'exemple City Racer.....	4
VI - Potentiel d'optimisation.....	4
VII - Tutoriel d'optimisation.....	5
VIII - Examinez l'image.....	6
IX - Optimisation 1 - Élimination des objets invisibles.....	8
X - Optimisation 2 - Instanciation.....	9
XI - Optimisation 3 - Tri d'avant en arrière.....	10
XII - Optimisation 4 - Nettoyage rapide.....	11
XIII - Conclusion.....	13

## I - Introduction

Ce tutoriel est constitué d'un guide étape par étape pour analyser les performances, identifier les goulots d'étranglement, et optimiser le rendu d'une application **OpenGL ES 3.0** sur Android. L'application que nous utiliserons comme exemple, intitulée « City Racer », simule une course dans un paysage de style urbain. Les analyses de performances de l'application sont réalisées en utilisant la suite d'outils **Intel® INDE Graphics Performance Analyzers (Intel® INDE GPA)**.

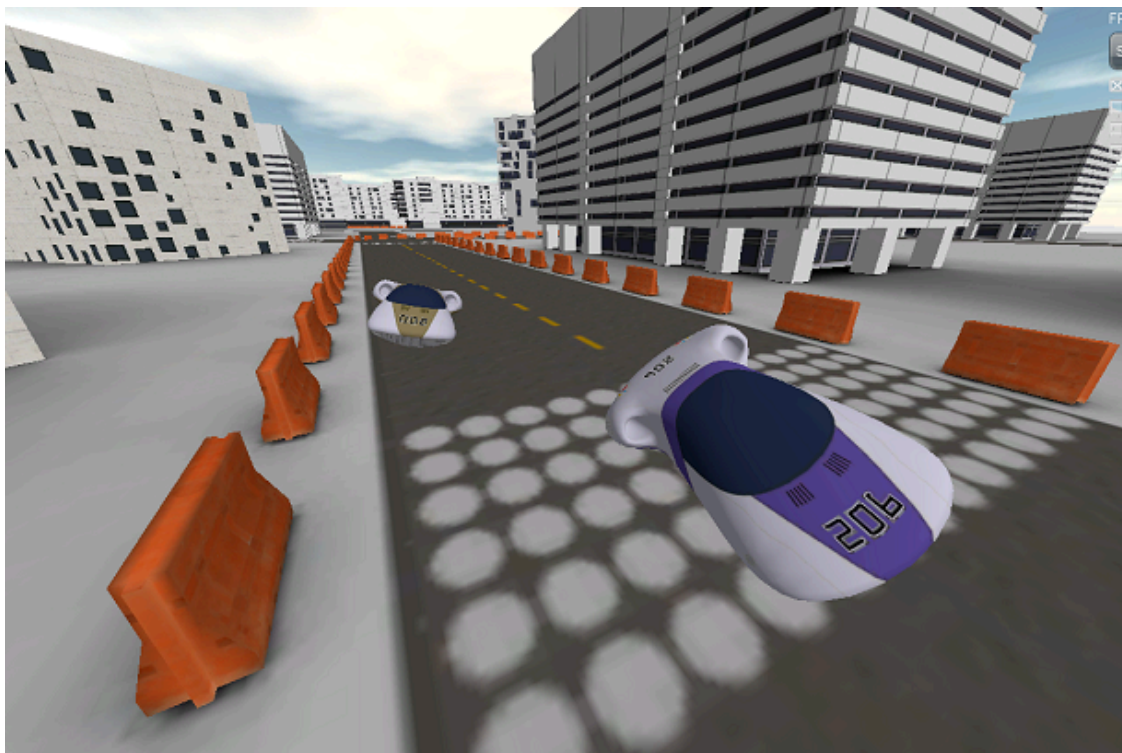


Figure 1

La géométrie combinée de la ville et du véhicule se compose d'approximativement 230 000 polygones (690 000 sommets) avec des éléments de carte éclairés par une unique lumière directionnelle sans ombre. Les éléments de sources fournis incluent le code, les fichiers de projets, et les éléments artistiques nécessaires pour construire l'application, incluant les optimisations de code source identifiées le long de ce tutoriel.

## II - Remerciements

Ce tutoriel est une version Android et OpenGL ES 3.0 du **Workshop de Performances Graphiques Intel pour la 3e Génération de Processeurs Intel® Core™ (Ivy Bridge)** (PDF) créé par David Houlton. Il concerne Intel GPA.

## III - Organisation du tutoriel

Ce tutoriel vous guide à travers quatre étapes d'optimisation successives. À chaque étape, l'application est analysée avec Intel GPA pour identifier les goulots d'étranglement des performances spécifiques. Une optimisation appropriée est ensuite appliquée à l'application pour venir à bout du goulot d'étranglement et l'application est ensuite analysée de nouveau pour mesurer le gain de performances. Les optimisations appliquées sont généralement conformes aux PDF.

Au cours de ce tutoriel, les optimisations appliquées améliorent les performances de rendu de City Racer à hauteur de 83 %.

## IV - Prérequis

- L'exemple City Racer est généré en utilisant l'**API Android version 20** et le **NDK Android version 10**.
- Les analyses de performances sont réalisées en utilisant la **suite d'outils Intel GPA**.
- **Intel GPA** est compatible avec la plupart des appareils Android ; cependant, un appareil Android fonctionnant avec une architecture x86 fournira les indicateurs de profilage les plus détaillés.

## V - Application d'exemple City Racer

City Racer est logiquement divisé entre la simulation de course et le rendu de sous-composants. La simulation de course inclut la modélisation de l'accélération du véhicule, le freinage, les paramètres de tournant, et une intelligence artificielle pour suivre la course et éviter les collisions. Le code de simulation de la course est contenu dans les fichiers `track.cpp` et `vehicle.cpp` et n'est affecté par aucune des optimisations appliquées dans ce tutoriel.

Les composants de rendu se composent du dessin de la géométrie des véhicules et de la scène en utilisant OpenGL ES 3.0 et notre framework CPUT, développé en interne. La version initiale du code de rendu représente un premier jet, contenant plusieurs choix de conception ayant pour effet de limiter les performances.

Les éléments de mesh et de texture sont chargés depuis le fichier média `defaultScene.scene`. Les éléments de maillages individuels sont de plusieurs types : soit des éléments de décor déjà placés, soit des éléments de décor instanciés avec des données de transformations déterminant leur comportement, ou des véhicules pour lesquels la simulation détermine les données de transformation. Il y a plusieurs caméras dans le décor : une pour suivre chaque voiture, et une caméra additionnelle permet à l'utilisateur d'explorer librement le décor. Toutes les analyses de performances et optimisations du code sont dirigées sur le mode de la caméra consistant à suivre les voitures.

Pour les besoins de ce tutoriel, City Racer est conçu pour démarrer en mode pause, ce qui vous permet d'appliquer toutes les étapes de profilage avec des configurations de données rigoureusement identiques. City Racer peut sortir du mode pause en décochant la case à cocher Pause dans l'interface de City Racer ou en ajoutant « `g_Paused = false` » en haut du fichier `CityRacer.cpp`.

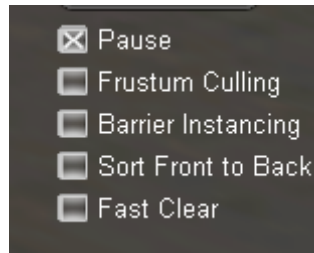
## VI - Potentiel d'optimisation

Considérez l'application City Racer comme un prototype fonctionnel, mais non optimisé. Dans son état initial, il fournit le rendu visuel souhaité, mais pas les performances de rendu. Il est composé d'un certain nombre de choix techniques et de conception qui sont représentatifs de ceux que vous trouveriez dans un jeu en développement typique, voyant ainsi ses performances de rendu limitées. L'objectif de la phase d'optimisation est d'identifier les goulots d'étranglement des performances un à un, d'apporter des changements au code afin de les surmonter, et de mesurer les améliorations réalisées.

Notez que ce tutoriel ne concerne qu'une petite partie des optimisations qui pourraient être appliquées à City Racer. Plus spécifiquement, il ne considère que les optimisations qui peuvent être exclusivement appliquées au code source, sans opérer de changement aux modèles ou aux textures. D'autres optimisations passant par des modifications de ces ressources sont volontairement exclues ici, tout simplement parce qu'elles deviennent assez lourdes à présenter dans un format de tutoriel, mais elles peuvent être identifiées en utilisant les outils Intel INDE GPA, et doivent être prises en compte dans une véritable optimisation de jeu.

Les valeurs de performances montrées dans ce document ont été capturées sur un système basé sur un processeur Intel® Atom™ (nom de code Bay Trail) tournant sous Android. Ces valeurs peuvent différer sur votre système, mais les proportions entre les performances devraient être similaires et logiquement, mener aux mêmes optimisations de performances.

Les optimisations qui seront appliquées durant ce tutoriel se trouvent dans `CityRacer.cpp`. Elles peuvent être appliquées par le biais de l'interface de City Racer ou par des modifications directement réalisées dans `CityRacer.cpp`.



#### CityRacer.cpp

```

1. bool g_Paused = true;
2. bool g_EnableFrustumCulling = false;
3. bool g_EnableBarrierInstancing = false;
4. bool g_EnableFastClear = false;
5. bool g_DisableColorBufferClear = false;
6. bool g_EnableSorting = false;
  
```

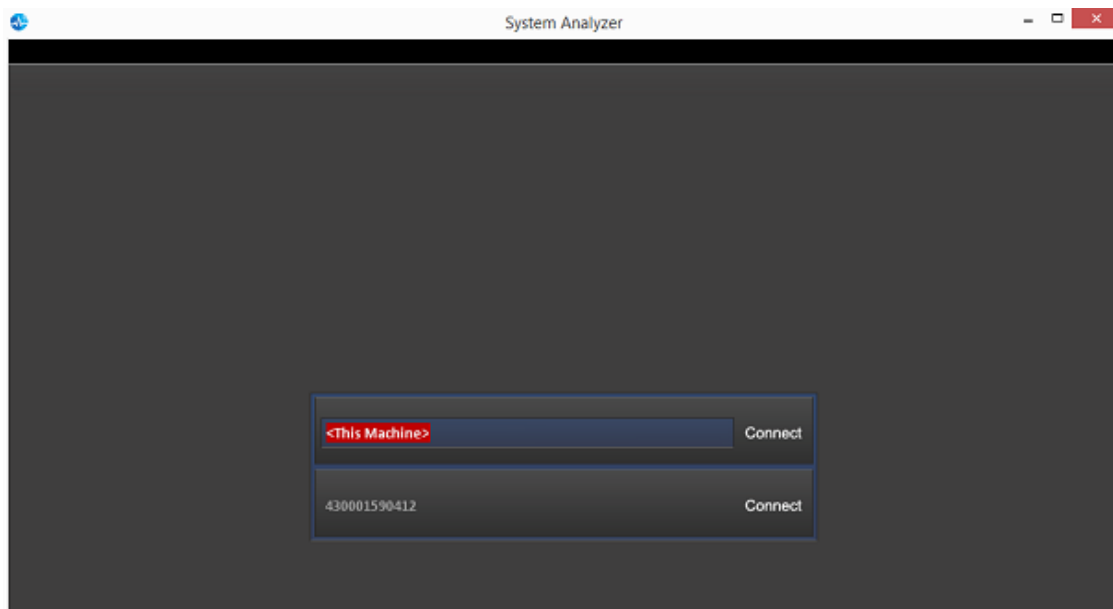
Elles sont activées une par une durant votre progression dans les étapes de l'optimisation. Chaque variable contrôle la substitution d'un ou de plusieurs segments de code pour réaliser l'optimisation de chaque étape du tutoriel.

## VII - Tutoriel d'optimisation

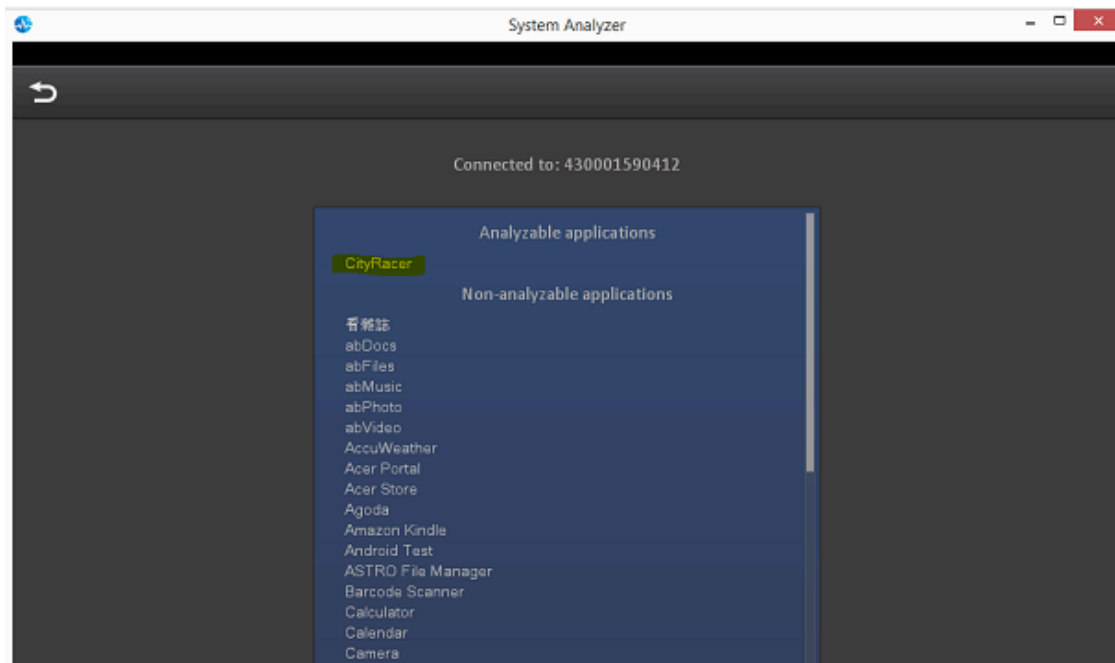
La première étape consiste à compiler et déployer City Racer sur un appareil Android. Si votre environnement Android est correctement réglé, le fichier buildandroid.bat situé dans CityRacer/Game/Code/Android réalisera ces étapes automatiquement.

Ensuite, lancez Intel GPA Monitor, faites un clic droit sur l'icône dans la barre système, et sélectionnez « System Analyzer ».

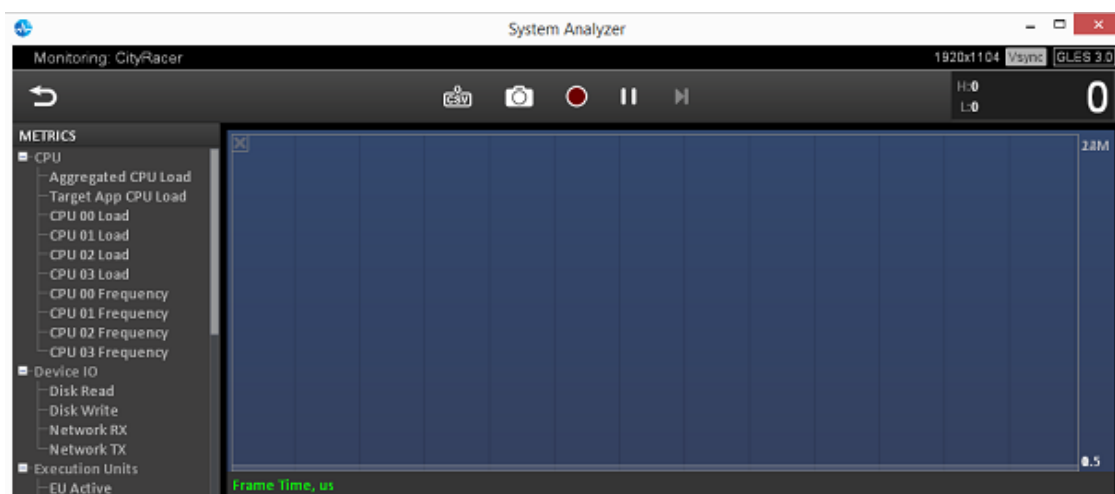
System Analyzer affichera une liste des plateformes auxquelles il peut se connecter. Choisissez votre appareil Android x86 et appuyez sur « Connect. »



Lorsque System Analyzer se connecte à votre appareil Android, il affiche une liste d'applications disponibles pour le profilage. Choisissez City Racer et attendez que l'application se lance.



Lorsque City Racer est lancé, appuyez sur le bouton de capture d'image pour obtenir une capture d'écran du processeur graphique à utiliser pour l'analyse.

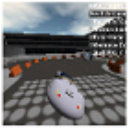




## VIII - Examinez l'image

Ouvrez Frame Analyzer pour OpenGL et choisissez les images de City Racer que vous venez de capturer, ce qui vous permet d'examiner en détail les performances GPU.

Type filter expression

Add



**Name:** CityRacer\_2015\_07\_15\_11\_29\_12

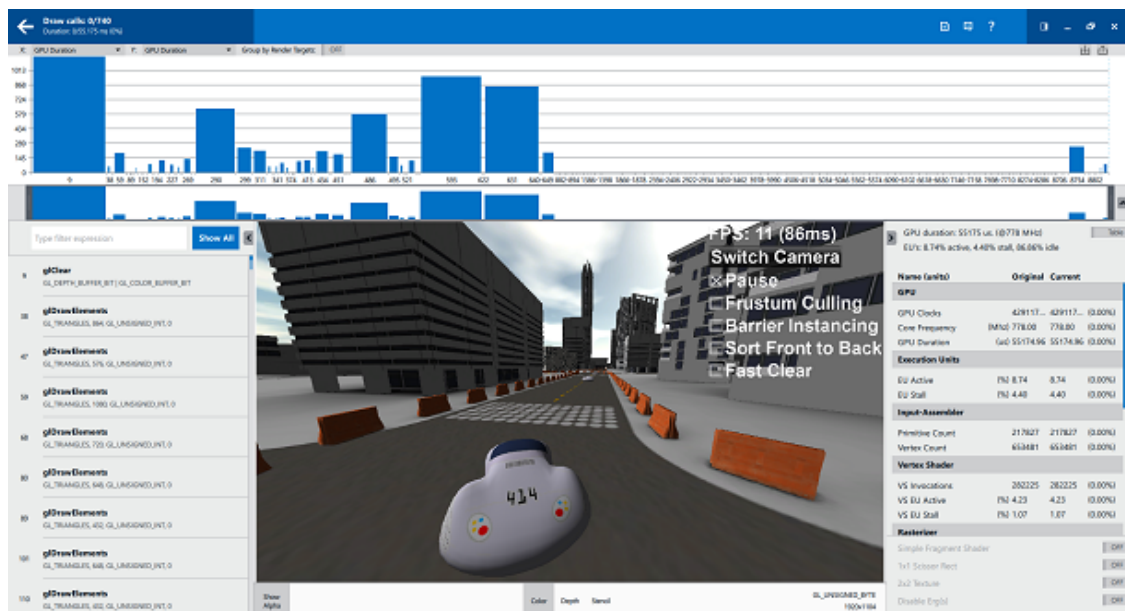
**API:** OpenGL ES 3.0

**Resolution:** 1920x1104

**Device:** 430001590412

**GPU:** Intel(R) HD Graphics for BayTrail

Open



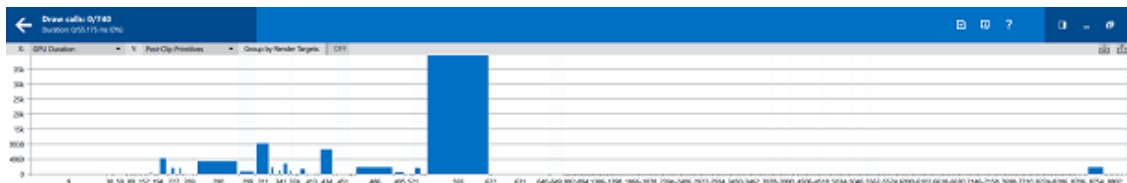
La chronologie du haut se compose d'« ergs » de travail régulièrement espacés, chacun correspondant à un appel de dessin OpenGL. Pour un affichage plus traditionnel de la chronologie, sélectionnez « GPU Duration » sur les axes X et Y. Cela vous montrera rapidement quels sont les ergs qui consomment le plus de temps de GPU et où sont les endroits où nous devrions concentrer nos efforts. Si aucun erg n'est sélectionné, le panneau de droite affiche le temps GPU pour toute l'image, qui est de 55 ms.



GPU duration: 55175 us. (@778 MHz)

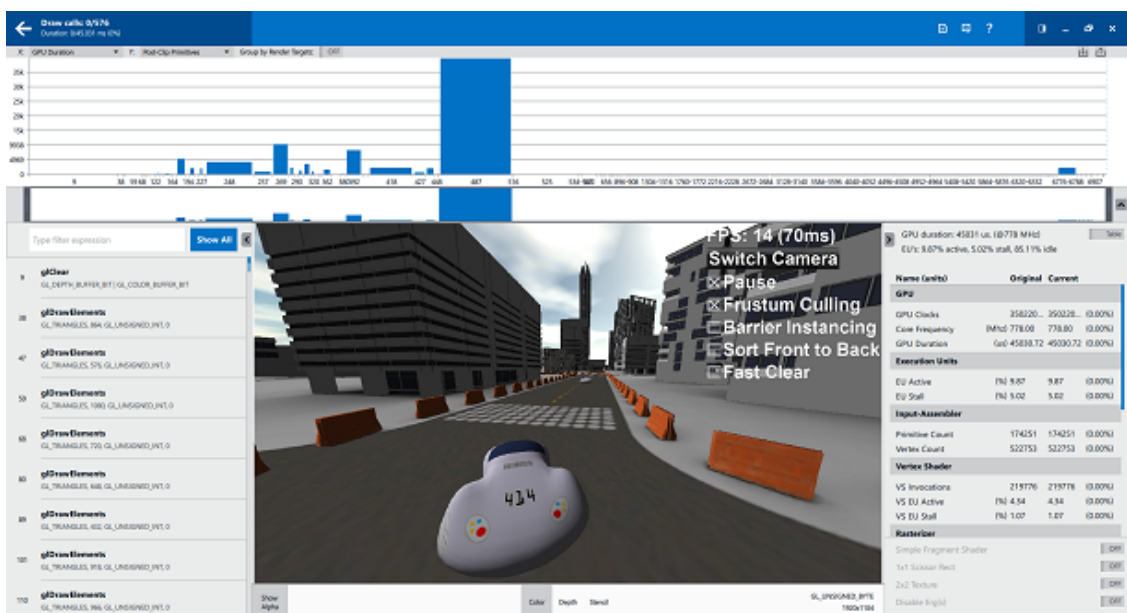
## IX - Optimisation 1 - Élimination des objets invisibles

En regardant tous ces images, nous pouvons remarquer qu'il y a beaucoup d'éléments dessinés qui n'apparaissent pas visuellement à l'écran. En changeant l'axe Y sur la position « Post-Clip Primitives », les interruptions servent, dans cette approche, à déterminer quels sont les dessins qui sont inutiles parce que leur géométrie est complètement sectionnée.



Les bâtiments dans City Racer sont combinés en groupes selon leur localisation spatiale. Nous pouvons retirer les groupes qui ne sont pas visibles, et éliminer ainsi le travail du GPU qui y est associé. En activant la case à cocher « Frustum Culling », chaque dessin sera réalisé à travers une routine d'élimination des objets invisibles par le CPU avant d'être envoyé au GPU.

Activez la case à cocher « Frustum Culling » et utilisez System Analyzer pour capturer une autre image. Une fois que l'image est capturée, ouvrez-la de nouveau dans Frame Analyzer.



En étudiant cette image, nous pouvons voir que le nombre de dessins est réduit de 22 %, passant de 740 à 576, et notre temps GPU global est réduit de 18 %.

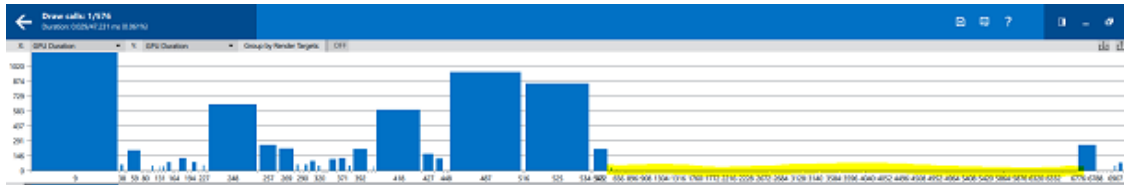
Draw calls: 0/576  
Duration: 0/52.166 ms (0%)

GPU duration: 45031 us. (@778 MHz)

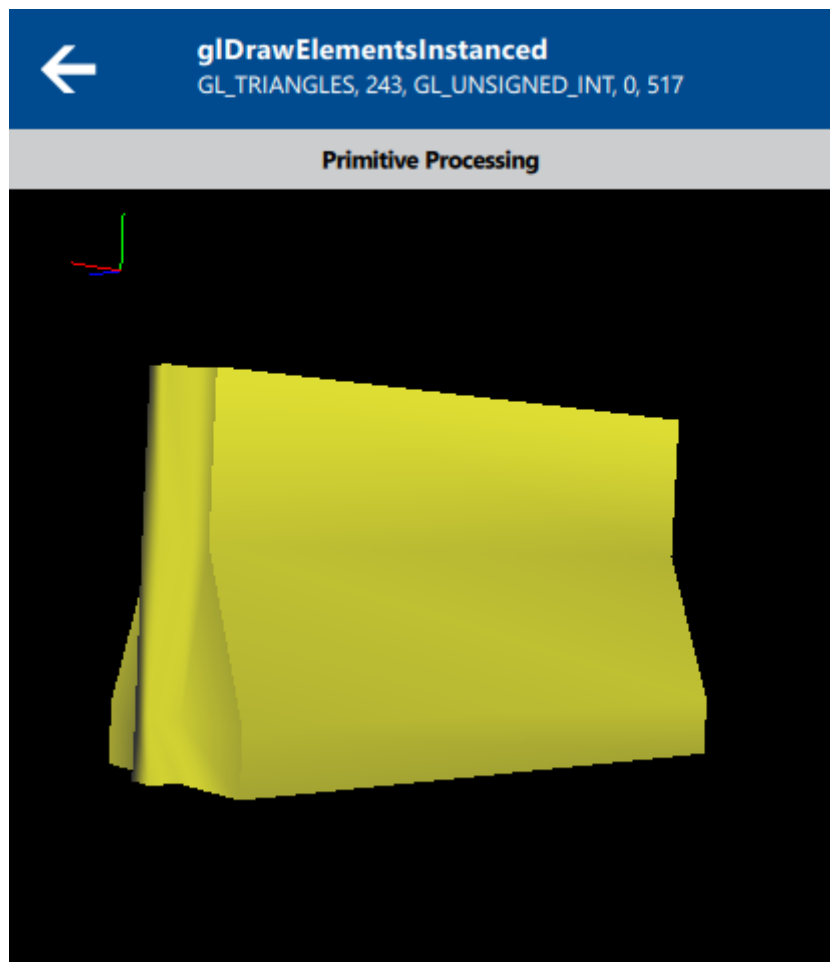


## X - Optimisation 2 - Instanciation

Alors que l'élimination des objets invisibles a réduit le nombre global d'ergs, il reste un grand nombre de petits ergs (surligné en jaune) qui, lorsqu'ils sont cumulés, ajoutent une quantité de temps GPU non négligeable.

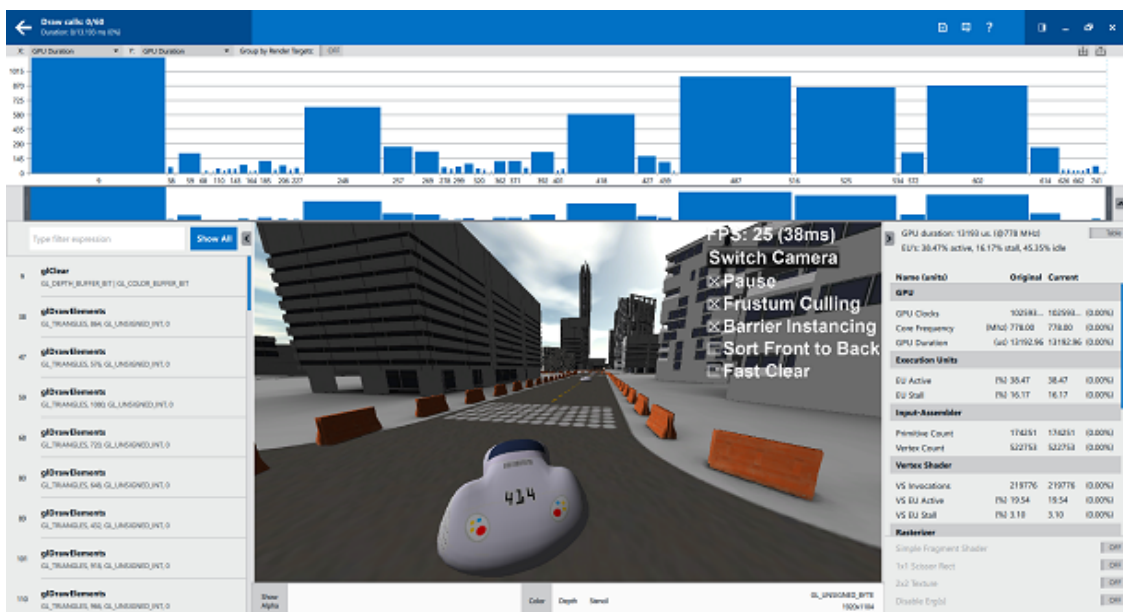


En examinant la géométrie de ces ergs, nous pouvons voir que la majorité d'entre eux constituent les barrières de béton qui délimitent les bords du circuit.



Nous pouvons éliminer une grande partie de la surcharge en cause dans ces dessins en les combinant dans un dessin à instance unique. En cochant la case à cocher « Barrier Instancing » les barrières seront combinées en un dessin à instance unique, retirant ainsi la nécessité pour le CPU de soumettre chacune d'entre elles au GPU.

Activez la case à cocher « Barrier Instancing » et utilisez System Analyzer pour capturer une autre image. Une fois l'image capturée, ouvrez-la dans Frame Analyzer.



En examinant cette image, nous pouvons constater que le nombre de dessins est réduit de 90 %, passant de 576 à 60.



Appels de dessins avant l'instanciation des barrières en béton (en haut) et après l'instanciation (en bas)

De plus, la durée GPU est réduite de 71 % pour atteindre 13 ms.

GPU duration: 13198 us. (@778 MHz)

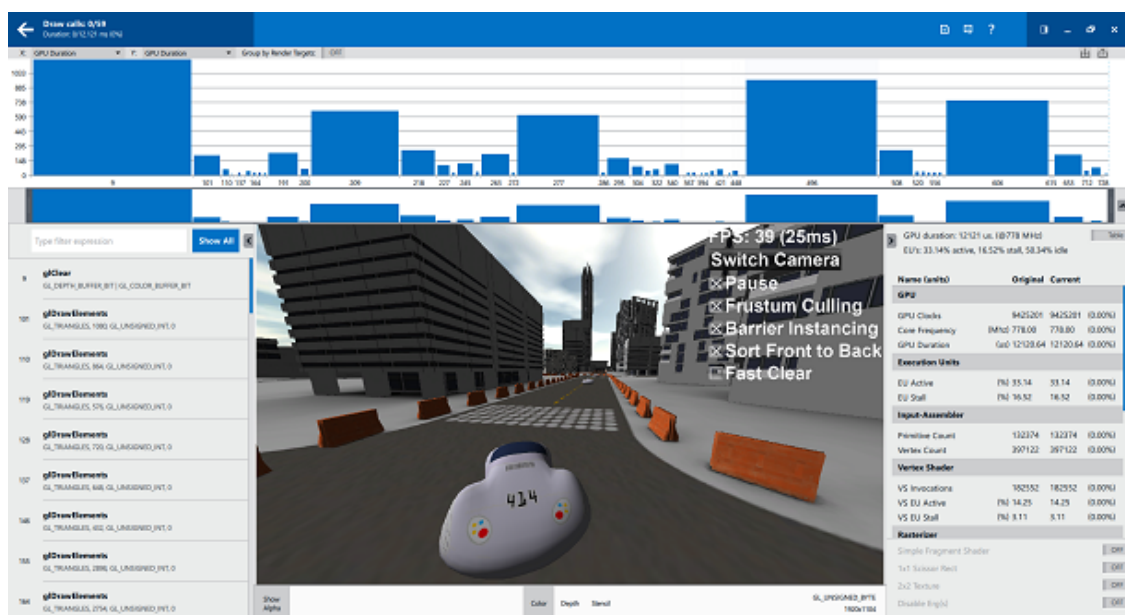
## XI - Optimisation 3 - Tri d'avant en arrière

Le terme d’“overdraw” fait référence au fait d’écrire chaque pixel plusieurs fois ; cela peut impacter le débit de remplissage des pixels et augmenter la durée de rendu des images. Examiner les indicateurs « Samples Written » nous montre que chaque pixel est écrit en moyenne 1,8 fois par image (accessible dans « Resolution / Samples Written »).

Output-Merger			
Alpha Test Fails	27656	27656	(0.00%)
Early Z Failed	2025463	2025463	(0.00%)
Early Z Passed	2868718	2868718	(0.00%)
Samples Written	3848699	3848699	(0.00%)

Trier les dessins d'avant en arrière avant de réaliser le rendu est une façon relativement radicale de réduire l'overdraw parce que le pipeline GPU rejettera tout pixel obstrué par des dessins précédents.

Activez la case à cocher « Sort Front to Back » et utilisez System Analyzer pour capturer une autre image. Une fois l'image capturée, ouvrez-la avec Frame Analyzer.

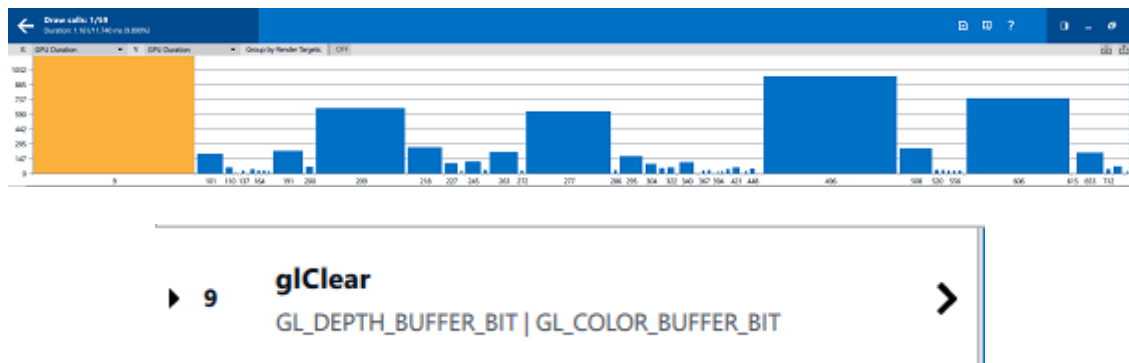


En examinant cette image, nous pouvons constater que l'indicateur « Samples Written » a baissé de 6 % et que notre temps GPU global est réduit de 8 %.

Output-Merger			
Alpha Test Fails	27711	27711	(0.00%)
Early Z Failed	2141885	2141885	(0.00%)
Early Z Passed	2637975	2637975	(0.00%)
Samples Written	3622166	3622166	(0.00%)

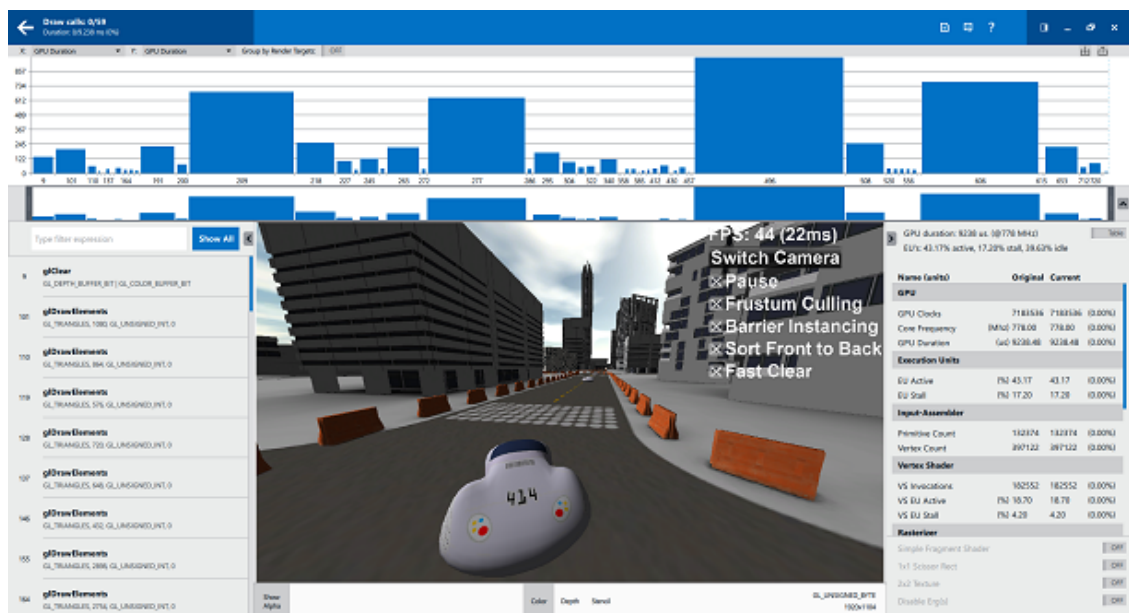
## XII - Optimisation 4 - Nettoyage rapide

Un coup d'œil final à nos temps de dessin montre que le premier erg est celui qui prend le plus de temps GPU individuel. Sélectionner cet erg révèle qu'il ne s'agit pas d'un appel de dessin, mais d'un appel glClear.



Le GPU Intel possède une optimisation matérielle qui réalise un « nettoyage rapide » en un temps bien plus faible qu'un nettoyage traditionnel. Un nettoyage rapide peut être réalisé en réglant glClearColor sur noir absolu ou blanc absolu (0, 0, 0, 0 ou 1, 1, 1, 1).

Activez la case à cocher « Fast Clear » et utilisez System Analyzer pour capturer une autre image. Une fois que l'image est capturée, ouvrez-la avec Frame Analyzer.



En examinant cette image, nous pouvons constater que le temps GPU pour le nettoyage a réduit de 87 % par rapport au nettoyage normal, de 1,2 ms à 0,2 ms.

GPU duration: 1151 us. (@778 MHz)

GPU duration: 145 us. (@778 MHz)

Finalement, le temps global de génération d'image par le GPU a été réduit de 24 % pour passer à 9,2 ms.

GPU duration: 9238 us. (@778 MHz)

## XIII - Conclusion

Ce tutoriel a porté sur une application de jeu représentative à un stade précoce, et utilisé Intel GPA pour analyser le comportement de l'application et réaliser des changements ciblés pour améliorer les performances. Les changements et les améliorations réalisés ont été :

Optimisation	Avant	Après	% amélioré
Élimination des objets invisibles	55,2 ms	45,0 ms	82 %
Instanciation	45,0 ms	13,2 ms	71 %
Tri	13,2 ms	12,1 ms	8 %
Nettoyage rapide	12,1 ms	9,2 ms	24 %
Optimisations globales GPU	55,2 ms	9,2 ms	83 %

Les logiciels et charges de travail utilisés dans les tests de performances peuvent avoir été optimisés pour des performances avec microprocesseurs Intel exclusivement. Les tests de performances, tels que SYSmark et MobileMark, sont mesurés en utilisant des systèmes informatiques, composants, logiciels, opérations et fonctions spécifiques. Tout changement à l'un de ces facteurs peut faire varier les résultats. Vous devriez consulter d'autres informations et tests de performances pour vous aider à évaluer complètement les achats que vous envisagez, dont la performance de ce produit lorsqu'il est combiné à d'autres produits. Pour de plus amples informations, rendez-vous sur <http://www.intel.com/performance>.

Globalement, de l'implémentation initiale de City Racer à la version la plus optimisée, nous avons réussi une amélioration des performances de 300 %, de 11 images par seconde à 44 images par seconde. Dans la mesure où cette implémentation est initialement significativement sous-optimisée, un développeur appliquant ces techniques ne constatera probablement pas le même gain absolu de performances sur un jeu réel.

Quoi qu'il en soit, l'objectif premier de ce tutoriel n'est pas l'optimisation d'une application d'exemple spécifique, mais le gain de performances que vous pouvez potentiellement obtenir en suivant les recommandations contenues dans le Guide Développeur pour Processeur graphique Intel Processor et l'utilité d'Intel GPA pour trouver et mesurer ces améliorations.

Pour davantage de ressources concernant le développement de jeux vidéo, consultez la **Zone des Développeurs Intel** ou rendez-vous sur le **forum dédié**.

Pour découvrir les technologies Intel, c'est par ici :

- **GPA**
- **Outils Intel pour le développement de jeux**
- **Intel INDE**
- **Unity sous Intel**