Résolution de problèmes de synchronisation entre processeur et processeur graphique dans Galactic Civilizations 3

Par Raja Bala

Date de publication : 2 juillet 2015

Galactic Civilizations 3 (GC3) est un jeu de stratégie 4X au tour par tour, développé et édité par **Stardock Entertainment**, et sorti le 14 mai 2015. Durant les périodes des versions de développement et d'accès anticipé, nous avons établi le profil et analysé les performances de rendu du jeu. L'une des grandes améliorations de performance réalisées a été la suppression de plusieurs problèmes de synchronisation entre processeur et processeur graphique qui étaient responsables de la perte d'un certain parallélisme entre le processeur et le processeur graphique. Cet article décrit le problème et sa résolution, et souligne l'importance d'utiliser des outils d'analyse de performance durant le développement, tout en gardant à l'esprit leurs forces et limites.

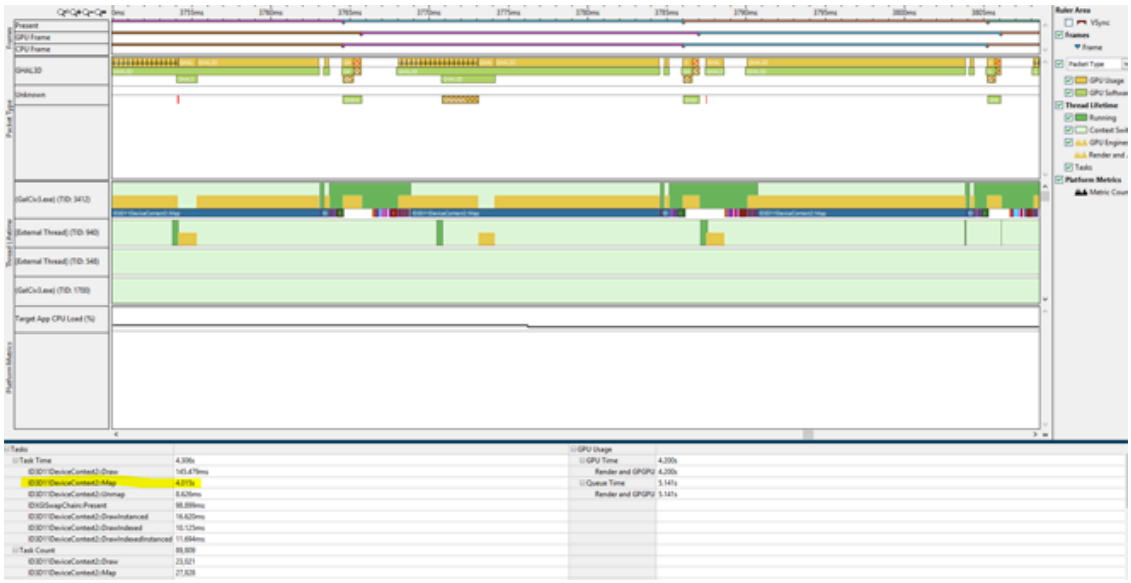
N'hésitez pas à donner votre avis sur cet article sur le forum systèmes : **Commentez**

I - Identification du problème....3
II - Élimination des bogues pour résoudre le problème....4
III - Résolution du problème....5
IV - L'importance des outils de profilage du processeur graphique durant le développement de jeux....7
V - Conclusion....7
VI - Crédits....8
VII - À propos de l'auteur....8
VIII - Traitement des ressources avec Direct3D....8

Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2016 Raja Bala. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.

I - Identification du problème

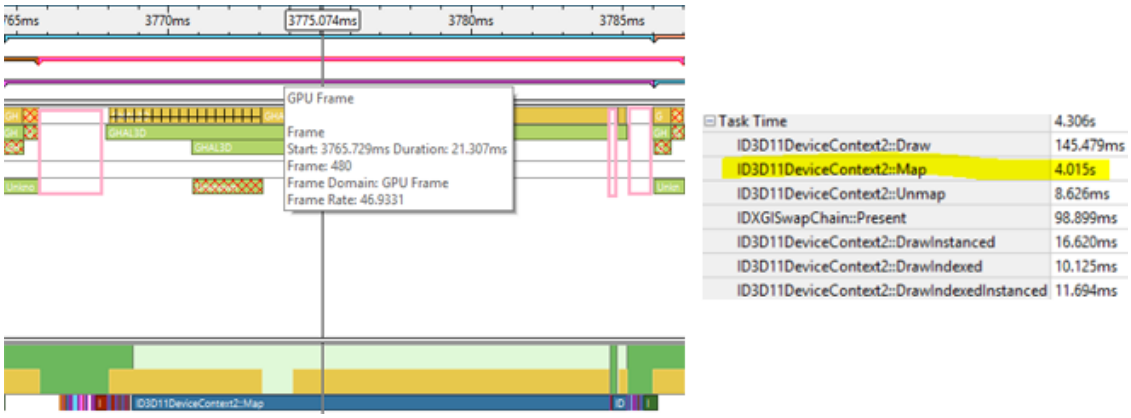
Nous avons commencé l'analyse des performances de rendu avec **Platform Analyzer** dans **Intel® INDE Graphics Performance Analyzers** (GPA). La capture d'écran ci-dessous est une capture de la trace du jeu (sans v-sync) avant que des optimisations soient réalisées. La file d'attente du processeur graphique a plusieurs trous dans la génération d'images ainsi qu'entre celles-ci, avec un rendement inférieur à une image en file d'attente. Si la file d'attente du processeur graphique n'est pas bien alimentée par le processeur et rencontre des trous, l'application ne rattrapera jamais cette période d'inactivité pour améliorer les performances ou la fidélité visuelle.



Avant : Intervalle = environ 21 ms - Moins d'une image en file d'attente - Lacunes dans la file d'attente du processeur graphique - Appel de Map très long

GPA Platform Analyzer montre aussi le temps de chaque appel à l'API Direct3D 11 (par exemple, application -> temps d'exécution -> pilote, et chemin retour). Dans la capture d'écran ci-dessus, vous pouvez voir un appel ID3D11DeviceContext::Map qui prend environ 15 ms à répondre, durant lequel la tâche principale de l'application ne fait rien.

L'image ci-dessous montre un agrandissement de l'évolution d'une image, depuis le début du processeur graphique jusqu'à la fin du processeur. Les erreurs sont affichées dans des cadres roses, à hauteur d'environ 3,5 ms par image. Platform Analyzer nous donne également la durée cumulée des divers appels API pour la trace, avec Map prenant 4,015 secondes du total de 4,306 secondes !



Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2016 Raja Bala. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.

Il est important de remarquer que Frame Analyzer ne peut pas repérer le long appel à Map avec une telle capture d'images. Frame Analyzer utilise des requêtes à l'horloge du processeur graphique pour mesurer le temps d'un **erg**, qui désigne les changements d'état, les ressources de liaison, et le rendu. La commande Map arrive donc au processeur, le processeur graphique n'en étant pas informé.

II - Élimination des bogues pour résoudre le problème

(Reportez-vous à la section des ressources Direct3D à la fin pour une introduction à l'utilisation et à la mise à jour des ressources.)

La recherche de bogues du pilote a révélé que le long appel de Map utilisait la commande DX11_MAP_WRITE_DISCARD (Platform Analyzer ne vous affiche pas les arguments de l'appel de Map) pour mettre à jour un grand tampon de vertex préalablement créé avec le drapeau D3D11_USAGE_DYNAMIC.

Il s'agit d'un scénario très usuel pour les jeux que d'optimiser le flux de données pour des ressources fréquemment mises à jour. En créant la carte d'une ressource dynamique avec DX11_MAP_WRITE_DISCARD, un alias est alloué dans le tas d'alias, et est renvoyé. Un alias fait référence à l'allocation de mémoire pour une ressource à chaque fois qu'elle est générée. À chaque fois qu'il n'y a pas de place pour des alias dans le tas d'alias actuel de la ressource, un nouveau tas d'alias parallèle est alloué. Cela continue jusqu'à ce que la limite de tas de la ressource soit atteinte.

Cela était précisément le problème pour GC3. À chaque fois que cela se produisait (donc lorsque plusieurs images de quelques ressources importantes étaient générées plusieurs fois), le pilote attendait que l'appel draw utilisant un alias de la ressource (qui avait été alloué plus tôt) finisse, pour pouvoir le réutiliser pour la requête actuelle. Il ne s'agissait pas d'un problème spécifique à Intel. Il est aussi apparu avec le pilote NVIDIA et a été vérifié avec **GPUView** pour confirmer ce que nous avions trouvé avec Platform Analyzer.

Le tampon de vertex était d'environ 560 Ko (taille trouvée avec l'aide du pilote) et a été rendu ~ 50 fois avec un rejet pour une image. Le pilote Intel alloue des tas multiples sur demande (chacun faisant 1 Mo) par ressource pour stocker ses alias. Les alias sont alloués dans un tas jusqu'à ce qu'ils ne le puissent plus, après quoi un autre tas d'alias parallèle de 1 Mo est assigné à la ressource et ainsi de suite. Dans le cas de la longue commande Map, seul un alias pouvait rentrer dans un tas ; donc, à chaque fois qu'une commande Map était appelée pour la ressource, un nouveau tas parallèle était créé pour cet alias jusqu'à ce que la limite de tas de la ressource soit atteinte. Cela arrivait à chaque image (ce qui explique pourquoi le même schéma se répète), pour laquelle le pilote attendait une commande draw préalable (pour la même image) soit appelée en utilisant son alias, afin de le réutiliser.

Nous avons étudié le journal de l'API dans Frame Analyzer pour filtrer les ressources qui avaient été générées plusieurs fois. Nous avons trouvé plusieurs cas de ce type, le système d'interface utilisateur étant le coupable principal, générant un tampon de vertex plus de 50 fois. La recherche de bogues dans le pilote a montré que chaque map ne mettait à jour qu'une faible portion du tampon.

Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2016 Raja Bala. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.

Filter:
map

Erg	Id	Function Name	Parameters
102	3065	ID3D11DEVICECONTEXT_MAP	2441, 0, D3D11_MAP_WRITE_DISCARD,
102	3067	ID3D11DEVICECONTEXT2_UNMAP	2441, 0
Unnamed Region 10, Erg 103			
103	3088	ID3D11DEVICECONTEXT_MAP	2322, 0, D3D11_MAP_WRITE_DISCARD,
103	3090	ID3D11DEVICECONTEXT2_UNMAP	2322, 0
Unnamed Region 10, Erg 104			
104	3111	ID3D11DEVICECONTEXT_MAP	2441, 0, D3D11_MAP_WRITE_DISCARD,
104	3113	ID3D11DEVICECONTEXT2_UNMAP	2441, 0
Unnamed Region 10, Erg 105			
105	3134	ID3D11DEVICECONTEXT_MAP	2322, 0, D3D11_MAP_WRITE_DISCARD,
105	3136	ID3D11DEVICECONTEXT2_UNMAP	2322, 0
Unnamed Region 10, Erg 106			
106	3157	ID3D11DEVICECONTEXT_MAP	2322, 0, D3D11_MAP_WRITE_DISCARD,
106	3159	ID3D11DEVICECONTEXT2_UNMAP	2322, 0
Unnamed Region 10, Erg 111			
111	3315	ID3D11DEVICECONTEXT_MAP	2322, 0, D3D11_MAP_WRITE_DISCARD,
111	3317	ID3D11DEVICECONTEXT2_UNMAP	2322, 0
Unnamed Region 10, Erg 112			
112	3338	ID3D11DEVICECONTEXT_MAP	2441, 0, D3D11_MAP_WRITE_DISCARD,
112	3340	ID3D11DEVICECONTEXT2_UNMAP	2441, 0
Unnamed Region 10, Erg 113			
113	3361	ID3D11DEVICECONTEXT_MAP	2322, 0, D3D11_MAP_WRITE_DISCARD,
113	3363	ID3D11DEVICECONTEXT2_UNMAP	2322, 0
Unnamed Region 10, Erg 114			
114	3386	ID3D11DEVICECONTEXT_MAP	2322, 0, D3D11_MAP_WRITE_DISCARD,
114	3388	ID3D11DEVICECONTEXT2_UNMAP	2322, 0
Unnamed Region 10, Erg 115			
115	3409	ID3D11DEVICECONTEXT_MAP	2322, 0, D3D11_MAP_WRITE_DISCARD,
115	3411	ID3D11DEVICECONTEXT2_UNMAP	2322, 0
Unnamed Region 10, Erg 116			
116	3435	ID3D11DEVICECONTEXT_MAP	2322, 0, D3D11_MAP_WRITE_DISCARD,
116	3437	ID3D11DEVICECONTEXT2_UNMAP	2322, 0
Unnamed Region 10, Erg 117			
117	3458	ID3D11DEVICECONTEXT_MAP	2322, 0, D3D11_MAP_WRITE_DISCARD,
117	3460	ID3D11DEVICECONTEXT2_UNMAP	2322, 0
Unnamed Region 10, Erg 118			
118	3482	ID3D11DEVICECONTEXT_MAP	2322, 0, D3D11_MAP_WRITE_DISCARD,
118	3484	ID3D11DEVICECONTEXT2_UNMAP	2322, 0
Unnamed Region 10, Erg 119			
119	3505	ID3D11DEVICECONTEXT_MAP	2322, 0, D3D11_MAP_WRITE_DISCARD,
119	3507	ID3D11DEVICECONTEXT2_UNMAP	2322, 0
Unnamed Region 10, Erg 120			
120	3528	ID3D11DEVICECONTEXT_MAP	2441, 0, D3D11_MAP_WRITE_DISCARD,
120	3530	ID3D11DEVICECONTEXT2_UNMAP	2441, 0
Unnamed Region 10, Erg 121			
121	3551	ID3D11DEVICECONTEXT_MAP	2322, 0, D3D11_MAP_WRITE_DISCARD,
121	3553	ID3D11DEVICECONTEXT2_UNMAP	2322, 0
Unnamed Region 10, Erg 122			
122	3574	ID3D11DEVICECONTEXT_MAP	2322, 0, D3D11_MAP_WRITE_DISCARD,
122	3576	ID3D11DEVICECONTEXT2_UNMAP	2322, 0
Unnamed Region 10, Erg 123			
123	3599	ID3D11DEVICECONTEXT_MAP	2322, 0, D3D11_MAP_WRITE_DISCARD,
123	3601	ID3D11DEVICECONTEXT2_UNMAP	2322, 0

Même ressource (handle 2322) générée plusieurs fois dans une image

III - Résolution du problème

Chez Stardock, nous avons **orchestré tous les systèmes de rendu afin d'obtenir des marqueurs supplémentaires dans la vue chronologique de Platform Analyzer**, en partie pour vérifier que l'interface utilisateur du système était derrière cette commande lourde et pour les profilages futurs.

Nous disposions de plusieurs possibilités pour résoudre le problème :

régler le drapeau Map sur D3D11_MAP_WRITE_NO_OVERWRITE plutôt que D3D11_MAP_WRITE_DISCARD :

- le grand tampon de vertex était partagé par plusieurs entités. Par exemple, la plupart des éléments de l'interface utilisateur à l'écran partageaient un grand tampon. Chaque appel de commande Map mettait seulement à jour une petite portion indépendante du tampon. Les vaisseaux et les astéroïdes qui utilisaient des instances partageaient également un grand vertex/tampon d'instance de données. D3D11_MAP_WRITE_NO_OVERWRITE serait ici le choix idéal puisque l'application garantit qu'elle ne réécrira pas de régions du tampon qui pourraient être utilisées par le processeur graphique ;

diviser le grand tampon de vertex en plusieurs tampons plus petits :

- puisque l'allocation d'alias était la raison derrière le ralentissement, réduire considérablement la taille du tampon de vertex permet à plusieurs alias de rentrer dans un tas. GC3 ne soumet pas trop d'appels draw, et ainsi, réduire la taille de 10 ou 100 fois (560 Ko à 5-50 Ko) résoudrait ce problème ;

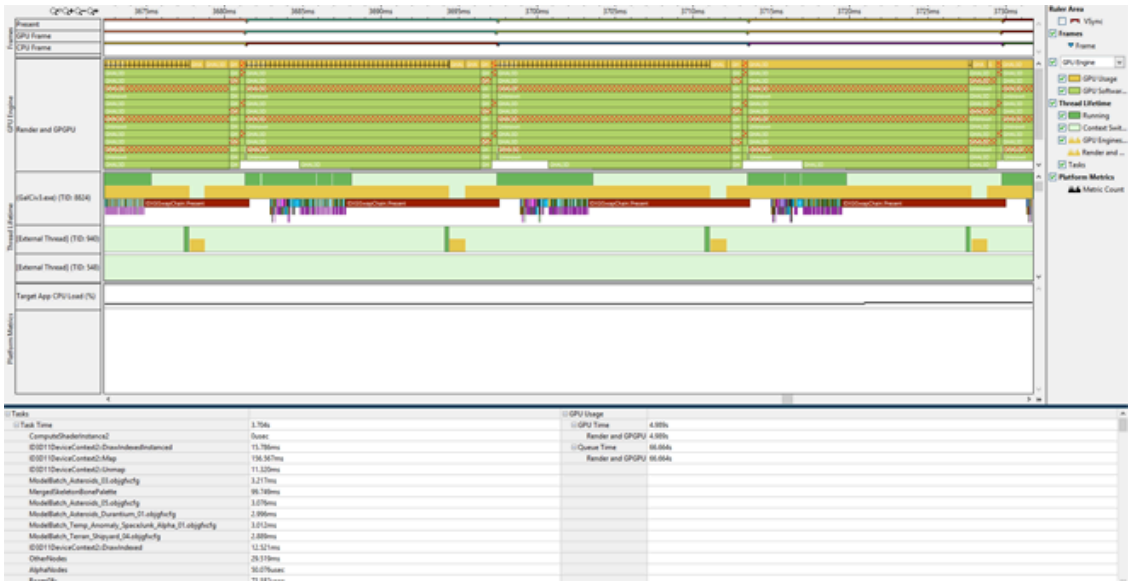
Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2016 Raja Bala. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.

utiliser le drapeau D3D11_MAP_FLAG_DO_NOT_WAIT :

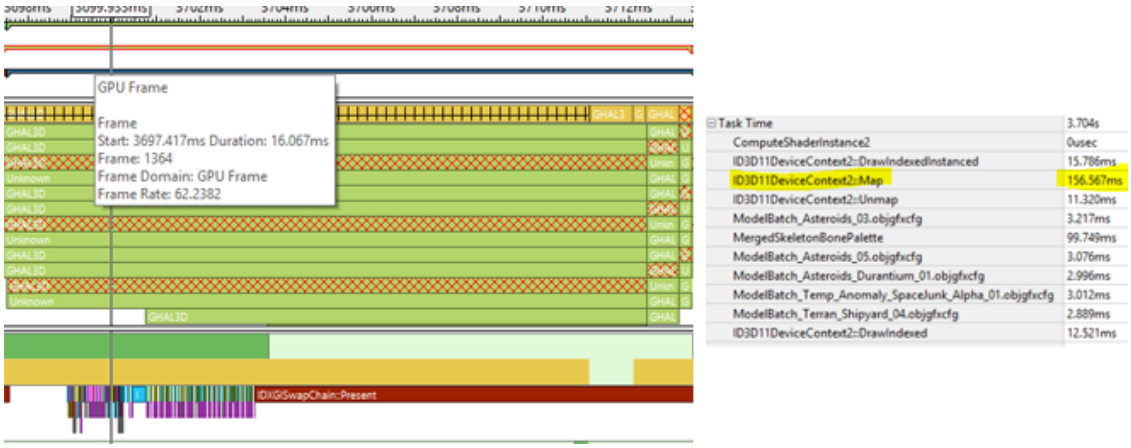
- vous pouvez utiliser ce drapeau pour détecter quand le processeur graphique est occupé à utiliser des ressources et réaliser un autre travail avant de générer la ressource de nouveau. Bien que cela laisse le processeur faire un travail utile, cela serait une mauvaise résolution du problème dans notre cas.

Nous avons utilisé la deuxième option et changé la constante utilisée dans la logique de création du tampon. Les tailles de tampon de vertex étaient codées en dur pour chaque sous-système, et nécessitaient simplement d'être abaissées. Plusieurs alias pouvaient alors prendre place dans chaque tas de 1 Mo, et avec le nombre d'appels draws, faible dans GC3, le problème n'allait pas ressurgir.

Chaque résolution de sous-système de rendu renvoyait le problème sur un autre, nous l'avons donc résolu pour tous les sous-systèmes de rendu. Une capture de la trace avec les résolutions de problèmes et les instrumentations, suivie par un aperçu détaillé d'une image, est affichée ci-dessous :



Après : Intervalle = environ 16 ms - 3 images en file d'attente - Pas de trou dans la file d'attente du processeur graphique - Pas de grands appels de Map.



Le temps total pris par Map est passé de 4 secondes à 157 millisecondes !

Les trous dans la file d'attente du processeur graphique ont disparu.

Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2016 Raja Bala. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.

Le jeu dispose de 3 images en file d'attente à tout moment et attendait que le processeur graphique finisse les images pour soumettre la suivante ! Le processeur graphique était toujours occupé après quelques changements simples.

Les performances ont augmenté d'environ 24 % avec chaque image prenant environ 16 ms au lieu d'environ 21 ms.

IV - L'importance des outils de profilage du processeur graphique durant le développement de jeux

Voici ce que Stardock avait à dire à ce sujet.

Sans des outils tels que GPA Platform Analyzer ou GPUView, nous n'aurions pas su ce qu'il se passait avec le processeur graphique parce que les informations qui sont retournées par DirectX ne concernent que la réussite ou non de la commande. Traditionnellement, nous aurions désactivé des systèmes ou des parties des systèmes, pour essayer d'isoler la source de la chute de performances. C'est un processus très chronophage, qui peut rapidement coûter des heures ou des journées entières sans aucun bénéfice tangible, surtout dans le cas où les goulots d'étranglement ne sont pas dans les systèmes que vous suspectiez.

Aussi, mesurer des systèmes isolés peut régulièrement amener à rater des problèmes qui nécessitent que de multiples systèmes interagissent pour causer le problème. Par exemple, si vous avez un goulot d'étranglement dans le système d'animation, vous pourriez ne pas l'identifier si vous avez suffisamment d'autres systèmes désactivés de façon à ce que le système d'animation (qui cause votre problème de performance) ait suffisamment de ressources pour tourner sans encombre. Dans ce cas, vous perdez du temps à dépanner le mauvais système, celui que vous avez retiré, à la place de la véritable source du problème.

Nous avons également essayé d'intégrer des outils de profilage dans nos jeux. Même si cela fonctionne, nous n'obtenons que des données de mesure concernant les systèmes que nous mesurons explicitement, nous empêchant à nouveau d'observer les problèmes dans les systèmes dont nous ne nous méfions pas. C'est aussi un travail considérable à implémenter, et à maintenir durant le développement des jeux pour être utilisable. Et nous devons recommencer depuis le début à chaque jeu que nous faisons. Nous obtenons donc des informations partielles pour un coût de développement élevé. À cause de cela, les problèmes peuvent être difficiles à trouver en étudiant simplement le code, même en pas-à-pas, parce qu'il peut sembler correct et avoir un rendu satisfaisant, mais, en réalité, provoquer des attentes ou du travail inutile du processeur graphique.

C'est ce pour quoi il est important de comprendre ce qu'il se passe dans le processeur graphique. Les outils de profilage du processeur graphique sont essentiels pour montrer rapidement aux développeurs où leur code fait attendre le processeur graphique ou là où l'image nécessite le plus de temps. Les développeurs peuvent identifier les zones de leur code qui bénéficieraient le plus d'optimisations, afin qu'ils puissent se concentrer sur des améliorations qui augmenteront les performances de façon significative.

V - Conclusion

Optimiser les performances de rendu d'un jeu est une tâche complexe. Les outils de capture de trace et d'image procurent des aperçus différents et importants des performances d'un jeu. Cet article s'est concentré sur les problèmes de synchronisation entre processeur et processeur graphique qui requièrent un outil de visualisation de trace tels que GPA Platform Analyzer ou GPUView pour l'identifier.

Tout le support au développement Windows est disponible sur la **Zone des Développeurs Intel** et sur le **forum**.

Découvrez aussi les nouvelles **technologies Intel** pour Windows.

Ressources:

- **Ressources Intel pour Windows**

Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2016 Raja Bala. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.

- **Forum UltraBook**
- **Intel INDE**

VI - Crédits

Nous remercions Derek Paxton (vice-président) et Jesse Brindle (développeur graphique en chef) de Stardock Entertainment pour ce formidable partenariat et l'incorporation de ces changements dans Galactic Civilizations 3.

Nous remercions particulièrement Robert Blake Taylor pour le débogage du pilote, Roman Borisov et Jeffrey Freeman pour leur aide concernant GPA, et Axel Mamode et Jeff Laflam d'Intel pour leur relecture de cet article.

VII - À propos de l'auteur

Raja Bala est ingénieur d'application pour le groupe de relations avec les développeurs de jeux d'Intel. Il aime disséquer le processus de rendu des jeux et trouver des façons de l'améliorer, et est un grand fan de Dota2 et de Valve.

VIII - Traitement des ressources avec Direct3D

L'API Direct3D peut être utilisée pour créer/détruire des ressources, fixer l'état du pipeline de rendu, lier des ressources au pipeline, et mettre à jour certaines ressources. La plupart des créations de ressources ont lieu durant le chargement de la scène ou du niveau.

Une image typique de jeu consiste à lier différentes ressources avec le pipeline, à fixer l'état du pipeline, à mettre à jour les ressources avec le processeur (tampon constant, tampon de vertex/index…) basé sur un état de simulation, et de mettre à jour les ressources avec le processeur graphique (cibles de rendu, uavs…) avec des appels draw, dispatch et clear.

Durant la création de ressources, l'enum **D3D11_USAGE** (1-08) est utilisé pour marquer la ressource comme nécessitant :

(a) un accès GPU en lecture-écriture (DÉFAUT - pour les cibles de rendus, uavs, tampons constants rarement mis à jour) ;

(b) un accès GPU en lecture seule (IMMUABLE - pour les textures) ;

(c) un accès CPU en écriture + un accès GPU en lecture (DYNAMIQUE - pour les tampons qui doivent fréquemment être mis à jour) ;

(d) un accès CPU, mais permettant au GPU de copier des données dessus (MONTAGE).

Veuillez noter que la ressource **D3D11_CPU_ACCESS_FLAGc & d** (1-09) nécessite d'être également réglée correctement pour respecter l'utilisation de c & d.

En matière de mise à jour des données d'une ressource, l'API Direct3D 11 API procure trois options, chacune d'entre elles étant utilisée pour une utilisation particulière (comme décrit plus tôt) :

- **Map/ Unmap**
- **UpdateSubresource**
- **CopyResource** / **CopySubresourceRegion**

Un scénario intéressant, dans lequel une synchronisation implicite est requise, est lorsque le processeur a un accès en écriture et le processeur graphique un accès en lecture à la ressource. Ce scénario survient généralement

Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2016 Raja Bala. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.

durant une image. Mettre à jour la matrice de vue/modèle/projection (stockée dans un tampon constant) et les transformations d'un squelette (animé) d'un modèle en sont des exemples. Attendre que le processeur graphique ait terminé d'utiliser la ressource serait trop coûteux. Créer plusieurs ressources indépendantes (copies de ressource) pour gérer cela serait fastidieux pour le programmeur de l'application. Par conséquent, Direct3D (9 à 11) remet cela au pilote avec le drapeau de Map DX11_MAP_WRITE_DISCARD. À chaque fois que la ressource est mappée avec ce drapeau, le pilote crée une nouvelle région de mémoire pour la ressource et laisse le processeur la mettre à jour à la place. Ainsi, des appels draw qui mettent à jour la ressource finissent par travailler sur différents alias de la ressource, ce qui, bien sûr est gourmand en mémoire du processeur graphique.

Pour plus d'information concernant la gestion de ressources avec Direct3D, consultez :

- **La présentation de John McDonald à GDC, « Gestion efficiente du tampon »**
- **Direct3D 11 Introduction aux ressources**
- **Direct3D 10 Choisir une ressource**
- **UpdateSubresource vs Map**

Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright ® 2016 Raja Bala. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.