Friday Facts #281 - Pour quelques images de plus

posila

2019-02-08

Pour quelques images de plus

Précédemment dans Factorio Friday Facts (#264) : *"Sans surprise, les scènes chargées en fumée ou en arbres peuvent accumuler des FPS, surtout en 4K. Peut être devrions nous faire quelque chose ..."*

Parfois, c'est juste un bug

Écrire des FFF me permet de résumer ce que je sais, d'examiner les problèmes sous un angle différent et d'essayer parfois de répondre à des questions que je n'avais même pas pensé à poser auparavant.

Par exemple, j'ai créé la visualisation de surimpression pour la capture d'écran de FFF-264 et je ne les ai pas analysées lors de l'écriture de la FFF. Juste après la publication du billet, j'ai jeté un coup d'œil et commencé à réfléchir : pourquoi la fumée crée-t-elle un tel blob alors qu'elle est à peine perceptible dans la vue du jeu ? Eh bien, il s'est avéré que c'était en grande partie à cause d'un bug. Il y a quelque temps, nous avons optimisé les particules de fumée afin qu'elles ne soient mises à jour qu'une fois toutes les 120 ticks (2 secondes) et que son animation (mouvement, échelle et opacité) soit interpolée pendant le dessin.

Le problème est que les particules ne sont détruites que lors de leur mise à jour. Si la durée de vie d'une particule se terminait quelque part au milieu de la fenêtre des 120 ticks, la particule serait toujours dessinée jusqu'à sa destruction. Parce que la fumée s'évapore et s'intensifie au cours de sa vie, elle se dessine complètement transparente sur une grande surface. Faire en sorte que la particule de fumée ne se dessine pas au-delà de sa durée de vie diminue de 15 % le nombre de particules rasterisées et réduit davantage le nombre de pixels en cours de tramage. De plus, les particules dont l'opacité est inférieure à 2% ne semblent pas vraiment ajouter d'effet à l'image finale. Nous ne pouvons donc pas en afficher, sans risque d'obtenir celles-ci, afin d'obtenir un coup de pouce supplémentaire.

Optimisation de la visualisation de la portée des tourelle

Il est probable que seul un très petit nombre de nos joueurs soit confronté à ce problème en mode de jeu normal, mais c'est un problème simple, à savoir que nous en apprenions plus sur la façon d'utiliser plus efficacement les GPU.

Les plages de tourelles de la vue Carte sont rendues sous forme de géométrie opaque (sans sprites) dans un tampon hors écran qui est ensuite dessiné de manière semi-transparente dans la vue du jeu. Cela fait que les portées de tourelles se fondent dans une seule forme solide de la même couleur. Chaque pixel est écrit pour chaque tourelle dans la plage, mais si il est écrit une fois, les écritures ultérieures sont inutiles.

Nous avions deux idées pour optimiser cela. Tout d'abord, activez le test du pochoir pour que les pixels ne puissent être écrits qu'une seule fois. Dans le second cas, passez une liste de tourelles au GPU et testez si chaque pixel est à portée d'une tourelle utilisant un pixel shader. L'idée du test du pochoir a donné environ 3 fois plus de rapidité dans nos cas de test extrêmes (tourelles 20x20), ce qui n'était pas suffisant - si votre GPU avait un problème avec la grille 3x3, il aurait de nouveau un problème avec la grille 9x3, qui n'est pas si absurde configuration pour avoir. L'idée de pixel shader s'est avérée être mixte : si tout l'écran était dans la portée, le shader serait ultra-rapide, mais dès qu'il y aurait des pixels hors de portée, le shader devait parcourir la liste complète des tourelles pour figurer si aucun ne le couvre, les performances ont commencé à baisser rapidement. Dans le pire des cas, ce serait pire que sans optimisation.

Jiri a eu l'idée de faire un prepass, dans lequel nous rendrions la géométrie dans un tampon beaucoup plus petit (disons 16x plus petite en largeur et en hauteur), puis dans le shader range de la tourelle, nous vérifierions si un pixel est vraiment dans la portée (pixel dans le tampon prepass est totalement opaque), définitivement hors de portée (le pixel dans le tampon prepass est totalement transparent) ou nous ne le savons pas (le pixel dans le tampon prepass est semi-transparent en raison du filtrage linéaire) et nous avons besoin du pixel contre la tourelle liste. Il a fait cela et la performance s'est légèrement améliorée, mais pas autant que nous l'espérions. Après plus d' investigation nous avons découvert que le processeur graphique n'aime vraiment pas la sortie anticipée dans le pixel shader. Jiri a réussi à l'enlever en rendant d'abord tous les cas déterminés, tout en marquant les pixels incertains dans la mémoire tampon du gabarit et, dans une autre passe, il exécutait le shader range de tourelle uniquement sur les pixels au pochoir. Cette solution finissait par être 20 fois plus rapide dans des cas trop lents avec la solution non optimisée, mais si vous réduisiez le zoom et que plus de tourelles couvraient moins de pixels à l'écran, la solution d'origine devenait meilleure. Nous n'activons donc l'optimisation que lorsque vous effectuez un zoom suffisant.

À propos, les portées d'artillerie ont connu le même problème dans les premières versions de 0,16, mais elles sont si grandes qu'il nous a suffi de tester si tout l'écran est dans la portée et de ne dessiner qu'un seul rectangle plein écran dans ce cas. Les tourelles risquaient fort de causer des problèmes sur les GPU bas de gamme, même si elles ne couvraient pas tout l'écran.

GPU performances

Comme toujours, la racine de toute dégradation des performances est l'accès à la mémoire. Comme nous ne faisons que du simple dessin d'image-objet, le GPU doit lire les pixels d'une texture et les mélanger dans un framebuffer. Ce qui, techniquement, signifie lire des pixels à partir d'une autre texture, faire des calculs simples avec les deux valeurs et écrire le résultat. Les GPU sont conçus pour ce faire sur une échelle massivement parallèle et sont optimisés pour une bande passante mémoire élevée tout en sacrifiant la latence de la mémoire. L'hypothèse est que vous voudrez faire au moins un peu de maths sur les couleurs extraites des textures, afin de donner à l'image résultante des détails dynamiques. Chaque cœur de GPU peut alors avoir de nombreuses tâches planifiées et basculer entre elles lorsque la tâche en cours commence à attendre qu'une opération de mémoire soit terminée.

Lorsque vous ne faites pas vraiment de calculs pour ajouter des détails dynamiques, et que tous les détails proviennent du dessin de plusieurs couches d'images-objets, les cœurs du processeur graphique atteignent rapidement leur limite de tâches maximales et chacun d'eux est bloqué par un accès à la mémoire. Cela ne veut pas dire que nous n'atteignons pas les limites de bande passante mémoire sur certains matériels. Dans les cas où nous atteignons les limites de bande passante mémoire, nous avons ajouté une option permettant d'effectuer le rendu avec une profondeur de couleur de 16 bits (par opposition à la version 32 bits normale). Cette option est destinée aux anciens GPU et aux GPU intégrés.

Un moyen évident d'améliorer cela consiste à ombrer moins de pixels. C'est quelque chose que j'ai mentionné auparavant dans FFF-227 en séparant les ombres des arbres des troncs pour supprimer de grandes zones de pixels complètement transparents. Cela peut être encore amélioré en dessinant les images-objets non pas sous forme de rectangles, mais sous forme de polygones génériques qui enveloppent les images-objets de sorte que

la plupart des zones entièrement transparentes ne soient pas rasterisées. C'est quelque chose que nous ferons probablement pour les sprites les plus problématiques (arbres et décoratifs).

Un autre moyen de réduire le nombre de pixels ombrés consiste simplement à effectuer un rendu avec une résolution inférieure. Cela fait déjà longtemps que nous faisons cela pour les lumières, mais cela pourrait être utilisé pour d'autres effets qui n'ont pas de détail haute fréquence important - par exemple la fumée. En fin de compte, certains GPU ne sont pas conçus pour rendre le jeu en résolution FullHD, quoi qu'il en soit (par exemple, les cartes Intel HD Graphics 2500 ou les cartes multimédias telles que GeForce GT 710 et Radeon HD 6450), alors ils bénéficieraient d'une option de rendu des images. affichage du jeu en résolution inférieure avec incrustation de l'interface graphique à résolution native.

Dans FFF-227, j'ai également mentionné les mipmaps, des copies à échelle réduite des textures utilisées lors de la réduction d'une texture. Cela aide à mieux utiliser les caches sur le processeur graphique et à réduire ainsi le besoin d'accéder à la VRAM principale. Nous utilisions déjà des mipmaps pour les arbres et les objets de décoration en 0.16, mais paradoxalement, certaines personnes ont eu des problèmes de performances lorsqu'elles ont activé les mipmaps. Le problème est qu'en 0.16, nous utilisons toujours le filtrage trilinéaire pour les textures mappées. Cela signifie que lorsque vous voulez dessiner en tant qu'image-objet à l'échelle de 75 %, le GPU obtiendra un pixel du mipmap à l'échelle 100 % et un mipmap à l'échelle de 50 % et les calculera en moyenne pour obtenir le pixel de la version à l'échelle de 75 %. . L'accès à deux niveaux différents de mappage rendrait les choses plus lentes. Dans le nouveau code de rendu, nous sommes en mesure de contrôler cela. Ainsi, pour les sprites susceptibles de poser des problèmes de performances (par exemple, fumée), nous pouvons simplement extraire des pixels de la mipmap détaillée la plus proche.

Compression de texture

Les GPU prennent en charge de manière native les formats compressés par bloc - la texture est divisée en 4x4 pixels (ou éventuellement en différentes tailles dans des formats qui ne sont généralement pas pris en charge par les GPU de bureau) et chaque bloc est compressé en un nombre d'octets fixe. Les formats couramment pris en charge sur les GPU de bureau sont : BC1-7 . Les plus intéressants pour nous sont :

- BC1 (anciennement DXT1) - Destiné à être RVB sans alpha (ou alpha sur 1 bit) avec une utilisation de 4 bits par pixel (par opposition aux 32 bits bruts utilisés par RGBA).
- BC3 (anciennement DXT5) - Utilise 8 bits par pixel - le même format de couleur que BC1, mais utilise 4 bits supplémentaires pour le canal alpha.
- BC4 - Ne stocke qu'un seul canal de couleur dans le même format que le canal alpha de BC3 (donc 4 bits par pixel).

Ces formats fonctionnent assez bien avec les textures normales, mais pas aussi bien avec les images 2D qui contiennent beaucoup de petits détails. Dans notre art, ce n'est pas si grave tant que les sprites sont statiques, mais dès que nous appliquons la compression sur les animations, même de minuscules modifications de pixels individuels entraînent des modifications plus importantes dans les blocs qui les contiennent, et l'animation résultante a un aspect très bruyant.

Loading Video

Loading Video

Non compressé vs. BC3 compression

Les développeurs d'Awesomenouts ont décrit cela dans leur blogpost sur le fonctionnement des formats compressés pris en charge par les GPU, sur le type d'artefacts qu'il crée dans leur art et sur ce qu'ils font pour améliorer la qualité. Ils stockent leurs images-objets dans une résolution légèrement supérieure (par exemple, une largeur et une hauteur supérieures de 41 %) afin d'écarter davantage le détail de fréquence élevée. Cela rend la compression moins efficace mais en vaut toujours la peine. Le problème, c'est que nous ne pouvons pas vraiment faire cela car cela rendrait nos sprites plutôt mauvais et flous.

Lors de nos premiers essais de compression, nous avons jugé la qualité de la compression trop faible. De plus, cela a quelque peu ralenti le démarrage du jeu, car nous avons d'abord créé des atlas de sprites dans un format non compressé avant de les compresser à la fin du processus de chargement des sprites. J'ai laissé l'option 'Compression de texture' dans, pour les personnes qui en avaient vraiment besoin, et elle compresserait uniquement les images-objets qui se fondent généralement par-dessus d'autres graphiques (comme les masques de couleur, les ombres et la fumée).

Les formats de compression d'image ou de vidéo les plus couramment utilisés (tels que JPEG ou H.264) exploitent le fait que la vision humaine est en fait assez mauvaise pour reconnaître les couleurs et est beaucoup plus sensible aux changements de luminosité. Au lieu d'un espace colorimétrique RVB, ils utilisent un espace colorimétrique basé sur la luma (luminosité) et la chrominance (couleur). Ils appliquent une compression de meilleure qualité au composant luma et une qualité inférieure aux composants colorés, ce qui permet d'obtenir une image de très haute qualité pour l'œil humain. Certaines personnes intelligentes pensaient que cette technique pourrait être utilisée pour améliorer la qualité des formats de compression de bloc GPU.

Une des idées est YCoCg-DXT compression qui utilise BC3 comme format sous-jacent et stocke la luma dans le canal alpha pour une compression et une chrominance de meilleure qualité dans les canaux RGB. Les pixel shaders qui utilisent ces textures doivent faire un peu de calcul pour convertir les couleurs de l'espace colorimétrique YCoCg en RVB. Nous avons essayé d'intégrer cela (en utilisant une texture compressée distincte BC4 pour le canal alpha) et avons été agréablement surpris par le résultat. Vous ne remarquerez probablement pas d'artefacts à moins de zoomer et de les rechercher. En fait, il y a deux semaines, j'ai activé la compression de texture par défaut (et je n'en ai parlé à personne) et chaque fois que je demande à un membre de l'équipe s'il l'a désactivée, il dit qu'il ne savait pas qu'elle était activée. Donc je suis plutôt content de ça. Le petit inconvénient est la nécessité d'utiliser deux textures (BC3 + BC4), ce qui donne 12 bits par pixel, mais le mieux est que, malgré le pixel shader devant extraire de 2 textures au lieu d'une, le GPU est capable de restituer deux fois plus vite, car les caches peuvent contenir plus de pixels dans les formats compressés que dans les formats bruts.

Heureusement, le papier contient le code pixel shader permettant de compresser ce format sur un GPU. Il nous a donc fallu adapter notre code de chargement des images-objets pour utiliser efficacement le GPU afin de compresser les images-objets au fur et à mesure de leur chargement. ne ralentit pas le chargement du jeu.

En 0.17, le paramètre graphique de compression de texture devient une liste déroulante contenant «Aucune», «Haute qualité» et «Basse qualité» :

- Haute qualité utilisera la compression personnalisée YCoCg-DXT et sera la valeur par défaut sur la plupart des ordinateurs.
- Basse qualité utilisera BC3 et est destiné à être utilisé uniquement sur des GPU vraiment faibles.

Il ne devrait y avoir aucune raison pour que vous désactiviez la compression, vous avez le choix principalement au cas où un problème technique surviendrait. La compression est appliquée à toutes les images-objets, à l'exception de l'interface graphique, qui doit être aussi nette que possible. De plus, quelle que soit l'option de compression de texture, les images-objets d'ombre seront toujours compressées au format BC4.

Loading Video

Loading Video

Loading Video

Non compressé vs. compression Haute qualité vs. compression Basse qualité

Après l'ajout des vers, des biters et des spitters haute résolution, l'utilisation de la VRAM a augmenté jusqu'à 3,5 Go (avec la haute résolution activée, bien sûr) sans compression (même celle de l'ombre). En ne compressant que les ombres, l'utilisation de la VRAM a été réduite à 0,16, soit environ 2,5 Go. Avec la compression de haute qualité activée, l'utilisation de la VRAM des atlas de sprites est actuellement d'environ 1 Go (sans mipmaps). Cela signifie que la vanille devrait être parfaitement lisible sur les GPU haute résolution équipés de VRAM de 2 Go. En combinaison avec la diffusion en continu des textures, ces GPU devraient pouvoir suivre le rythme élevé, même dans les cas où les mods ajoutent de nombreux nouveaux sprites. Les sprites haute résolution étaient à l'origine destinés aux joueurs dotés des ordinateurs les plus puissants, mais en 0.17, ils deviendront essentiellement un nouveau standard. L'objectif est de supprimer à terme les options de résolution des images-objets «très basses» et «très basses», car la compression de texture «de basse qualité» sur les images-objets normales et le streaming de texture devraient pouvoir s'exécuter même sur des GPU avec des tailles de VRAM très faibles.

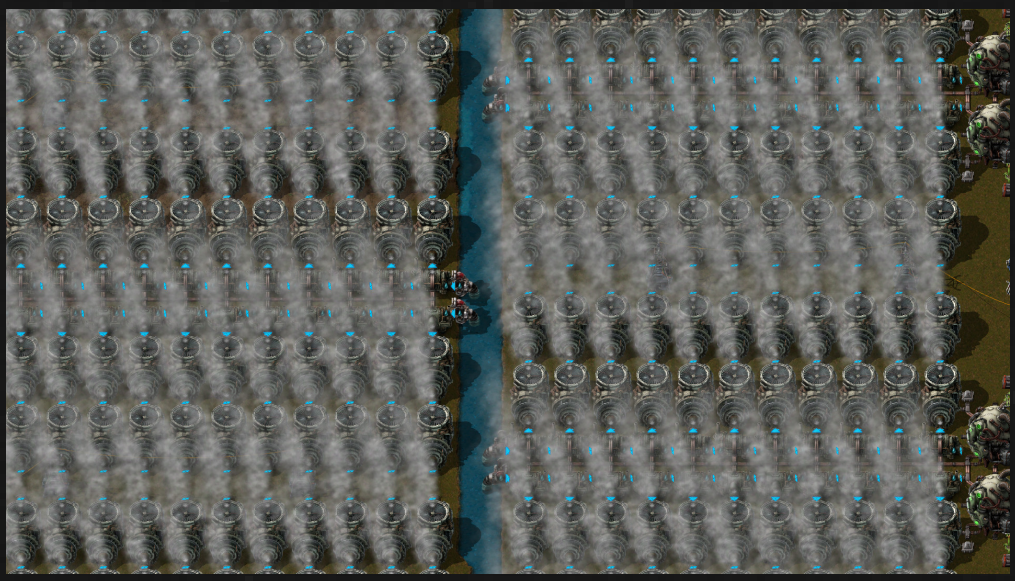
Note latérale : j'ai mentionné qu'il y a 7 formats en Colombie-Britannique. BC7 est destiné aux textures RVB ou RGBA et présente une qualité potentiellement bien meilleure que celle de BC3 avec le même taux de compression. Le problème est que le format a été introduit avec DirectX 11, de sorte qu'il n'est pas pris en charge par le matériel de classe DirectX 10 et qu'il n'est pas disponible sous OpenGL sous macOS. Le deuxième problème est qu'il faut beaucoup de temps pour compresser quelque chose dans ce format, car le compresseur doit essayer un grand nombre de configurations pour chaque bloc afin de trouver celui qui offre la meilleure qualité. Comme Factorio est un peu aberrant dans la mesure où son art est distribué sous forme de fichiers PNG au lieu d'être parfaitement emballé de manière à ce que les ressources graphiques puissent être chargées directement sur le GPU, sans conversion préalable dans un format différent ni création dynamique d'atlas sprite, nous avons besoin de une solution de compression en temps réel. Nous ne souhaitons pas trop changer la façon dont le jeu est distribué, car le fait de tout disposer dans des fichiers séparés rend les mises à jour assez petites lorsque nous en modifions certaines, et le fait que les données à la vanille soient complètement ouvertes facilite la modification du jeu.

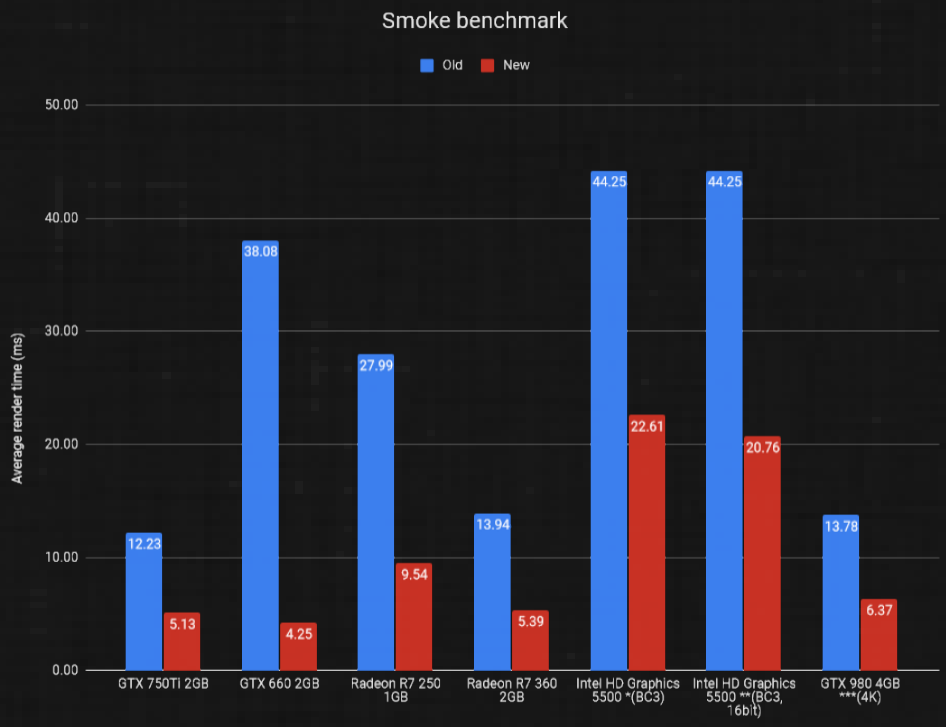
Observez ces graphes !

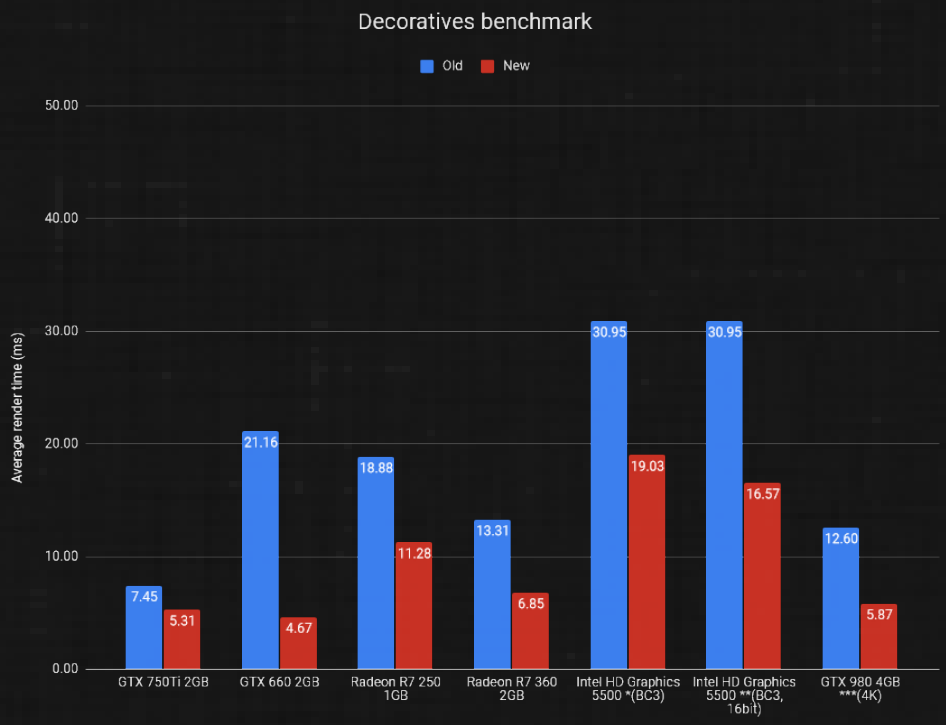
Nous avons pris quelques repères de scènes extrêmes. Le premier est en sauvegarde de l'événement public Clusterio de l'année dernière. Twinsen m'a envoyé la sauvegarde, parce qu'il recevait des gouttes de mémoire d'images au milieu d'un grand réseau de turbines à vapeur. Le second est sauvegardé depuis un rapport de bogue. J'ai choisi la sauvegarde parce que les rails vont dans la forêt et que, même en 0.16, je pouvais vraiment obtenir des FPS bas, j'ai utilisé un éditeur de carte pour recouvrir le sol de motifs décoratifs en gazon.

Nous avons testé sur des GPU avec 2 Go ou 1 Go de VRAM, que nous avons au bureau. Les benchmarks sur les GPU de bureau ont été exécutés sur un seul PC (nous avons simplement échangé des GPU entre les exécutions) - Intel Core i7-4790K, 16 Go de RAM, Windows 10. Nous avons mesuré le temps de traitement d'une image par GPU, pour 1000 images, et une moyenne des résultats. . Nous nous sommes comparés à la version 0,17 d'avant la mise en œuvre des optimisations côté GPU. Malheureusement, nous n'avons aucun moyen de capturer les timings du GPU dans la version 0,16. Les tests ont fonctionné en résolution FullHD (1920x1080) avec les sprites haute résolution activés, avec une configuration graphique que je crois être la meilleure pour les performances de rendu. Pour les anciennes versions : Mipmaps, spécialisation Atlas, atlas d'objet inférieur séparé, options de compression de texture activées, Utilisation de la mémoire vidéo définie sur Tout. Et configuration équivalente pour la nouvelle version, à l'exception de l'utilisation de la nouvelle option de compression de haute qualité.

Nous avons également inclus le résultat d'un processeur graphique haute performance (GeForce GTX 980 4 Go), mais ce dernier s'exécutait en résolution 4K (3840x2160) par opposition à FullHD. La référence d'Intel HD Graphics 5500 a été exécutée sur un ordinateur portable doté de processeurs Intel Core i7-5600U et de 16 Go de RAM et qui utilisait plutôt une compression de texture de qualité inférieure. Nous avons également inclus les résultats avec la profondeur de couleur 16 bits activée.




Smoke benchmark
Old
New
Average render time (ms)
50.00
40.00
30.00
20.00
10.00
0.00
12.23
5.13
38.08
4.25
27.99
9.54
13.94
5.39
44.25
22.61
44.25
20.76
13.78
6.37
GTX 750Ti 2GB
GTX 660 2GB
Radeon R7 250 1GB
Radeon R7 360 2GB
Intel HD Graphics 5500 *(BC3)
Intel HD Graphics 5500 **(BC3, 16bit)
GTX 980 4GB ***(4K)



Les temps ont été mesurés en millisecondes. Les temps inférieurs sont meilleurs, et pour 60 FPS, un cadre doit prendre moins de 16,66ms.

Si vous êtes intéressés par la façon dont les GPU fonctionnent plus en profondeur, Fabian Giesen a écrit une jolie série d'articles sur le sujet.

Comme toujours, laissez-nous savoir ce que vous pensez sur notre forum .