[Note du traducteur : ceci est une traduction en français libre et non officielle du FFF #264]

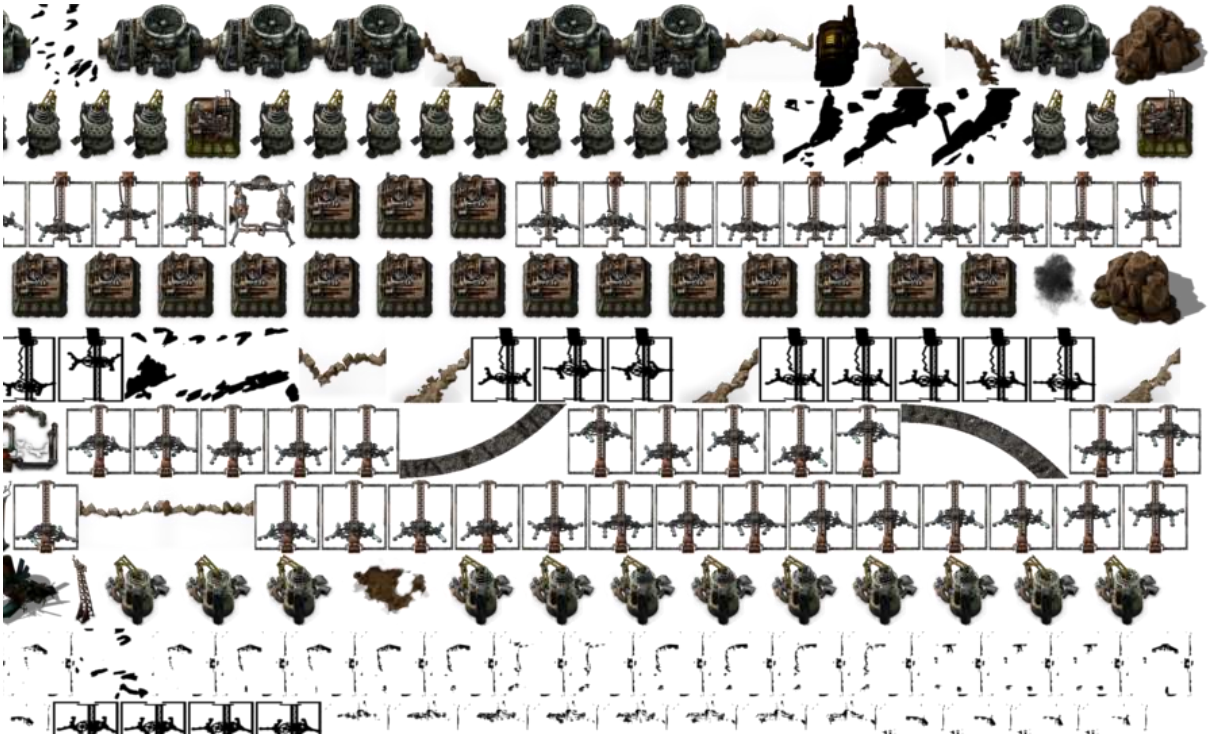
(par posila)

Flux de textures

Bonjour, c'est moi, posila, avec un autre article technique. Sorry.

Cache Bitmap

Avec la version 0.16, nous avons ajouté l'option graphique mystérieusement appelée le mode "*Low VRAM*" [NdT: mode de mémoire vidéo faible], elle permet une implémentation de base du flux de texture. Mais je ne voulais pas l'appeler ainsi, car je craignais que sa mauvaise performance ne donne une mauvaise réputation au flux de texture. Comment cela fonctionne-t-il ? Chaque sprite[1] a spécifié une priorité, et l'option "*Utilisation de la mémoire vidéo*" - malgré son nom - contrôle quelles priorités de sprites sont incluses dans l'atlas des sprites. Qu'arrive-t-il aux sprites qui ne vont pas dans l'atlas ? Ils sont chargés comme des sprites individuels. Normalement, ces sprites sont alloués en objets de texture. Le raisonnement derrière cela est que le pilote graphique a la possibilité de décider comment il veut disposer ces petites textures en mémoire, au lieu d'être forcé de travailler avec d'énormes atlas.



Voici comment ressemble une partie d'un atlas de sprites.

[NdT : [1] Sprite : dans le domaine de l'infographie et des jeux vidéo, désigne une image en deux dimensions qui peut être déplacée indépendamment du fond (ou décor) de l'affichage et est théoriquement superposé par le processeur vidéo, au moment d'envoyer le signal à l'écran. (Wikipédia)]

Lorsque vous activez le mode "*Low VRAM*", les sprites hors atlas sont chargés uniquement en RAM, et la texture leur est allouée uniquement lorsqu'ils sont utilisés pendant le rendu. Nous avons appelé "*BitmapCache*" la classe qui gérait cela et il y avait une limite maximale du nombre de mégaoctets de données de texture que le cache bitmap pouvait utiliser à la fois. Lorsque cette limite était atteinte, il ne réussissait pas à convertir la mémoire bitmap en texture vidéo, et le système de dessin d'Allegro devait passer au rendu logiciel, ce qui bloquait complètement le FPS [NdT : nombre d'images par seconde], mais cela ne s'est pas produit... la plupart du temps.

Donc, à part le problème évident de retour au rendu logiciel (qui n'existe plus depuis la réécriture graphique, donc le jeu plantera ou sautera un sprite si cela arrive), il y a d'autres problèmes de performance. La plupart des sprites ont une taille unique, donc nous ne pouvons pas réutiliser les textures pour différents sprites. Au lieu de cela, lorsqu'un sprite doit être converti en une texture, une nouvelle texture est allouée, et lorsque le sprite est expulsé du cache, sa texture est détruite. Créer et détruire les textures ralentit considérablement le rendu. La façon dont nous le faisons fragmente aussi la mémoire, et donc tout d'un coup il peut échouer à allouer une nouvelle texture parce qu'il n'y a plus de blocs de mémoire consécutifs assez grands. De plus, comme nos sprites ne sont pas dans un atlas, la mise en lots des sprites ne fonctionne pas et nous obtenons un autre cap de performance en émettant des milliers d'appels de dessin au lieu de seulement des centaines.

J'ai considéré qu'il s'agissait d'une expérience, et j'ai été un peu surpris que ses performances n'étaient pas aussi mauvaises que je le pensais. Bien sûr, cela peut faire chuter le FPS à un seul chiffre de temps en temps, mais dans l'ensemble le jeu reste jouable (j'ai une longue histoire de jeu sur console, donc mes standards en tant que joueur peuvent ne pas être très élevés :)).

Pouvons-nous le rendre assez bon pour qu'il ne soit plus une option expérimentale, mais plutôt quelque chose qui pourrait être activé par défaut ? Voyons voir. Le problème est l'allocation des textures, donc allouons une texture pour l'ensemble du cache bitmap - ce serait un atlas de sprite que nous mettrions à jour dynamiquement. Cela permettrait également d'améliorer le dosage des sprites, mais quand j'ai commencé à réfléchir à la façon de le mettre en œuvre, j'ai rapidement rencontré un problème de fragmentation de l'espace. J'ai envisagé de faire une "défragmentation" de temps en temps, mais cela a commencé à me sembler un problème insurmontable, avec un résultat très incertain.

Mappage de texture virtuel

Comme je l'ai mentionné dans la FFF-251, il est très important pour notre performance de rendu que les commandes de dessin de sprite soient utilisées par lots. Si plusieurs commandes de dessin consécutifs utilisent la même texture, nous pouvons les regrouper en un seul appel de dessin. C'est pourquoi nous construisons de grands atlas de sprites. Le mappage de texture virtuel - une technique de flux de texture popularisée par *id Software* sous le nom de *Mega Textures*, semble être la solution idéale pour nous. Tous les sprites sont placés dans un atlas virtuel unique, dont la taille n'est pas limitée par des limites matérielles. Vous devez quand même pouvoir stocker l'atlas quelque part, mais il n'est pas nécessaire qu'il s'agisse d'un bloc de mémoire consécutif. L'idée derrière est la même que dans la mémoire virtuelle - les allocations de mémoire assignent une adresse virtuelle qui correspond à un emplacement physique qui peut changer en interne (RAM, fichier de page, etc.), les sprites reçoivent des coordonnées de texture virtuelle qui sont affectées à un emplacement physique.

L'atlas virtuel est divisé en tuiles ou pages (dans notre cas 128x128 pixels), et lors du rendu, nous allons déterminer quelles tuiles sont nécessaires, et les charger dans une texture physique de dimensions beaucoup plus petites que celle du virtuel. Dans le nuanceur de pixel, nous transformons ensuite les coordonnées de texture virtuelle en coordonnées physiques. Pour cela, nous avons besoin d'une table de direction qui indique où trouver les tuiles de la texture virtuelle dans la texture physique. C'est tout un défi pour les moteurs 3D de déterminer quelles pages de textures virtuelles sont nécessaires, mais puisque nous passons par l'état du jeu pour déterminer quels sprites doivent être rendus, nous avons déjà cette information facilement disponible.

Cela résout le problème des affectations fréquentes - nous n'avons qu'une texture et nous la mettons juste à jour. De plus, puisque tous les sprites partagent le même espace de coordonnées de texture, nous pouvons faire des appels de dessin par lots qui les utilisent. Super !

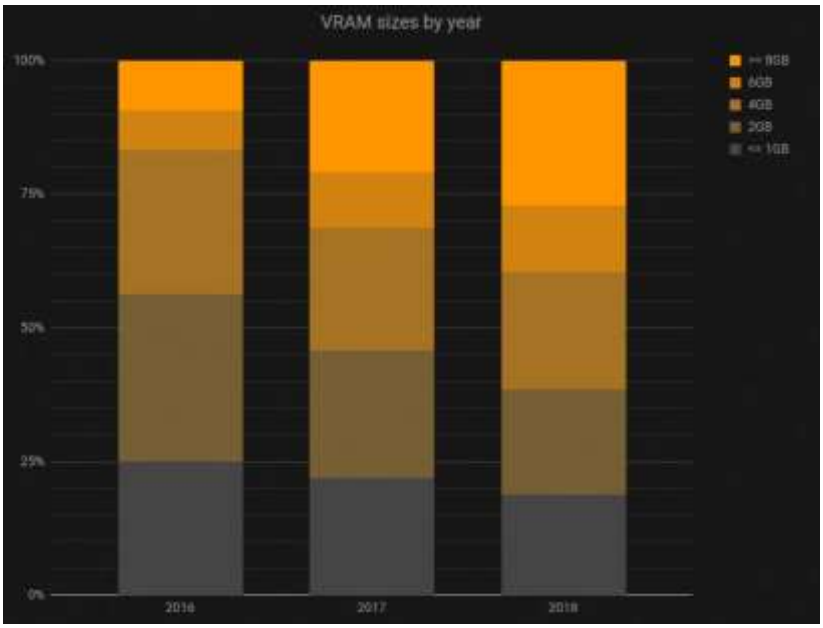
Cependant, nous pourrions encore manquer d'espace dans la texture physique. C'est plus probable quand le joueur fait un zoom arrière, car beaucoup plus de sprites différents peuvent être visibles en même temps. Eh bien, si vous faites un zoom arrière, les sprites sont réduits, et nous n'avons pas besoin de rendre les sprites dans leur pleine résolution. Pour cela, l'atlas virtuel possède plusieurs niveaux de détails (mipmaps), qui sont la même texture réduite à différentes tailles (0,5, 0,25, etc.) et nous ne devons diffuser en continu que les niveaux de mipmap qui sont nécessaires pour le niveau de zoom actuel. Nous pouvons utiliser des niveaux de mipmap plus bas aussi si vous avez zoomés et qu'il y a trop de sprites à l'écran. Nous pouvons également utiliser les détails moindres pour limiter le temps passé en flux par image afin d'éviter les blocages dans le rendu lorsqu'une mise à jour importante est nécessaire.

La technique de l'atlas virtuel est une grande amélioration par rapport à l'ancien mode "*Low VRAM*", mais elle n'est toujours pas suffisante. Dans l'idéal, j'aimerais que cela fonctionne si bien que nous puissions supprimer les options de qualité de sprite basse et très basse, et que tout le monde puisse jouer le jeu normalement. Ce qui empêche cela, c'est que tout l'atlas virtuel doit être en RAM. Le flux à partir d'un disque dur a une latence très élevée, et nous ne savons pas encore s'il nous sera possible de le faire sans introduire de mauvais flashes de sprite, etc.

Si vous souhaitez en savoir plus sur le mappage de textures virtuelles, vous pouvez lire l'article "Advanced Virtual Texture Topics", ou peut-être même plus en détail "Software Virtual Textures".

Performances de rendu du processeur graphique

La motivation principale derrière le flux de texture est de s'assurer que le jeu est capable de fonctionner avec des ressources limitées, sans avoir à réduire trop la qualité visuelle. Selon l'enquête sur le matériel Steam, près de 60% de nos joueurs (qui ont un processeur graphique dédié) ont au moins 4 Go de VRAM et ce nombre augmente au fur et à mesure que les gens mettent à jour leurs ordinateurs :



[NdT : Taille de la mémoire vidéo par année]

Nous avons reçu pas mal de rapports de bugs concernant des problèmes de performances de rendu de la part de personnes ayant des processeurs graphiques corrects, surtout depuis que nous avons commencé à ajouter des sprites haute résolution. Nous avons supposé que les problèmes étaient causés par le fait que le jeu voulait utiliser plus de mémoire vidéo que disponible (le jeu n'est pas la seule application qui veut utiliser de la mémoire vidéo) et que le pilote graphique doit passer beaucoup de temps pour optimiser les accès aux textures.

Pendant la réécriture graphique, nous avons beaucoup appris sur le fonctionnement des processeurs graphiques actuels (et nous continuons à apprendre), et nous avons pu utiliser les nouvelles API pour mesurer leur temps de rendu.

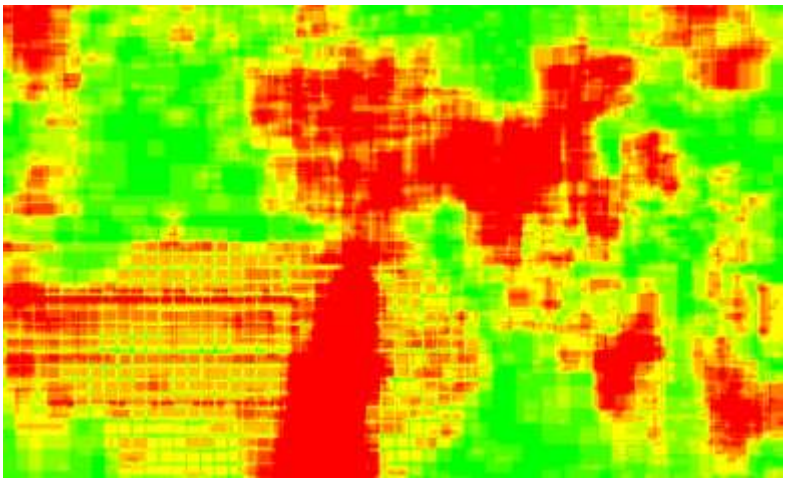
Pour simplement dessiner une image 1920x1080 sur une cible de rendu de la même taille, il faut approximativement :

- 0,1 ms sur GeForce GTX 1060.
- 0,15 ms sur Radeon Vega 64.
- 0,2 ms sur GeForce GTX 750Ti ou Radeon R7 360.
- 0,75 ms sur GeForce GT 330M.
- 1 ms sur Intel HD Graphics 5500.
- 2 ms sur Radeon HD 6450.

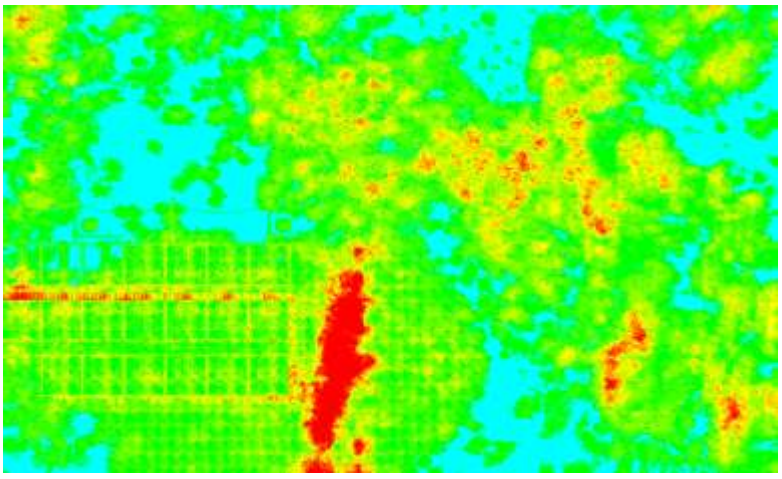
Cela semble se mettre à l'échelle linéairement avec le nombre de pixels écrits, il faudrait donc approximativement 0,4 ms pour que le GTX 1060 rende la même chose en 4K.

C'est assez rapide, mais nos sprites ont beaucoup de pixels semi-transparents. Nous utilisons également la transparence d'autres façons - du dessin de fantômes et de l'application de masques de couleur, aux visualisations de dessins comme la zone logistique ou les plages de tourelles. Il en résulte une grande quantité de recouvrements - les pixels sont écrits à plusieurs reprises. Nous savions que le recouvrement était quelque chose à éviter, mais nous n'avions pas de données fiables sur l'ampleur du phénomène dans Factorio, jusqu'à ce que nous ajoutions la visualisation du recouvrement :

La scène de jeu en cours de rendu.



Visualisation du recouvrement (cyan = 1 dessin, vert = 2, rouge >= 10).



Visualisation du recouvrement lorsque l'on supprime les pixels transparents.

Il est intéressant de noter que l'élimination des pixels totalement transparents ne semble pas faire de différence en termes de performances sur les processeurs graphiques modernes, y compris Intel HD. Dessiner des sprites avec beaucoup de pixels complètement transparents est plus rapide qu'un sprite opaque sans avoir à éliminer explicitement les pixels transparents avec des nuanceurs. Cependant, cela a fait la différence sur Radeon HD 6450 et GeForce GT 330M, donc peut-être que les processeurs graphiques modernes jetteraient automatiquement des pixels qui n'auraient aucun effet sur le résultat ?

Quoi qu'il en soit, une GTX 1060 rend une scène de jeu comme celle-ci en 1080p en 1 ms. C'est rapide, mais cela signifie qu'en 4K, cela prendrait 4 ms, 10 ms sur les processeurs graphiques intégrés, et plus qu'un seul frame de temps (16,66 ms) sur les anciens processeurs graphiques. Ce n'est pas étonnant, des scènes lourdes de fumée ou d'arbres peuvent faire chavirer le FPS, surtout en 4K. Peut-être devrions-nous faire quelque chose à ce sujet...

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