[NdT : Ceci est une traduction française libre et non officielle du "Factorio Friday Facts" n°251]

Une poignée de cadres

Factorio à la Bibliothèque nationale de technologie de Prague (par Klonan)

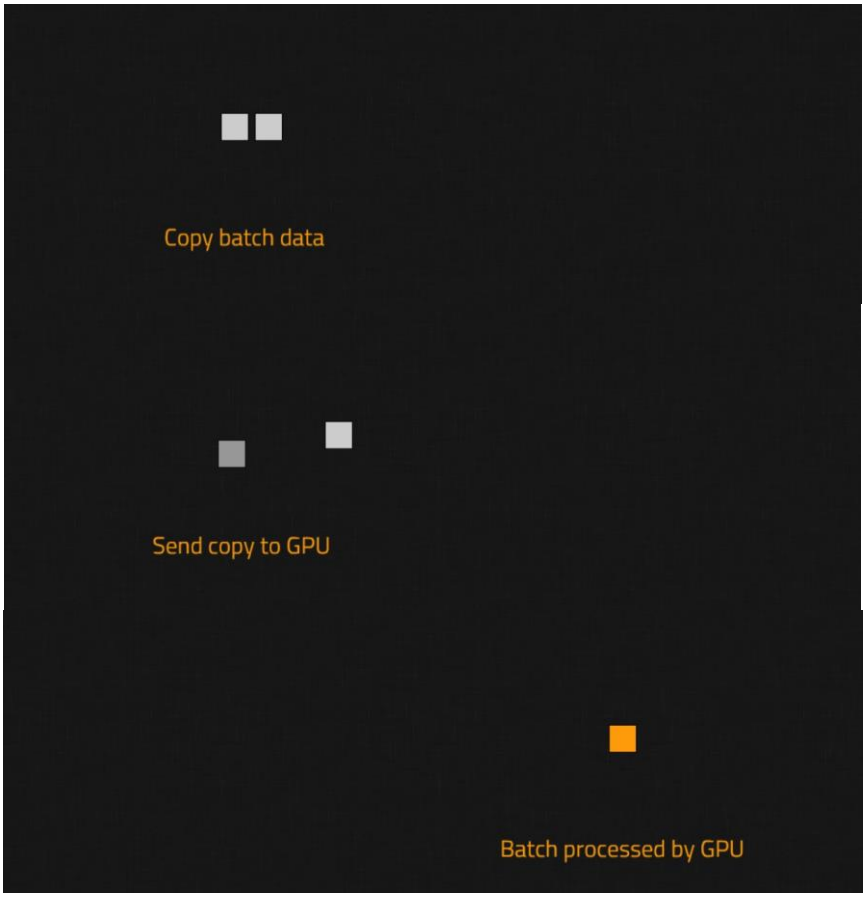
Si vous êtes à Prague cet été et que vous voulez satisfaire vos envies de Factorio, vous pouvez vous arrêter à la Bibliothèque nationale de technologie de Prague, où Factorio est installé sur 150 ordinateurs pour que tous puissent y jouer. L'entrée est gratuite pour tous les visiteurs du lundi au vendredi de 08h à 22h jusqu'au 31 août. Les PC fonctionnent sous Linux (Fedora), avec une version personnalisée du jeu (réalisée par HanziQ), et vous pouvez héberger des serveurs LAN et jouer avec vos amis.

Le 23 juillet, nous organiserons notre propre soirée *Factorio LAN party* à la bibliothèque à partir de 16h CEST (heure de Prague [NdT: mais aussi de Paris, Berne et Bruxelles]), afin que vous puissiez venir jouer avec nous. Il est conseillé d'apporter votre propre casque d'écoute si vous allez y assister.

Optimisation du rendu (par posila)

Nous avons commencé à moderniser notre système de rendu, l'absolu étant de le rendre au moins aussi rapide que l'ancien. Nous avons eu la chance de faire les choses comme nous le voulions, donc nous sommes enthousiasmés par les nouvelles APIs débloquées pour nous, et nous avons beaucoup d'idées pour dessiner des sprites [NdT : symboles graphiques] aussi vite que possible.

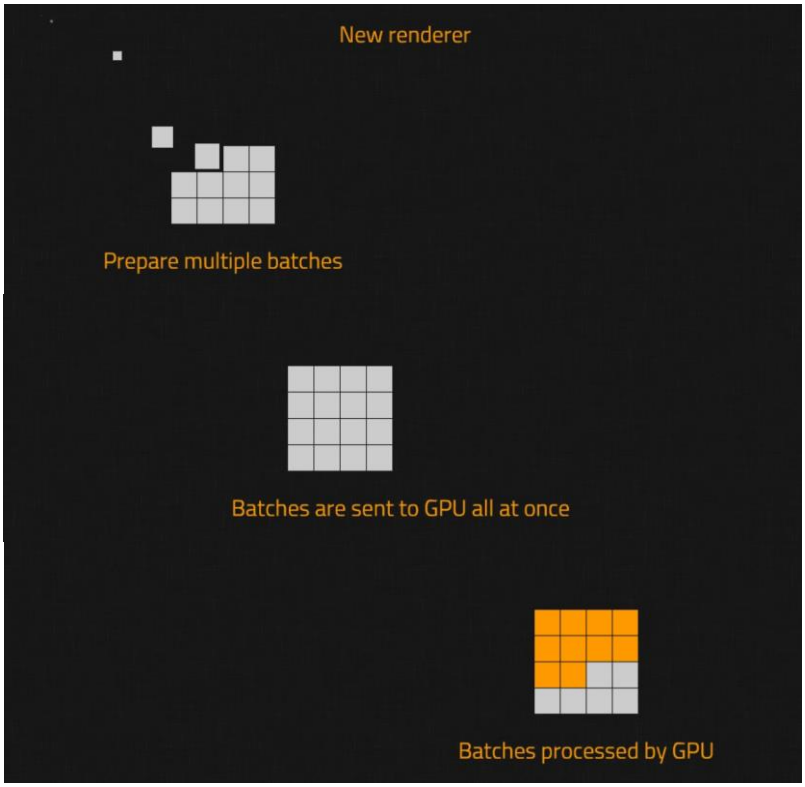
Mais d'abord, il n'est pas nécessaire de réinventer la roue, alors voyons comment Allegro rend la magie possible. Allegro utilise le sprite par lot, ce qui signifie qu'il dessine plusieurs sprites qui utilisent la même texture et le même état de rendu, en utilisant une seule commande envoyée au processeur graphique. Ces commandes de dessins sont généralement appelées *appels de dessin*. Allegro dessine des sprites en utilisant 6 sommets, et il les met en lots dans un tampon C alloué. Chaque fois qu'un lot se termine, il est transmis à la fonction de dessin OpenGL ou DirectX qui le copie (afin de ne pas bloquer le CPU) et envoie l'appel de dessin.



[NdT : Ancien rendu
Préparer un seul lot
Copier les données du lot
Envoyer la copie au processeur graphique
Le lot est traité par le processeur graphique]

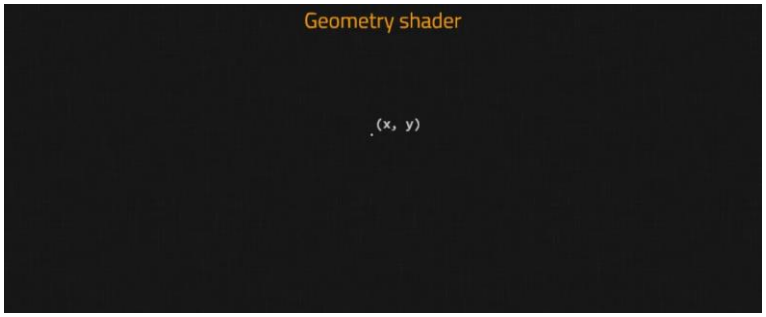
Cela semble assez raisonnable, mais nous ne pouvons pas faire exactement la même chose, parce que dans DirectX 10, il n'y a pas de fonctions pour dessiner directement à partir d'un tableau C, et il est obligatoire d'utiliser des tampons de sommets. Notre première version créait donc un tampon de sommets dans lequel le lot courant était toujours copié pour être utilisé dans un appel de dessin, et nous réaffections un tampon avec une taille plus grande si le lot courant ne pouvait pas y tenir. Il fonctionnait très bien, probablement pas aussi vite que la version Allegro, mais il traînait sensiblement à chaque fois que le tampon de sommets avait besoin d'être redimensionné.

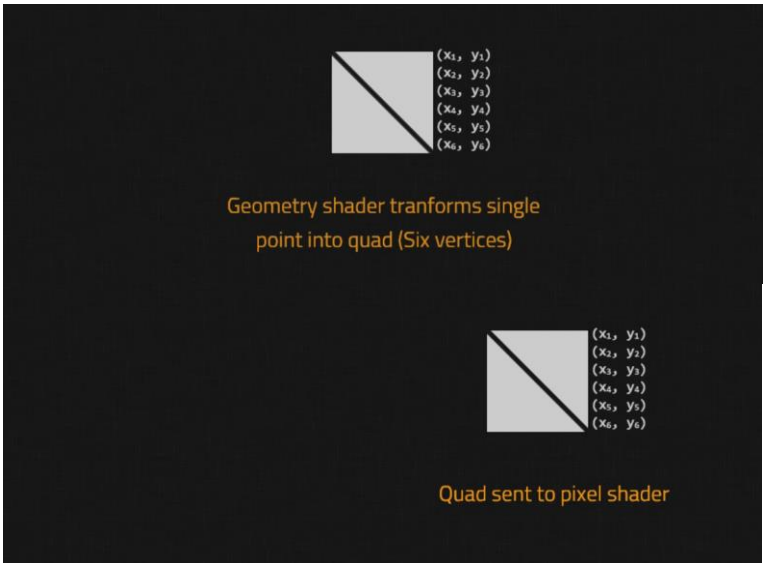
Après avoir lu quelques articles, par exemple "*Optimiser le rendu dans Galactic Civilizations 3*" et "*Le streaming d'objets tampon*" sur le Wiki OpenGL (qui était très utile), il est devenu clair que la façon de faire est d'avoir un tampon de sommets de taille fixe, et de continuer à ajouter jusqu'à ce qu'il soit plein. Lorsque nous terminons d'écrire un lot dans la mémoire tampon, nous n'envoyons pas un appel de dessin tout de suite, nous écrivons où ce lot commence et se termine dans une file d'attente, et nous continuons d'écrire dans la mémoire tampon. Lorsque le tampon est plein, nous le sortons de la mémoire système et nous envoyons tous les appels de dessin en attente en une seule fois. Cela permet d'économiser sur l'opération coûteuse de composition et de vider le tampon de sommets pour chaque lot.



[NdT : Nouveau rendu
Préparer plusieurs lots
Les lots sont envoyés tous ensemble vers le processeur graphique
Les lots sont traités par le processeur graphique]

Alors que nous essayions de trouver comment servir les données au processeur graphique de la manière la plus efficace, nous avons également expérimenté quel type de données à envoyer au processeur graphique. Moins nous envoyons de données, mieux c'est, et Allegro utilisait 6 sommets par sprite avec une taille totale de 144 octets. Nous voulions faire des sprites ponctuels qui ne nécessiteraient que 48 octets par sprite et donc moins de calculs pour le processeur graphique par sprite. Notre première idée était d'utiliser l'instanciation, mais nous avons rapidement changé d'avis sans même essayer, parce que lorsque nous avons fait des recherches sur la méthode, nous avons trébuché sur cette présentation mettant en garde contre l'utilisation de l'instanciation pour les objets à faible nombre de polygones (comme les sprites). L'idée suivante était d'implémenter des sprites ponctuels en utilisant un nuanceur de géométrie.



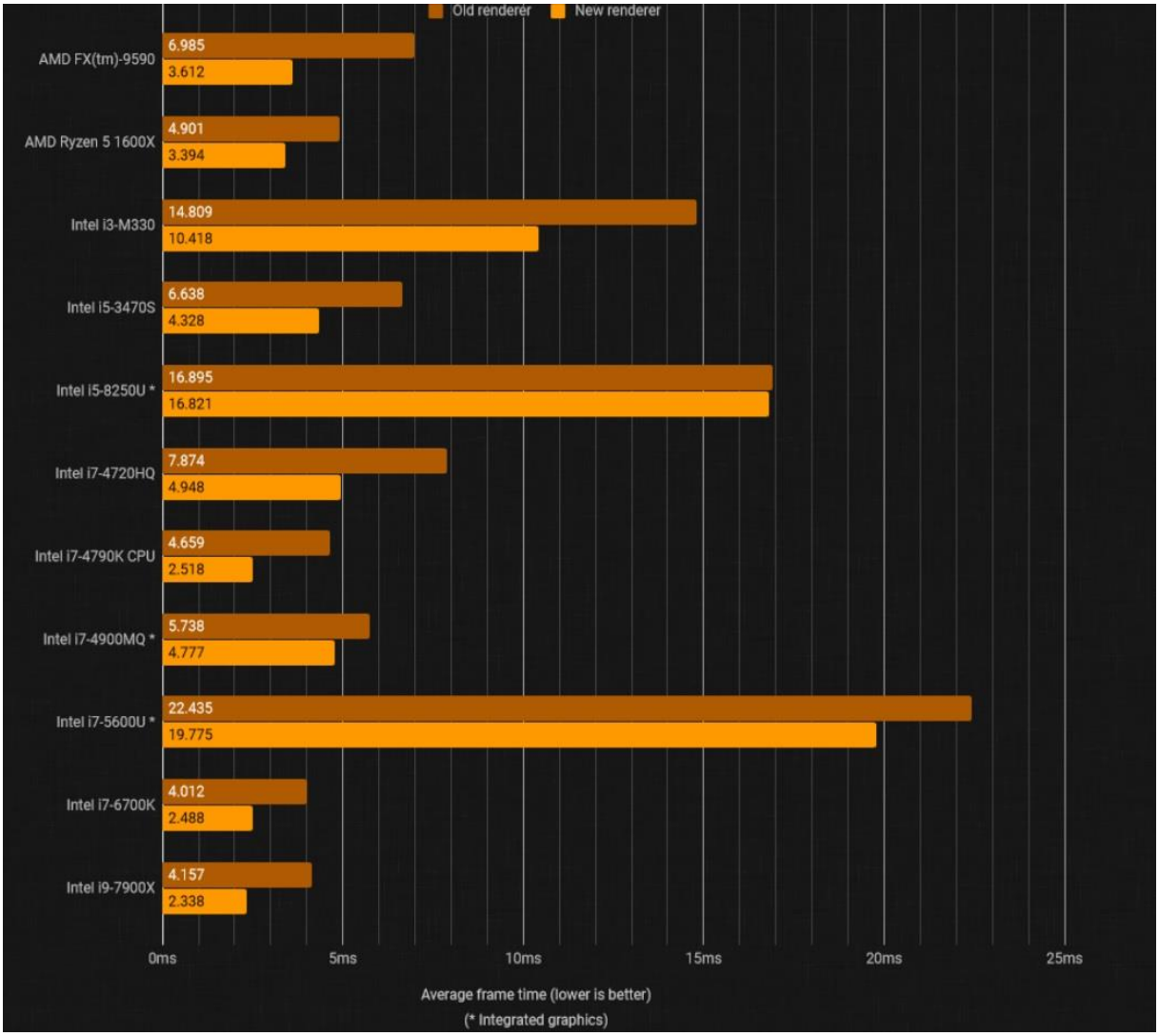


[NdT : Nuanceur de géométrie
Le nuanceur de géométrie transforme le simple sommet en un carré (6 sommets)
Le carré est envoyé au nuanceur de pixel]

Nous avons essayé, et ça a très bien fonctionné. Nous avons obtenu une certaine amélioration de la vitesse car le CPU doit préparer moins de données, mais en faisant des recherches sur la façon dont les différentes APIs fonctionnent avec les nuanceurs de géométrie, nous avons découvert que Metal (et donc MoltenVK) sur macOS ne supporte pas du tout les nuanceurs de géométrie. Après avoir creusé davantage, nous avons trouvé un article intitulé "*Pourquoi les nuanceurs de géométrie sont lents*". Nous avons donc testé le nuanceur de géométrie sur une série de PC dans le bureau, et nous avons constaté que, bien qu'il soit plus rapide sur les PC équipés de nouvelles cartes graphiques, les anciennes machines ont pris un coup remarquable sur les performances. En raison du manque de support sur macOS et du ralentissement possible sur des machines plus lentes, nous avons décidé d'abandonner l'idée.

Il semble que la meilleure façon de faire des sprites de sommets est d'utiliser un tampon constant ou tampon de texture pour passer les données des sommets à un nuanceur de sommets, et d'étendre les sommets en carrés. Mais à ce stade, nous avons déjà toutes les optimisations mentionnées dans la première partie, et la partie CPU du rendu est maintenant assez rapide pour que nous puissions mettre l'idée du sprite ponctuel au frigo pour le moment. Au lieu de cela, le CPU préparera 4 sommets par sprite avec une taille totale de 80 octets, et nous utiliserons un tampon d'index statique pour les étendre en deux triangles.

Les résultats suivants proviennent de divers ordinateurs. Le test a fait un rendu d'une seule image avec un zoom arrière maximum (environ 25 000 sprites) 600 fois le plus vite possible sans mettre à jour le jeu, et le graphique montre le temps moyen de préparation et de rendu de l'image. Sur les ordinateurs avec processeur graphique intégré, il y a eu peu d'améliorations parce que ceux-ci semblent être bloqués par le processeur graphique.



[NdT : Ancien rendu / Nouveau rendu
Temps moyen par image (le plus bas est le meilleur)
(* processeur graphique intégré)]

Nous avons également remarqué des accélérations plus élevées sur les cartes AMD par rapport aux cartes nVidia. Par exemple, dans mon ordinateur, j'ai un GTX 1060 6GB et Radeon R7 360. En 0.16, le rendu était beaucoup plus lent sur le Radeon que sur le GeForce, mais avec le nouveau moteur de rendu, les performances sont presque identiques (comme cela devrait l'être parce que le processeur graphique termine son travail plus vite que le CPU ne peut l'alimenter en commandes de dessins). Pour la suite, nous avons besoin d'améliorer le côté processeur graphique des choses, principalement l'utilisation excessive de la mémoire vidéo (VRAM), mais vous en saurez plus à ce sujet dans quelques futurs Friday Facts ...

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