42-Slab - Traitement des données sur le GPU

> **Remarque**: certaines techniques décrites dans ce didacticiel sont dépassées. Il est recommandé aux utilisateurs d'utiliser *jit.movie* avec **output_texture** activé au lieu de uyvy colormode pour l'efficacité de téléchargement des images vidéo vers le GPU. Voir l'article GL Texture Output pour plus d'informations.

Nous avons vu dans le didacticiel précédent comment des shaders personnalisés peuvent être exécutés sur l'unité de traitement graphique (GPU) pour appliquer des modèles d'ombrage supplémentaires aux objets 3D. Bien que le processeur de sommets et le processeur de fragments soient intrinsèquement conçus pour rendre la géométrie 3D, avec un peu de créativité, nous pouvons faire en sorte que ces puissantes unités d'exécution fonctionnent sur des ensembles de données matricielles arbitraires pour effectuer des tâches telles que le traitement d'image et d'autres activités. Vous pourriez vous demander: "Si nous pouvons déjà traiter des ensembles de données matricielles arbitraires sur le CPU avec des opérateurs matriciels Jitter (MOP), pourquoi faire quelque chose de stupide comme utiliser la carte graphique pour faire ce travail?" La réponse est la vitesse.

> **Configuration matérielle requise**: pour profiter pleinement de ce didacticiel, vous aurez besoin d'une carte graphique prenant en charge les shaders programmables, par ex. les cartes graphiques ATI Radeon 9200, NVIDIA GeForce 5000 ou plus récentes. Il est également recommandé de mettre à jour votre pilote OpenGL avec la dernière version disponible pour votre carte graphique. Sur Macintosh, cette mise à jour est fournie avec la dernière mise à jour du système d'exploitation.

Tendances récentes en matière de performances

Au cours du dernier demi-siècle , les performances des CPU ont plus ou moins suivi la loi de Moore, qui prévoit le doublement des performances des CPU tous les 18 mois. Cependant, l'architecture du GPU n'ayant pas besoin d'être aussi flexible que celle du CPU et étant intrinsèquement ***parallélisable*** (c'est-à-dire que plusieurs pixels peuvent être calculés indépendamment les uns des autres), les GPU ont été rationalisés au point que les performances progressent à un rythme beaucoup plus rapide, doublant même tous les 6 mois. Cette tendance a été qualifiée de loi de ***Moore's Law Cubed***. A l'heure où nous écrivons ces lignes, les cartes graphiques grand public haut de gamme possèdent jusqu'à 128 pipelines de sommets et 176 pipelines de fragments qui peuvent chacun fonctionner en parallèle, permettant ainsi des dizaines d'effets de traitement d'image en résolution HD avec une fréquence d'images maximale. Compte tenu de l'histoire récente, il semblerait que les performances des GPU continueront à augmenter plus rapidement que celles du CPU.

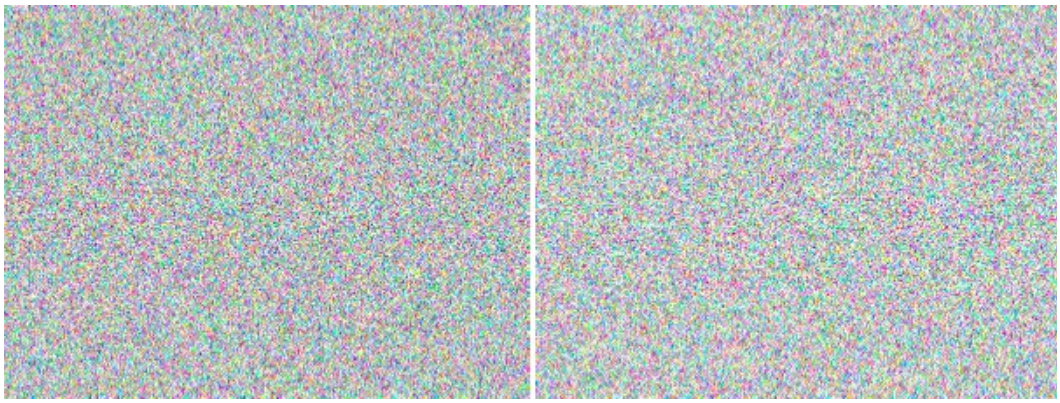
Mise en route

• Ouvrez le patch de Tutorial et double-cliquez sur le sub-patch **p slab-comparison-CPU** pour l'ouvrir. Cliquez sur le *toggle* relié à l'objet *qmetro*. Notez les performances du patch telles qu'elles sont affichées par l'objet *jit.fpsgui* en bas.

• Désactivez le *toggle*, fermez le sub-patch et double-cliquez sur le sub-patch **p slab-comparison-GPU** pour l'ouvrir. Cliquez deux fois sur le *toggle* connecté au message **sync** pour désactiver la synchronisation. Cliquez sur le *toggle* connecté à l'objet *qmetro*. Notez les performances dans **jit.fpsgui** et comparez-les à ce que vous avez obtenues avec la version CPU.

Les patchs ne font rien de particulièrement excitant; il s'agit simplement d'un ensemble d'additions et de multiplications en cascade fonctionnant sur une matrice de bruit **640x480** (valeurs aléatoires de type **char**). Un patch effectue ces calculs sur le CPU, l'autre sur le GPU. Dans les deux exemples, le bruit est généré par le CPU (ce qui n'est pas sans coût). Les résultats visibles de ces deux patchs devraient être similaires; cependant, comme vous le remarquerez probablement si vous avez une carte graphique récente, les performances sont beaucoup plus rapides lorsqu'elles sont exécutées sur la carte graphique (GPU). Notez que nous n'effectuons que quelques opérations mathématiques simples sur un ensemble de données, et cette même technique pourrait être utilisée pour traiter des ensembles de données matricielles arbitraires sur la carte graphique.

> Qu'est-ce que le message **sync**? Il est généralement inutile de rendre les images plus rapidement que l'ordinateur ne peut les afficher. En fait, si le logiciel devance le matériel, il peut essayer d'afficher deux images en même temps: vous obtiendrez une partie d'une image en haut de la fenêtre et une partie d'une autre en bas, un effet appelé " déchirement". Nous n'avons pas non plus besoin de gaspiller des cycles sur des images qui ne seront jamais vues - le système a d'autres choses à faire. L'attribut **sync** de l'objet *jit.window* synchronise les calculs de Jitter avec le taux d'affichage de la fenêtre (généralement 60 ips). Cela ne signifie pas que le GPU n'est plus aussi rapide, mais qu'il doit juste prendre une pause entre les images.



CPU (à gauche) et GPU (à droite) traitant du bruit.

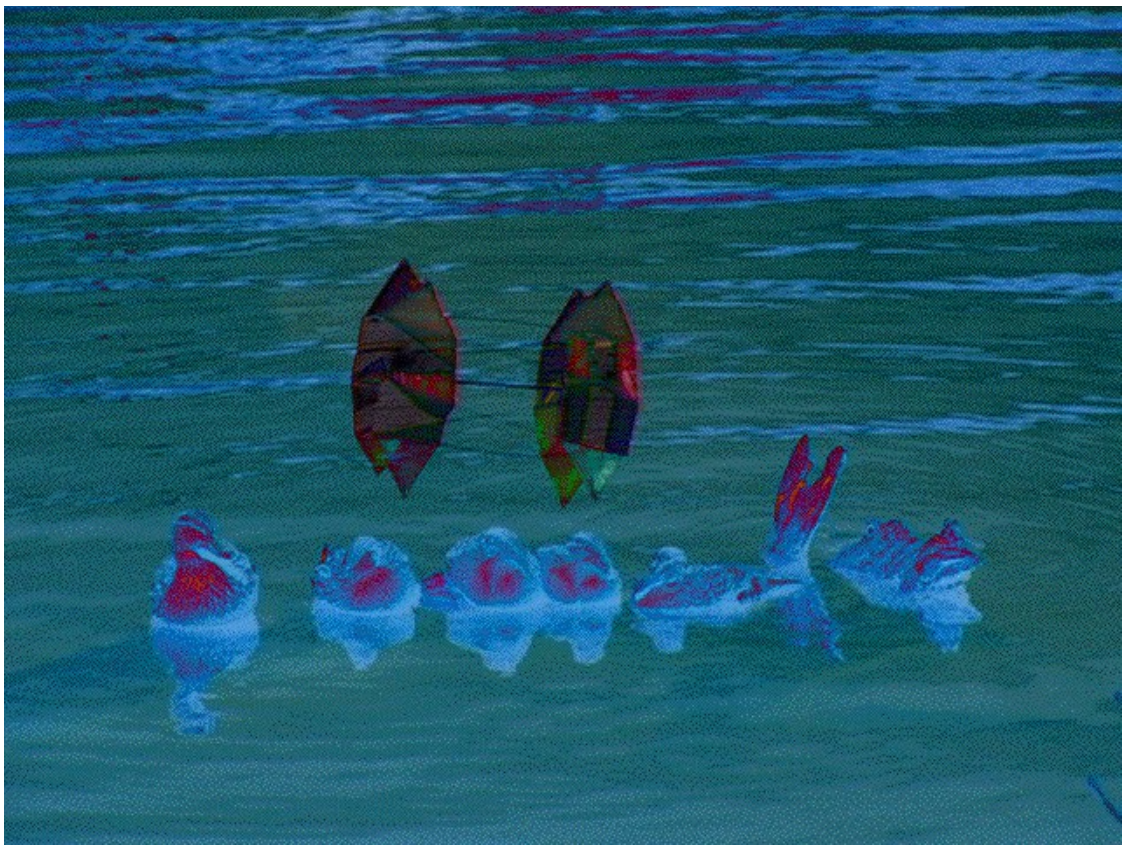
Qu'en est-il des modèles d'ombrage?

Contrairement au dernier tutoriel, nous n'effectuons pas le rendu de ce qui semble être une géométrie 3D en fonction des propriétés d'éclairage ou de matériaux. Par conséquent, cela ne semble pas vraiment être la même chose que les shaders que nous avons déjà couverts, n'est-ce pas? En fait, nous utilisons toujours le même processeur de sommets et le même processeur de fragments, mais avec une géométrie extrêmement simple où les pixels des coordonnées de la texture appliquée à notre géométrie correspondent aux coordonnées en pixels de notre tampon de sortie. Au lieu des calculs d'éclairage et de matériaux, nous pouvons effectuer des calculs arbitraires par pixel dans le processeur de fragments. De cette façon, nous pouvons utiliser les programmes de shaders de manière similaire aux objets Jitter qui traitent les matrices sur le CPU (Jitter MOP).

• Ouvrez le patch de Tutorial et double-cliquez sur le sub-patch **p slab-composite-DV** pour l'ouvrir. Cliquez sur le *toggle* connecté à l'objet *qmetro* le plus à gauche.

• Cliquez sur les boîtes de *message* contenant **dvducks.mov** et **dvkite.mov** pour charger deux films DV et activez les objets *metro* correspondants pour permettre la lecture.

• Chargez un opérateur de composition souhaité à partir de l'objet *umenu* connecté à l'instance la plus élevée de *jit.gl.slab*.



Séquences DV UYVY composites sur GPU utilisant "différence" op.

Pour autant que notre matériel puisse suivre, nous mélangeons maintenant deux sources DV en temps réel sur le GPU. Vous remarquerez que les objets *jit.movie* et l'objet *jit.gl.slab* le plus haut ont chacun leur attribut **colormode** défini sur **uyvy**. Comme nous l'avons vu dans le *didacticiel 49*: *Colorspaces*, cela indique aux objets *jit.movie* de rendre le métrage DV en données YUV 4: 2: 2 à chrominance réduite, et à l'objet *jit.gl.slab* d'interpréter les matrices entrantes comme telles. Nous sommes en mesure d'obtenir une décompression plus efficace du métrage DV en utilisant des données **uyvy** parce que le DV est nativement un format YUV à réduction chromatique. Comme les données **uyvy** occupent la moitié de la mémoire des données ARGB, nous pouvons réaliser un transfert de mémoire plus efficace vers la carte graphique.

Ajoutons un traitement supplémentaire à cette chaîne.

• Cliquez sur les boîtes de *message* contenant **read cf.emboss.jxs** et **read cc.scalebias.jxs** connectés aux deux instances inférieures de *jit.gl.slab*.

• Ajustez ces deux effets en jouant avec les boîtes de *nombres* à droite pour modifier les paramètres des deux effets.

Traitement supplémentaire sur GPU.

Comment ça marche?

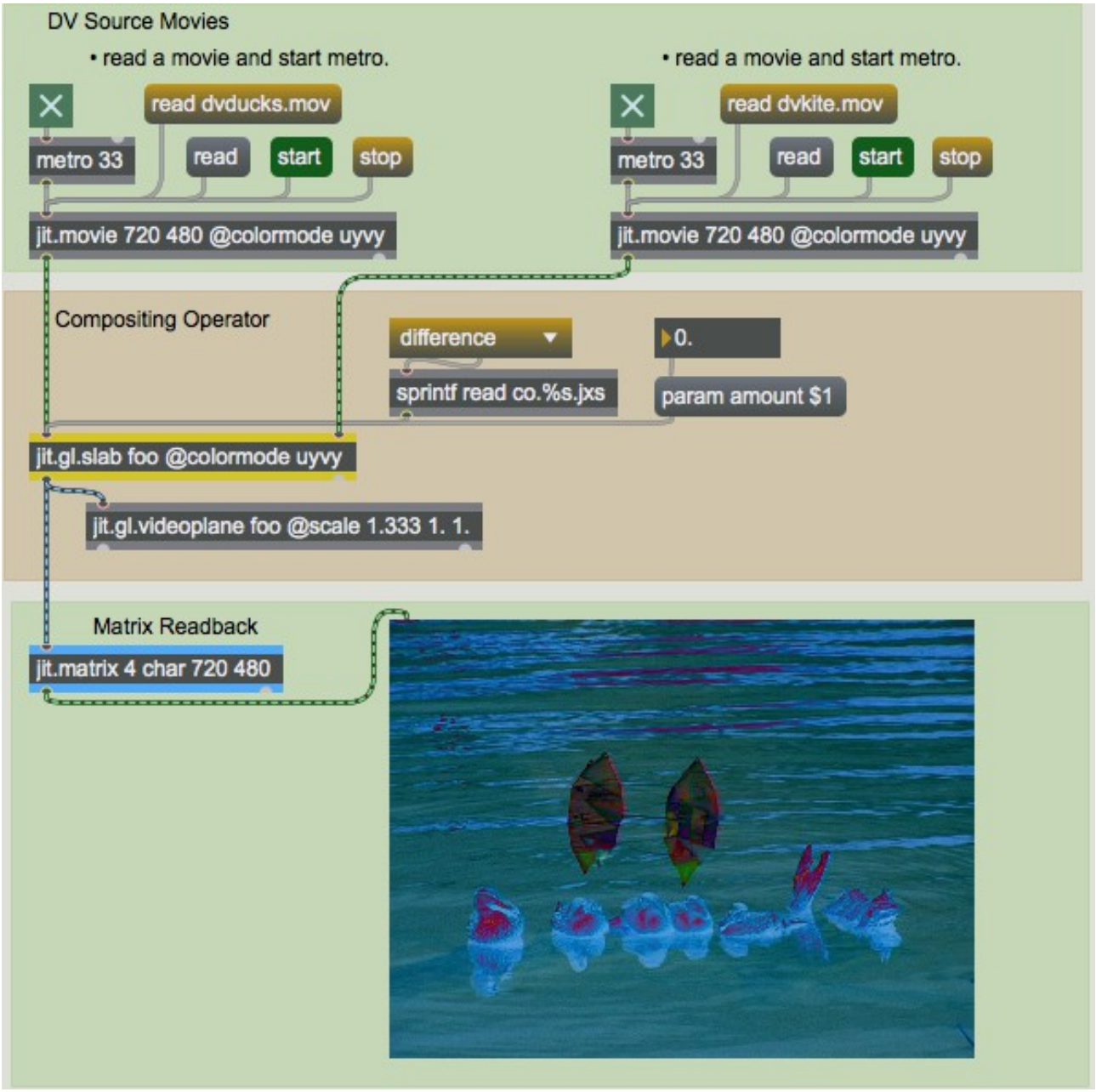
L'objet *jit.gl.slab* gère cette magie, mais comment fonctionne-t-il? L'objet *jit.gl.slab* reçoit en entrée des messages **jit_matrix** ou **jit_gl_texture**, les utilise comme textures d'entrée pour rendre cette géométrie simple avec un shader appliqué, capturant les résultats dans une autre texture qu'il envoie en aval via le message **jit_gl_texture <texturename>**. Le message **jit_gl_texture** fonctionne de la même manière que le message **jit_matrix**, mais au lieu de représenter une matrice résidant dans la mémoire du système principal, il représente une image de texture résidant dans la mémoire du matériel graphique.

L'affichage final de notre vidéo composite est réalisé à l'aide d'un objet *jit.gl.videoplane* qui peut accepter un message **jit_matrix** ou **jit_gl_texture**, en utilisant l'entrée reçue comme une texture pour la géométrie planaire. Cet objet peut éventuellement être connecté à un autre objet comme *jit.gl.gridshape* pour texturer une sphère, par exemple.

Passer de la VRAM à la RAM

Les instances de *jit.gl.texture* qui sont transmises entre les objets *jit.gl.slab* par leur nom font référence à des ressources qui existent sur la carte graphique. C'est très bien lorsque l'application finale de la texture sur une géométrie 3D comme *jit.gl.videoplane* ou *jit.gl.gridshape*, mais que faire si nous voulons utiliser cette image dans une chaîne de traitement basée sur le CPU, ou la sauvegarder sur le disque comme une image ou un fichier vidéo? Nous avons besoin d'un moyen de transférer cette image vers la mémoire du système. L'objet *jit.matrix* accepte le message **jit_gl_texture** et peut effectuer ce qu'on appelle ***texture readback***, qui transfère les données de texture de la carte graphique (VRAM) vers la mémoire système principal (RAM).

• Ouvrez le patch de Tutorial et double-cliquez sur le sub-patch **p slab-readback** pour l'ouvrir. Cliquez sur les *toggle* connectés à l'objet *qmetro* le plus à gauche. Comme dans le dernier patch que nous avons examiné, lisez les films en cliquant sur les boîtes de *message* et démarrez l'objet *metro* sur le côté droit du patch.



Relecture de la matrice depuis le GPU.

Nous voyons ici que l'image est traitée sur le GPU avec l'objet *jit.gl.slab*, puis recopiée vers la RAM en envoyant le message **jit_gl_texture <texturename>** à l'objet *jit.matrix*. Ce processus n'est généralement pas aussi rapide que l'envoi de données à la carte graphique, et ne prend pas en charge la relecture dans un format UYVY à chroma réduit. Cependant, si le GPU effectue une quantité raisonnable de traitement, même avec le transfert du CPU au GPU et vice-versa, cette technique peut être plus rapide que d'effectuer l'opération de traitement équivalente sur le CPU. Il convient de noter que les performances de lecture sont améliorées dans les GPU de dernière génération.

Résumé

Dans ce didacticiel, nous avons vu comment utiliser l'objet *jit.gl.slab* pour utiliser le GPU pour le traitement de données à usage général. Bien que l'accent a été mis sur le traitement des images, les mêmes techniques pourraient être appliquées à des ensembles de données matricielles arbitraires. Nous avons également abordé l'amélioration des performances en utilisant des données **uyvy** à chrominance réduite, ainsi que la manière de relire une image du GPU vers le CPU.