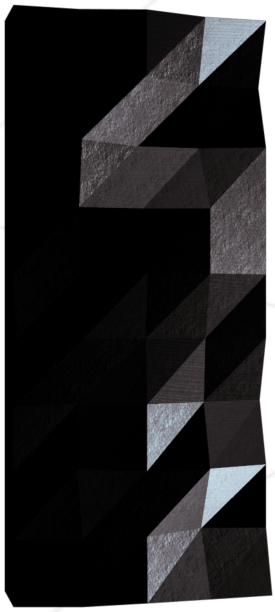


Particle System

Introduction à OpenCL

Rémi DAVID rdavid@student.42.fr
Paul VINCENT pvincent@student.42.fr

Résumé: Ce sujet est une introduction au calcul parallèle sur GPU avec OpenCL.

Table des matières

Chapitre I

Préambule

Compute shaders operate differently from other shader stages. All of the other shader stages have a well-defined set of input values, some built-in and some user-defined. The frequency at which a shader stage executes is specified by the nature of that stage ; vertex shaders execute once per input vertex, for example (though some executions can be skipped via caching). Fragment shader execution is defined by the fragments generated from the rasterization process.

Compute shaders work very differently. The "space" that a compute shader operates on is largely abstract ; it is up to each compute shader to decide what the space means. The number of compute shader executions is defined by the function used to execute the compute operation. Most important of all, compute shaders have no user-defined inputs and no outputs at all. The built-in inputs only define where in the "space" of execution a particular compute shader invocation is.

Therefore, if a compute shader wants to take some values as input, it is up to the shader itself to fetch that data, via texture access, arbitrary image load, shader storage blocks, or other forms of interface. Similarly, if a compute shader is to actually compute anything, it must explicitly write to an image or shader storage block.

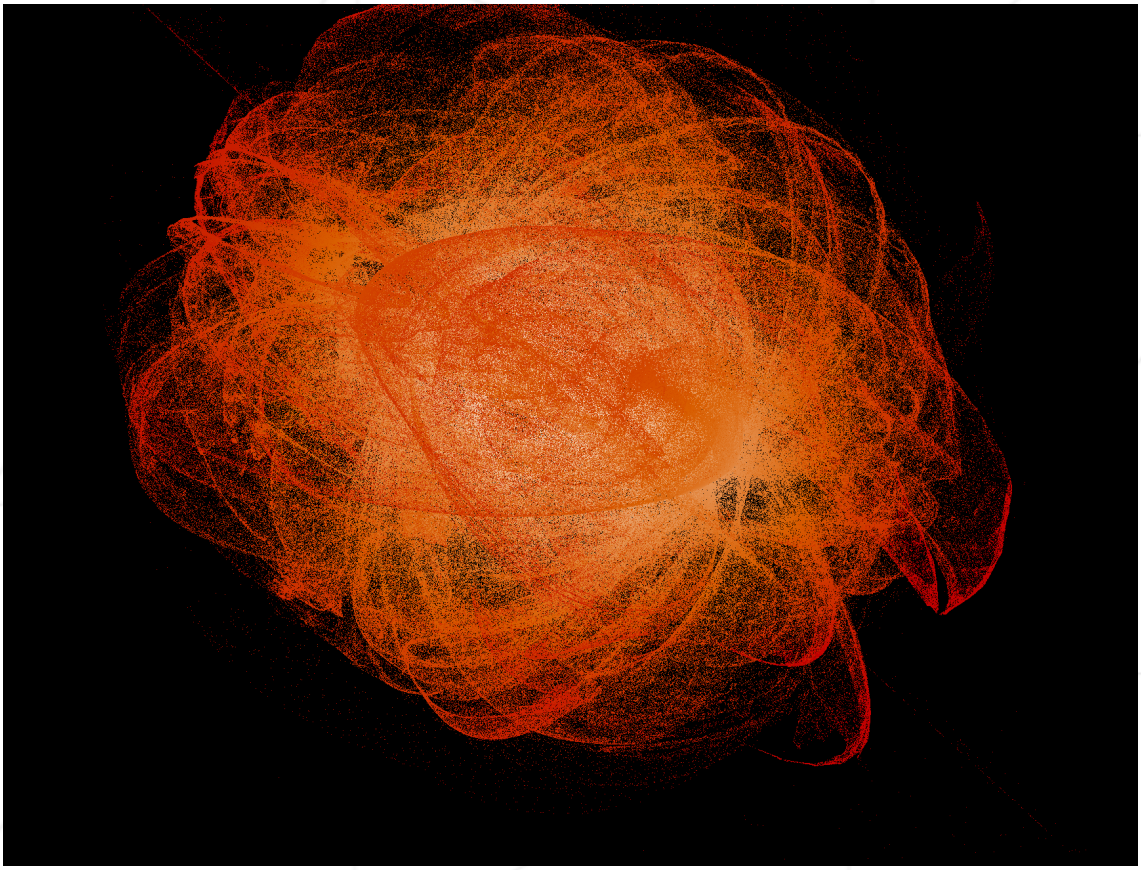
- Core in version 4.5 (2014)
- Core since version 4.3 (2012)

Les compute shaders c'est génial, simple à utiliser et ça fait le café quand il s'agit de mixer le rendu et le calcul. Ils sont présent depuis OpenGL 4.3 (2012 donc). En 2015, MacOSX supporte OpenGL 4.1 MAXIMUM (2009). Donc.. les compute shaders c'est top. Mais pour ce sujet, vous ne pouvez pas les utiliser. Pourquoi ? Parce que Mac.

Chapitre II

Introduction

Dans ce projet, vous allez devoir implémenter un système de particules. Avec un grand nombre de particules. Pour ce faire il vous sera nécessaire d'utiliser le GPU pour sa capacité à paralléliser massivement les calculs.



Voici ce que le projet pourrait donner.

Chapitre III

Objectifs

En réalisant ce projet, vous apprendrez à utiliser l'API OpenCL et son langage de kernels OpenCL C pour animer vos particules.

De plus, vous les afficherez en utilisant l'OpenGL, ce qui vous permettra de consolider vos connaissances de cette API.

Ce sujet est aussi l'occasion de commencer à réfléchir à l'optimisation. N'ayez pas peur de chercher et tester différentes approches. En fait.. vous devrez le faire. Donc faites le, mais ce sera plus simple si vous êtes content de le faire.

Chapitre IV

Consignes générales

- Vous devrez utiliser OpenGL 4.0 au minimum (avec des shaders bien entendu) ainsi qu'OpenCL 1.2 minimum.

- Les particules ne devront jamais être allouées sur la RAM. Jamais. Même à l'initialisation. Jamais. Si vous vous demandez si à tel ou tel moment c'est possible, la réponse est : "Jamais.". Tout sera alloué sur la mémoire vidéo.

- Au niveau des performances, vous devez faire tourner au moins un million de particules à 60 FPS et trois millions (au minimum) à 20 FPS. Faites en sorte que le nombre de particules soit modifiable facilement.

- Vous devez faire de l'interopérabilité et par soucis de propreté, vous devez synchroniser la mémoire entre les différentes API. Les fonctions d'acquisition et de libération sont vos amis.

- Vous êtes libres d'utiliser le langage que vous souhaitez.

- Vous pouvez utiliser la librairie graphique de votre choix (SDL2, SFML, GLFW, MLX, Glut...).

- Un Makefile ou assimilé est requis. Seul ce qui se trouve sur votre dépôt sera évalué.

Chapitre V

Partie obligatoire

Vous devrez implémenter un système de particules. Ces particules seront initialisées dans une sphère ou dans un cube et il sera possible de passer de l'un à l'autre avec des inputs. Elles pourront être attirées par un centre de gravité activable et désactivable grâce à une touche. Celui-ci pourra soit être statique (placé sous la souris au moment de l'input), soit suivre la souris selon l'input.

Elles devront être allouées sur la mémoire du GPU (VRAM). De plus, il sera nécessaire de respecter certaines contraintes de performances malgré le grand nombre de particules : un million de particules à 60 fps et trois millions à au moins 20 (et là, vous comprendrez pourquoi le GPU, c'est le bien).

Vous devrez aussi afficher le nombre de FPS dans la fenêtre (en titre par exemple), cela rendra les tests de performances plus simple.

Pour le plaisir de la rétine il sera nécessaire de mettre des couleurs. Il faudra qu'elles dépendent au moins de la distance des particules par rapport au curseur.

Rappelez vous que la propreté du code pour l'interopérabilité est vraiment importante. Si vous ne voyez pas de quoi on parle, relisez les consignes générales.

`Les particules n'ont pas besoin d'avoir de mesh.`

Chapitre VI

Partie bonus

Quand vous serez sûrs que tout fonctionne et que vos performances sont bonnes, voici quelques bonus qu'il serait bon d'ajouter :

- Une caméra qui peut bouger dans les particules, contrôlable avec WASD et la souris. Parce que c'est la classe.
- Des émetteurs générant des particules ayant une durée de vie limitée.

Il y aura des points dédiés à ces bonus et d'autres à votre créativité.

Pour les bonus, les performances demandées dans les contraintes ne s'appliquent pas... n'abusez pas non plus. 10.000 particules à 20 fps, c'est de l'abus.

Chapitre VII

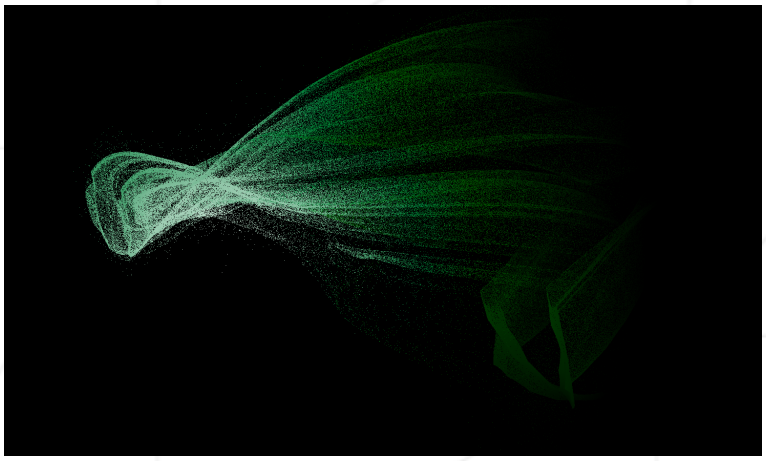
Rendu et peer-évaluation

Rendez-votre travail sur votre dépot `GiT` comme d'habitude. Seul le travail présent sur votre dépot sera évalué en soutenance.
Chaque contrainte sera vérifiée donc soyez sûr de les respecter. Une contrainte non respectée signifiera un échec du projet.

Chapitre VIII

Illustrations

Des particules suivant le curseur :



Et tournant autour d'un centre de gravité :