M1 - Aquisition, Analyse et Traitement d'Images

OpenGL et compute shaders

J.C. Iehl

May 20, 2014

Résumé des épisodes précédents

programmation GPU :

- parallélisme de données,
- espace d'itération, threads et tache associée,
- débit d'instructions,
- debit de données,
- utilisation mémoire partagée.

Bilan

mais :

- processeurs parallèles,
- pas de pile, donc pas de récursion,
- uniquement parallélisme de données.

certains algorithmes sont plus "délicats" à réaliser avec ces contraintes.

Exemple : histogramme

construire l'histogramme d'une image :

- définir un ensemble de classes de couleurs,
- compter le nombre de pixels par classe de couleur.

Exemple : histogramme

```
std::vector<int> T(256, 0);

gk::Image *image= gk::readImage(...);

for(int y= 0; y < image->height; y++)
for(int x= 0; x < image->width; x++)
{
    gk::VecColor color= image->pixel(x, y);
    int classe= 255.f * (color.r + color.g + color.b) / 3.f;
    if(classe > 255) classe= 255;
    T[classe]++;
}
```

Algorithme parallèle

rappel :

- découpage en tâche,
- association des tâches aux threads de calculs.

Détails : opérations atomiques

opérations atomiques ?

- chaque thread exécute `T[classe]++;`
- quel est le résultat ?

rappel : ajouter 1 à une variable

- lire la valeur,
- ajouter 1,
- écrire la nouvelle valeur,
- == 3 opérations exécutées en parallèle sur la même variable...

Détails : opérations atomiques

garantir le résultat :

- utiliser les fonctions dédiées :
- `atomicAdd( )`,
- `imageAtomicAdd( )`
- `atomicCounterIncrement( )`, etc.
- en fonction de la variable à modifier.