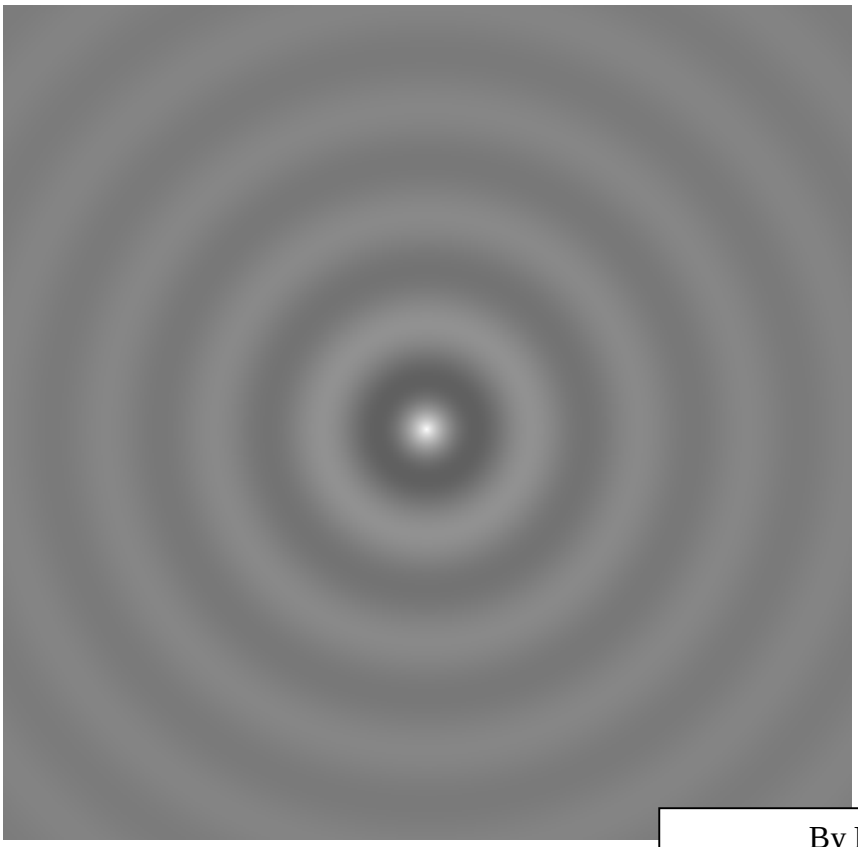
Problèmes Parallelisation



By Professeur
Cédric Bilat
Cedric.Bilat@he-arc.ch

Rippling Effect

Problème Rippling

On considère une image **carrée** de dimension $[\dim x \dim]$. La couleur du pixel (i, j) qui dépend du temps t est définit par

$$color_t(i,j) = 128 + 127 \frac{\cos\left(\frac{d(i,j)}{10} - \frac{t}{7}\right)}{\frac{d(i,j)}{10} + 1} \in [0,255] \in \mathbb{R}$$

Avec

$$\begin{cases} d(i,j) = \sqrt{f_i^2 + f_j^2} \\ f_i = i - \dim/2 \\ f_j = j - \dim/2 \end{cases}$$

Implémenter une animation qui dessine l'image et qui fait bouger $t \in [1,T]$. L'image est en niveau de gris. Si l'image est de type RVBA elle est défini par

$$\begin{cases} R_t(i,j) = color_t(i,j) \\ V_t(i,j) = color_t(i,j) \\ B_t(i,j) = color_t(i,j) \\ A_t(i,j) = 255 \end{cases}$$

Ne calculez $d(i,j)$ qu'une seule fois par pixel !

Implémentation

Utiliser la version « bitmap » de l'API image fourni. Il n'est pas nécessaire ici d'utiliser la version « fonctionnelle ».

Comme dimension d'image, prenez par exemple

```
int w=16*60 ;
int h=w ;
```

Artefact visuel

Synchronisation Verticale

Il est conseillé de désactiver la synchronisation verticale des drivers du GPU. Dans quel cas, vous êtes limités à **60Fps** pour un écran standard de **60Hz.** Une fois la synchronisation désactivée entre le GPU et l'écran, la barrière des 60Fps est levé, mais des artéfacts visuels risquent d'apparaitre, notamment avec le *Rippling*. En effet, l'écran continue à afficher 60 images par secondes, mais le GPU en calcul beaucoup plus. Il y a alors échantillonnage, et seule une partie des images est affichée.

glxgears

Essayer avec *glxgears*. En bureau à distance il faut passer par le wrapper *gl* pour lancer du code openGL. Dans une console (CTRL ALT t)

gl glxgears

Parfois on a l'impression que les engrenages tournent dans le mauvais sens, ou que la vitesse de rotation change au cours du temps. Il n'en est rien. Il s'agit d'artéfacts visuels dont la cause vient des images manquantes.

Speedup

Perturbation

(P1) Interopérabilité OpenGL-Cuda

Avec l'API image fournit, le chef d'orchestre est *OpenGL*. C'est lui qui dicte l'animation. Pour minimiser les transferts mémoire, la zone mémoire contenant l'image est partagée entre *OpenGL* et *Cuda*. Pour des raisons d'intégrité de données, lorsque *OpenGL* parcours l'image pour la rendre, *Cuda* ne peut accéder à cette zone mémoire, et vice versa.

(P2) Bureau à distance

Le bureau à distance ne peut offrir le même niveau de fps qu'un bureau local.

(P3) Multi-utilisateur

Plusieurs utilisateur peuvent utilisez les ressources du serveur en même temps

Solutions

(S1) MOO-only, free-OpengL

Effectuez un rendu « à sec » sans OpenGL en n'utilisant que le modèle MOO de votre implémentation. Laisser la vue de côté ! (cf projet Tuto)

(S2) Device

Dans le cas d'implémentation GPU, changez le device (dans main.cpp), et appropriez-vous un GPU que de vos collègues n'utilisent pas !

Fps

Référentiel :

GCC mono-thread :

- Mettez en commentaire #pragma omp paralel for de la version **forAutoOMP**
- Passer en mode **OMP_FORAUTO** pour le « parallelPatern »

Comparaison :

(C1) **gcc** parallel entrelacement only
(C2) **gcc** parallel forAuto only

(C3) **intel** parallel entrelacement only
(C4) **intel** parallel forAuto only

(C5) **cuda** 1 seul thread
(C6) **cuda** multi-thread

Dans le cas cuda multithread, effectuez des variations de

- dg
- db

et observer la grande volatilité des résultats !!!

Trouver le dg,db optimal, comparer à vos collègues ! Sur cuda1 (tessla m2090), vous devrez pouvoir obtenir environ 9000 fps et sur cuda2 (K6000) vous devriez obtenir environ 22000 fps.

Note

Le meilleur dg,db dépend de

- L'algorithme
- Du GPU

Vous changez l'un des deux, il faut « tuner » à nouveau le meilleur dg,db.

Présentation des résultats

Utiliser le classeur Excel : *speedup_simple_tp_graphique.xlsx*

End