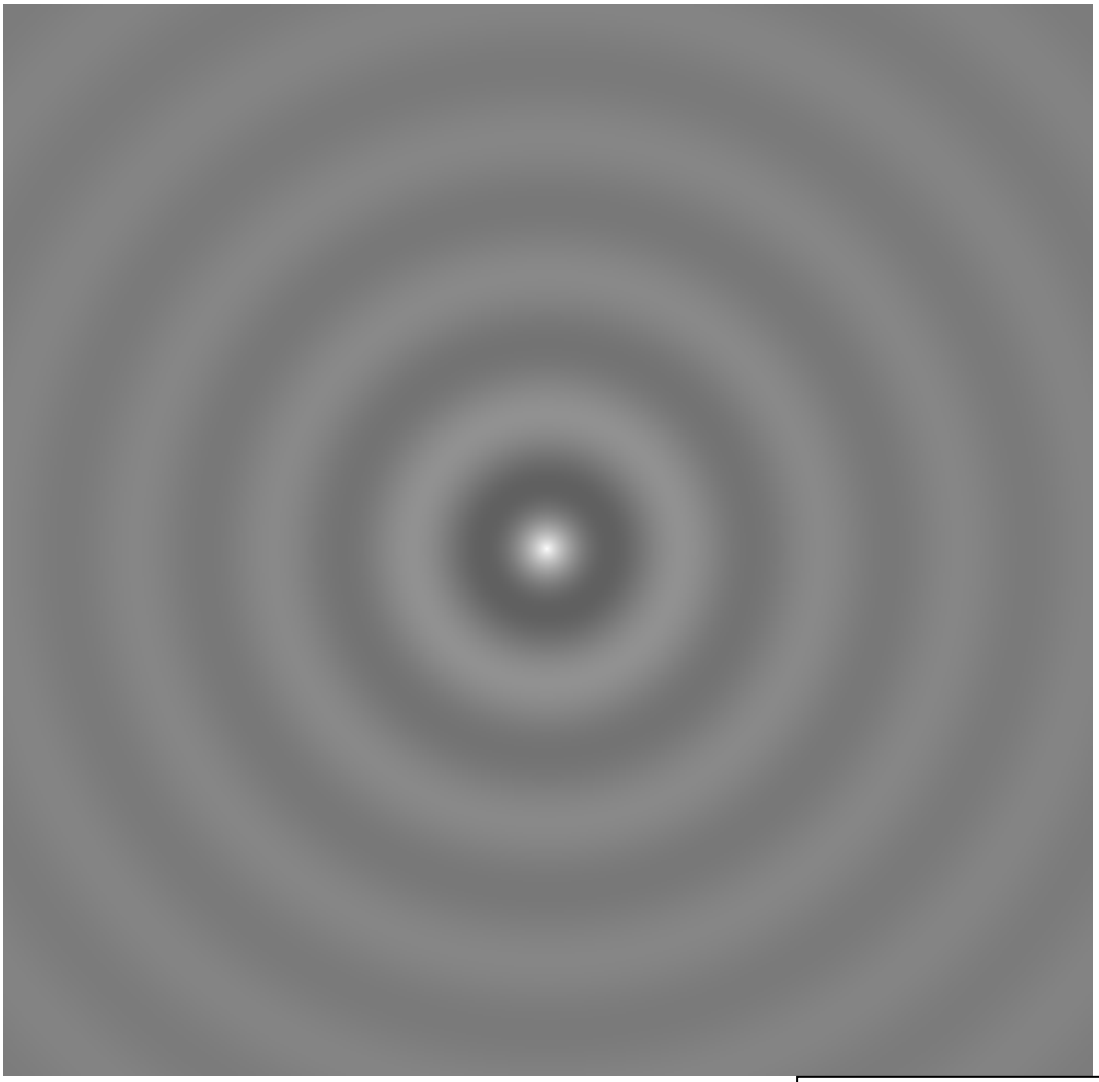
GPGPU



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

Rippling

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/zoomable ».

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.

Optimisation Cuda

(O1) Utilisez dans *main.cpp* le launcher en mode BARIVOX et/ou BRUTEFORCE pour trouver les meilleurs *(dg,db).*

Vous pouvez customiser le BARIVOX et BRUTEFORCE dans

mainBarivox.cpp
mainBruteForce.cpp

Par exemple, changer le nombre d'itération pour ajuster la longueur du test, ou choisissez un maillage de recherche plus fin ou plus grand !

(O2) Essayer aussi des valeusr de *(dg,db).*par vous même

(O3) Vérifiez que toute votre chaine de calcul est en ***float***. Le moindre dérapage à cet effet, et vos performances peuvent être réduites d'un facteur 4 !! Un GPU calcul *nativement* en float. Il est presque 2x plus performant dans ce mode. Les changements de type

double < --> float

coute aussi des cycles d'horloges non négligeables dans le cas de calcul mixte mélangeant des *float* et des *double*.

Tips :

cos → cos**f**
sin → sin**f**
sqrt → sqrt**f**
pow(x,2) x*x

4+3sin(x) -> 4.0**f** + 3.0 * sin**f**(x)

…

Pensez-y dans vos prochains TP :

Note

Le meilleur *(dg,db).*dépend

- De l'algorithme
- Du GPU
- Du type de mémoire utilisée (GM,SM,CM, …)

Vous changez l'un deux, il faut « tuner » à nouveau le meilleur *(dg,db).*.
A cet effet, trouver le meilleur *(dg,db).*pour les serveurs :

- Cuda1
- Cuda2

- Cuda3

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) Animable

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). Utilisez à cet effet le *launcher* en mode ***ANIMABLE*** dans *main.cpp*.

(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 ! Après avoir validé sous forme graphique votre TP, utilisez dans *main.cpp* le *launcher* en mode ***ANIMABLE***

Référentiel

Comme référentiel, utiliser un code

gcc séquentiel

A cet effet, mettez en commentaire

```
#pragma omp paralel for
```

de la version **forAutoOMP**

Indication

Avec un *(dg,db)*.optimal vous devriez quelque chose comme :

Cuda1 : GPU TesslaM2090	environ 10000 fps
Cuda2 : GPU M6000	environ 36000 fps

Comparer avec vos les fps de vos collègues !

Présentation des résultats

Utiliser le classeur Excel : *speedup_simple_tp_graphique.xlsx*

End