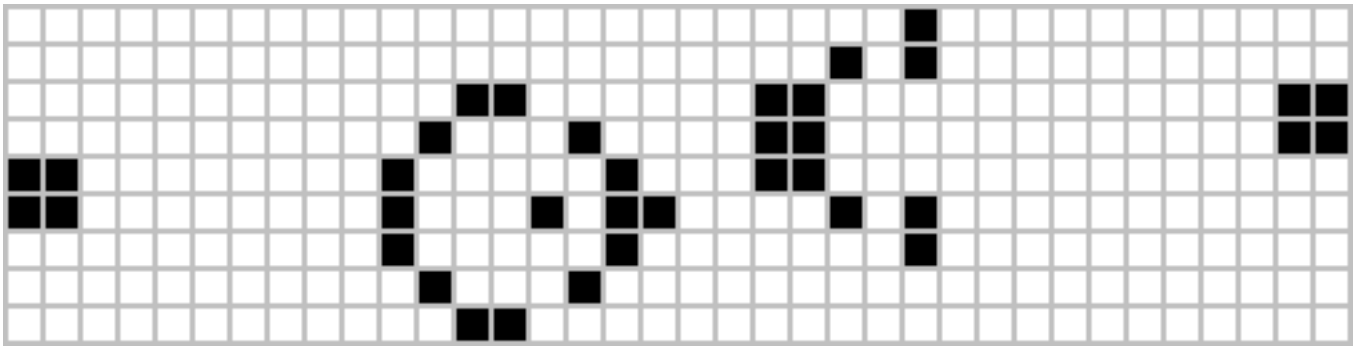
Jeux de la vie en *OpenCL*

Pascal BALLET

March 9, 2016



1 Préambule

Le Jeu de la vie est un automate cellulaire créé dans les années 1960 par John Conway. A l'aide de règles simples, il est capable d'exprimer des phénomènes complexes comme la reproduction, la disparition et le déplacement de formes.

Comme tout automate cellulaire, dont l'un des père fondateur est John Von Neumann, son état est représenté par une matrice. Dans notre cas, elle sera à 2 dimensions de taille 512 x 256.

Chaque cellule de la matrice ne prend que deux valeurs possibles : mort (valeur 0 = noir dans Matrix Studio) ou vivant (valeur 0xFFFFFF = blanc). Le voisinage d'une cellule (morte ou vivante) prend en compte les 4 côtés + les 4 diagonales.

Les règles sont les suivantes :

1. Si une *cellule morte* est entourée de 2 ou 3 cellules vivantes, elle devient vivante (reproduction),
2. Si une *cellule vivante* est entourée de 0 ou 1 cellule vivante, elle meurt (isolement),
3. Si une *cellule vivante* est entourée de 4 ou plus de cellules vivantes, elle meurt également (surpopulation),
4. Si une *cellule vivante* est entourée de 2 ou 3 cellules vivantes, elle reste vivante (survie).

2 Démarche

2.1 A) Matrices

Pour développer un programme OpenCL du Jeu de la vie, il faut 2 matrices de type *Integer* et de taille 512 x 256.

1. La première matrice, *MatEtat_t* correspond à l'état courant (à l'instant t) de l'automate. Pour avoir une matrice courante non vide, on pourra utiliser le logiciel *Paint* avec une image de taille 512 x 256 dont le fond est noir et utiliser l'outils de dessin *Spray* pour ajouter aléatoirement des points blancs (cellules vivantes). Ensuite, dans *Matrix Studio*, il faudra sélectionner *Mat_t1* et cliquer sur le bouton avec l'icone *Crayon* pour importer cette image (au format *png*).
2. La deuxième matrice, *MatEtat_t1* est l'état futur (à l'instant t+1) de l'automate. Autrement dit, c'est grâce à *MatEtat_t* et aux règles que *MatEtat_t1* est remplie. De plus, une fois que *MatEtat_t1* est entièrement remplie, elle devient le nouvel état courant et est donc recopiée intégralement dans *MatEtat_t*.

2.2 B) Kernels

Pour faire ce programme, il faut aussi 2 *Kernels* (fonctions *OpenCL*).

1. Le Kernel *KerVie* qui va appliquer les règles du Jeu de la vie sur chaque cellule de *MatEtat_t* et qui va écrire 0 (morte) ou 0xFFFFFF (vivante) dans *MatEtat_t1*.
2. Le Kernel *kerCopie* qui va simplement copier *MatEtat_t1* dans *MatEtat_t*, une fois que *KerVie* est terminé.

2.3 C) Scheduler

KerVie doit s'exécuter sur toute la matrice *MatEtat_t* : il faut donc ajouter une *Task* (Tâche) dont la matrice des processus est de 512 x 256 et qui exécute *KerVie*.

De plus, il faut une autre *Task* qui exécute *KerCopie*.

Dans *Matrix Studio*, une fois la dernière *Task* exécutée, le cycle d'exécution reprend à la première *Task*, et ce jusqu'à ce que l'utilisateur appuie sur le bouton *Pause* ou *Stop*.

3 Questions

Q1) Lancer *Matrix Studio* et créer les 2 matrices de type *Integer MatEtat_t* et *MatEtat_t1*. Lancer le logiciel *Paint*, créer une image de 512 x 256 avec un fond noir et peindre avec le *Spray* des points blancs au hasard. Enregistrer l'image et l'importer dans *MatEtat_t*. Penser à enregistrer régulièrement votre programme (*CTRL+S*).

Q2) Créer les *Kernels KerVie* et *KerCopie*. Pour information *KerVie* doit comporter environ 50 lignes et *KerCopie* environ 5 lignes seulement.

Q3) Ajouter les *Tasks* qui vont exécuter vos 2 *Kernels* dans le bon ordre.

Q4) Tester votre programme et noter les formes intéressantes qui apparaissent selon la norme suivantes :

- Formes fixes
- Formes à 2 états alternatifs
- Formes se déplaçants

Q5) Chercher sur Internet les noms de ces formes (Carré, TicTac, Glider, etc)

Q6) Chercher sur Internet comment créer une forme *Glider Gun* (Canon à Planneur). Ajouter cette forme à votre image *png* et l'importer dans *MatEtat_t*. Tester votre *Glider Gun*.

Q7) De la même manière, ajouter une forme de type *SpaceShip* et tester la.