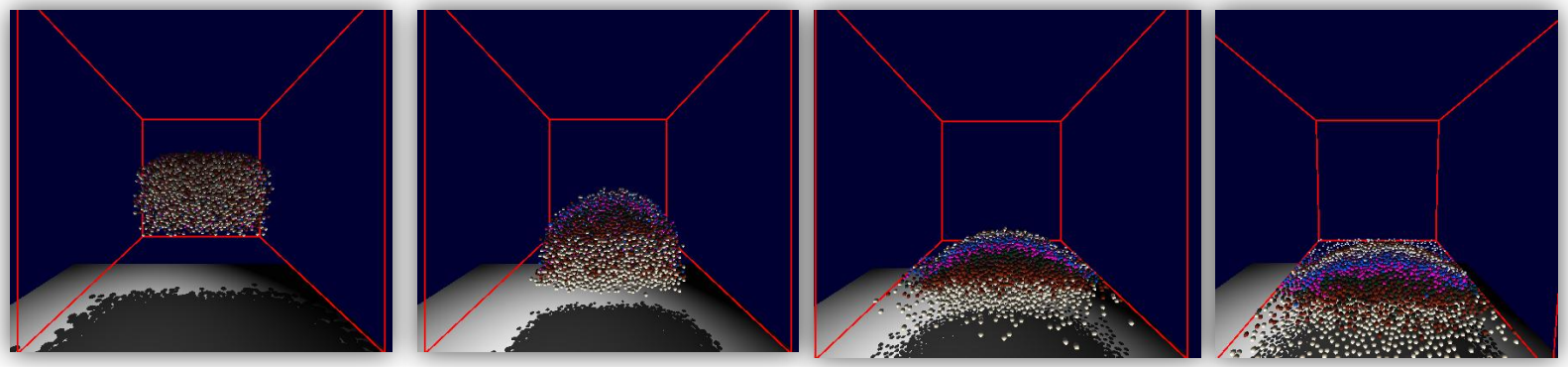
Sliced Grid on GPU

Paul Demeulenaere

Sliced Grid on GPU

Paul Demeulenaere

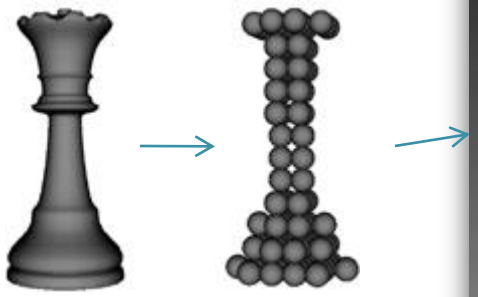


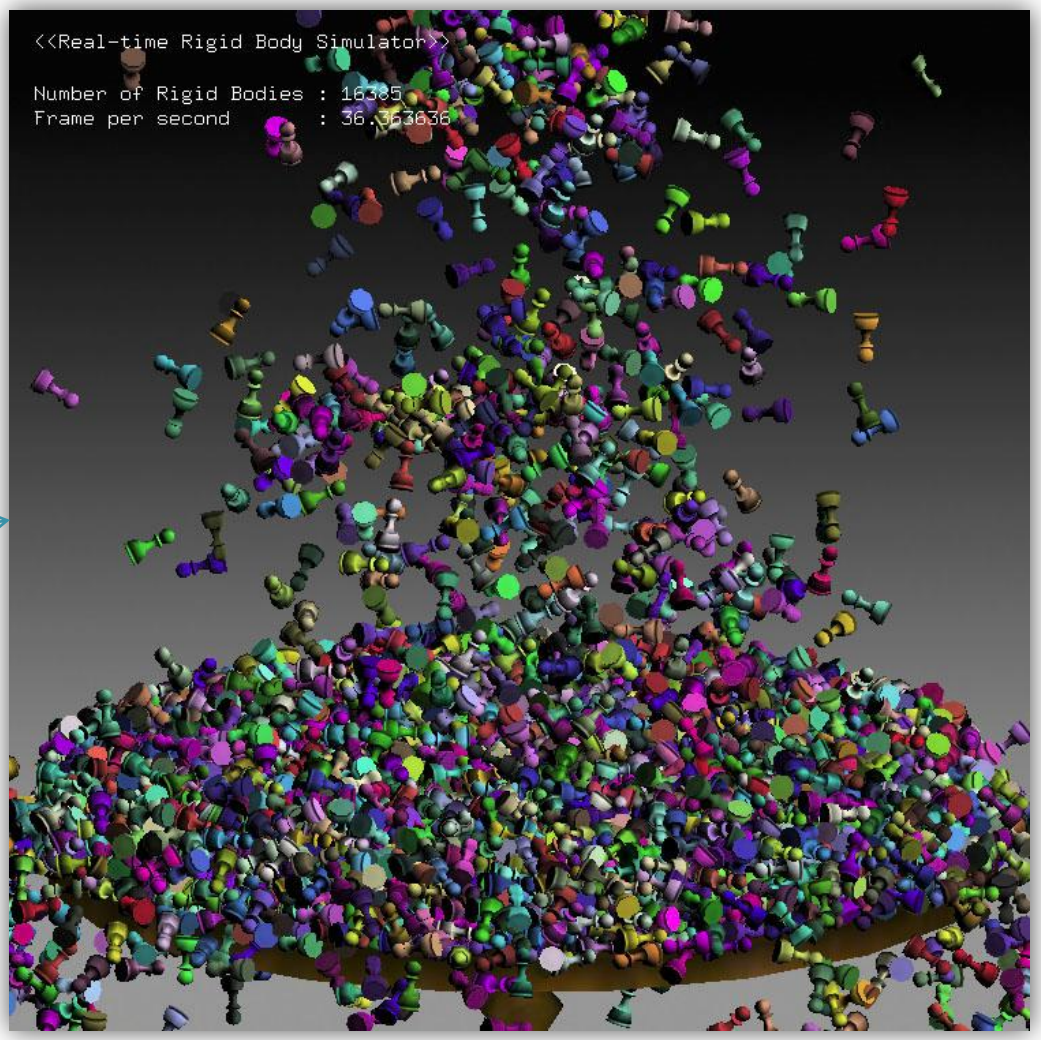
Introduction

- GPGPU est de plus en plus répandu notamment pour les simulations physiques

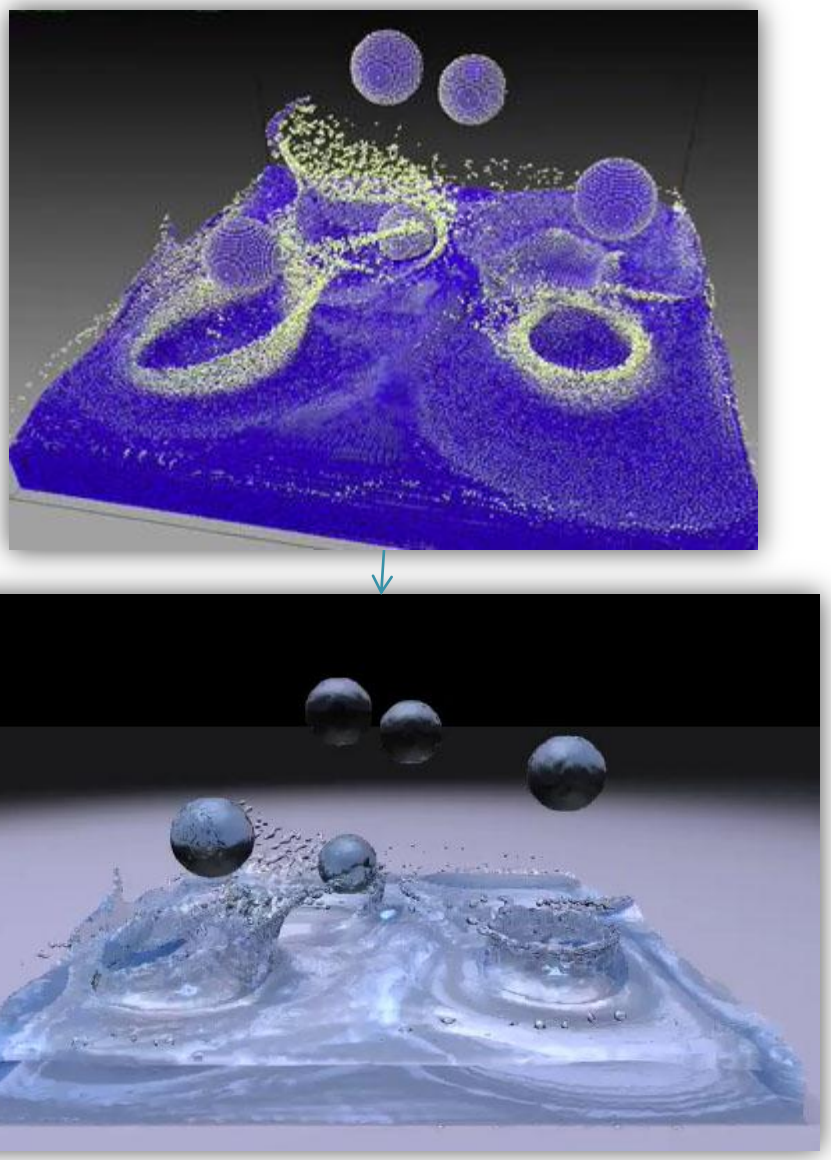
- Particule-based simulation est une des méthodes de simulation physique
  - Rigid body simulation
  - Smoothed-particle hydrodynamics
  - Flock simulation

Introduction

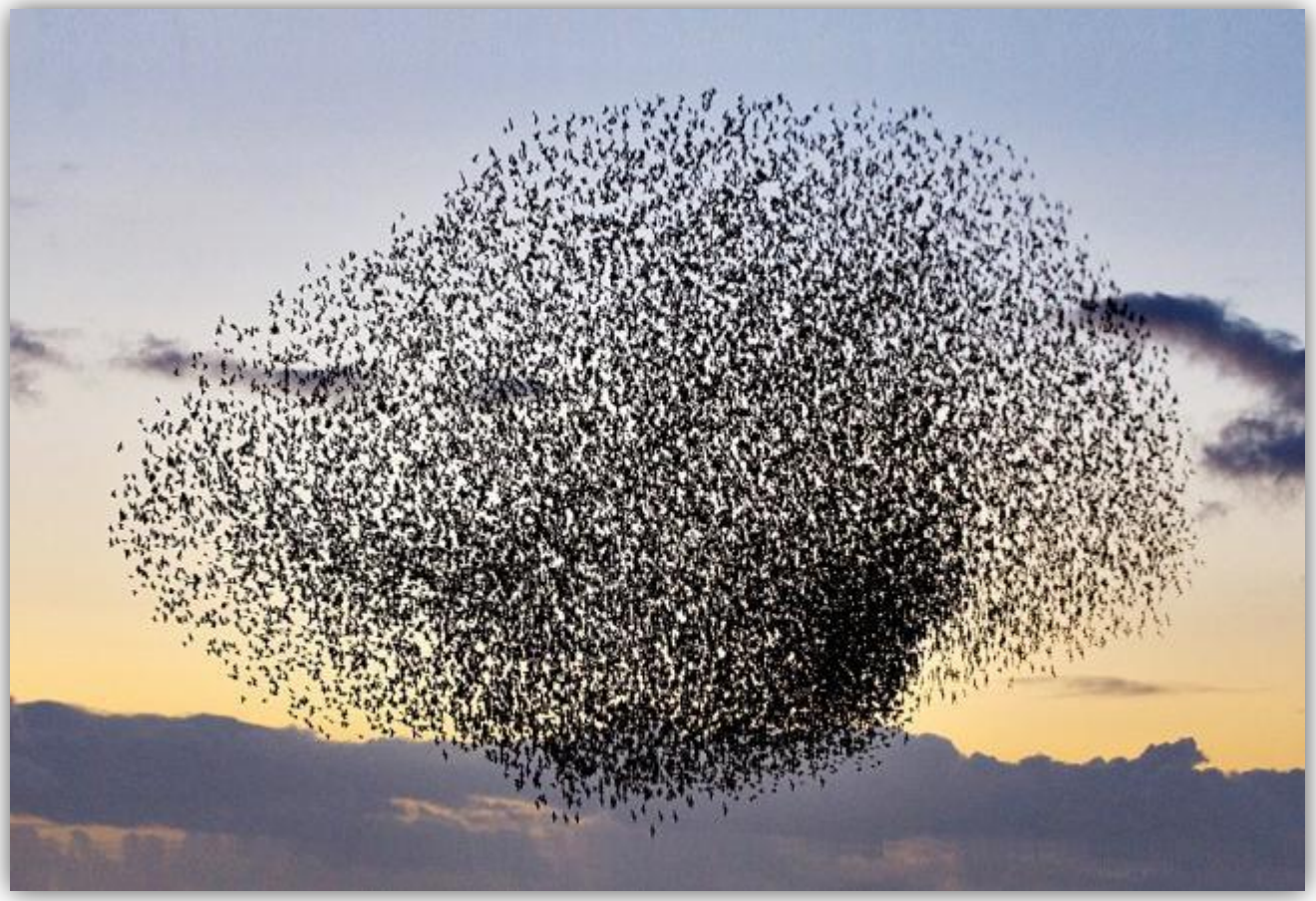




Introduction



Introduction



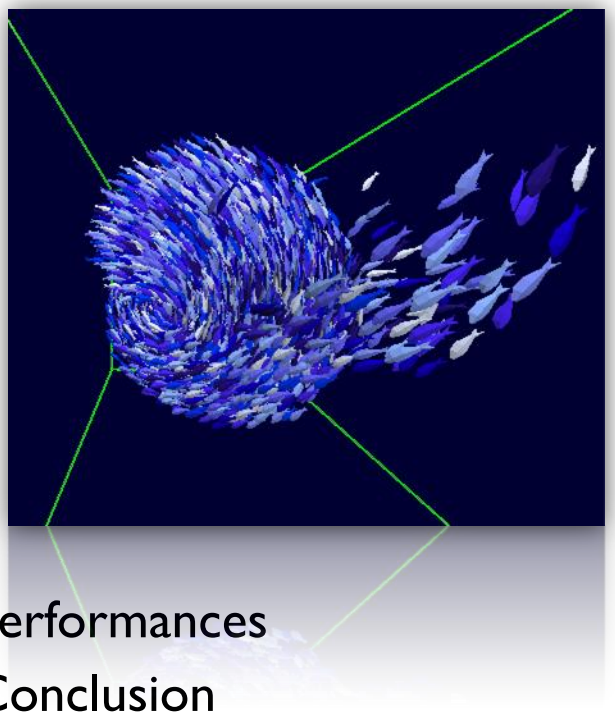
Introduction

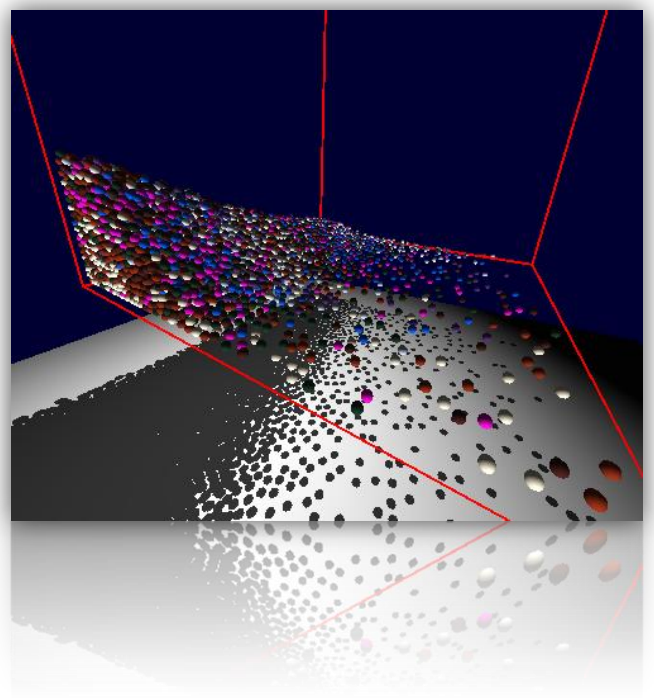
- Pour ces simulations, on a besoin d'accéder facilement aux informations des voisins d'une particule pour éviter la complexité quadratique

- Découpage de l'espace
  - En grille
    - Uniforme
    - Tranchée (sliced)
    - « Adaptable »
  - En arbre
    - Pas forcement adapté dans cette situation

Plan

- Uniform grid
- Sliced grid
- *Adaptative grid*





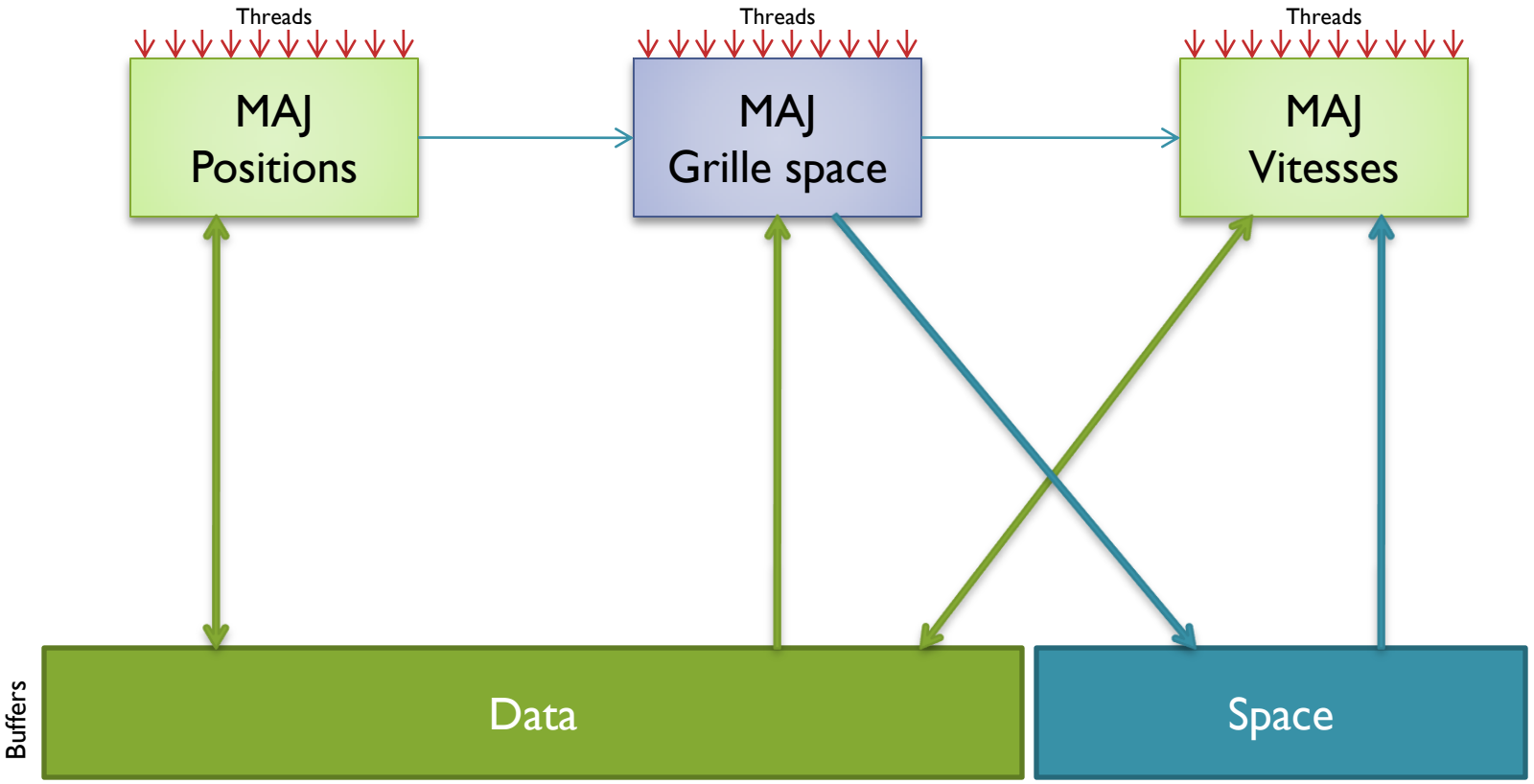
- Performances
- Conclusion

Uniform Grid

- Découpage de base mais reste plutôt efficace
  - Tableau1D (Data) contenant les informations sur chaque particule
    - Position
    - Vitesse
    - …
    - Taille dans notre cas : 2^{13} particules de 2 float4
  - Tableau3D aplati (Space) contenant les cases ou voxels de l'espace (id de Data si occupé, NULL sinon)
    - Contient id de Data si occupé, NULL sinon
    - 2^{18} voxels de 1 uint

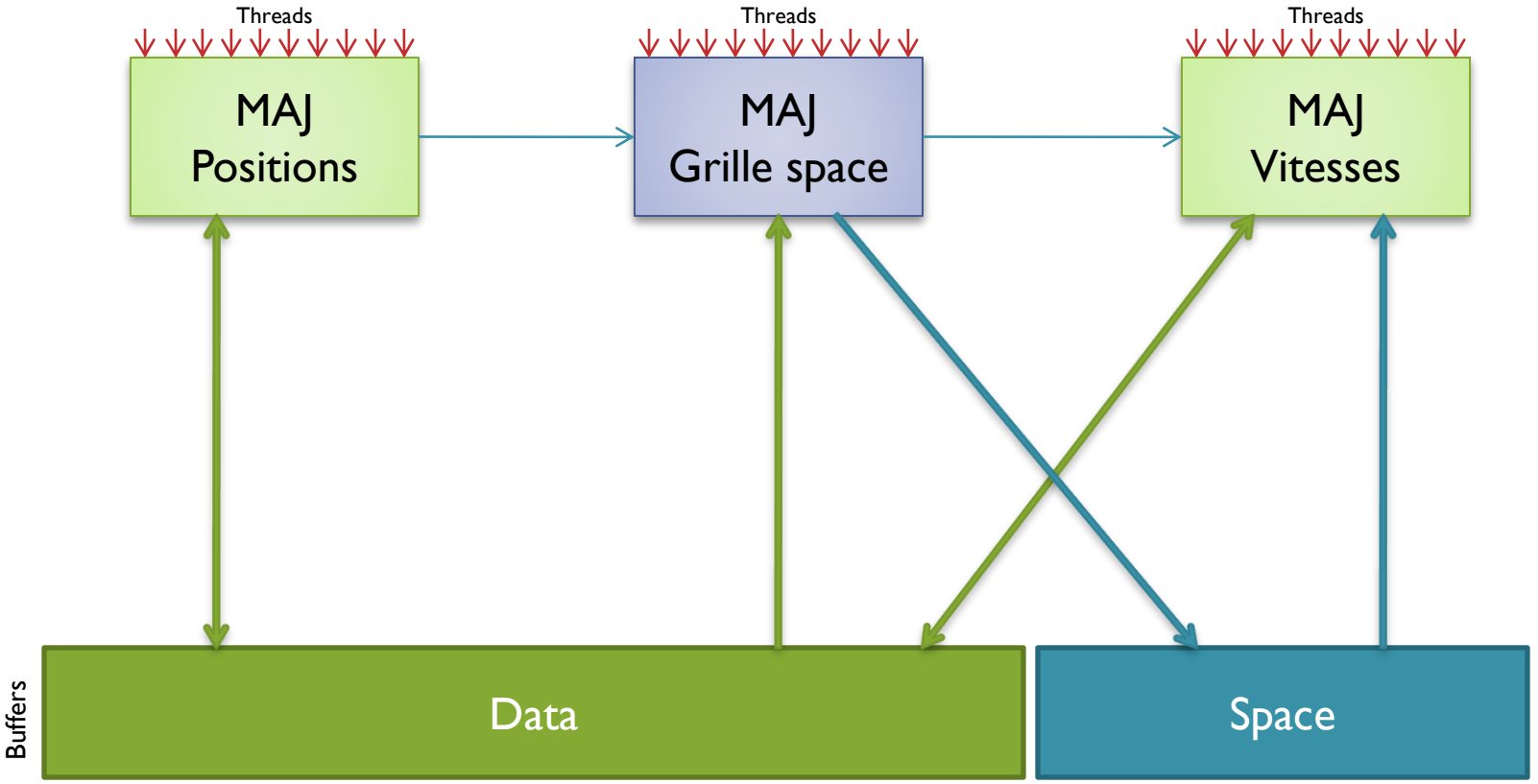
Uniform Grid

- Chaque bloc correspond à un compute shader 4.0



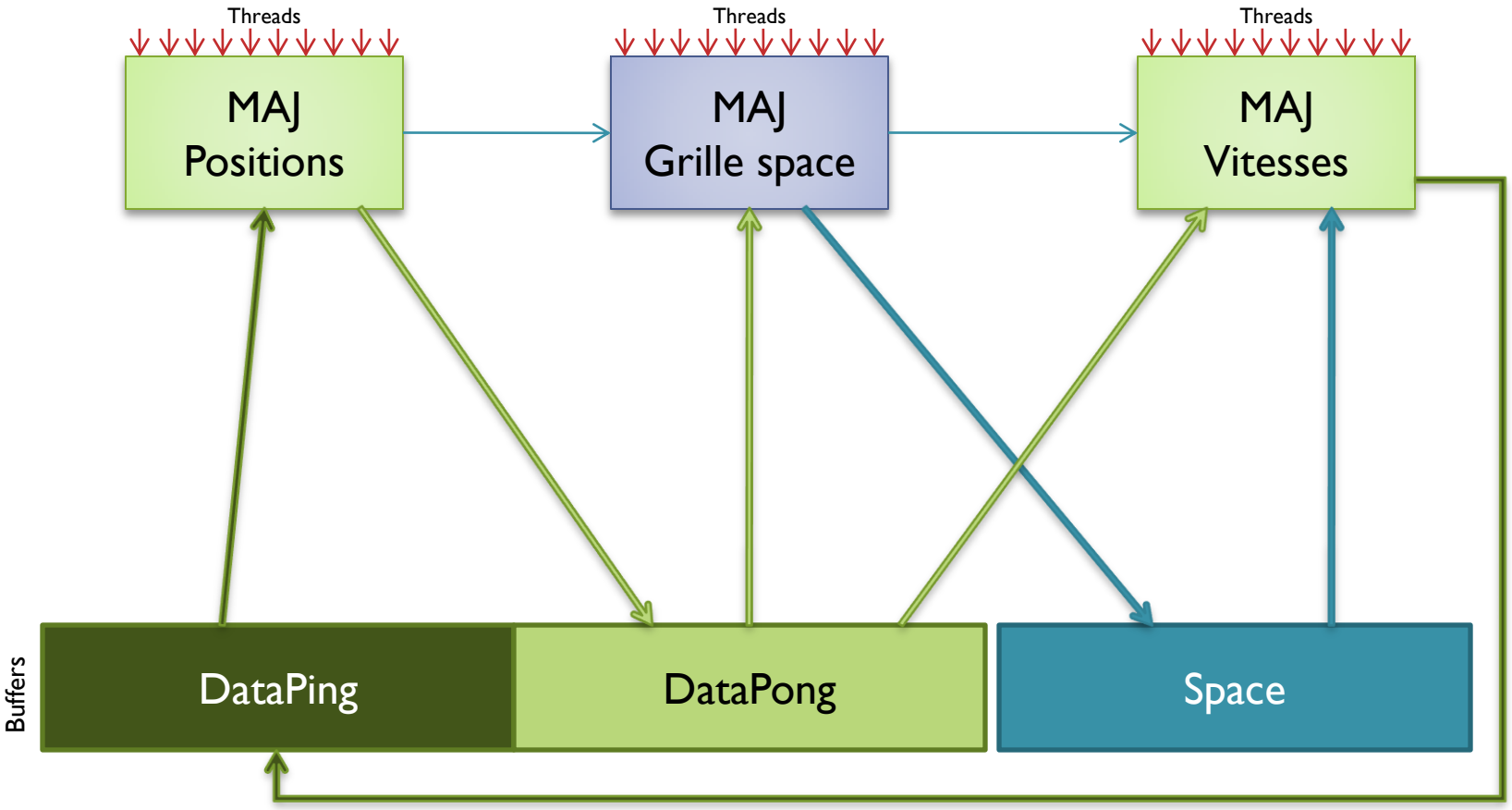
Uniform Grid

- Problèmes :
  - Lire dans un UAV (unordered access view) qui est potentiellement en cours de modification n'est pas une bonne idée
  - La lecture dans un SRV (sharder ressource view) est plus efficace sur un même buffer



Uniform Grid

- Solution
  - On crée deux buffers jumeaux DataPing et DataPong



Uniform Grid

- Limitations du buffer space
  - Espace figé (même si recentré)
  - Mémoire allouée importante
  - Indices potentiellement dispersés

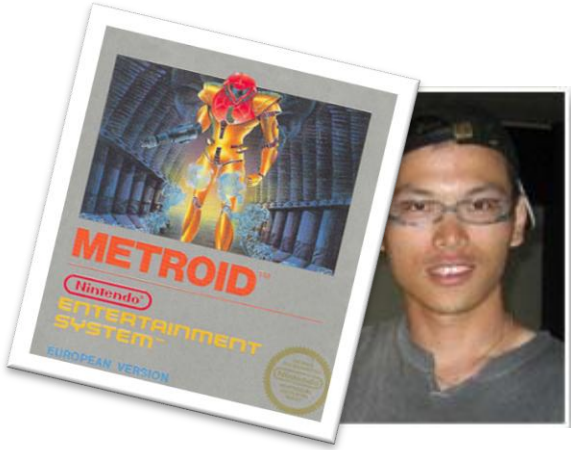
- Questions avant la suite ?

Sliced Grid

- Concept décrit dans le shaderX7 par Takahiro Harada (website)
- Présentation (rapide) de Takahiro Harada

Sliced Grid

- Concept décrit dans le shaderX7 par Takahiro Harada (website)
- Présentation de Takahiro Harada
  - Metroid – 1986



Sliced Grid

- Concept décrit dans le shaderX7 par Takahiro Harada (website)
- Présentation de Takahiro Harada
  - Metroid – 1986
  - Famicon Wars - 1988

Sliced Grid

- Concept décrit dans le shaderX7 par Takahiro Harada (website)
- Présentation de Takahiro Harada
  - Metroid – 1986
  - Famicon Wars – 1988
  - Super Mario Land – 1989

Sliced Grid

- Concept décrit dans le shaderX7 par Takahiro Harada (website)
- Présentation de Takahiro Harada
  - Metroid – 1986
  - Famicon Wars – 1988
  - Super Mario Land – 1989
  - Dr Mario - 1990

Sliced Grid

- Concept décrit dans le shaderX7 par Takahiro Harada (website)
- Présentation de Takahiro Harada
  - Metroid – 1986
  - Famicon Wars – 1988
  - Super Mario Land – 1989
  - Dr Mario – 1990
  - Metroid II – 1991

Sliced Grid

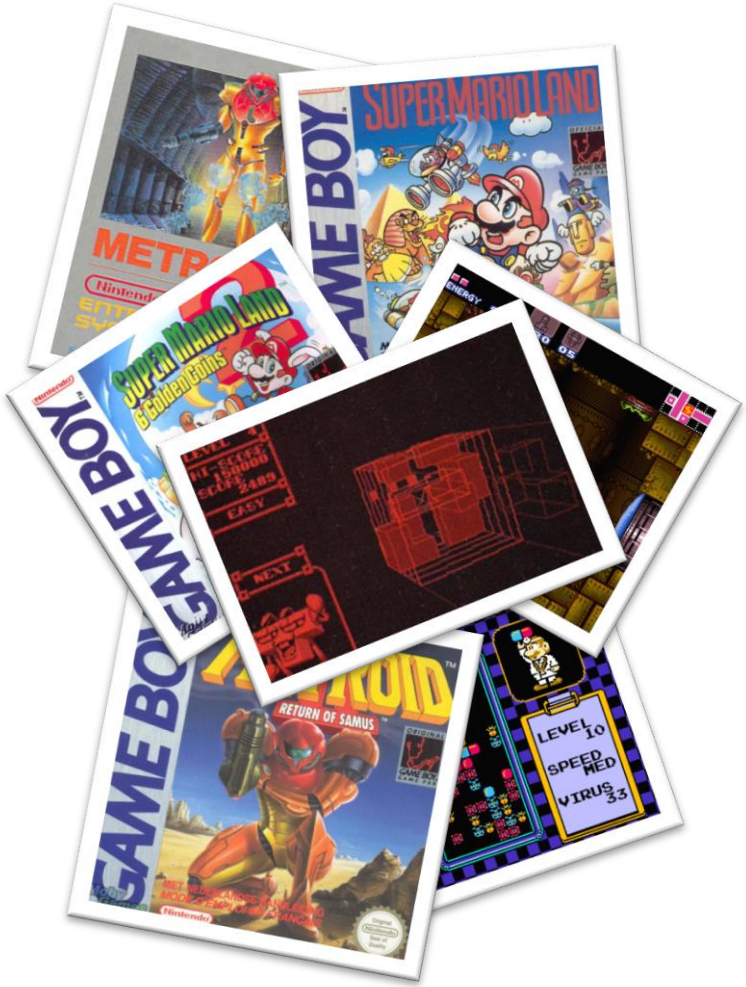
- Concept décrit dans le shaderX7 par Takahiro Harada (website)
- Présentation de Takahiro Harada
  - Metroid – 1986
  - Famicon Wars – 1988
  - Super Mario Land – 1989
  - Dr Mario – 1990
  - Metroid II – 1991
  - Super Mario Land II - 1992

Sliced Grid

- Concept décrit dans le shaderX7 par Takahiro Harada (website)
- Présentation de Takahiro Harada
  - Metroid – 1986
  - Famicon Wars – 1988
  - Super Mario Land – 1989
  - Dr Mario – 1990
  - Metroid II – 1991
  - Super Mario Land II – 1992
  - Super Metroid - 1994

Sliced Grid

- Concept décrit dans le shaderX7 par Takahiro Harada (website)
- Présentation de Takahiro Harada
  - Metroid – 1986
  - Famicon Wars – 1988
  - Super Mario Land – 1989
  - Dr Mario – 1990
  - Metroid II – 1991
  - Super Mario Land II – 1992
  - Super Metroid – 1994
  - 3D Tetris - 1996



Sliced Grid

- Concept décrit dans le shaderX7 par Takahiro Harada (website)
- Présentation de Takahiro Harada
  - Metroid – 1986
  - Famicon Wars – 1988
  - Super Mario Land – 1989
  - Dr Mario – 1990
  - Metroid II – 1991
  - Super Mario Land II – 1992
  - Super Metroid – 1994
  - 3D Tetris – 1996
  - Pokémon – 1996

Sliced Grid

- Concept décrit dans le shaderX7 par Takahiro Harada (website)
- Présentation de Takahiro Harada
  - Metroid – 1986
  - Famicon Wars – 1988
  - Super Mario Land – 1989
  - Dr Mario – 1990
  - Metroid II – 1991
  - Super Mario Land II – 1992
  - Super Metroid – 1994
  - 3D Tetris – 1996
  - Pokémon – 1996
  - Fire Emblem – 1996

Sliced Grid

- Concept décrit dans le shaderX7 par Takahiro Harada (website)
- Présentation de Takahiro Harada
  - Metroid – 1986
  - Famicon Wars – 1988
  - Super Mario Land – 1989
  - Dr Mario – 1990
  - Metroid II – 1991
  - Super Mario Land II – 1992
  - Super Metroid – 1994
  - 3D Tetris – 1996
  - Pokémon – 1996
  - Fire Emblem – 1996
  - Wario Land 4 - 2001

Sliced Grid

- Concept décrit dans le shaderX7 par Takahiro Harada (website)
- Présentation de Takahiro Harada
  - Metroid – 1986
  - Famicon Wars – 1988
  - Super Mario Land – 1989
  - Dr Mario – 1990
  - Metroid II – 1991
  - Super Mario Land II – 1992
  - Super Metroid – 1994
  - 3D Tetris – 1996
  - Pokémon – 1996
  - Fire Emblem – 1996
  - Wario Land 4 – 2001
  - F-Zero Maximum Velocity – 2001

Sliced Grid

- Concept décrit dans le shaderX7 par Takahiro Harada (website)
- Présentation de Takahiro Harada
  - Metroid – 1986
  - Famicon Wars – 1988
  - Super Mario Land – 1989
  - Dr Mario – 1990
  - Metroid II – 1991
  - Super Mario Land II – 1992
  - Super Metroid – 1994
  - 3D Tetris – 1996
  - Pokémon – 1996
  - Fire Emblem – 1996
  - Wario Land 4 – 2001
  - F-Zero Maximum Velocity – 2001
  - Metroid Prime – 2002

Sliced Grid

- Concept décrit dans le shaderX7 par Takahiro Harada (website)
- Présentation de Takahiro Harada
  - Metroid – 1986
  - Famicon Wars – 1988
  - Super Mario Land – 1989
  - Dr Mario – 1990
  - Metroid II – 1991
  - Super Mario Land II – 1992
  - Super Metroid – 1994
  - 3D Tetris – 1996
  - Pokémon – 1996
  - Fire Emblem – 1996
  - Wario Land 4 – 2001
  - F-Zero Maximum Velocity – 2001
  - Metroid Prime – 2002
  - Université de Kyoto 2003/2006

Sliced Grid

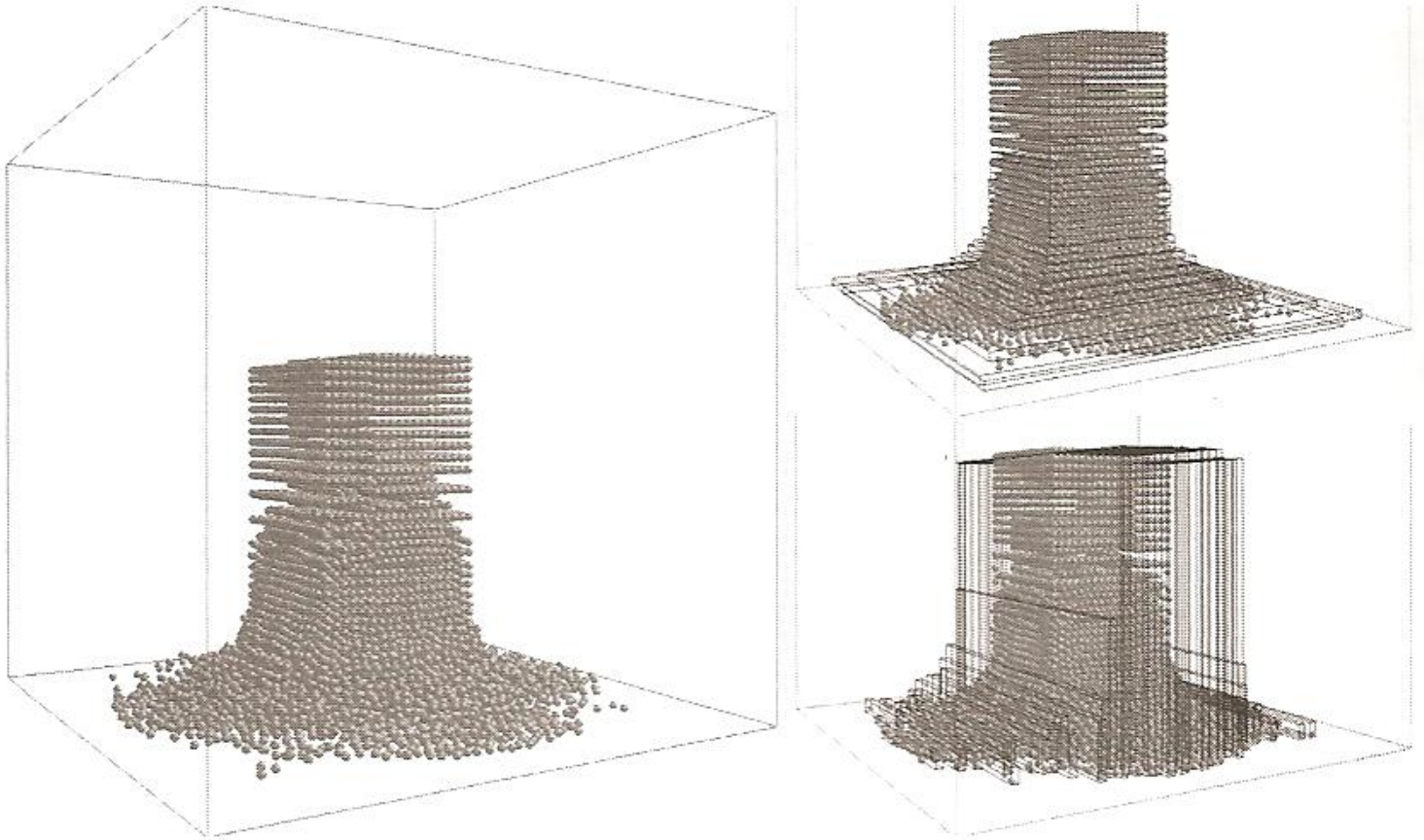
- Concept décrit dans le shaderX7 par Takahiro Harada (website)
- Présentation de Takahiro Harada
  - Metroid – 1986
  - Famicon Wars – 1988
  - Super Mario Land – 1989
  - Dr Mario – 1990
  - Metroid II – 1991
  - Super Mario Land II – 1992
  - Super Metroid – 1994
  - 3D Tetris – 1996
  - Pokémon – 1996
  - Fire Emblem – 1996
  - Wario Land 4 – 2001
  - F-Zero Maximum Velocity – 2001
  - Metroid Prime – 2002
  - Université de Kyoto 2003/2006
  - Université de Tokyo 2006/2008

Sliced Grid

- Concept décrit dans le shaderX7 par Takahiro Harada (website)
- Présentation de Takahiro Harada
  - Metroid – 1986
  - Famicon Wars – 1988
  - Super Mario Land – 1989
  - Dr Mario – 1990
  - Metroid II – 1991
  - Super Mario Land II – 1992
  - Super Metroid – 1994
  - 3D Tetris – 1996
  - Pokémon – 1996
  - Fire Emblem – 1996
  - Wario Land 4 – 2001
  - F-Zero Maximum Velocity – 2001
  - Metroid Prime – 2002
  - Université de Kyoto 2003/2006
  - Université de Tokyo 2006/2008
  - Ingénieur chez Havok à Dublin

Sliced Grid

- Concept décrit dans le shaderX7 par Takahiro Harada (website)
- Découpage de l'espace en plans de tailles variables



Sliced Grid

- Déroulement :
  - Choisir un axe a parmi { x, y, z } : ici, on choisit z

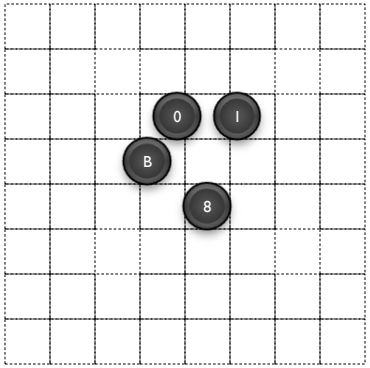
Sliced Grid

- Déroulement :
  - Choisir un axe a parmi { x, y, z } : ici, on choisit z
  - Chercher la valeur de a entière minimale dans les particules : ici zmin

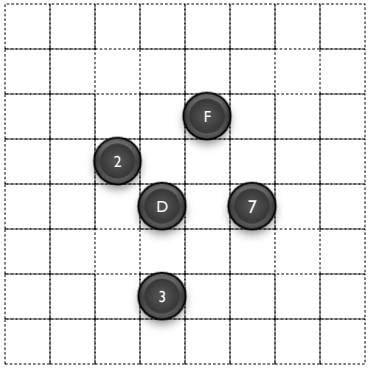
Sliced Grid

- Déroulement :
  - Choisir un axe a parmi { x, y, z } : ici, on choisit z
  - Chercher la valeur de a entière minimale dans les particules : ici zmin
  - Séparer les particules pour chaque z = zmin + x (slice)

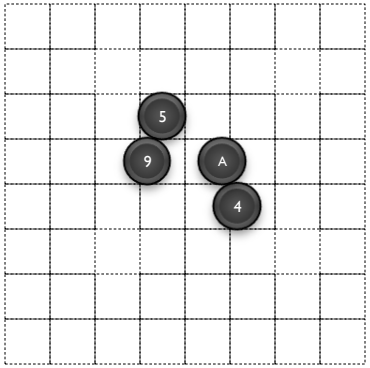
z = zmin+0



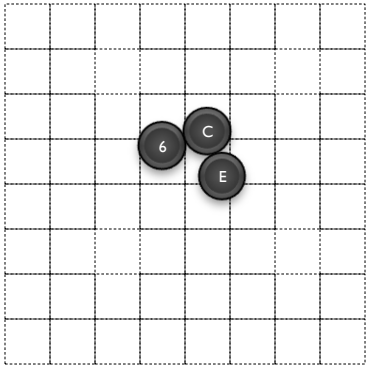
z = zmin+1



z = zmin+2

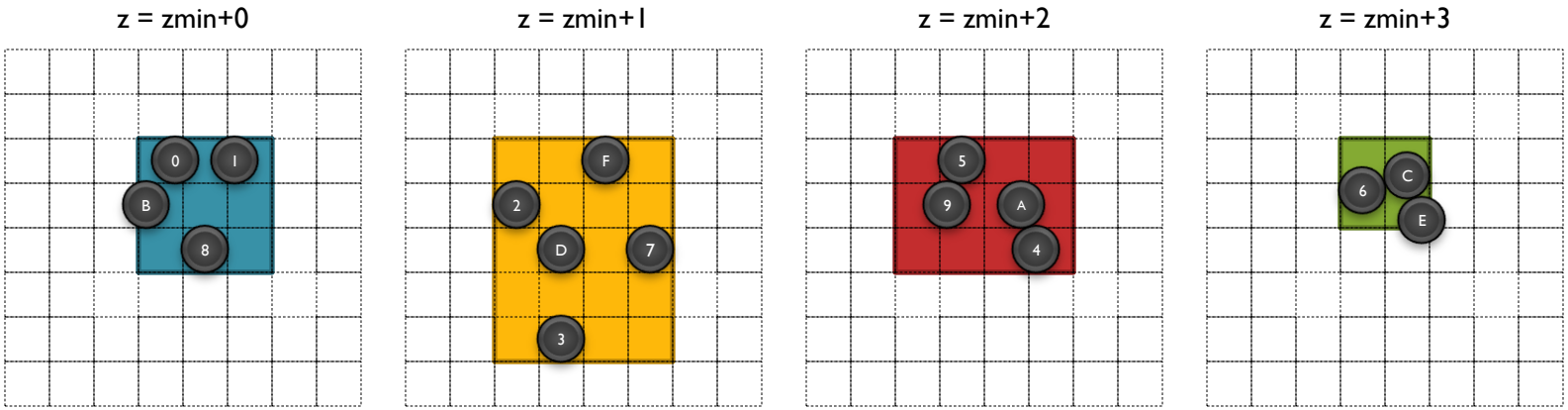


z = zmin+3



Sliced Grid

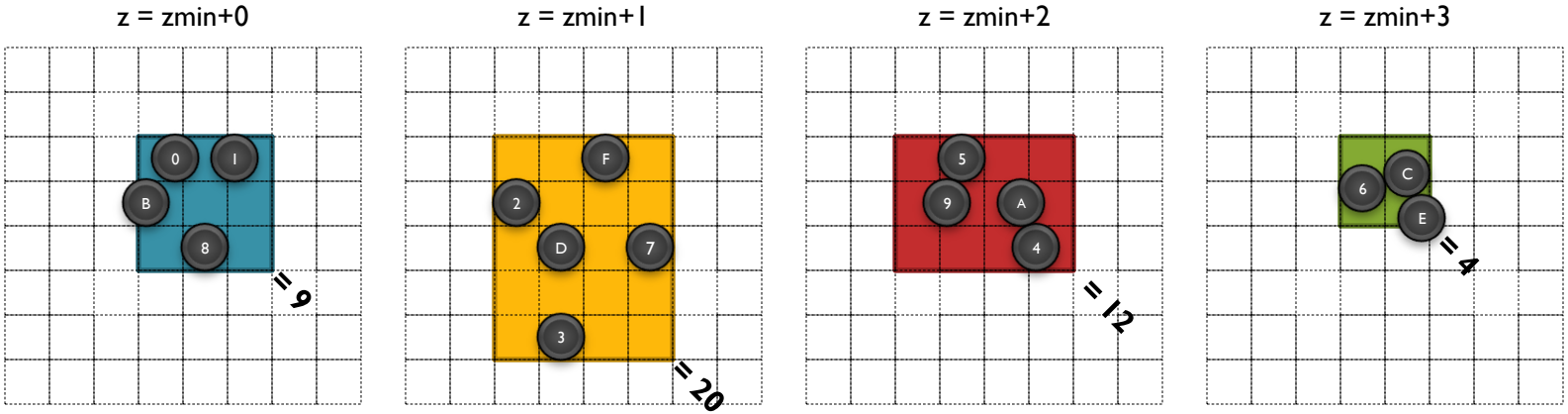
- Déroulement :
  - Séparer les particules pour chaque z = zmin + x (slice)



  - Chercher les bounding boxes (indices min et max et xy) pour chaque slice

Sliced Grid

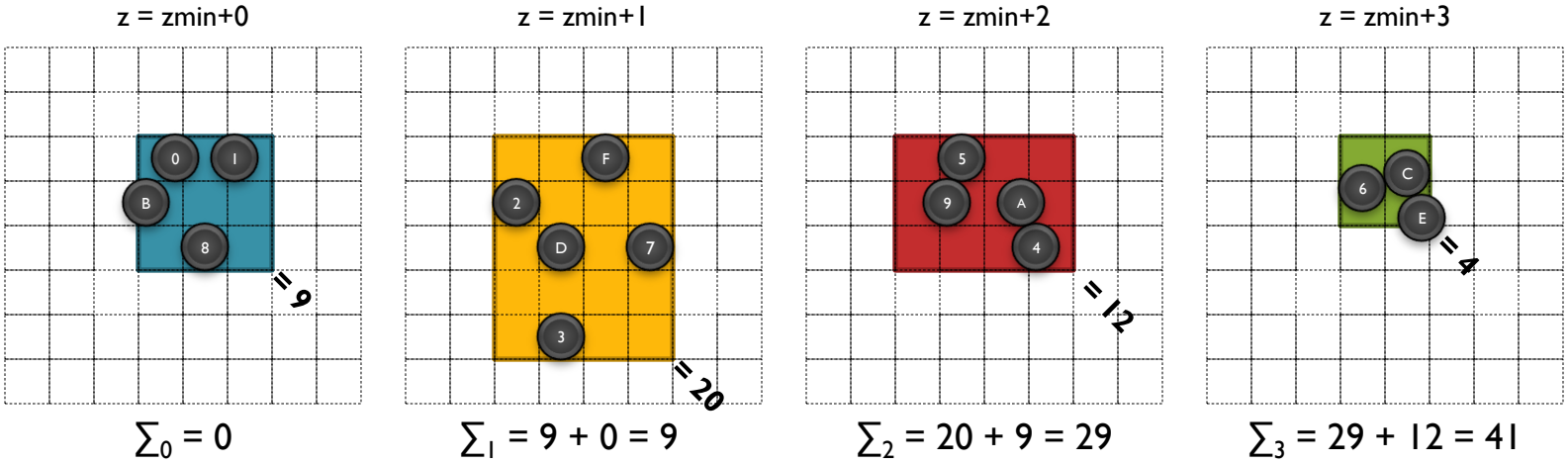
- Déroulement :
  - Séparer les particules pour chaque z = zmin + x (slice)



  - Chercher les bounding boxes pour chaque slice
  - Calculer la taille de chaque slice

Sliced Grid

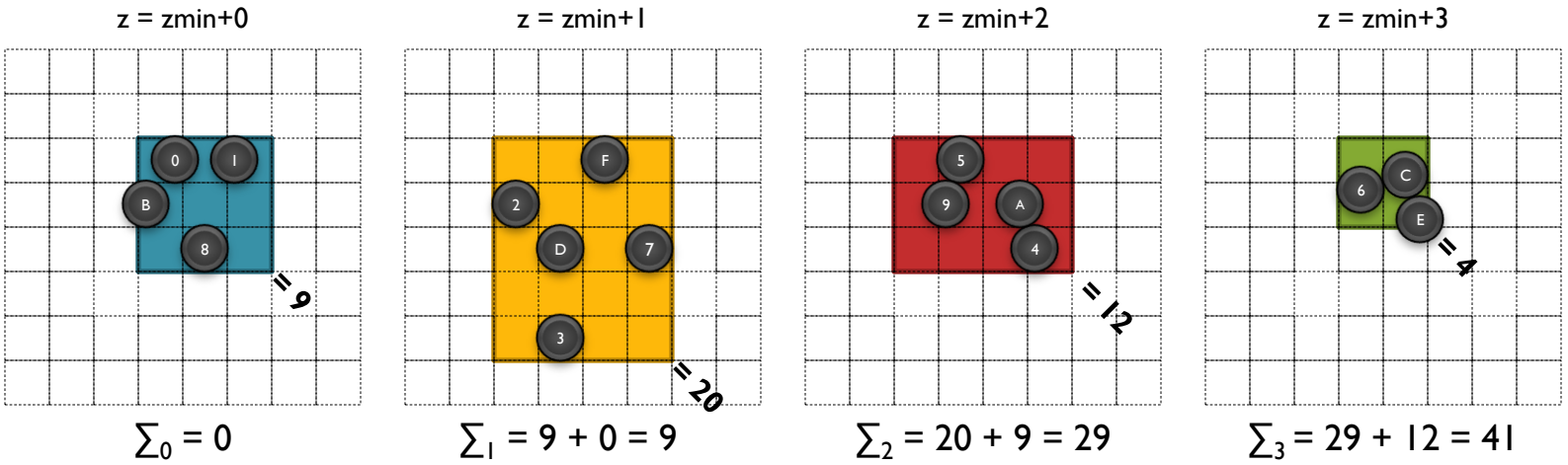
- Déroulement :
  - Séparer les particules pour chaque z = zmin + x (slice)

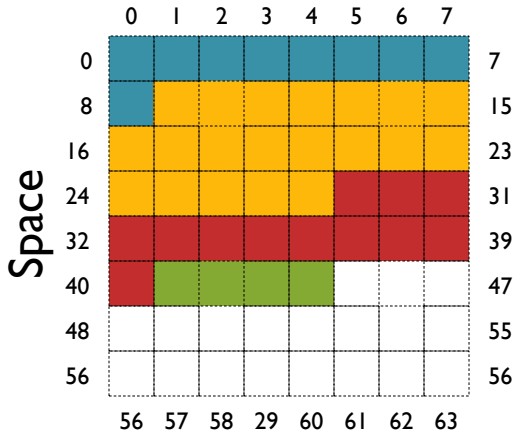


  - Chercher les bounding boxes pour chaque slice
  - Calculer la taille de chaque slice
  - Réaliser une somme préfixée de la taille (donne l'indice du premier voxel de chaque slice)

Sliced Grid

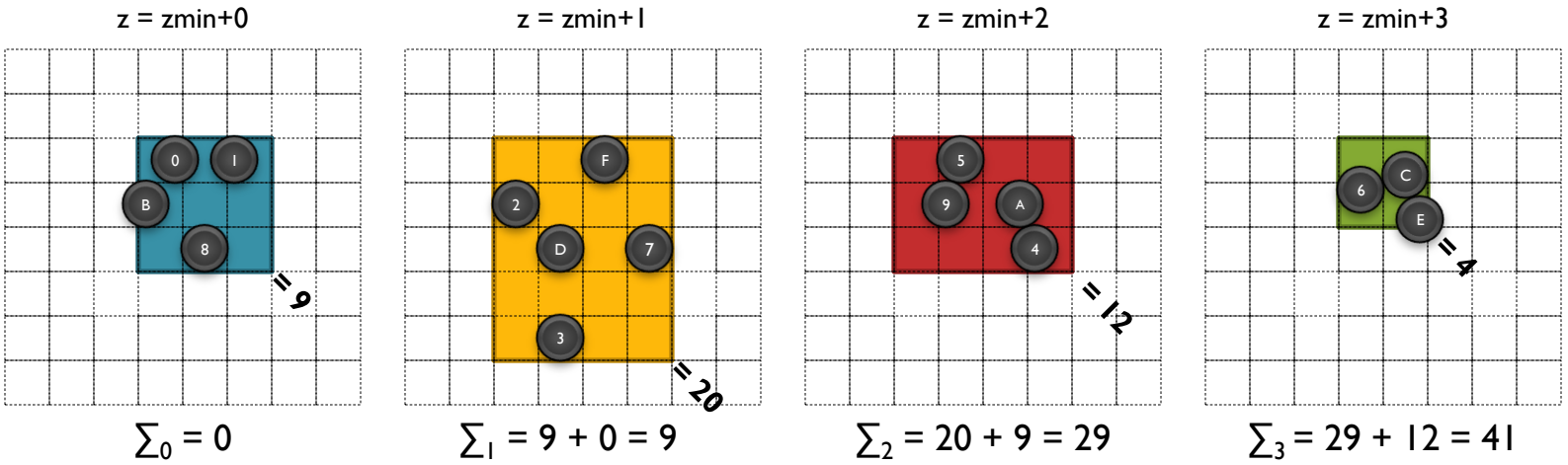
- On a toutes les données nécessaire pour remplir l'espace (zmin, bbslice, prefixsum)

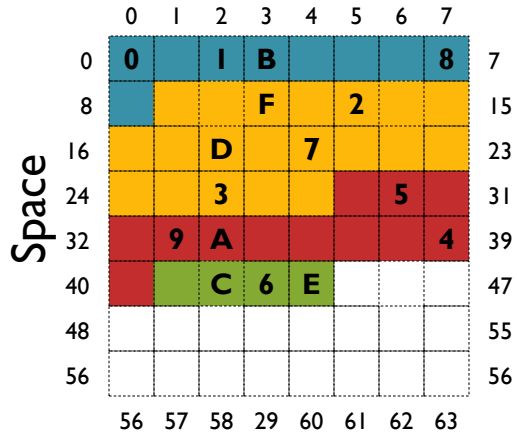


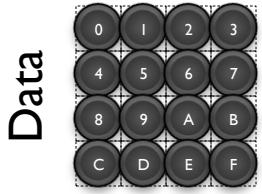


Sliced Grid

- On a toutes les données nécessaire pour remplir l'espace (zmin, bbslice, prefixsum)

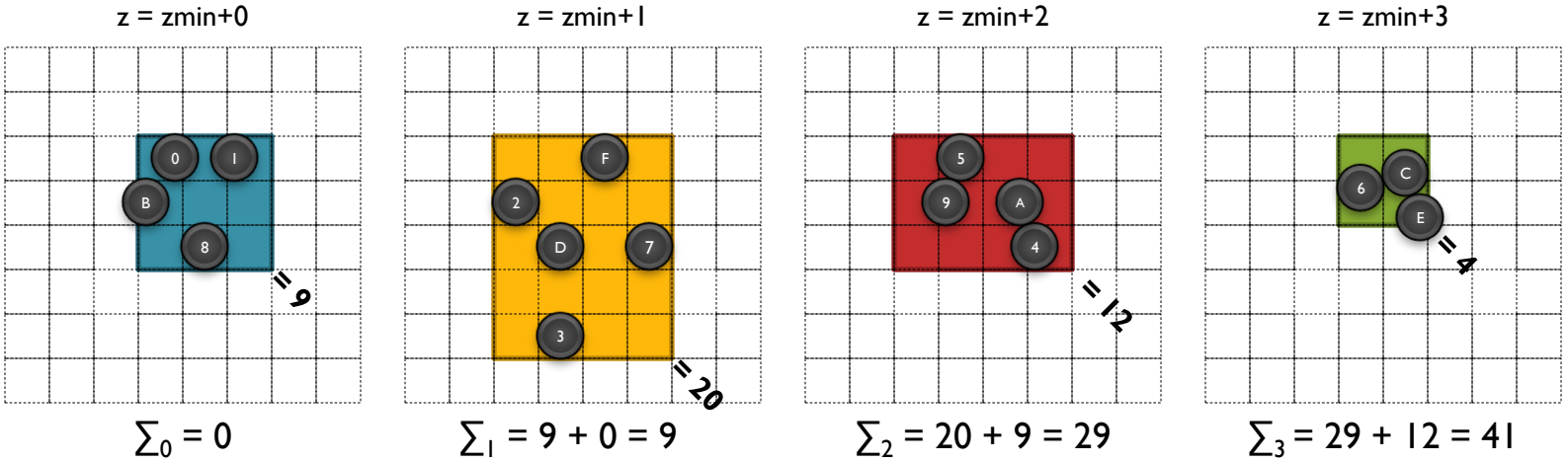


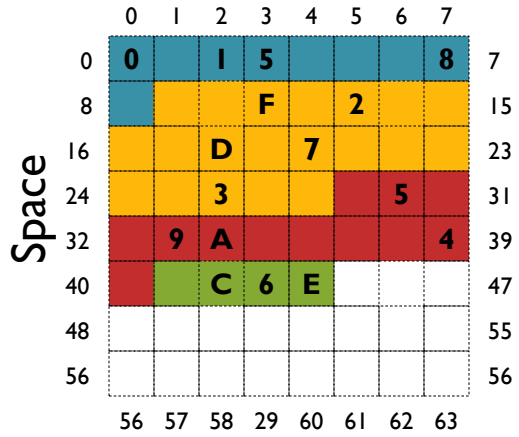


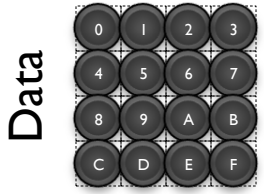


Sliced Grid

- Comment générer ce process sur GPU ?
- Méthode du Shader X7 expliqué sans CS







Sliced Grid

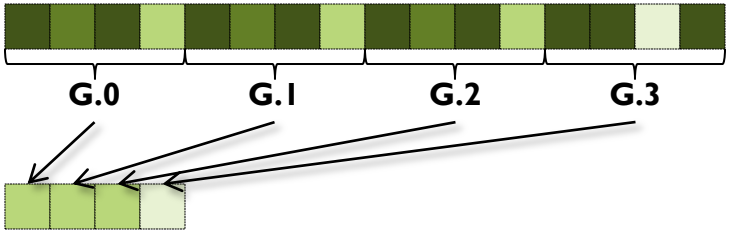
- 1er point, recherche du min(z)
  - Problème de réduction
  - Résolution d'une réduction CUDA ~ CS
    - Profite de la zone de mémoire partagée des threads group
    - Chaque thread group cherchant son minimum dans sa zone allouée
- Déroulement global :

Sliced Grid

- 1er point, recherche du min(z)
  - Problème de réduction
  - Résolution d'une réduction CUDA ~ CS
    - Profite de la zone de mémoire partagée des threads group
    - Chaque thread group cherchant son minimum dans sa zone allouée
- Déroulement global :

Sliced Grid

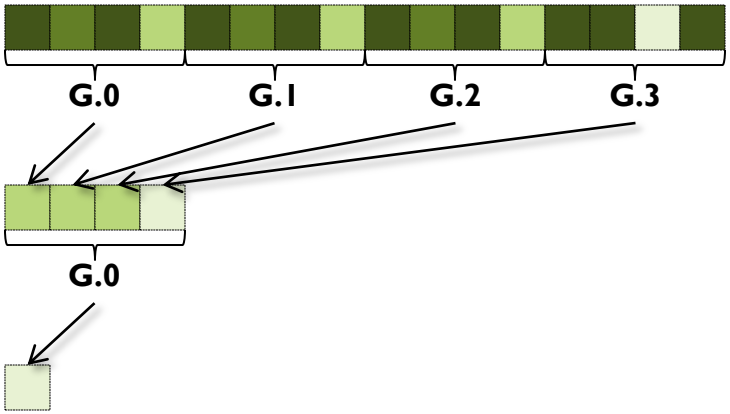
- 1er point, recherche du min(z)
  - Problème de réduction
  - Résolution d'une réduction CUDA ~ CS
    - Profite de la zone de mémoire partagée des threads group
    - Chaque thread group cherchant son minimum dans sa zone allouée
- Déroulement global :



Dispatch de 4 thread group de taille 4 (16 threads)

Sliced Grid

- 1er point, recherche du min(z)
  - Problème de réduction
  - Résolution d'une réduction CUDA ~ CS
    - Profite de la zone de mémoire partagée des threads group
    - Chaque thread group cherchant son minimum dans sa zone allouée
- Déroulement global :



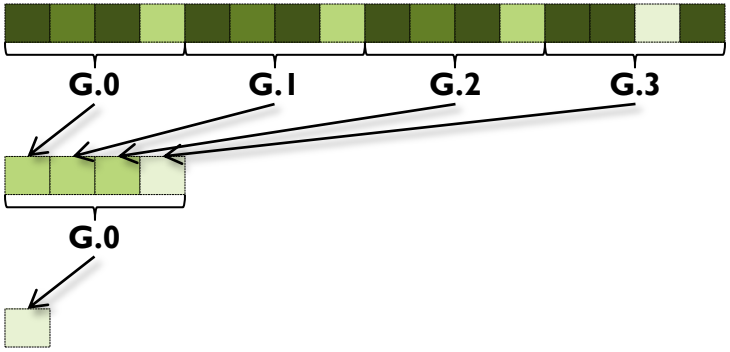
Dispatch de 4 thread group de taille 4 (16 threads)

Dispatch de 1 thread group de taille 4 (4 threads)

On a le résultat dans la case 0

Slice Grid

- Déroulement global :

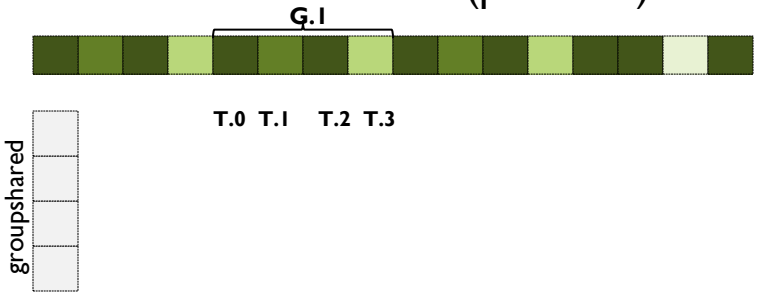


Dispatch de 4 thread group de taille 4 (16 threads)

Dispatch de 1 thread group de taille 4 (4 threads)

On a le résultat dans la case 0

- Déroulement local (pour G.1) :

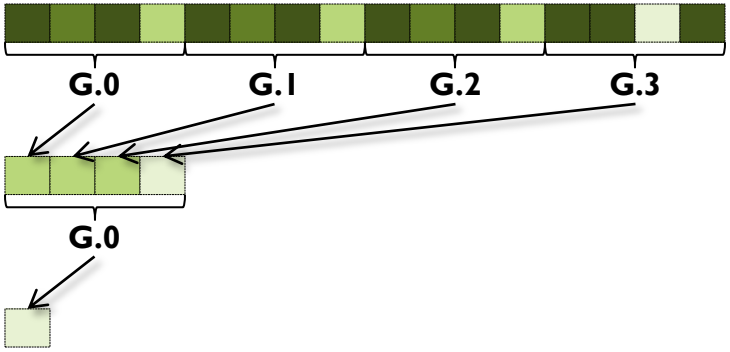


Nos 4 threads doivent écrire en 1 (car G.1) le minimum

Rappel sur la mémoire partagée :
•On a un emplacement réservé en écriture
•On peut lire n'importe où

Sliced Grid

- Déroulement global :

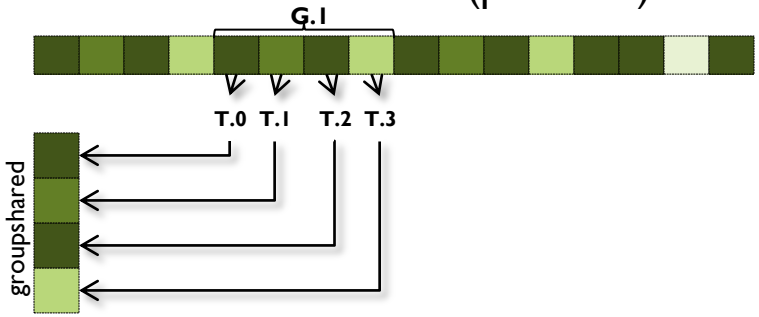


Dispatch de 4 thread group de taille 4 (16 threads)

Dispatch de 1 thread group de taille 4 (4 threads)

On a le résultat dans la case 0

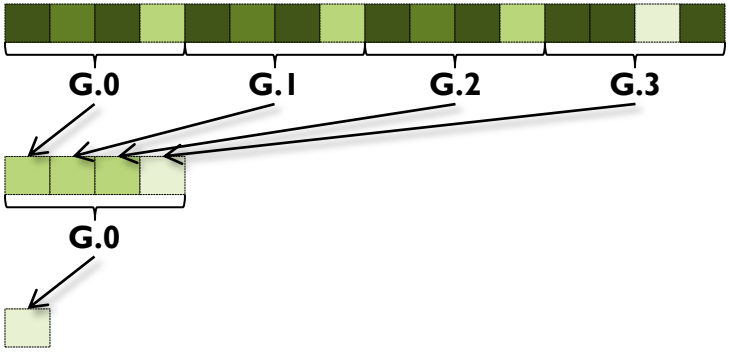
- Déroulement local (pour G.1) :



• On copie en mémoire partagée la valeur à traiter

Sliced Grid

- Déroulement global :

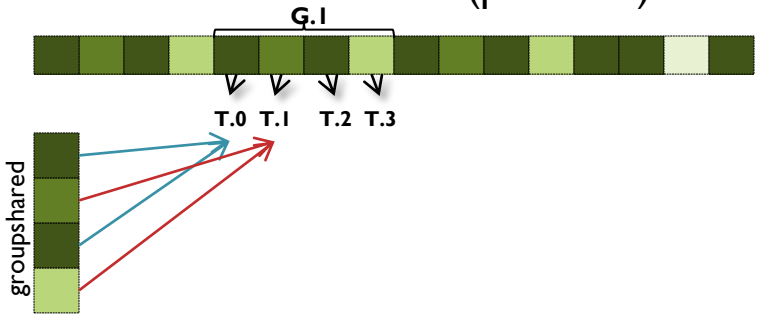


Dispatch de 4 thread group de taille 4 (16 threads)

Dispatch de 1 thread group de taille 4 (4 threads)

On a le résultat dans la case 0

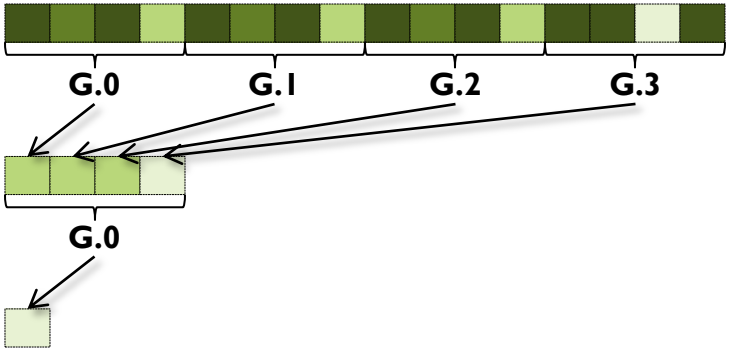
- Déroulement local (pour G.1) :



- On copie en mémoire partagé la valeur à traiter
- Synchronisation (GroupMemoryBarrierWithGroupSync)
- Si (T.x<2) GS[T.x] = min(GS[T.x],GS[T.x+2])

Sliced Grid

- Déroulement global :

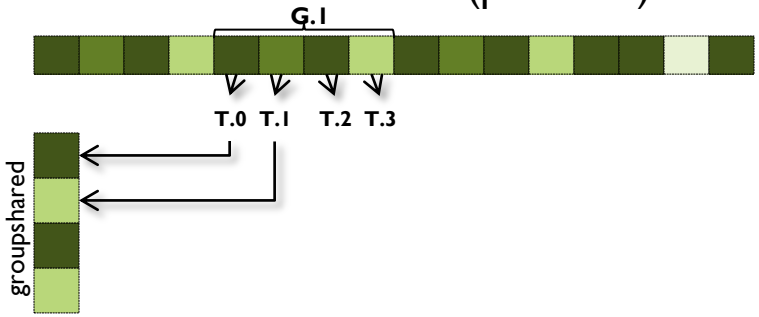


Dispatch de 4 thread group de taille 4 (16 threads)

Dispatch de 1 thread group de taille 4 (4 threads)

On a le résultat dans la case 0

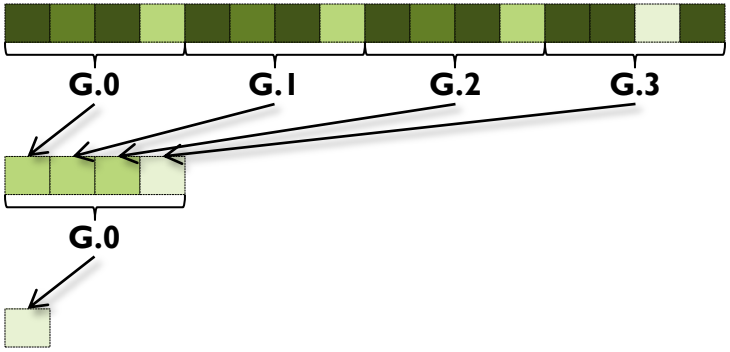
- Déroulement local (pour G.1) :



- On copie en mémoire partagé la valeur à traiter
- Synchronisation (GroupMemoryBarrierWithGroupSync)
- Si (T.x<2) GS[T.x] = min(GS[T.x],GS[T.x+2])

Sliced Grid

- Déroulement global :

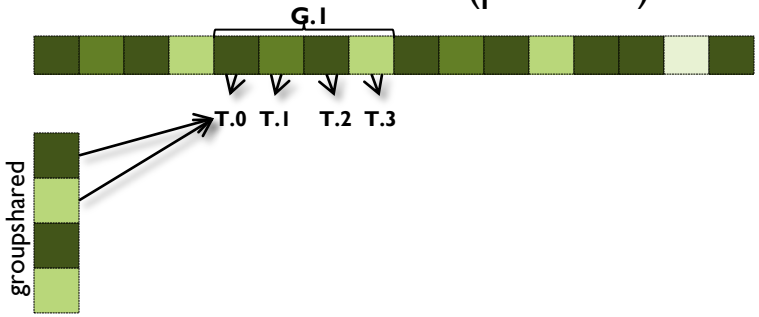


Dispatch de 4 thread group de taille 4 (16 threads)

Dispatch de 1 thread group de taille 4 (4 threads)

On a le résultat dans la case 0

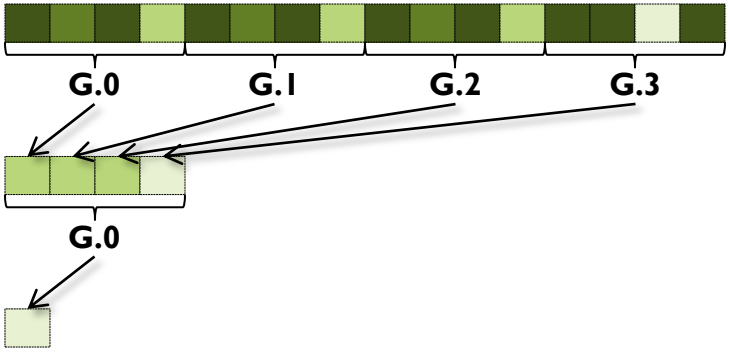
- Déroulement local (pour G.1) :



• On copie en mémoire partagé la valeur à traiter
• Synchronisation (GroupMemoryBarrierWithGroupSync)
• Si (T.x<2) GS[T.x] = min(GS[T.x],GS[T.x+2])
•Synchronisation (GroupMemoryBarrierWithGroupSync)
• Si (T.x<1) GS[T.x] = min(GS[T.x],GS[T.x+1])

Sliced Grid

- Déroulement global :

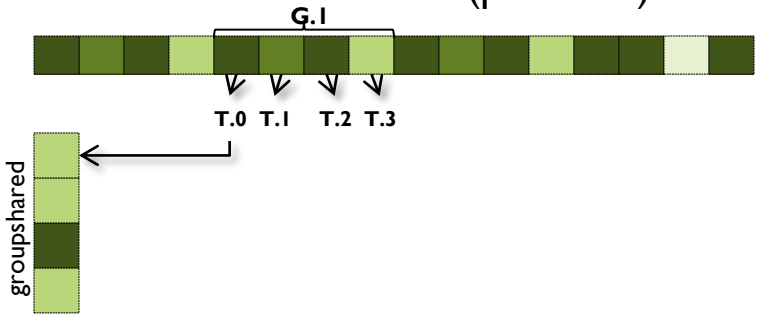


Dispatch de 4 thread group de taille 4 (16 threads)

Dispatch de 1 thread group de taille 4 (4 threads)

On a le résultat dans la case 0

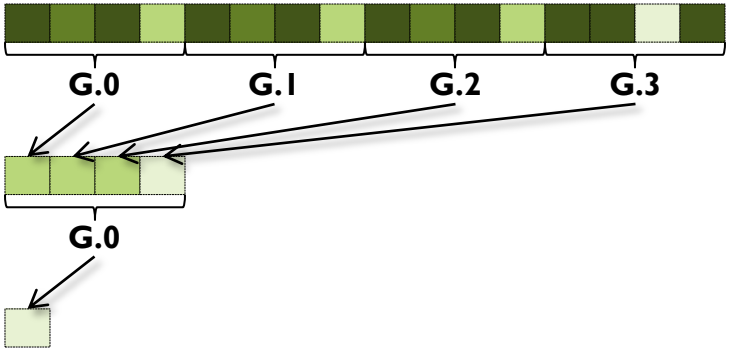
- Déroulement local (pour G.1) :



• On copie en mémoire partagé la valeur à traiter
• Synchronisation (GroupMemoryBarrierWithGroupSync)
• Si (T.x<2) GS[T.x] = min(GS[T.x],GS[T.x+2])
•Synchronisation (GroupMemoryBarrierWithGroupSync)
• Si (T.x<1) GS[T.x] = min(GS[T.x],GS[T.x+1])

Sliced Grid

- Déroulement global :

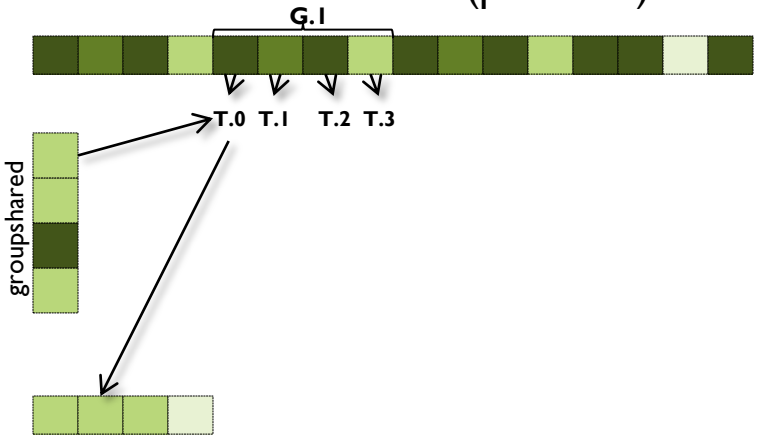


Dispatch de 4 thread group de taille 4 (16 threads)

Dispatch de 1 thread group de taille 4 (4 threads)

On a le résultat dans la case 0

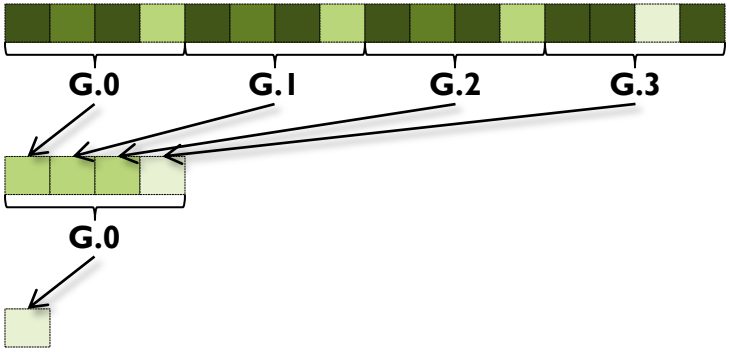
- Déroulement local (pour G.1) :



• On copie en mémoire partagé la valeur à traiter
• Synchronisation (GroupMemoryBarrierWithGroupSync)
• Si (T.x<2) GS[T.x] = min(GS[T.x],GS[T.x+2])
•Synchronisation (GroupMemoryBarrierWithGroupSync)
• Si (T.x<1) GS[T.x] = min(GS[T.x],GS[T.x+1])
•Synchronisation (GroupMemoryBarrierWithGroupSync)
• Si (T.x == 0) Res[G.x] = GS[0]

Sliced Grid

- Déroulement global :

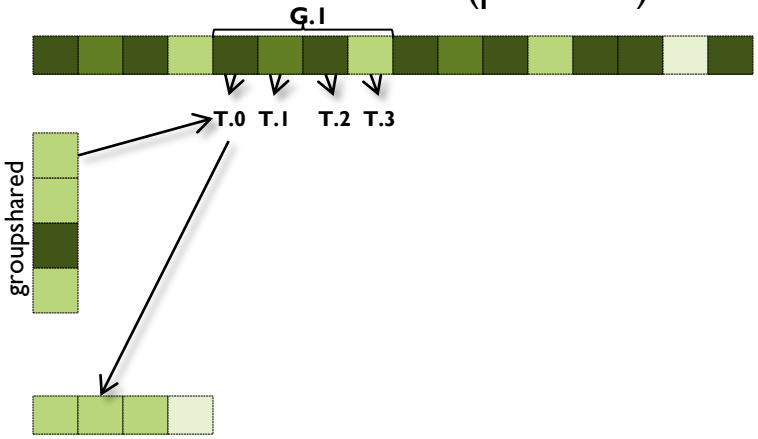


Dispatch de 4 thread group de taille 4 (16 threads)

Dispatch de 1 thread group de taille 4 (4 threads)

On a le résultat dans la case 0

- Déroulement local (pour G.1) :

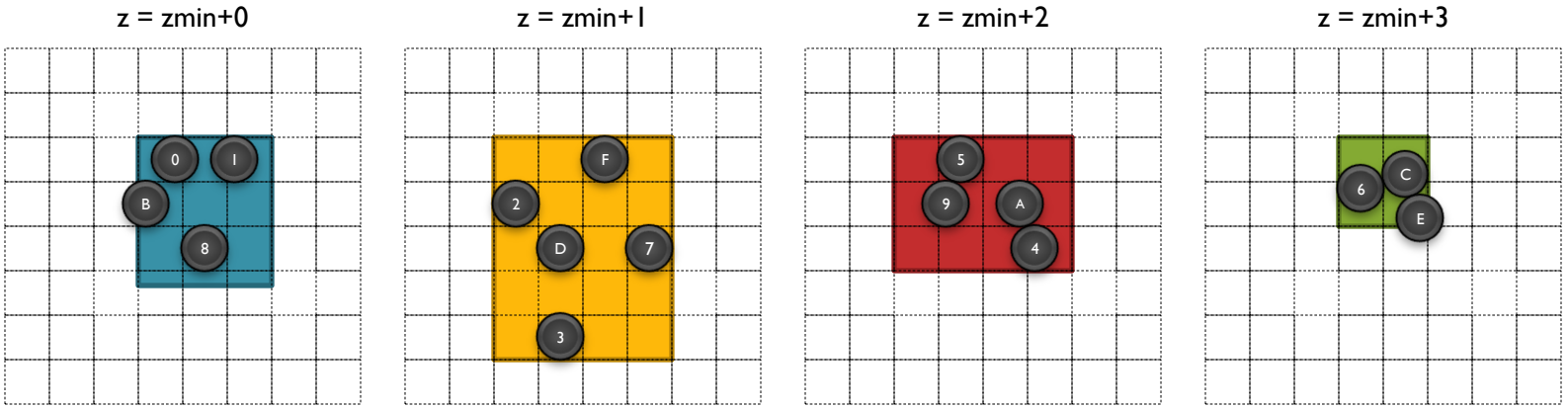


• On copie en mémoire partagé la valeur à traiter
• Synchronisation (GroupMemoryBarrierWithGroupSync)
• Si (T.x<2) GS[T.x] = min(GS[T.x],GS[T.x+2])
•Synchronisation (GroupMemoryBarrierWithGroupSync)
• Si (T.x<1) GS[T.x] = min(GS[T.x],GS[T.x+1])
•Synchronisation (GroupMemoryBarrierWithGroupSync)
• Si (T.x == 0) Res[G.x] = GS[0]

Pour plus de code, voir le sampleDX « HDRToneMappingCS11 » (MSDN)

Sliced Grid

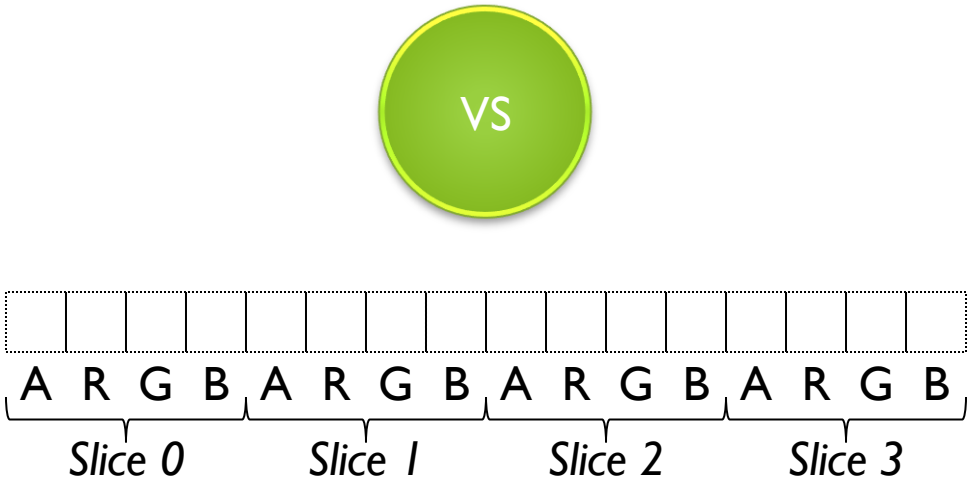
- 2ème étape, recherche des bouding boxes



- Pourrait ressembler à un problème de réduction mais la quantité de données en sortie est trop importante
- Pas de solution pour le moment de mon côté en utilisant les compute shaders
- Solution en utilisant l'unité de blending du rasterizer

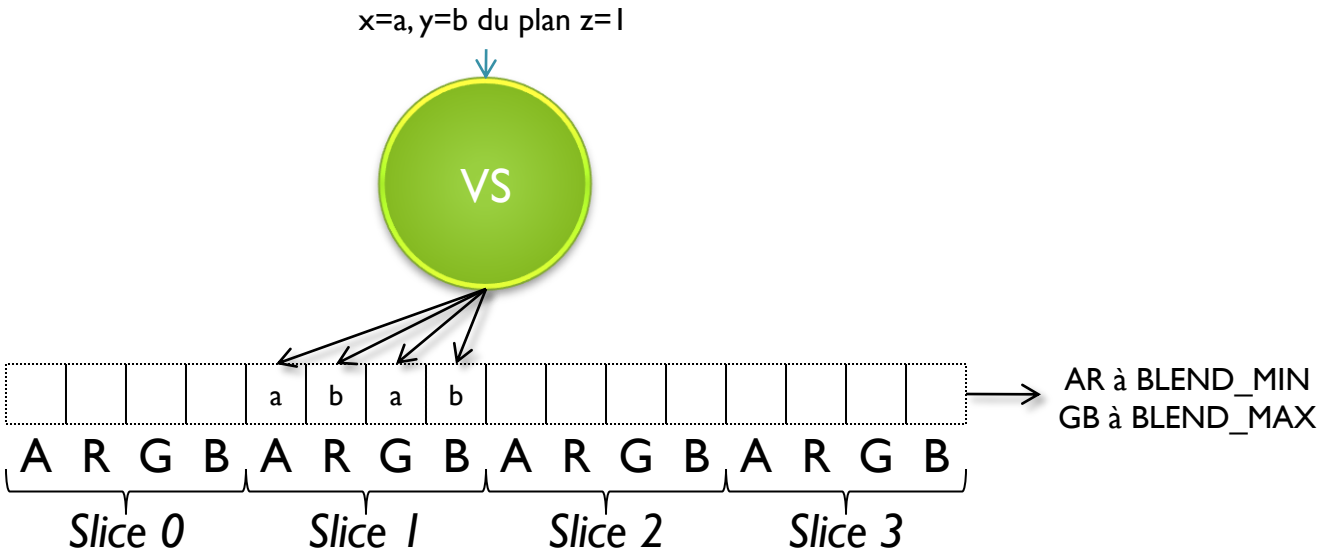
Sliced Grid

- 2ème étape, recherche des bouding boxes, en utilisant un vertex shader :
  - Chaque particule est un vertex
  - Le render target est un tableau de N pixels correspondant au N slices.
  - Le vertex shader envoie chaque vertex au pixel correspondant au slice
  - Les fonctions de blending sont configurées de manière à ne récupérer que le MAX ou le MIN des pixels de la RTT



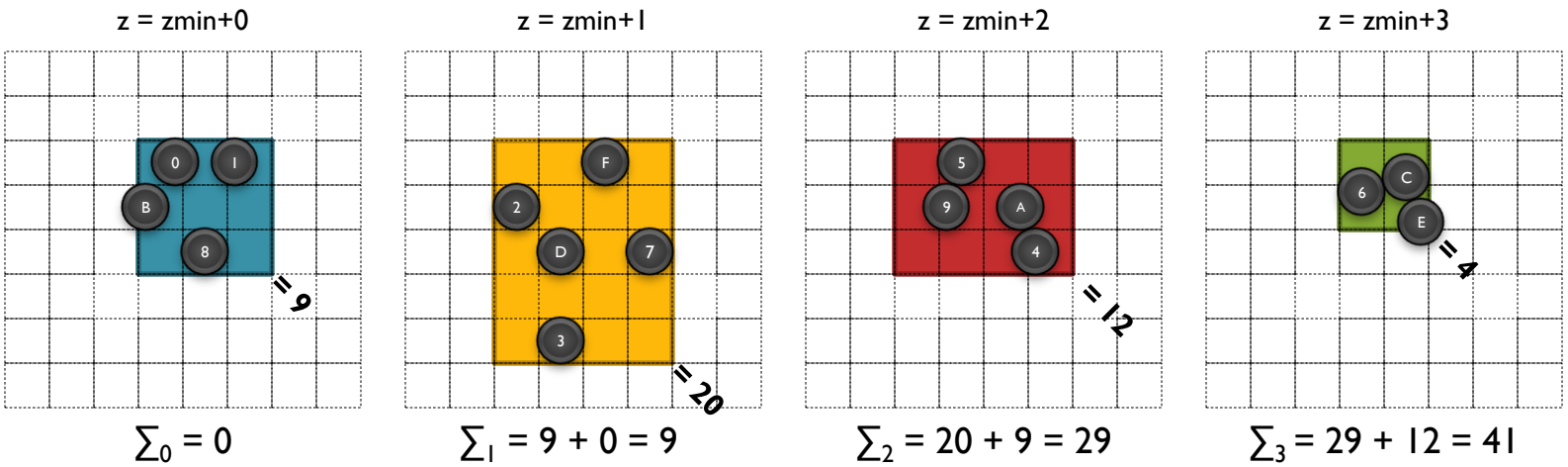
Sliced Grid

- 2ème étape, recherche des bouding boxes, en utilisant un vertex shader :
  - Chaque particule est un vertex
  - Le render target est un tableau de N pixels correspondants au N slices.
  - Le vertex shader envoie chaque vertex au pixel correspondant au slice
  - Les fonctions de blending sont configurés de manière à ne récupérer que le MAX ou le MIN des pixels de la RTT



Sliced Grid

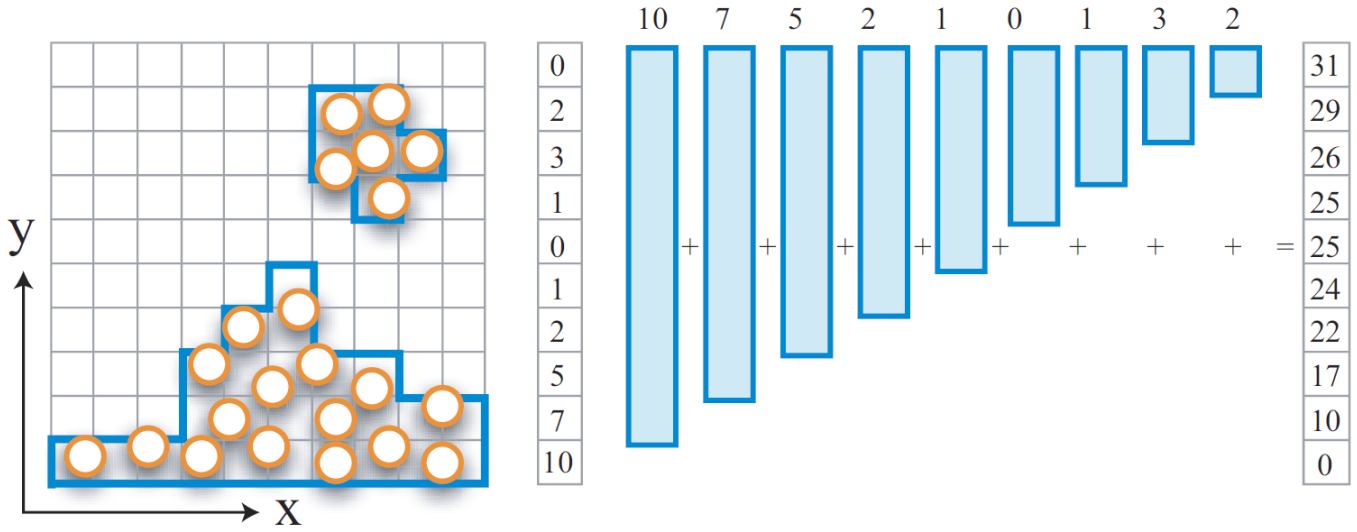
- 3ème étape, calcul de la somme préfixée



  - Pas de solution non plus en compute shader pour le moment
  - Solution en utilisant encore l'unité de blending du rasterizer

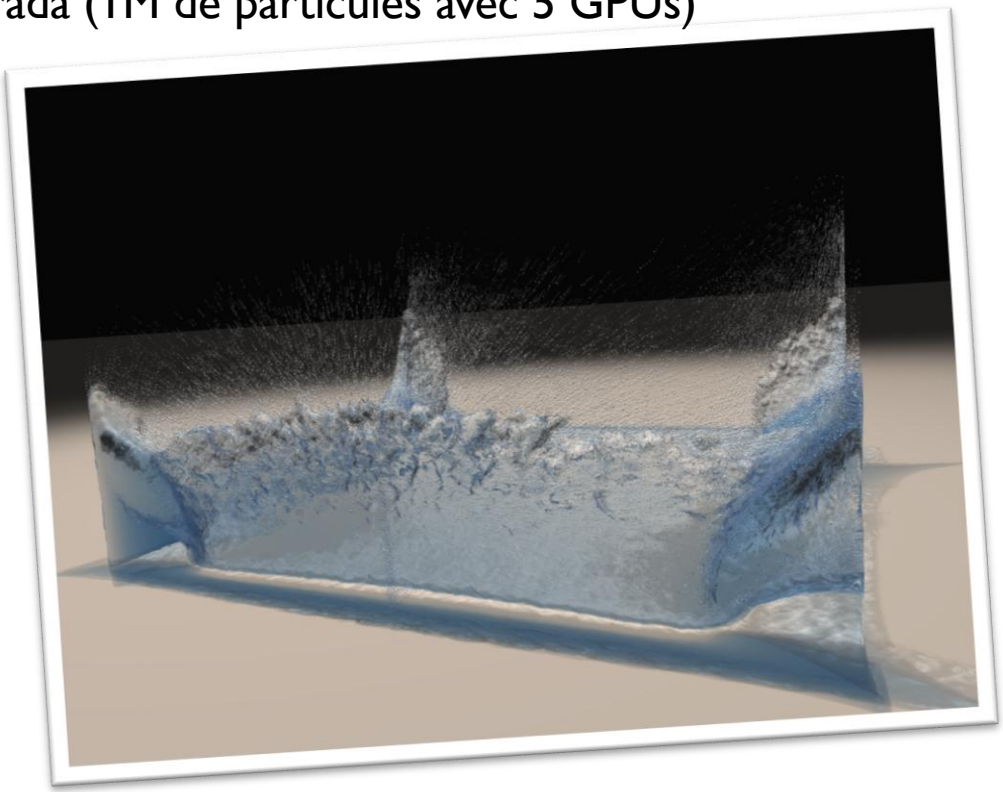
Sliced Grid

- 3ème étape, calcul de la somme préfixée, en utilisant un vertex shader
  - Chaque slice dessine une ligne commençant au pixel 0 de taille « numéro du slice »
  - Chaque ligne contient un entier de la taille du slice
  - La render target est un tableau de N pixels correspondant aux N slices
  - L'opération de blending est maintenant l'addition



Sliced Grid

- Pistes de résolution
  - Utiliser des variables atomiques (cs_5_0) pour reproduire le comportement de l'opération de blending ?
  - Trouver plus d'informations sur l'implémentation CUDA de Takahiro Harada (1M de particules avec 5 GPUs)



Adaptative Grid

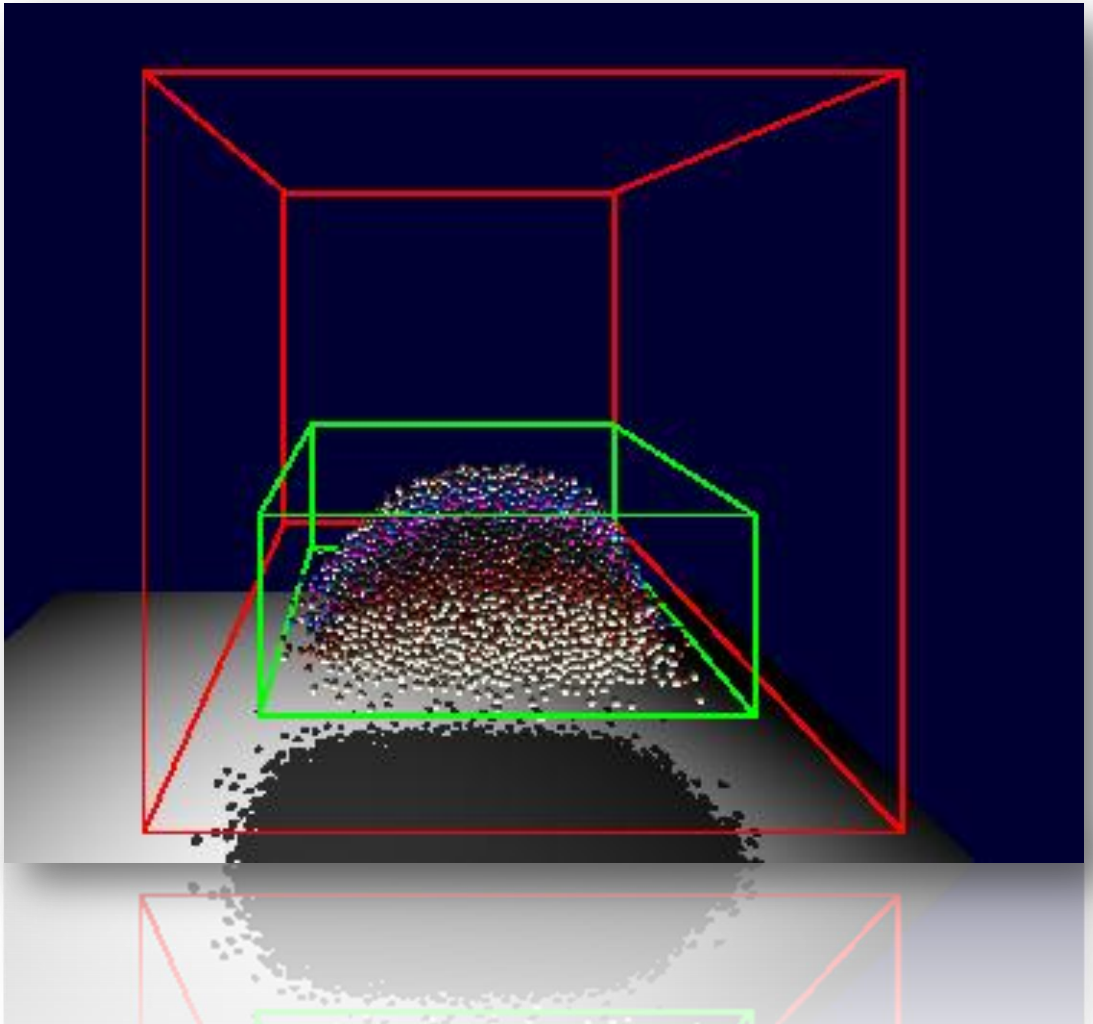
- « Adaptative Grid » est une méthode maison :
  - Très simple
  - Permet d'occuper moins d'espace pour la grille
  - Permet de constater les gains d'un espace moins dispersé

- Principe général
  - Recherche de bounding box global (deux voxels)
    - Même algorithme de réduction que lors du zmin du sliced grid
  - Utilisation de cette bounding box pour aplatir nos indices

- Gain de performance mesuré jusqu'à 43%

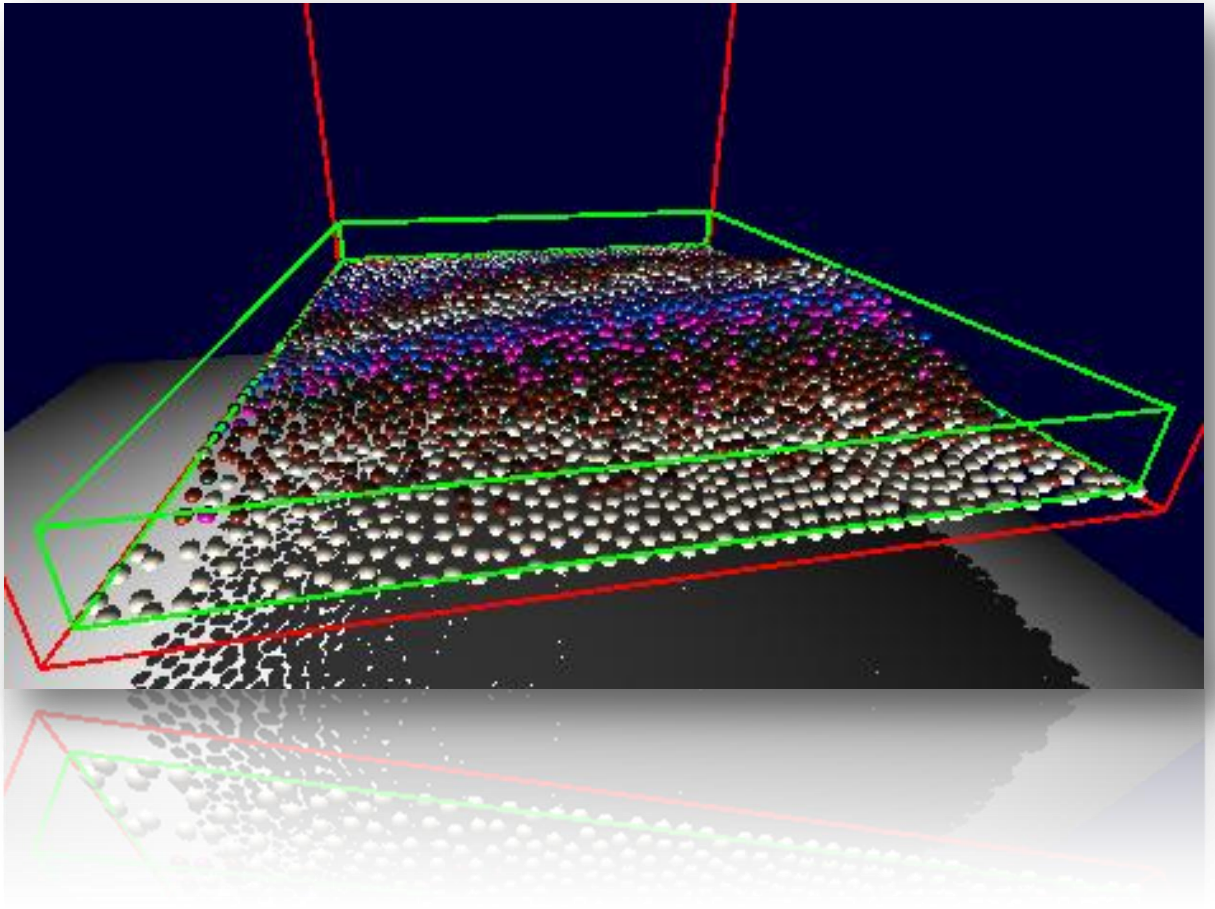
Adaptative Grid

- « Adaptative Grid » est une méthode maison :



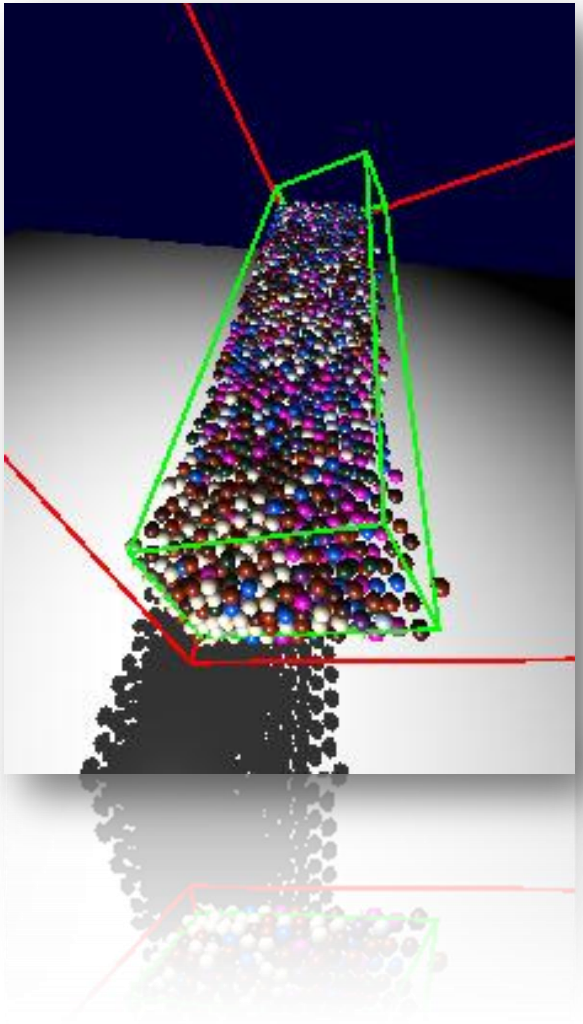
Adaptative Grid

- « Adaptative Grid » est une méthode maison :



Adaptative Grid

- « Adaptative Grid » est une méthode maison :



Performances

- Deux aspects
  - Choix de l'espace uniform ou adapté
  - Choix de la composante du structured buffer « space » (*alignement sur DWORD apparemment*)
    - uint (un voxel par élément)
    - uint4 (quatre voxels par élément)
    - uint8, pas adapté, trop d'écritures concurrentes
- Test de perf. réalisé avec les queries sur la moyenne d'un calcul de 100 frames

Performances

	Uniform	Adaptative
uint	1.167284	0.689202
uint4	0.870611	0.661326

Performances

	Uniform	Adaptative
uint	1.167284	0.689202
uint4	0.870611	0.661326

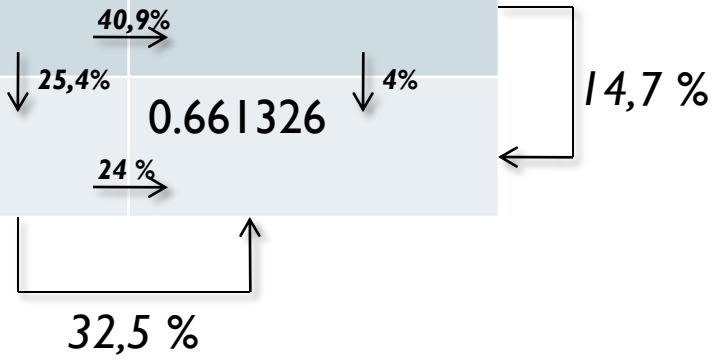
25,4%

4%

14,7 %

Performances

	Uniform	Adaptative
uint	1.167284	0.689202
uint4	0.870611	0.661326



Performances

	Uniform	Adaptative
uint	1.167284	0.689202
uint4	0.870611	0.661326

40,9%
25,4%
4%
14,7 %
24 %
32,5 %

ShaderX7

Nombre de particules	Uniform Grid	Sliced Grid
65535	60.9	53.1
262144	280.5	231.3
589824	685.9	567.9
1048576	1160.9	1070.3

Peut gérer plus de particules

13,9 %

Améliorations

- Le rendu contient
  - 8192 particules en multi-instanced (300 triangles par particules)
  - Un plan
  - Tout est éclairé par pixel avec du phong
  - Shadow map
- La physique est mise à jour environ cinq fois par frame

- Part rendu/CS : 72% du temps global est consacré à l'affichage
  - Bottleneck n'est pas forcement où on l'imagine
  - Nécessité d'utiliser des LOD
  - Intérêt du tri des particules selon la profondeur

- Prochaines étapes :
  - Travailler sur des maillages skinnés et animés en multi-instance et LOD
  - Profiter des variables inutiles (alignement en float4) des particules pour améliorer la simulation de foule (prédateur, proie,…)

Conclusion

- Démonstration
- Ce qui n'a pas été abordé :
  - L'implémentation en elle-même
    - Le choix de la taille des threads groups
    - Les algorithmes appliqués sur les particules
    - Les méthodes de rendu
  - Les autres variantes
    - Approche en CUDA ?
    - Approche sur un rendu différent ?
- Questions ?

Références

- Sliced Data Structure for Particle-Based Simulations on GPUs (2007) de Takahiro Harada, http://www.iii.u-tokyo.ac.jp/~yoichiro/report/report-pdf/harada/international/2007graphite.pdf
- Chapter 29. Real-Time Rigid Body Simulation on GPUs de Takahiro Harada, http://http.developer.nvidia.com/GPUGems3/gpugems3_ch29.html
- Flocks, Herds, and Schools: A Distributed Behavioral Model the SIGGRAPH '87 boids paper.
- Improving Boids Algorithm in GPU using Estimated Self Occlusion http://boid.alessandrosilva.com/sbgames08-boids.pdf
- Paul Richmond (2008) Agent Based GPU, a Real-time 3D Simulation and Interactive Visualisation Framework for Massive Agent Based Modelling on the GPU
- Professeur Wen-mei W. Hwu et David Kirk, directeur scientifique NVIDIA (2007) : CUDA http://www.nvidia.fr/object/cuda_education_fr.html
- Programmation GPU Avancée (2010) http://sead.univ-reims.fr/courses/GPU/index.php