M2-Images

Rendu différé

J.C. Iehl

November 25, 2015

Bilan

openGL :

- dessine les primitives dans l'ordre définit par l'application,
- transforme tous les sommets,
- (transforme toutes les primtives),
- colorie chaque fragment de chaque primitive (a priori visible),
- teste la visibilité du fragment et écrit dans le framebuffer.

Bilan

exemple :

- dessine quelques objets,
- chaque objet est éclairé par un ensemble de sources de lumière (chaque fragment calcule l'énergie réfléchie par toutes les sources)
- pas d'ombres portées...

Exemple

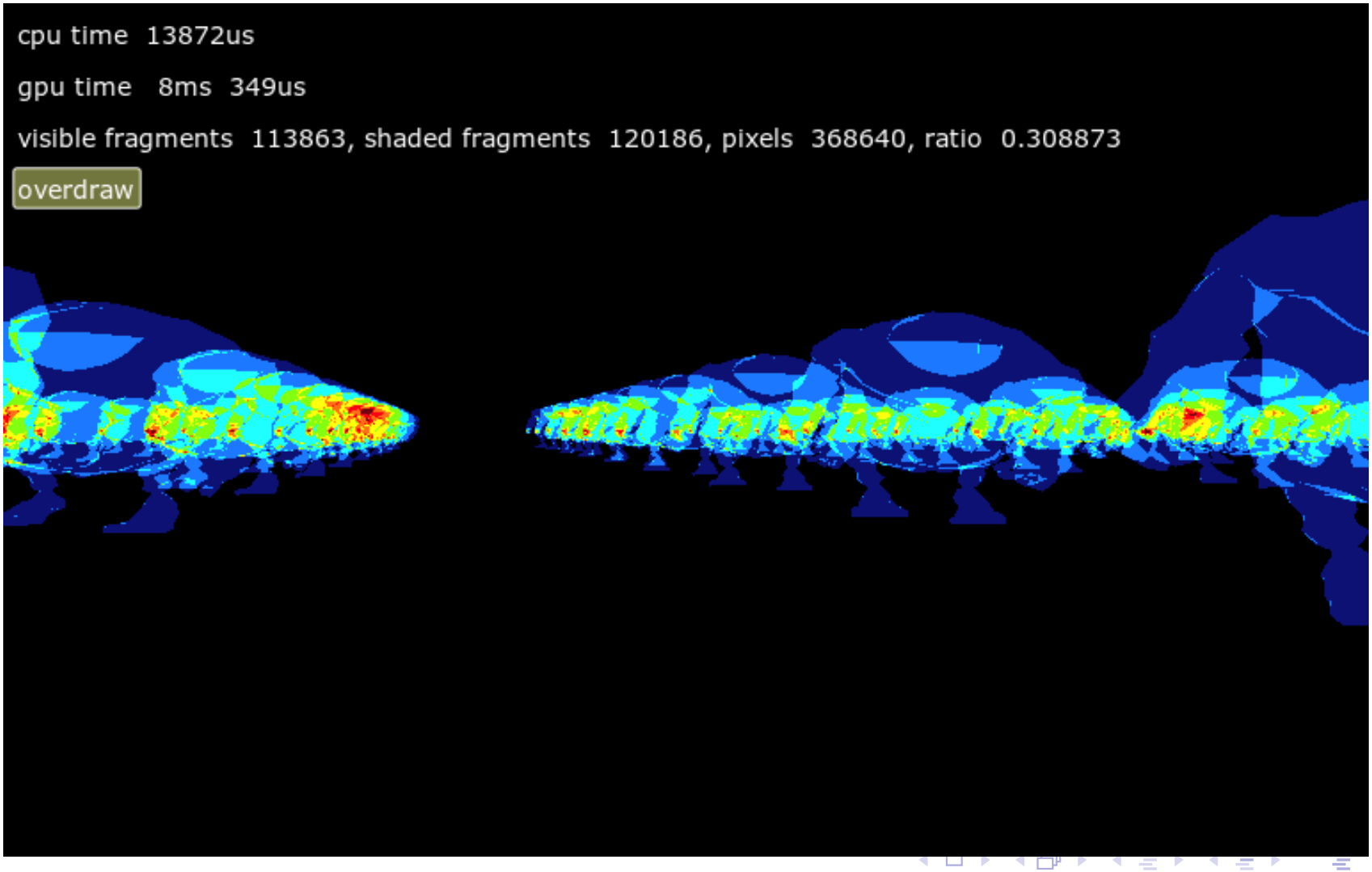

cpu time 42us
gpu time 2ms 158us
visible fragments 113863, shaded fragments 0, pixels 368640, ratio 0.308873
overdraw

Peut mieux faire ?

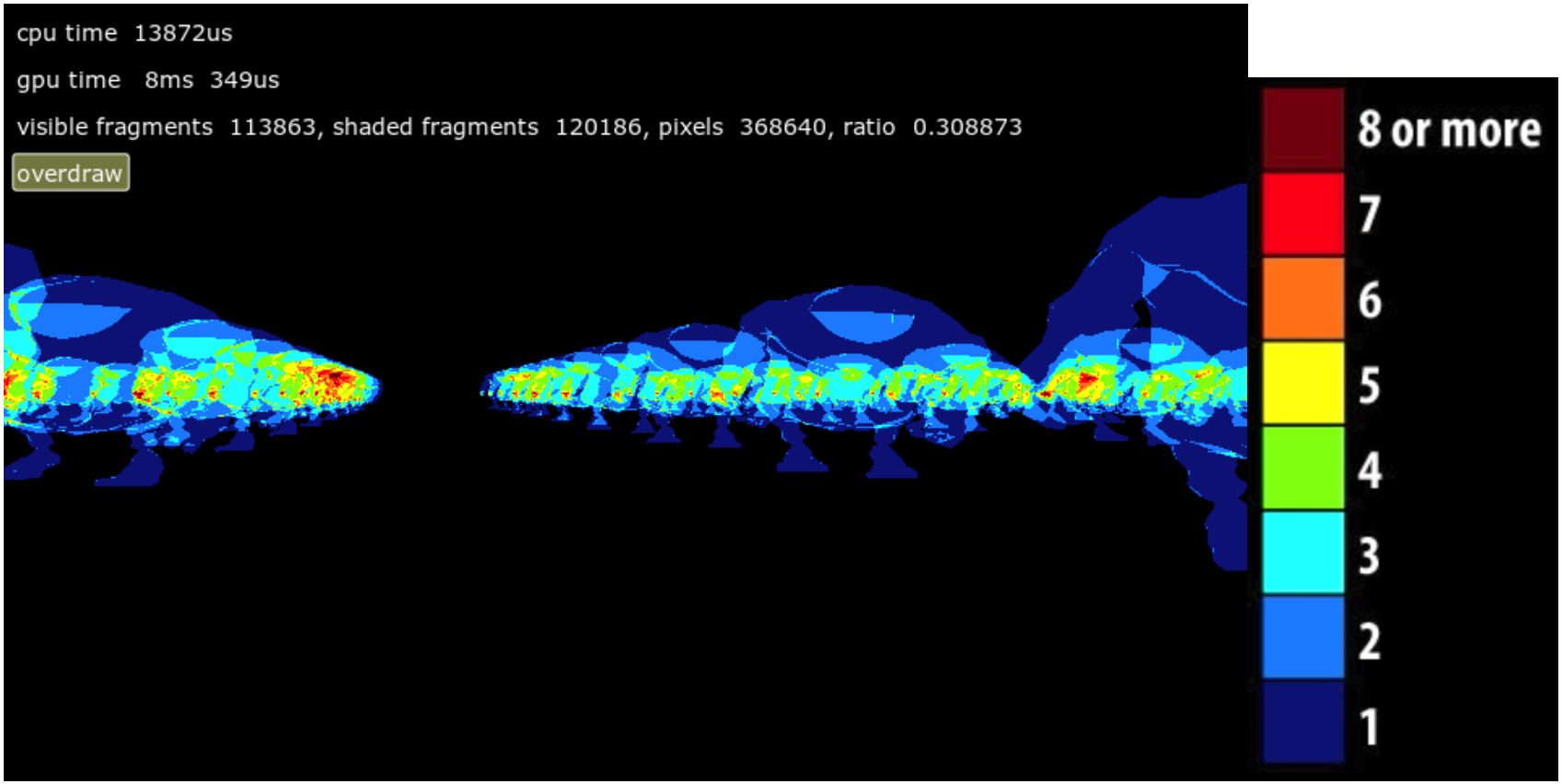
peut mieux faire :

- combien de fois est executé le fragment shader par pixel ?
- combien de fragments sont nécessaires pour construire l'image finale ?

Exemple


cpu time 13872us
gpu time 8ms 349us
visible fragments 113863, shaded fragments 120186, pixels 368640, ratio 0.308873
overdraw

Exemple


cpu time 13872us
gpu time 8ms 349us
visible fragments 113863, shaded fragments 120186, pixels 368640, ratio 0.308873
overdraw
8 or more
7
6
5
4
3
2
1

Exemple

certains pixels :

- sont recouverts par > 4 fragments,
- le fragment shader est exécuté > 4 par pixel...
- $==$ 4 fois trop pour construire l'image finale...

pourquoi ?

- la visibilité du fragment est testée après l'exécution des fragments shaders...
- les fragments non visibles sont quand meme calculés (pour rien)

N'exécuter qu'une seule fois le fragment shader par pixel...

comment ?

- ne pas laisser openGL dessiner les primitives et exécuter les fragments shaders dans l'ordre imposé par le pipeline...
- ??

N'exécuter qu'une seule fois le fragment shader par pixel...

comment ?

- ne pas laisser openGL dessiner les primitives et exécuter les fragments shaders dans l'ordre imposé par le pipeline...
- solution pratique : 2 étapes,
- étape 1 : utiliser un fragment shader le plus simple possible...
- ... et stocker les informations nécessaires aux calculs,
- étape 2 : relire les informations stockées et finir les calculs d'éclairage.

En pratique :

étape 1 :

- stocker la position et la normale associée à chaque fragment,
- dans un framebuffer avec 2 textures / drawbuffers

étape 2 :

- relire la position et la normale de chaque fragment visible,
- calculer l'influence des sources de lumière...

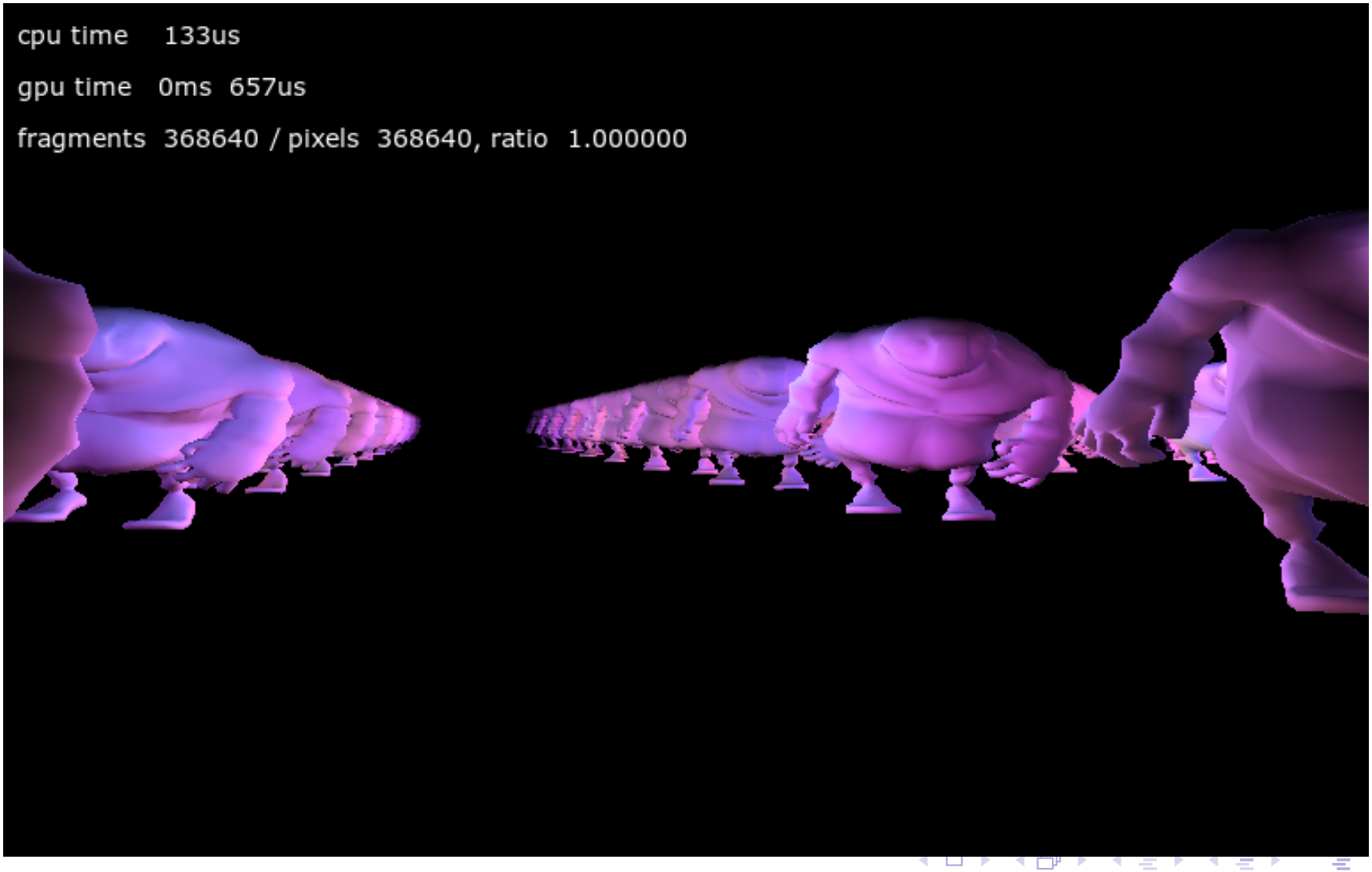
Résultat :


cpu time 145us
gpu time 1ms 907us
fragments 368640 / pixels 368640, ratio 1.000000

Résultat :

- en moyenne > 2 plus rapide que la solution directe !
- peut mieux faire ?

Résultat :

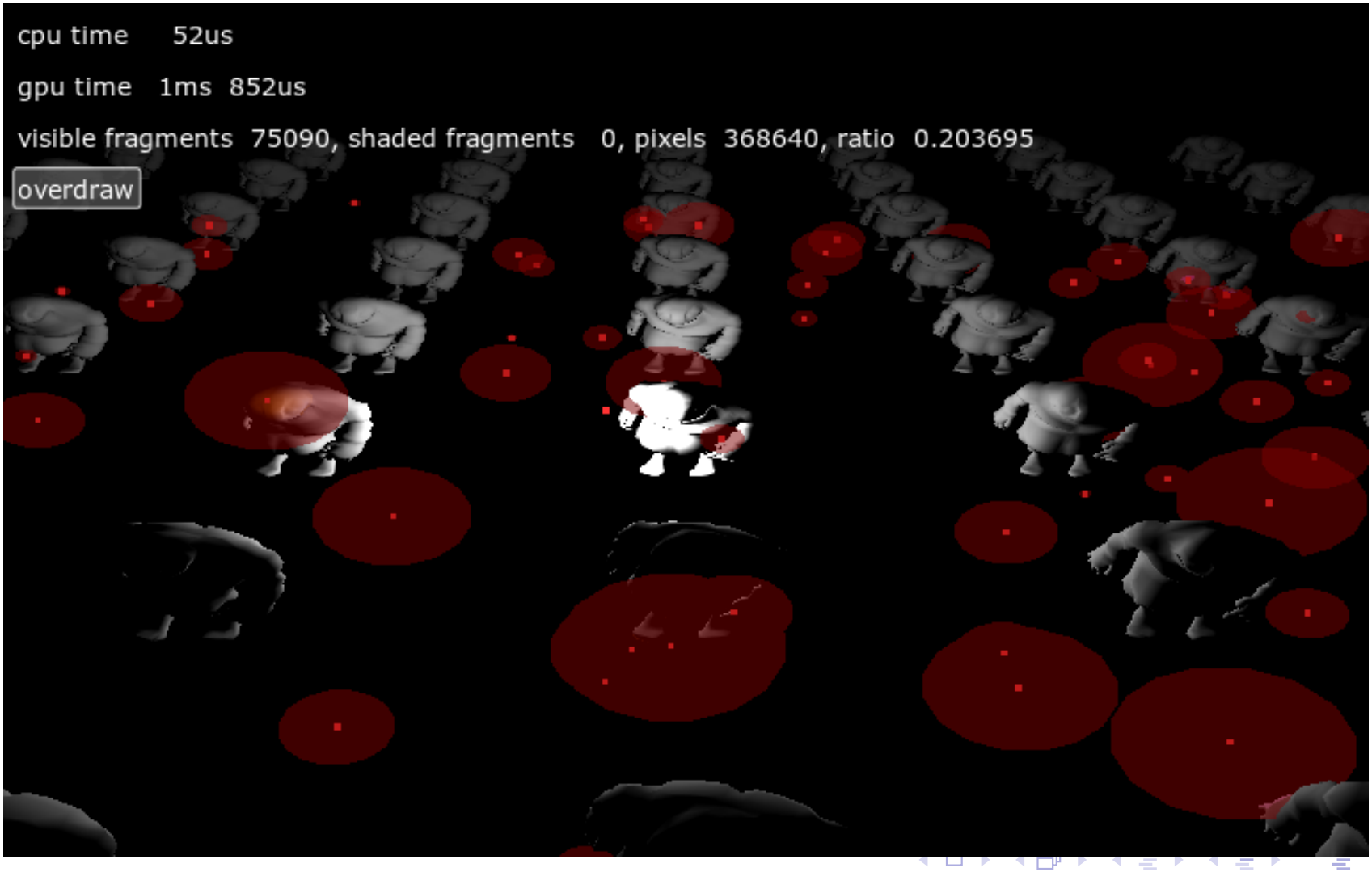

cpu time 133us
gpu time 0ms 657us
fragments 368640 / pixels 368640, ratio 1.000000

Résultat :

- les fragment shaders ne sont exécutés qu'une seule fois par pixel de l'image...
- ... mais uniquement pour les pixels contenant des objets !

peut mieux faire ?

peut mieux faire ?



Peut mieux faire ?

- il est inutile de calculer l'influence des sources de lumières trop loin...
- (rappel : le flux incident est inversement proportionnel au carre de la distance entre la source et le point...)
- de nombreuses sources ont une influence (quasi-) nulle sur un point.

Eliminer les sources sans influence

à lire :

- comment calculer, dans le repère de l'image, un englobant de la zone d'influence de chaque source ?
- "2D Polyhedral Bounds of a Clipped, Perspective-Projected 3D Sphere"
  M. Mara, M. McGuire, jgt
- "Intersecting Lights with Pixels"
  A. Lauritzen, siggraph 2012

Eliminer les sources sans influence

idée :

- pour chaque pixel,
- pour chaque source,
- si la source influence le pixel,
  calculer la lumière réfléchie.

comment accélérer cet algorithme ?
Eliminer rapidement les sources sans influence...

Eliminer les sources sans influence

idée :

- pour chaque *groupe* de pixels,
- pour chaque source,
- si la source influence un pixel du *groupe*,
  calculer la lumière réfléchie (par chaque pixel du groupe).

quelle complexite ?

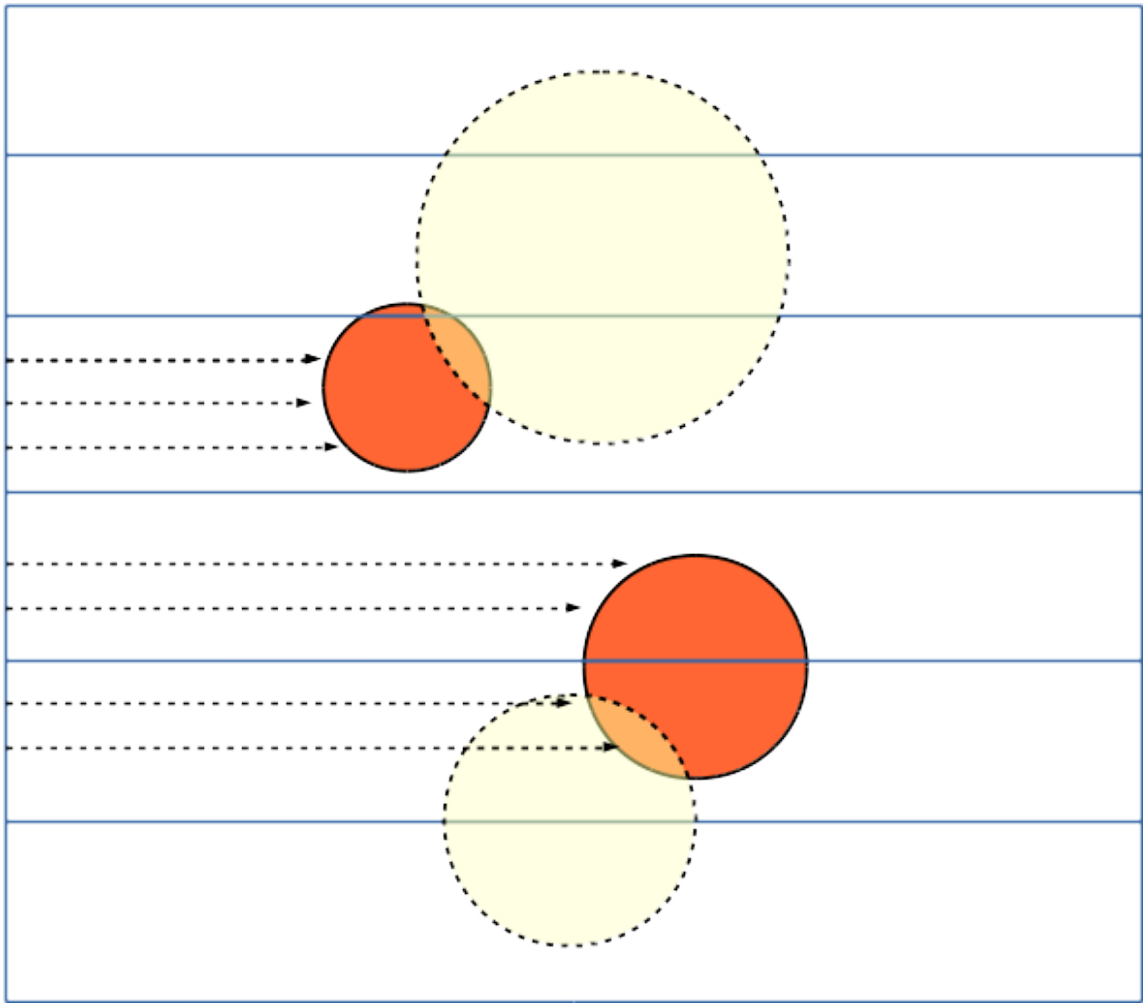
Eliminer les sources sans influence

influence sur un groupe de pixels :

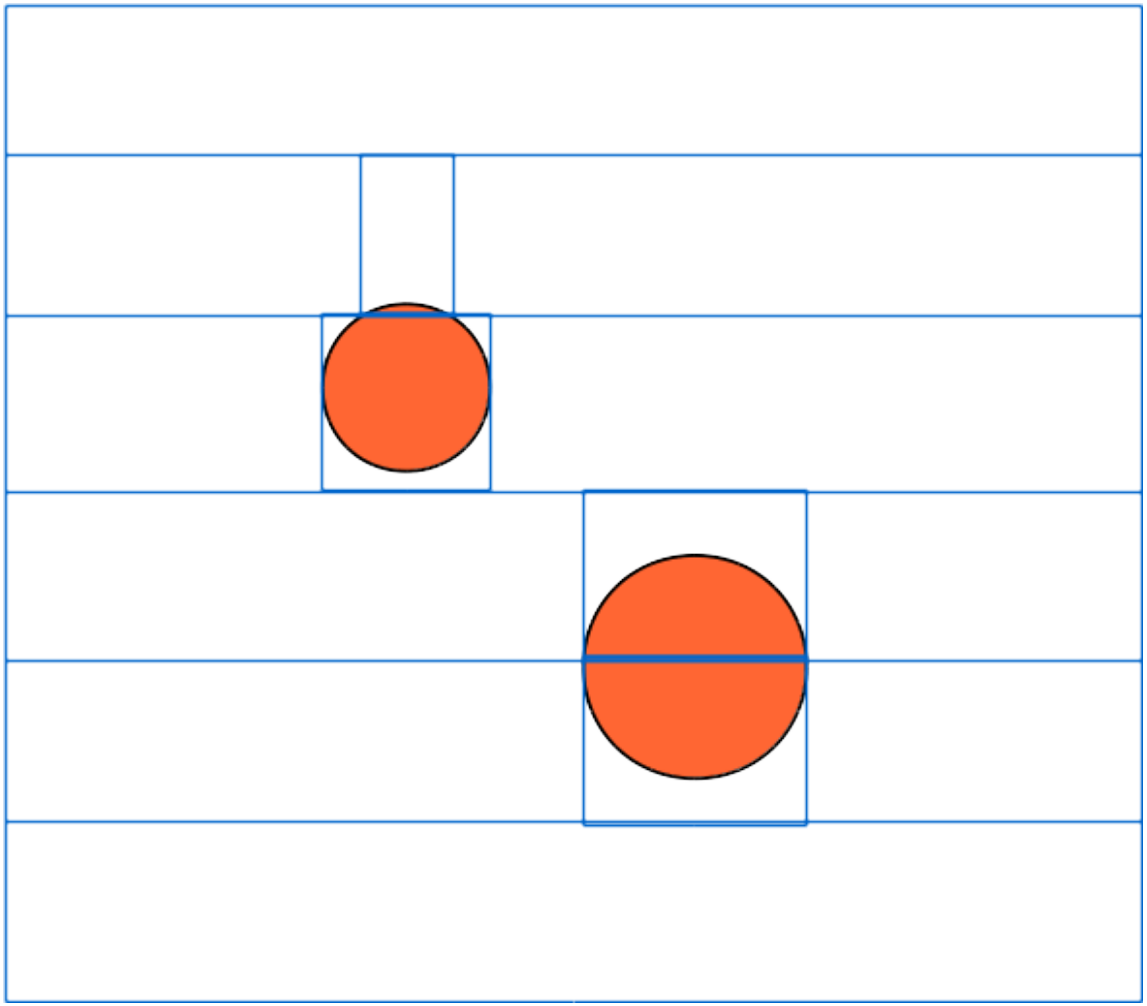
- la source à une zone (sphère) d'influence,
- les pixels correspondent à des points / fragments,
- si un des fragments est dans la sphère d'influence de la source, finir le calcul d'éclairage.

grouper les fragments et tester efficacement l'intersection avec la sphère d'influence de la source ?

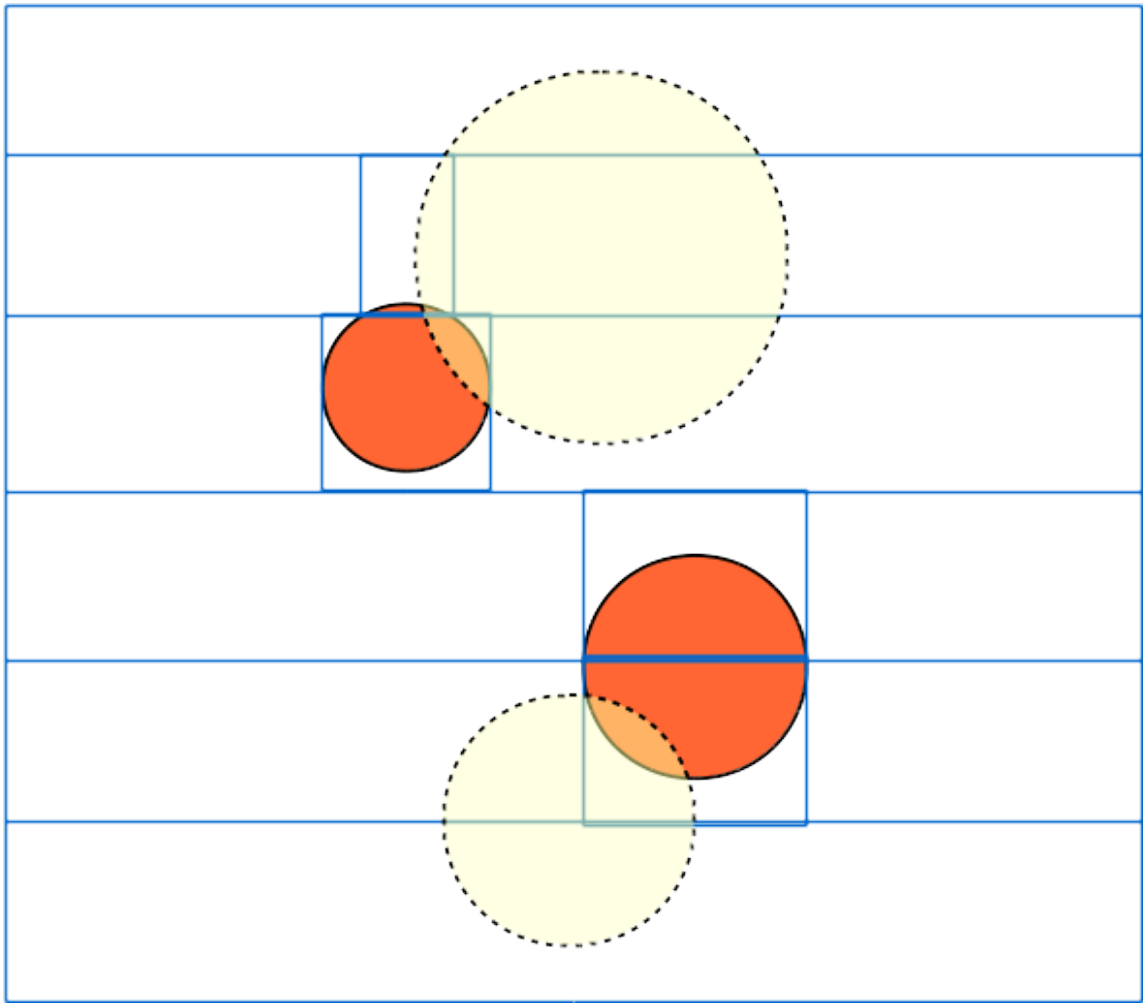
Eliminer les sources sans influence



Eliminer les sources sans influence



Eliminer les sources sans influence



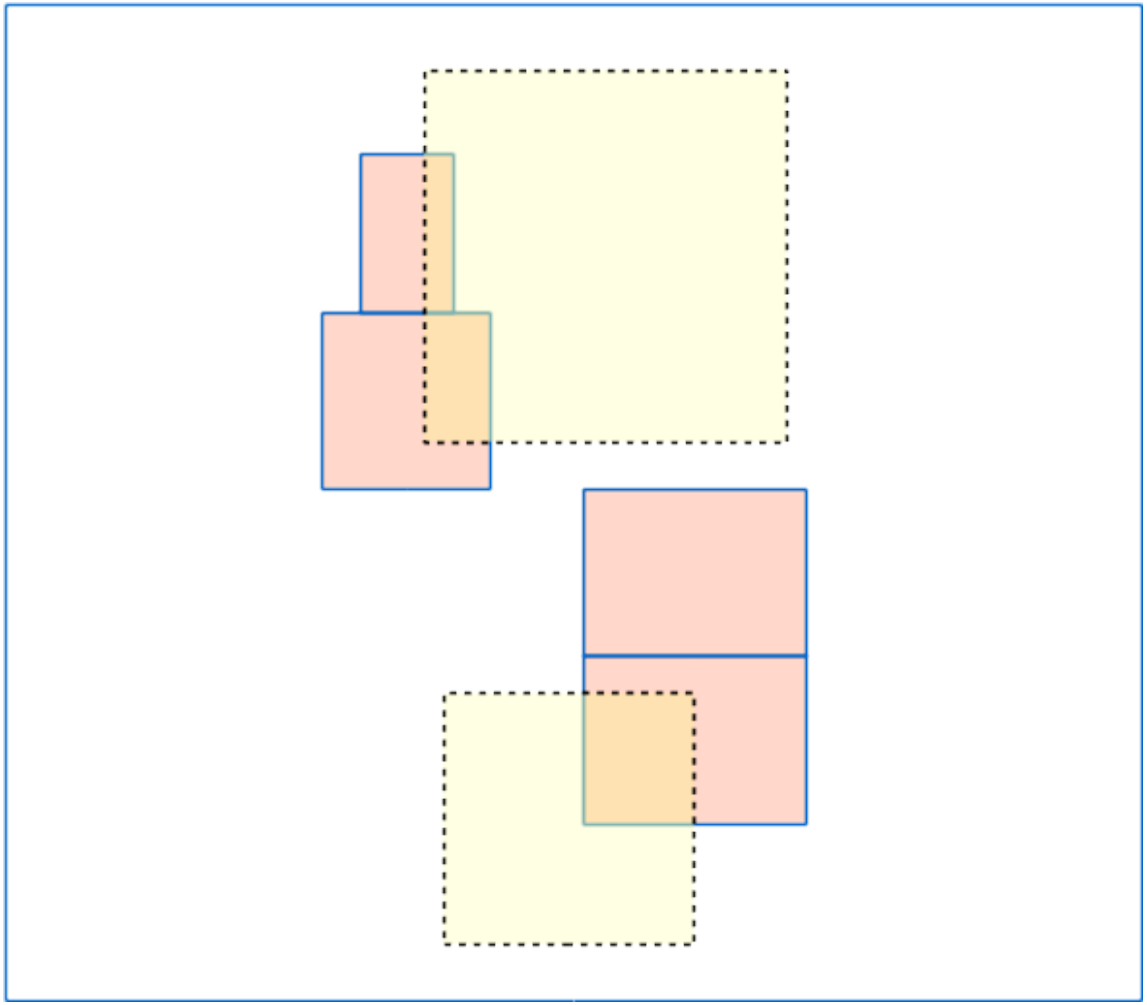
Eliminer les sources sans influence

englobants :

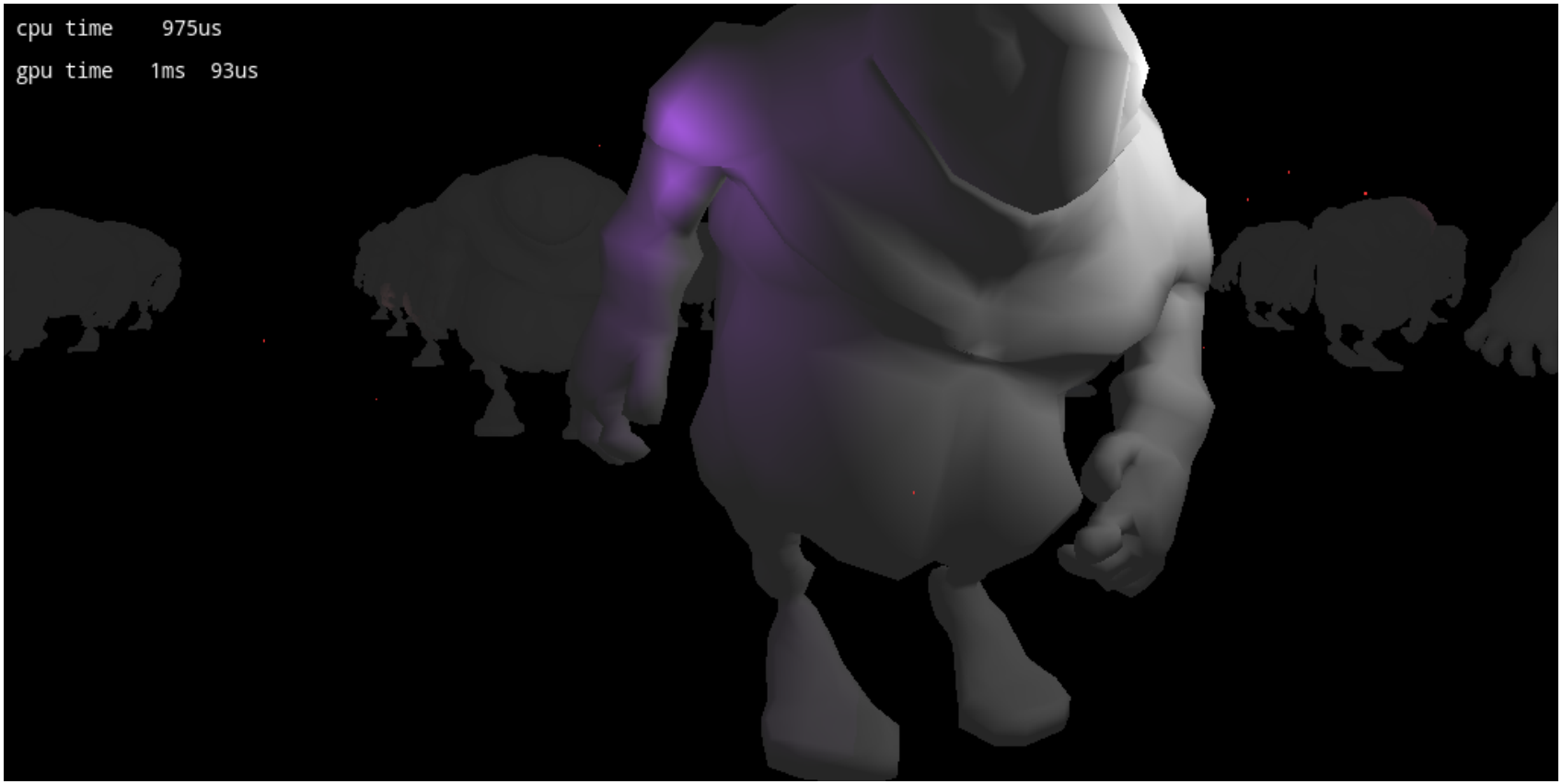
- construire les englobants des fragments de chaque groupe,
- tester si l'englobant est séparé de la sphère / englobant des sources.

rappel: 2 englobants convexes ne s'intersectent pas si un plan les sépare...

Eliminer les sources sans influence



Eliminer les sources sans influence



Eliminer les sources sans influence

nombre de sources par bloc (bon detecteur de discontinuités... pourquoi ?)

réalisation efficace

idée :

- construire les englobants des groupes de pixels,
- (découper l'image en blocs),
- tester *parallèle* l'influence de chaque source sur chaque bloc...

construire l'englobant d'un groupe de fragments

algo classique :

- calculer le min et le max d'un ensemble de valeurs
- ??

en parallèle ?

Réductions parallèles

algo classique :

- réduction :
- pour chaque valeur z :
- zmin= min(zmin, z)
- zmax= max(zmax, z)
- version parallèle ?

propriété : pas d'ordre particulier pour extraire le min ou le max d'un ensemble, ou d'un sous ensemble.

Réductions parallèles

décomposition type "diviser pour régner" :

- étape 1 :
- pour chaque sous ensemble, extraire le min et le max,
- étape 2 :
- traiter les résultats intermédiaires...
  (extraire le min et le max des résultats de l'étape 1).

étape 1 : le traitement de chaque sous ensemble est indépendant des autres == simple à paralléliser.

Réduction parallèles : solution alternative

autre solution :

- utiliser la synchronisation :
- zmin= atomicMin(zmin, z),
- zmax= atomicMax(zmax, z).

ca marche, mais peut mieux faire...

Eliminer les sources sans influence

épaisseur de chaque bloc (bon detecteur de discontinuités... pourquoi ?)

Eliminer les sources sans influence

idée de base correcte :

- mais peut mieux faire (toujours !)
- "A 2.5D culling for forward+"
  T. Harada, 2012
- "Practical Clustered Shading"
  E. Persson, 2013