M2-Images

Scènes complexes

J.C. Iehl

January 6, 2014

Résumé des épisodes précédents...

- méthodes externes / out-of-core,
- ré-organiser le pipeline graphique : solutions logicielles,
- notions de représentations multi-échelle...

En pratique

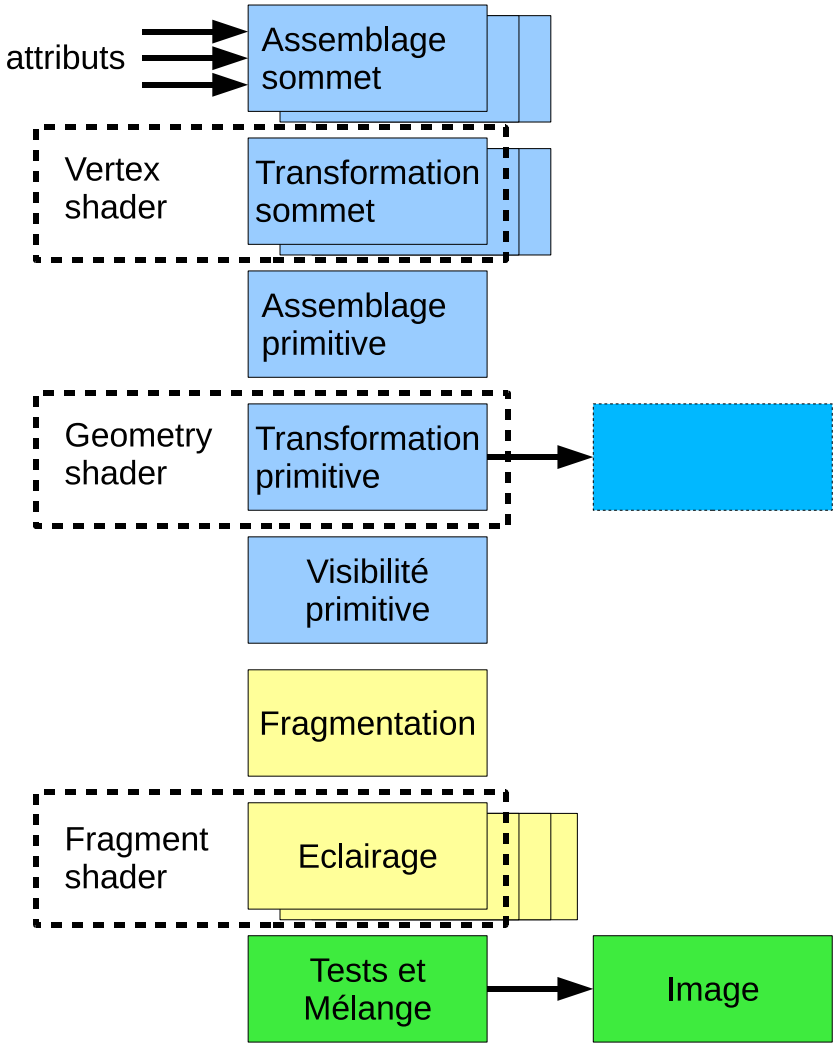
- que peut-on faire avec openGL ?
- comment fonctionne le pipeline ?
- utilisation efficace du pipeline ?

Introduction

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.

Introduction

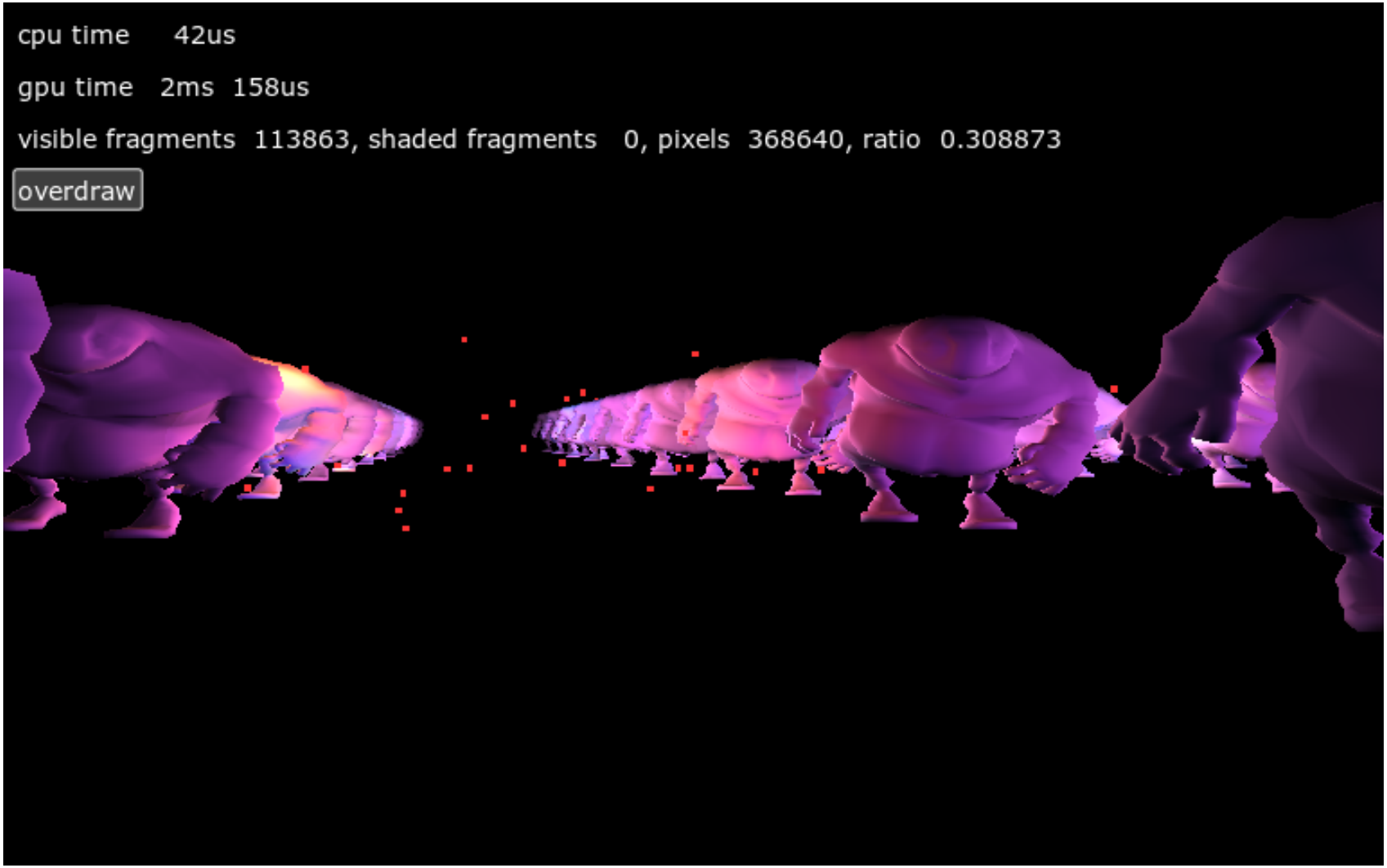


Introduction

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

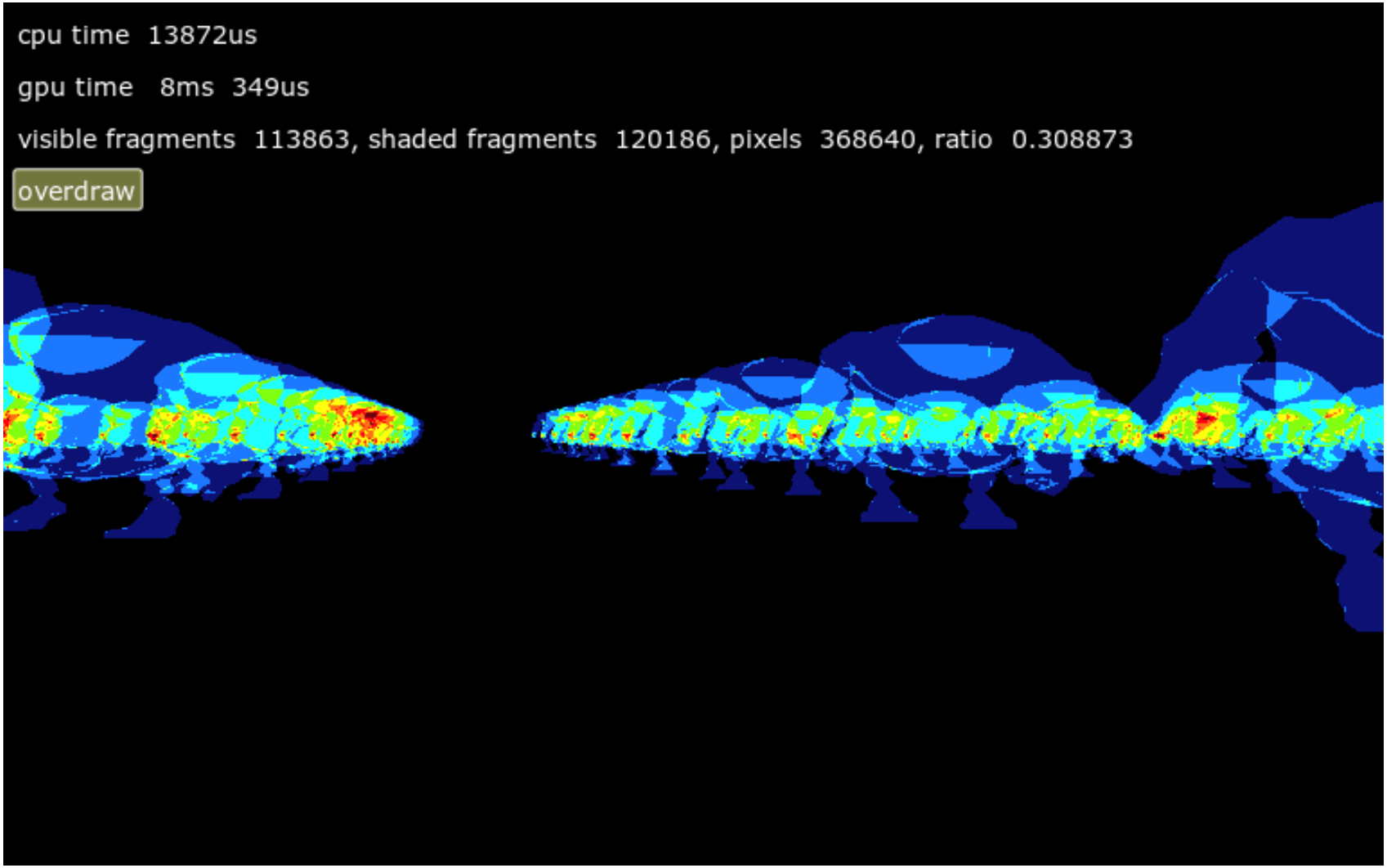

cpu time 42us
gpu time 2ms 158us
visible fragments 113863, shaded fragments 0, pixels 368640, ratio 0.308873
overdraw

Peut mieux faire ?

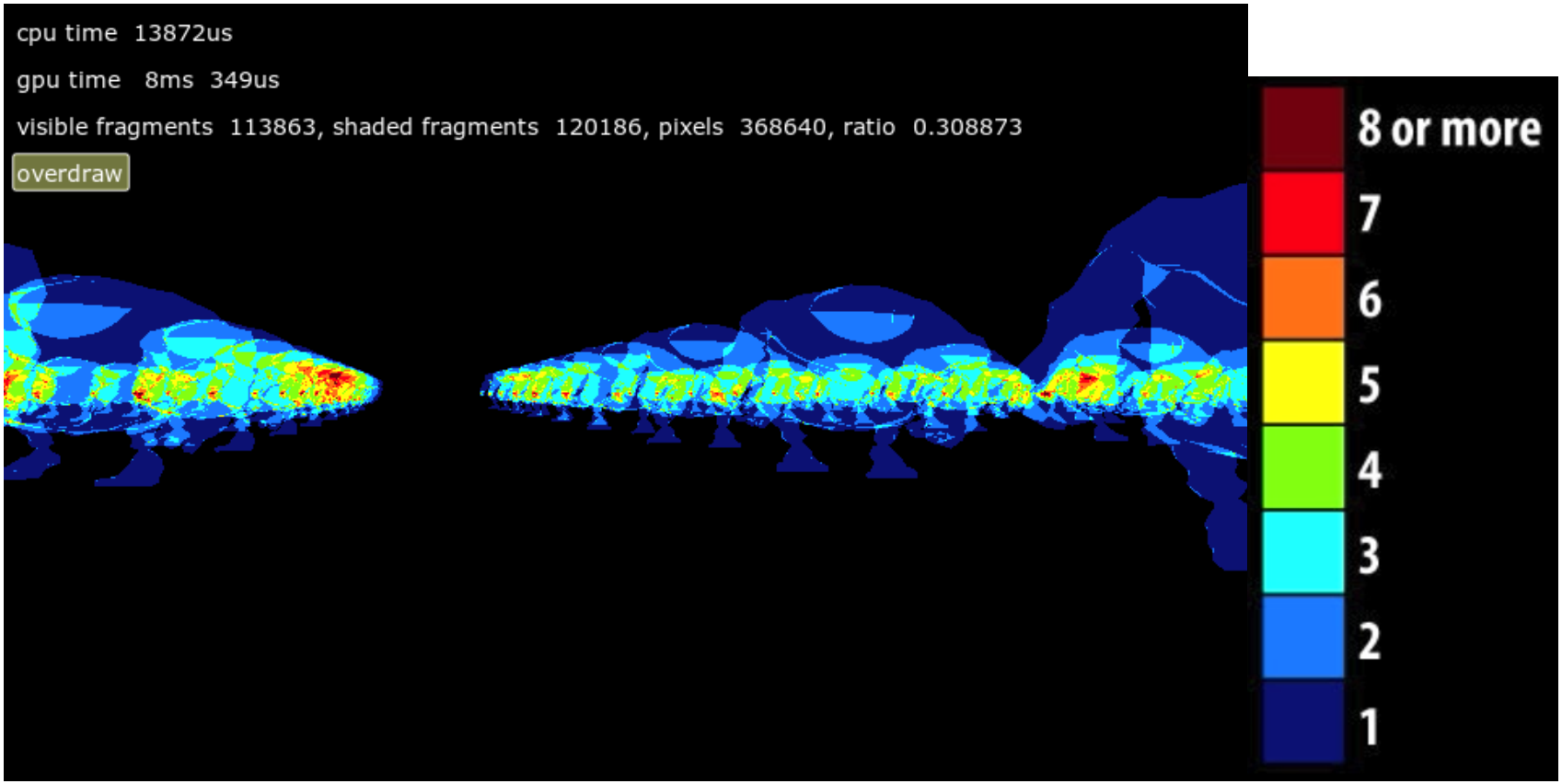
Peut mieux faire :

- 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 :

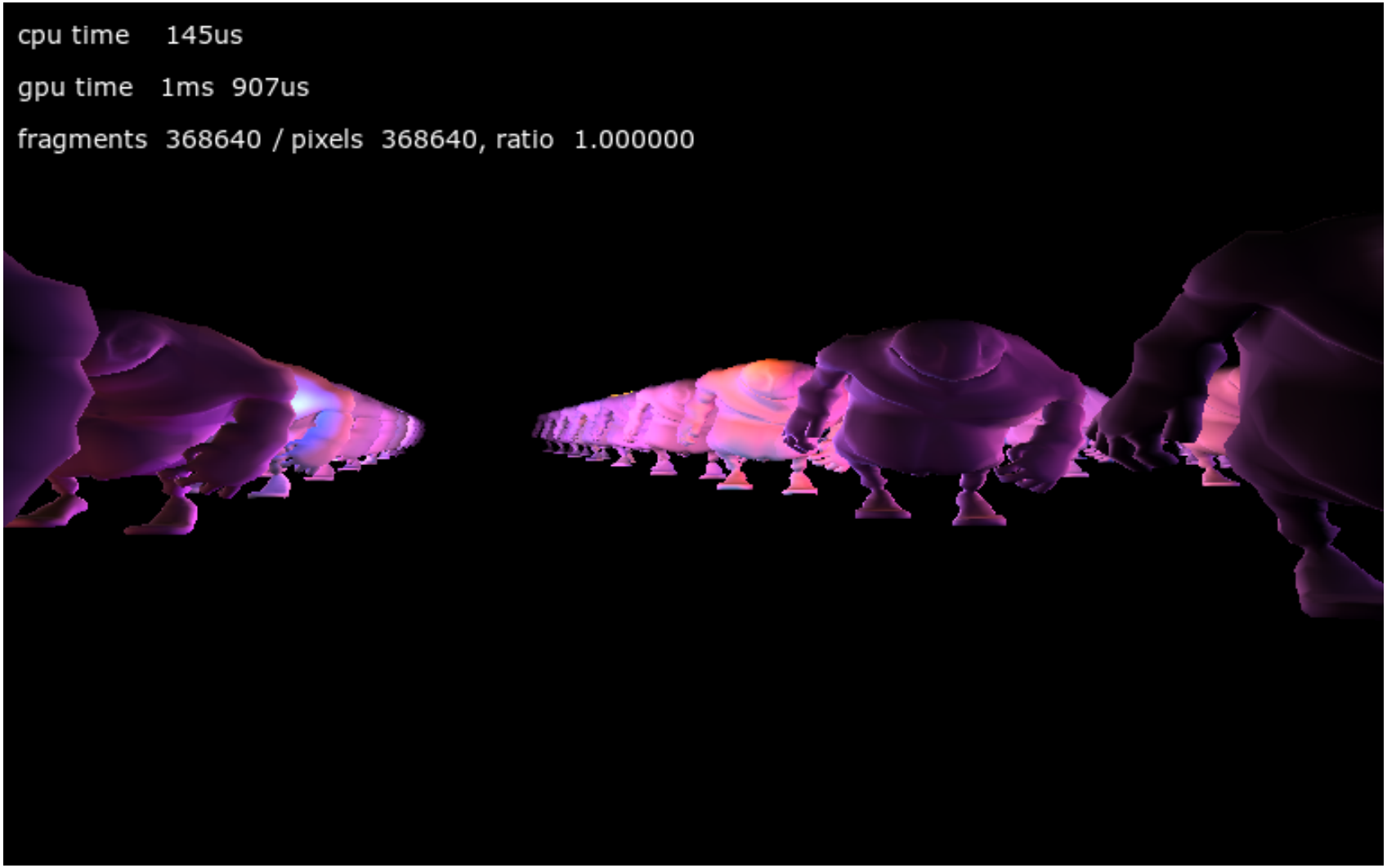
étape 1 :

- 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...

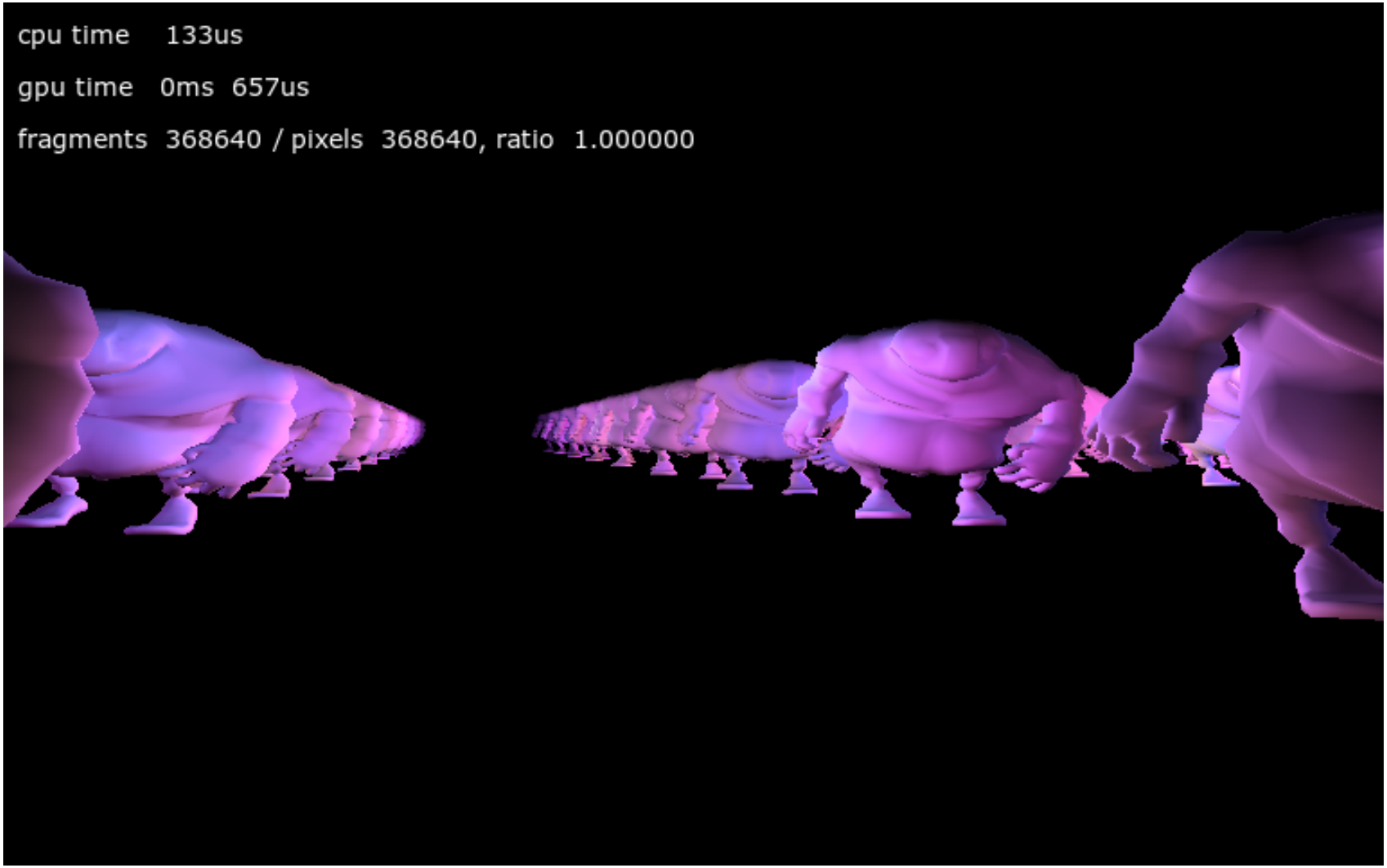
résultat :


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 :

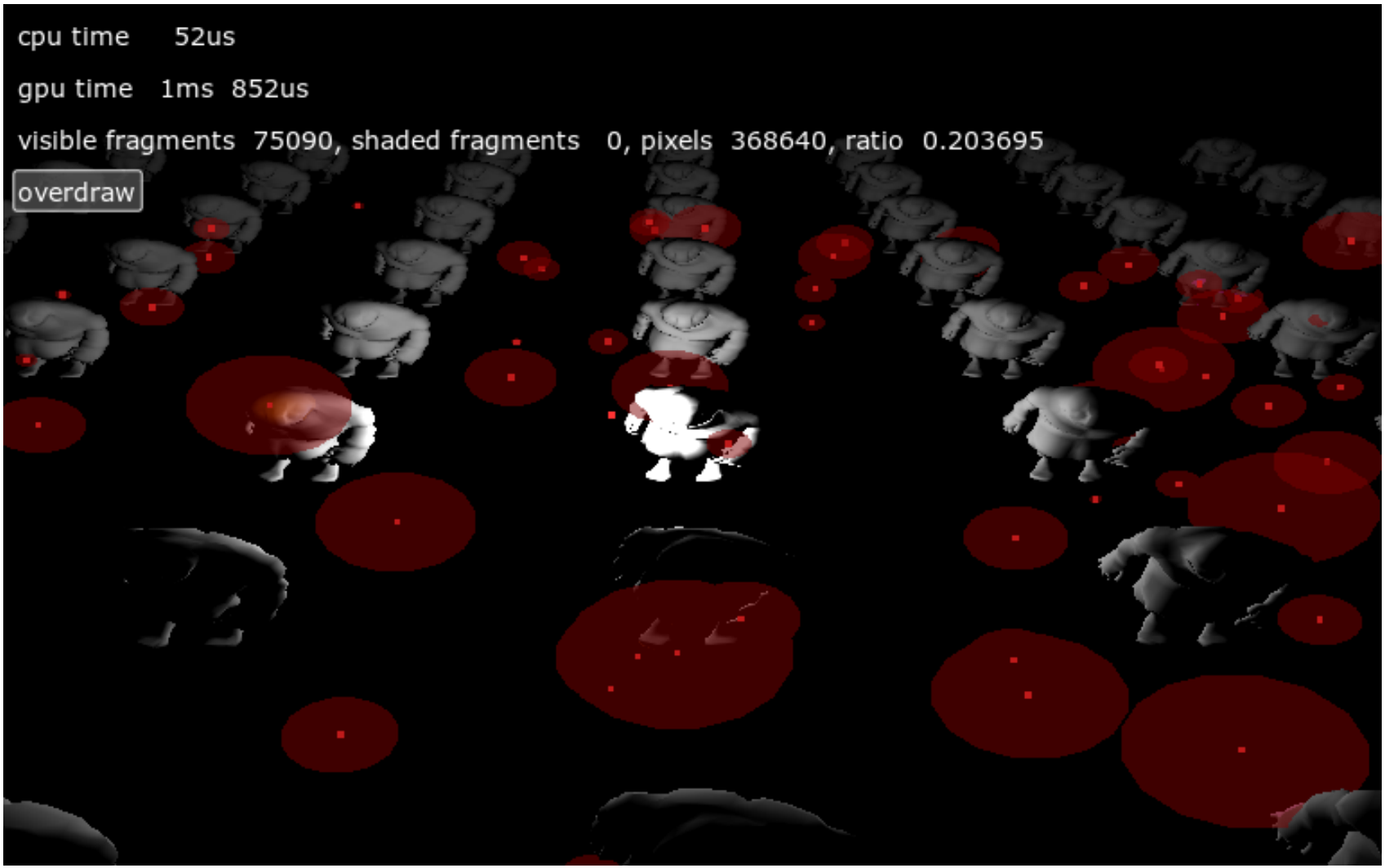

cpu time 133us
gpu time 0ms 657us
fragments 368640 / pixels 368640, ratio 1.000000

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.

éliminer 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