M2-Images

Scenes Complexes

J.C. Iehl

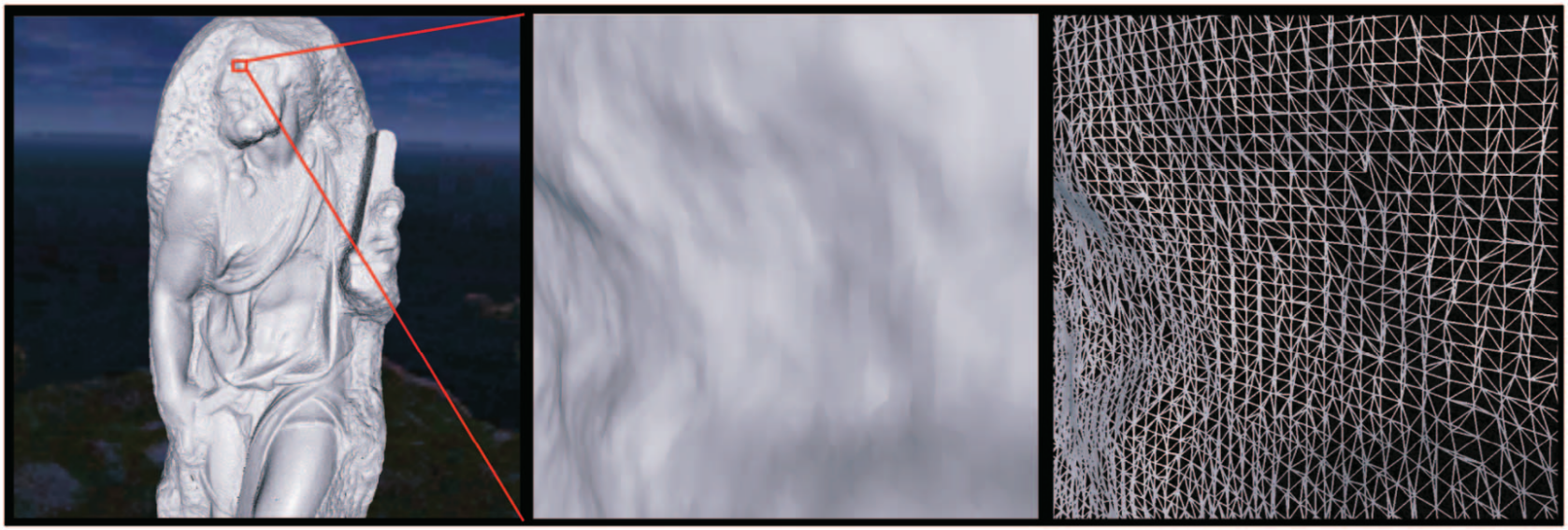
September 16, 2013

Qu'est ce que c'est ?

Complexe ?

exemples

exemples



"Cache-Oblivious Ray Reordering" (2009)

exemples

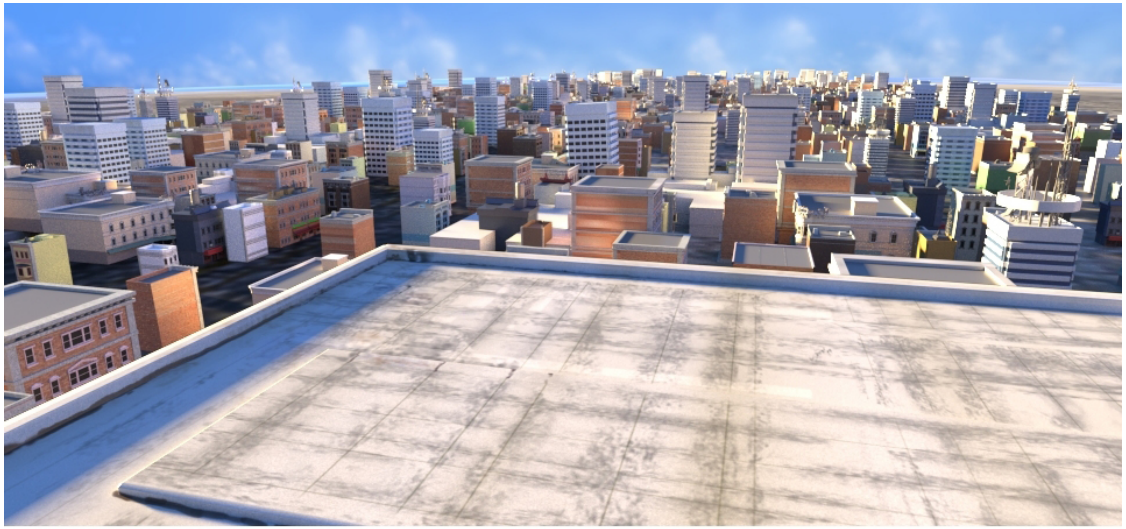
"Cache-Oblivious Ray Reordering" (2009)

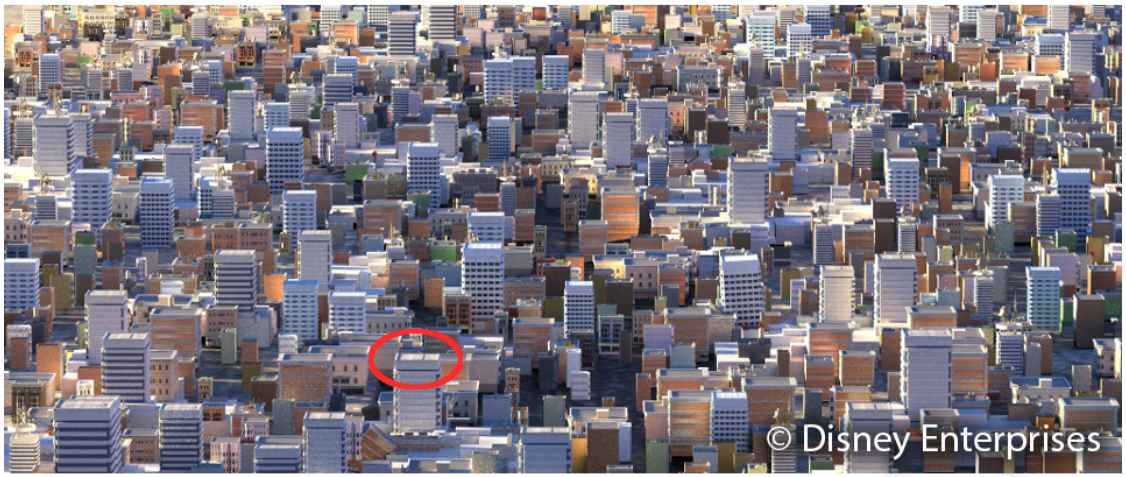
exemples

Figure 11: *Out-of-core city scene: 4096×1716, 512 SPP, path length 4; 530 M triangles, no instancing, 16 hours.*

"Sorted Deferred Shading for Production Path Tracing" (egsr 2013)

exemples



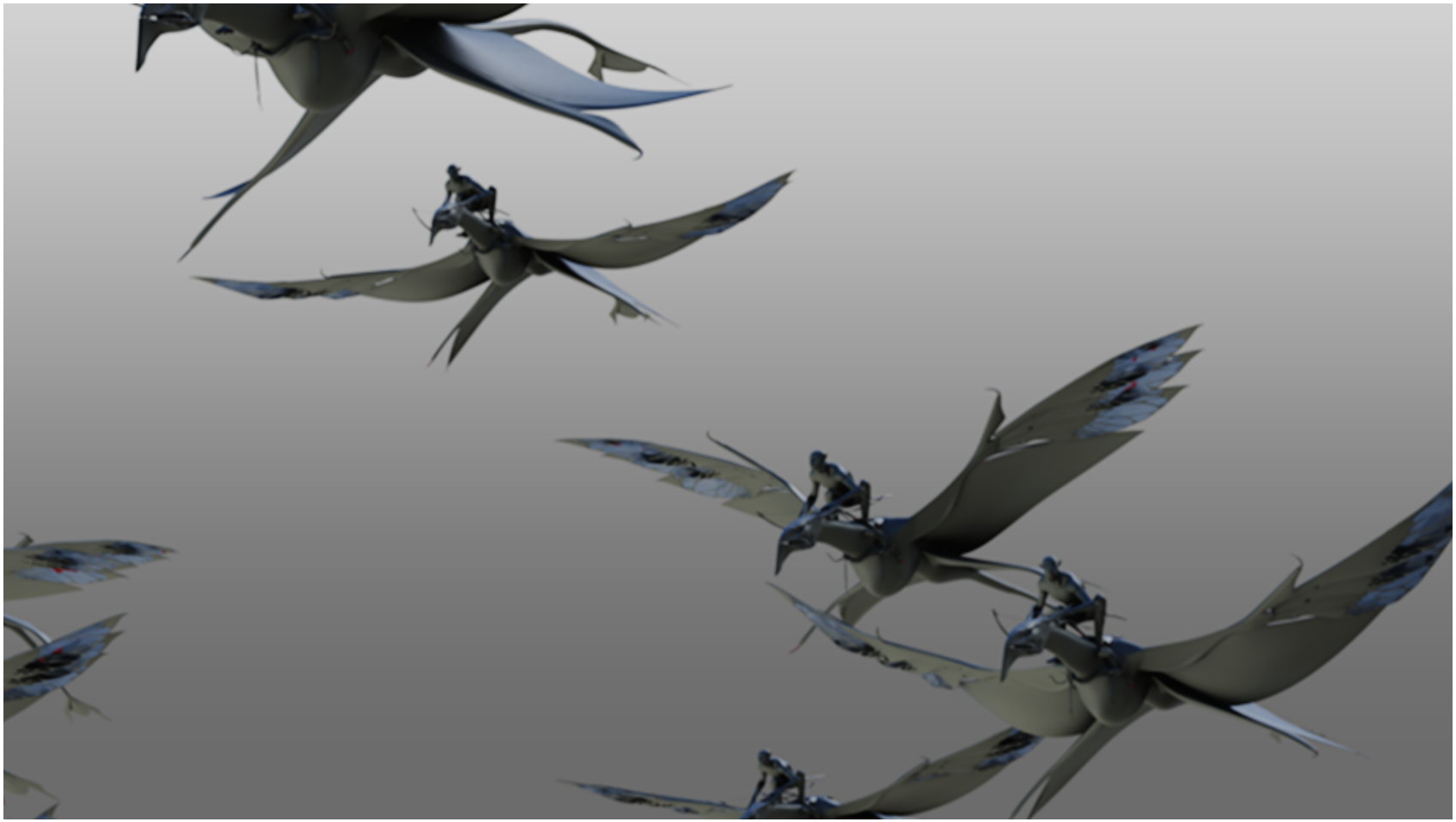


"Sorted Deferred Shading for Production Path Tracing" (egsr 2013)

exemples

"Avatar" (2009)

exemples



"Avatar" (2009)

exemples

3DRender.com lighting challenge (2010)

exemples

3DRender.com lighting challenge (2010)

exemples

"Cloudy with a chance of meatballs" (2009)

exemples

"How to train your dragon" (2010)

Pourquoi ?

différents types de complexité :

- nombre d'objets,
- détails géométriques,
- détails / subtilité de l'aspect, de l'apparence, textures,
- éclairage "naturel",
- éclairage par un grand nombre de sources de lumière,
- subtilités de l'éclairage,
- (shaders...)

Complexité géométrique

Pourquoi ?

- la nature est complexe,
- chaque objet est différent des autres,
- peu de régularité,

volonté de reproduire cette variété pour céer des environnements réalistes.

Complexité géométrique

problème principal :

- volume de données.
- très petites primitives (>100M pour une image 2M pixels ?).

solutions existantes :

- modélisation procédurale :
  générer des variations d'un seul modèle,
- nier le problème : méthodes out-of-core,
- réorganiser le pipeline graphique pour limiter / contrôler la quantité de mémoire nécessaire,
- construire des représentations multi-échelles des objets.

Méthodes out-of-core

simple :

- ajouter quelques disques supplémentaires à la zone de swap...
- mais pas très efficace :
  environ 2% des performances normales...

construire un système de cache plus “intelligent” ?

- ne charger qu'une partie des données,
- utiliser le fonctionnement de la méthode de calcul pour anticiper et pre-charger les données ?
- ne les charger qu'une seule fois ?

Cache “intelligent” ?

principe :

- un cache n’est utile que si les données sont accédées de manière “cohérente” ?
- $==$ *elles sont utilisées plusieurs fois,*
  et on ne veut les charger qu’une seule fois,
- quel type de cohérence peut-on exploiter ?

Cache et cohérence

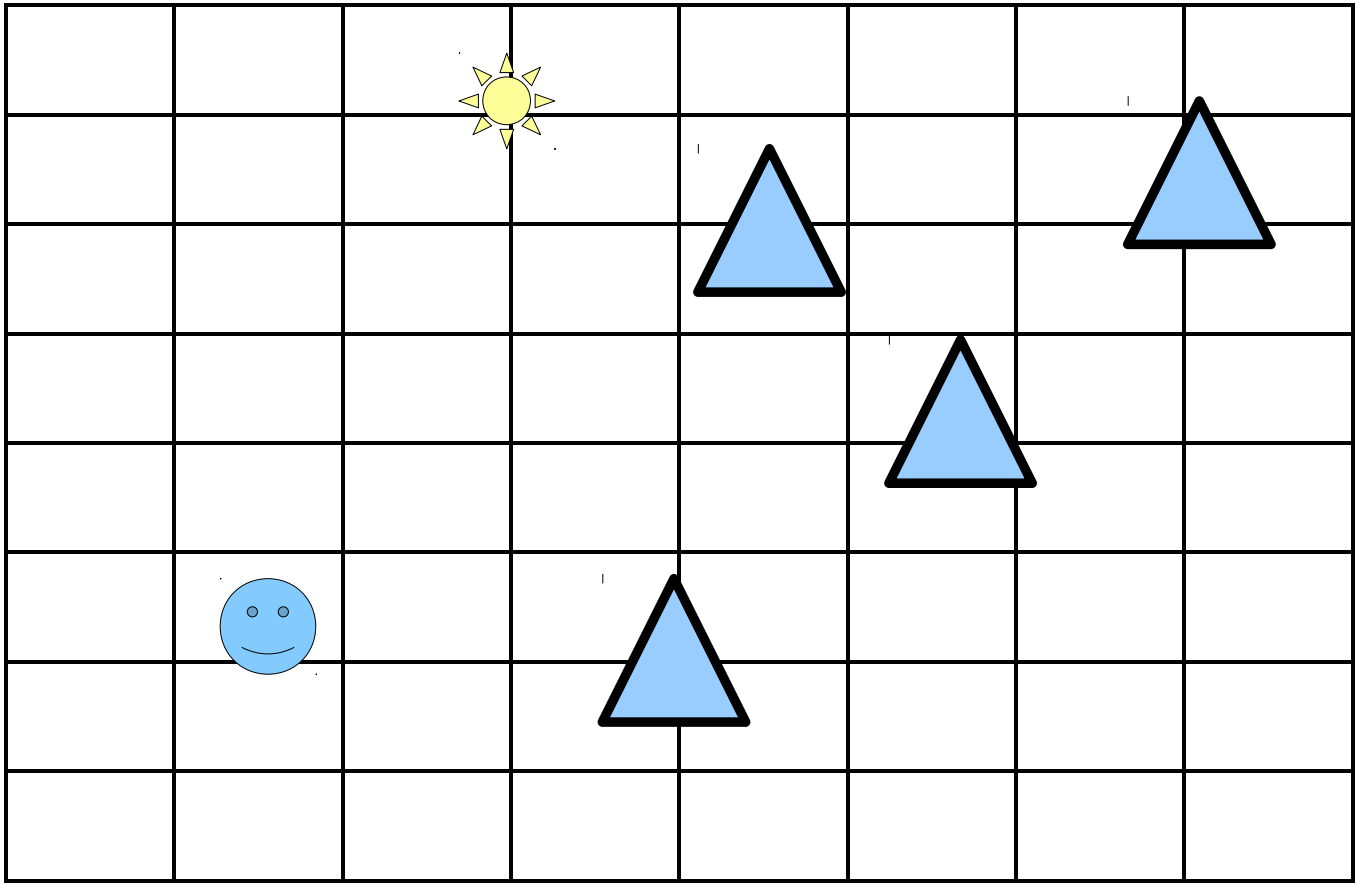
"Rendering Complex Scenes with Memory-Coherent Ray Tracing"
M. Pharr, C. Kolb, R. Gershbein, P. Hanrahan, siggraph 97

principes :

- triangule les objets de la scène,
- découpe la scène en blocs / grille régulière :
  chaque bloc stocke un ensemble de triangles,
- propage les rayons de bloc en bloc,
- groupe les rayons par bloc,
- re-ordonne les calculs d'intersection :
  calculs sur un bloc de rayons et un bloc de triangles.

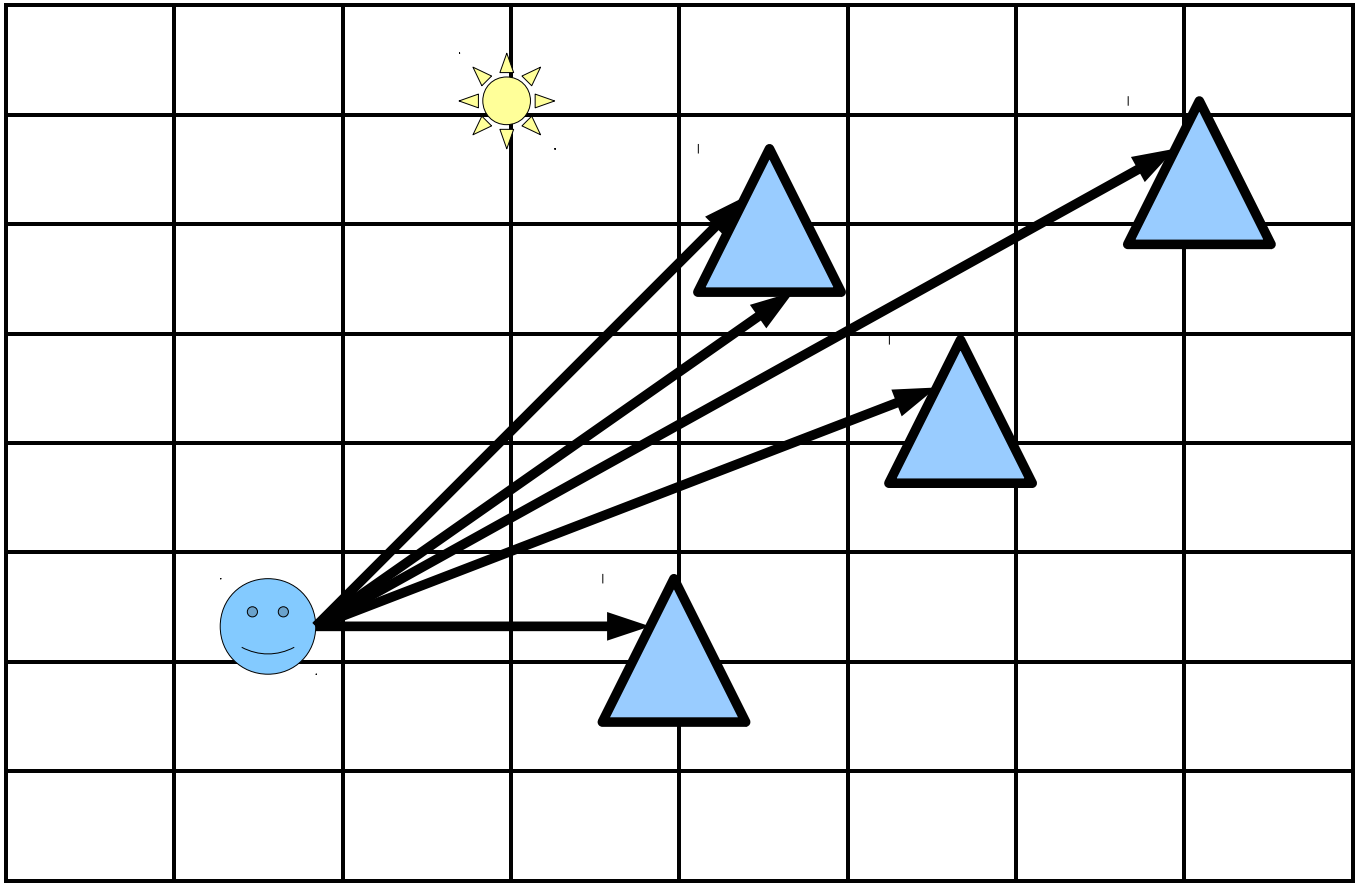
combien de fois est chargée la géométrie de chaque bloc ? quels types de cohérence ? combien de rayons ?

exemple



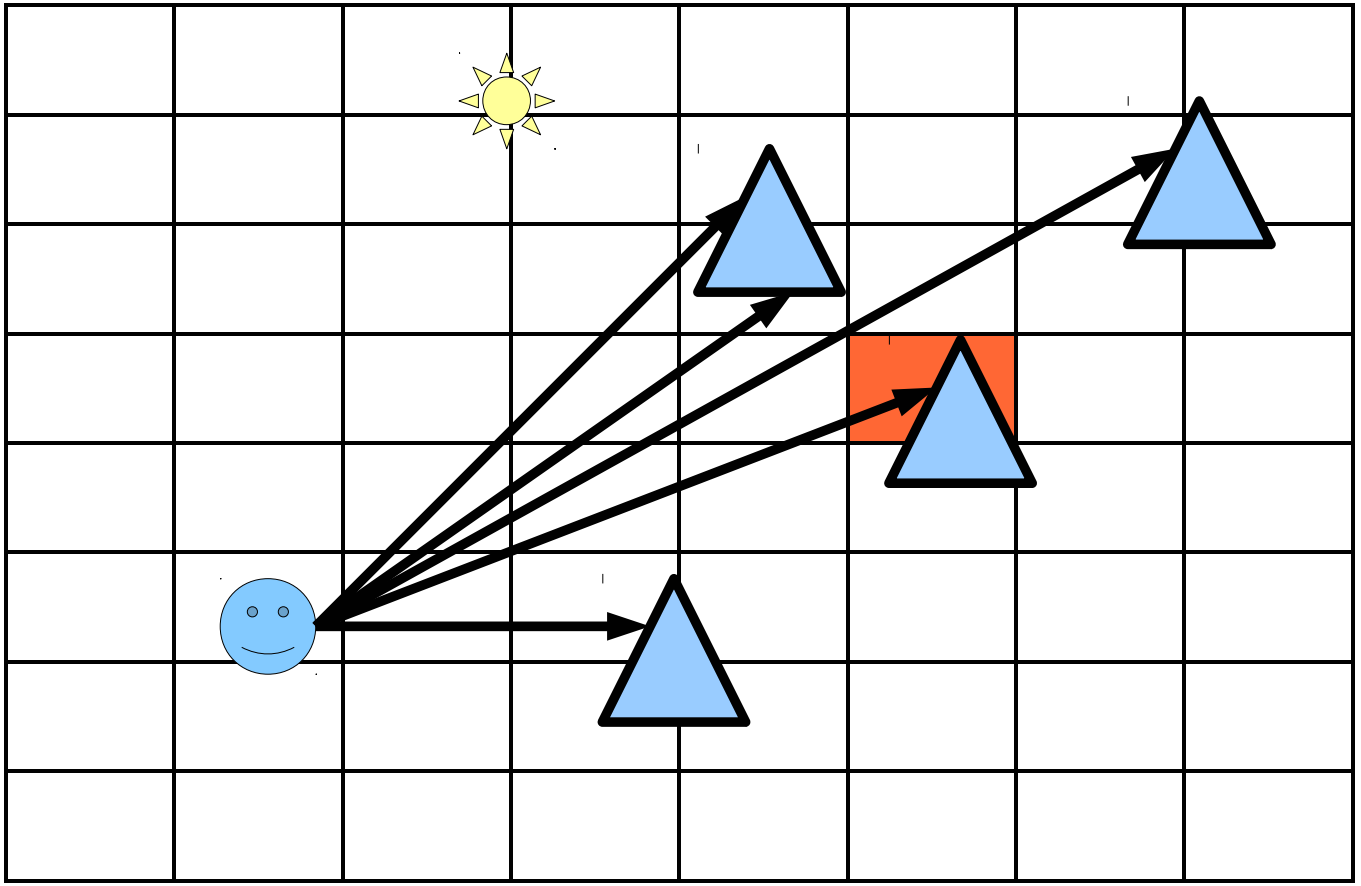
découpage de la scène

exemple



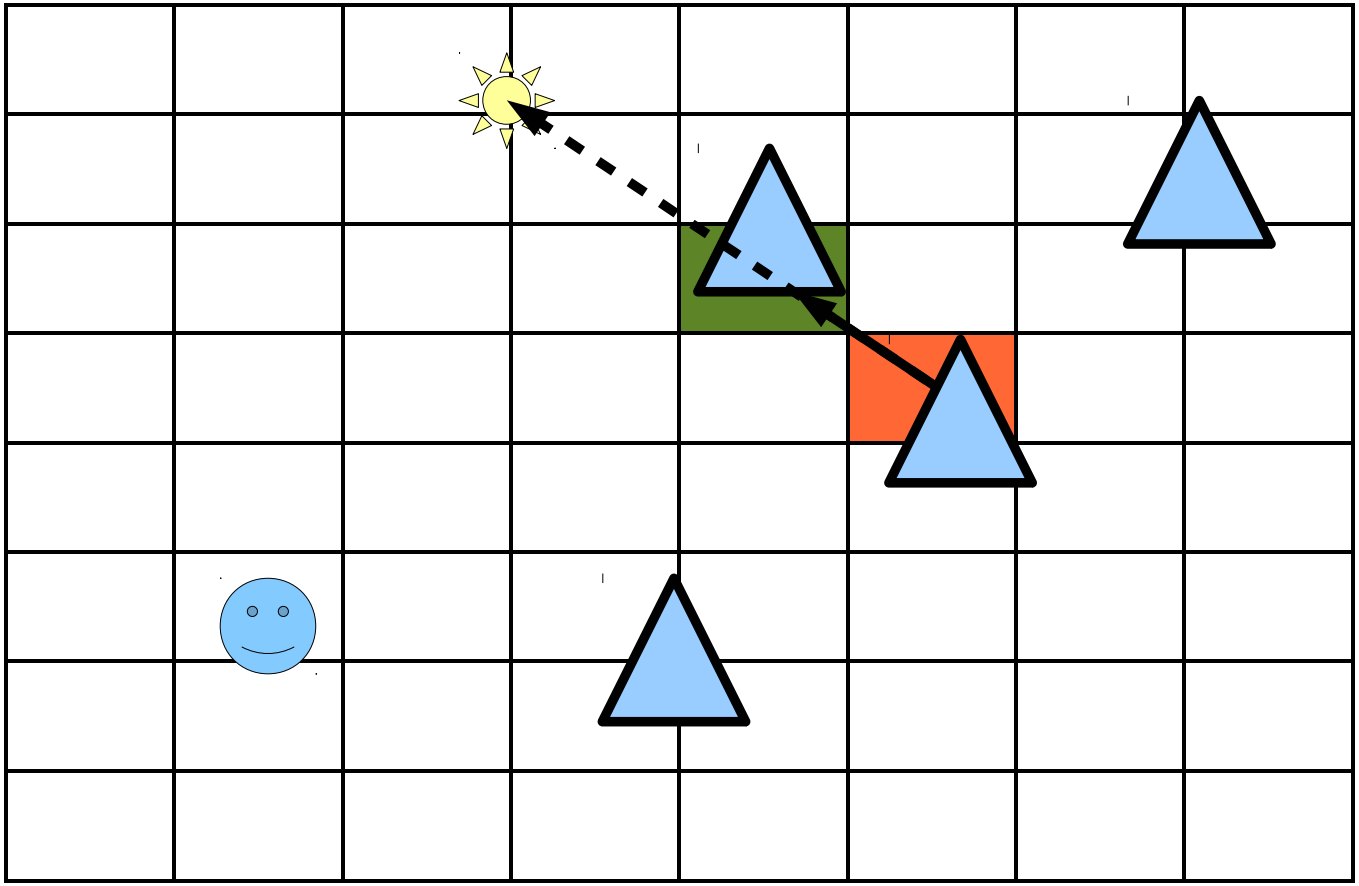
propagation des rayons, de bloc en bloc, jusqu'à trouver de la géométrie

exemple



charger la géométrie d'un bloc, intersection avec les rayons

exemple



calcul d'éclairage == tester la visibilité des sources de lumière

repropager les rayons par bloc, charger la géométrie

ordonnancement

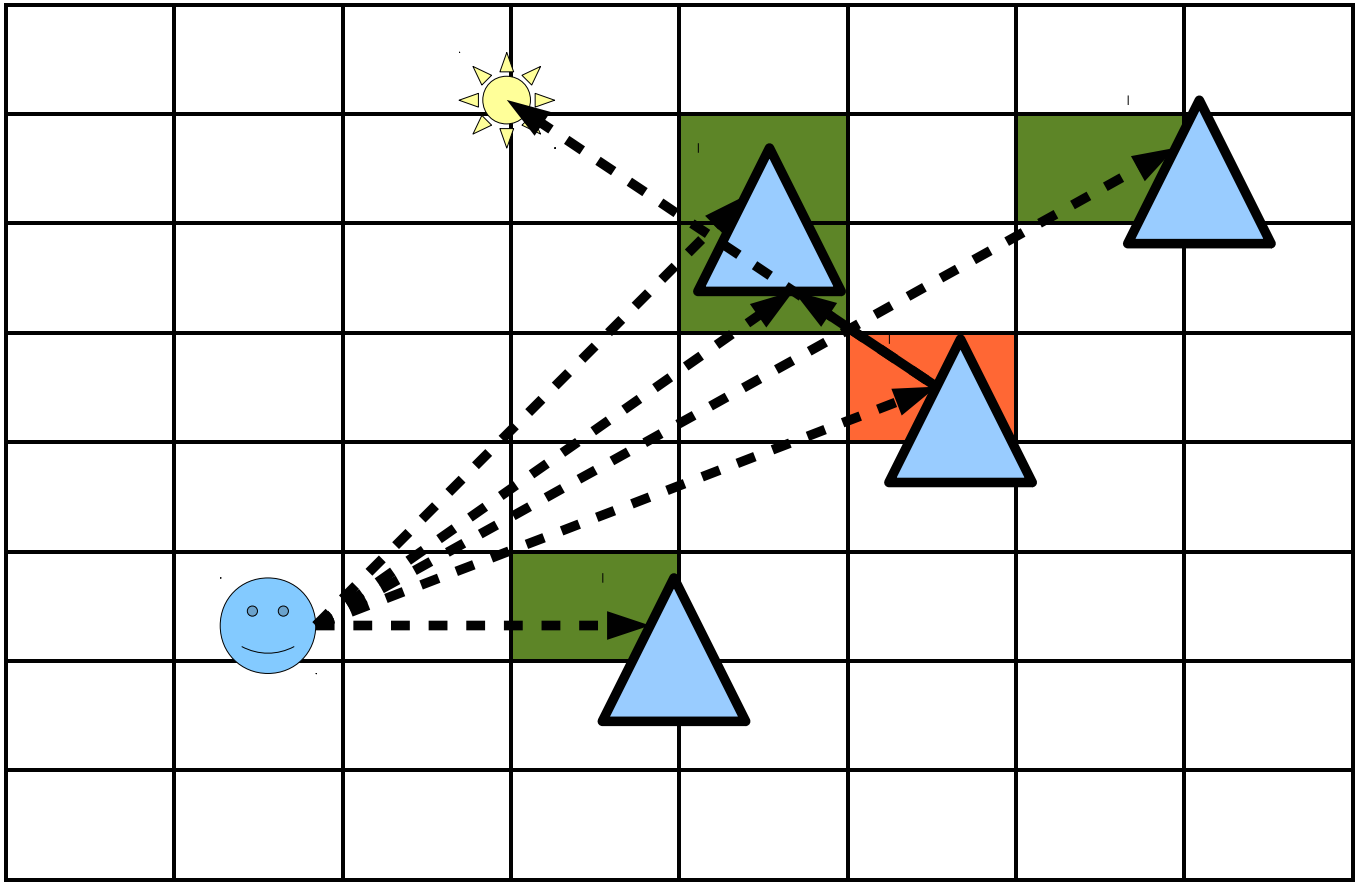
stratégie d'ordonnancement :

- il y a des rayons en attente de calcul d'intersection,
- il y a des blocs avec de la géométrie chargée,
- comment choisir quel bloc de rayons traiter ?

objectif :

minimiser le chargement de la géométrie.

exemple



quel bloc traiter / charger ?

ordonnancement

conséquence :

- ne pas faire les calculs dans l'"ordre"...
- mais dans l'ordre qui permet de minimiser les chargements,
- == privilégier les blocs dont la géométrie est chargée
- + privilégier les blocs qui permettent d'obtenir le plus de "résultats" (*de progresser*)

ordonnancement

ne pas faire les calculs dans l'ordre ?

- restructurer l'algorithme de calcul d'éclairage...
- et stocker les informations permettant de finir le calcul une fois le résultat de l'intersection connu.

le pixel auquel contribue le rayon + les paramètres nécessaires à évaluer la "contribution" du point d'intersection.

Cache et cohérence

avantages :

-
-

inconvénients :

-
-

Cache et cohérence

avantages :

- ne charge que la géométrie nécessaire,
- une fois par génération de rayon :
  visibilite, ombres, rebond 1, rebond 2...,
- gérer une scène $\times 10$ plus volumineuse...

inconvénients :

- contraintes sur la méthode de simulation,
  réorganisation des calculs == Monte Carlo “propre”,
- choisir la taille des blocs / caches...
- stocker les rayons en attente...
- gérer une scène $\times 10$ plus volumineuse...

Amélioration :

choisir la taille des blocs :

- utiliser une représentation hiérarchique / multi-échelle : un octree
- construction out-of-core d'un octree ?
- tri fusion externe par axe...

exemple d'algorithme de construction :
"Coherent Out-of-Core Point-Based Global Illumination"
J. Kontkanen, E. Tabellion, R.S. Overbeck, egsr 2011

et alors ?

et si la complexité n'est pas que géométrique ?

- données géométriques (primitives, surfaces ?)
- données supplémentaires (textures, shaders ?)

et si :

- le chargement des textures,
- le temps de compilation des shaders,

dominent le temps de rendu ?

exemple :

exemple :

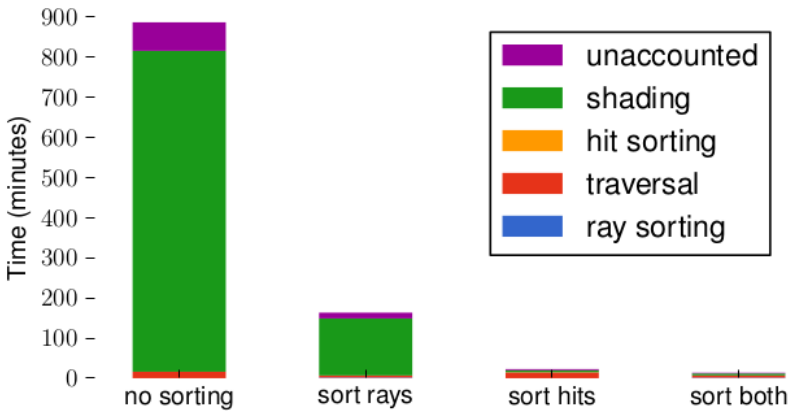


Figure 5: ***Interior scene with a single texture layer.*** *Sorting significantly reduces shading time (the dominant cost) due to more coherent texture lookups.*

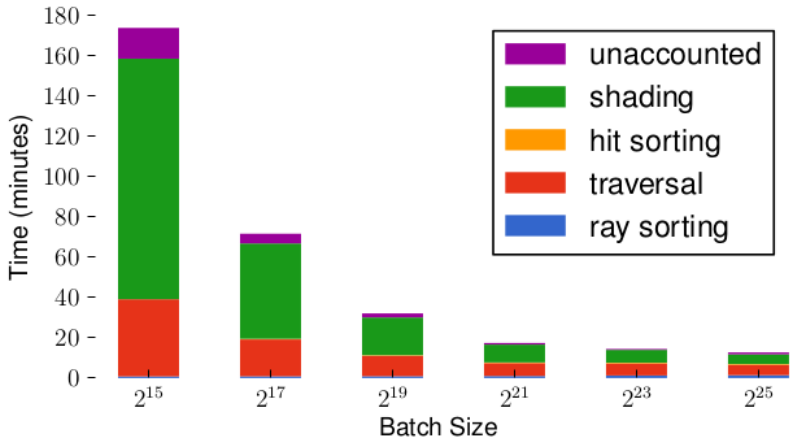


Figure 7: ***Interior scene with single texture layer.*** *Larger ray batches allow our method to discover more coherence through sorting, leading to better performance.*

"Sorted Deferred Shading for Production Path Tracing" (egsr 2013)

extraire la cohérence...

"Sorted Deferred Shading for Production Path Tracing"
C. Eisenacher, G. Nichols, A. Selle, B. Burley, egsr 2013.

idée :

- partitionner les rayons par origine / direction,
- partitionner les rayons par cellule intersectée,
- partitionner les rayons par mesh id / face id
- charger la géométrie, les textures, compiler les shaders,
- calculer...

... c'est simple !

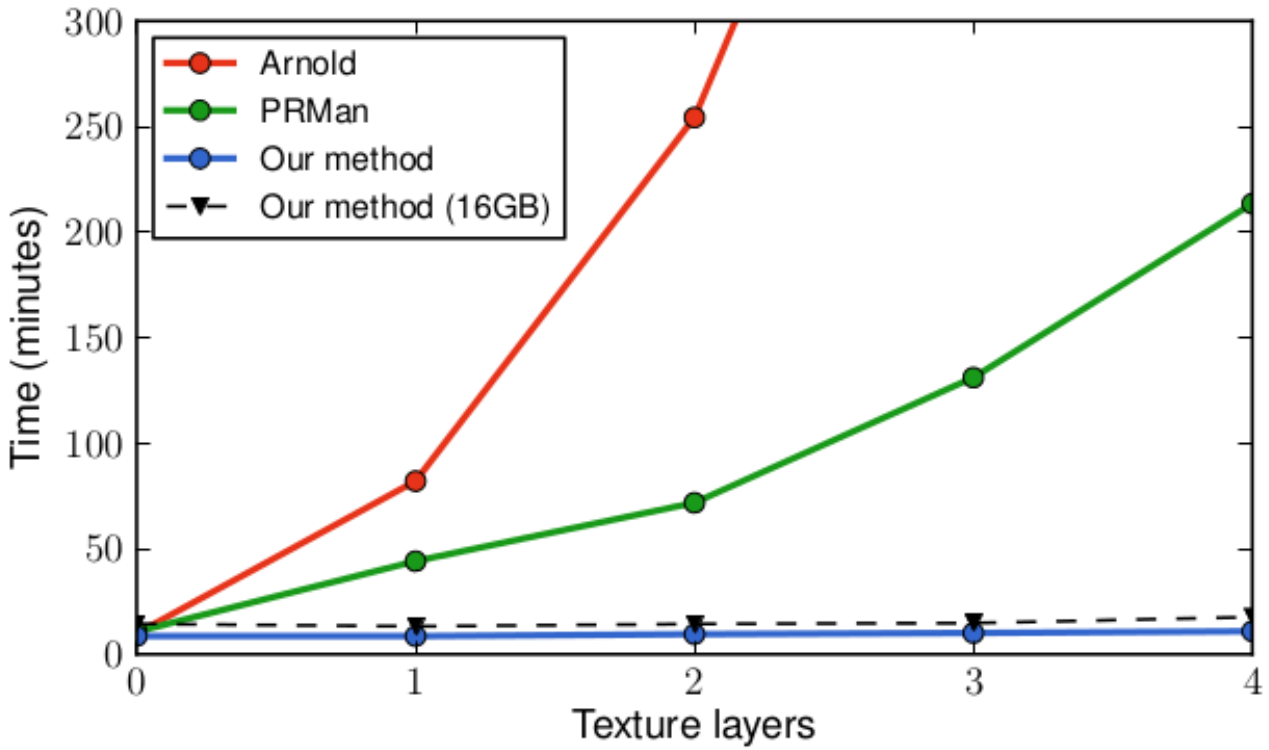


Figure 9: *Comparing texture performance of our method against production renderers on the interior scene.*

et alors ?

quelle est la primitive visible à travers chaque pixel ?

- ?

combien de primitives se projettent sur un pixel ?

- dépend de la distance et de la taille des primitives...

== un rayon par *triangle* pour évaluer la couleur du pixel ?

et alors ?

"The A-buffer, an antialiased hidden surface method"
L. Carpenter, siggraph 84

A-buffer :

- échantillonne un pixel par une grille 8×4,
- conserve *tous* les *fragments* (couleur + Z),
- calcule l'aire du pixel couverte par chaque primitive,
- détermine quel fragment est visible pour chaque *échantillon*,
- $\approx$ 32 Z-buffers classiques...
- mais traite également les objets semi-transparents.

"Compositing Digital Images"
T. Porter, T. Duff, siggraph 84

et alors ?

est ce que le problème est réglé ?

et alors ?

est ce que le problème est réglé ?

lequel ?

et alors ?

a lire :

"Ray Differentials and Multiresolution Geometry Caching for Distribution Ray Tracing in Complex Scenes"

P.H. Christensen, D.M. Laur, J. Fong, W.L. Wooten, D. Batali, eurographics 2003

Pipeline graphique

Modèles multi-échelles