M2-Images

construction hiérarchie d'englobants / BVH

J.C. Iehl

December 12, 2018

Résumé des épisodes précédents

résumé :

- déterminer la visibilité de 2 points,
- permet de calculer les transferts d'énergie,
- ombre, pénombre, reflet, transparence, ...

trop d'intersections calculées ...

Rappels : intersection rayon / arbre

algorithme :

- init : $t = \infty$
- si le rayon $o + [0...t] \cdot \vec{d}$ touche l'englobant du noeud :
  calculer les intersections avec les fils : t_{gauche}, t_{droite}
  visiter le fils proche, puis le fils loin
- si le noeud est une feuille :
  calculer l'intersection avec les primitives de la feuille,
  $t_{feuille} = min(t_1, t_2, ...)$
  $t = min(t, t_{feuille})$

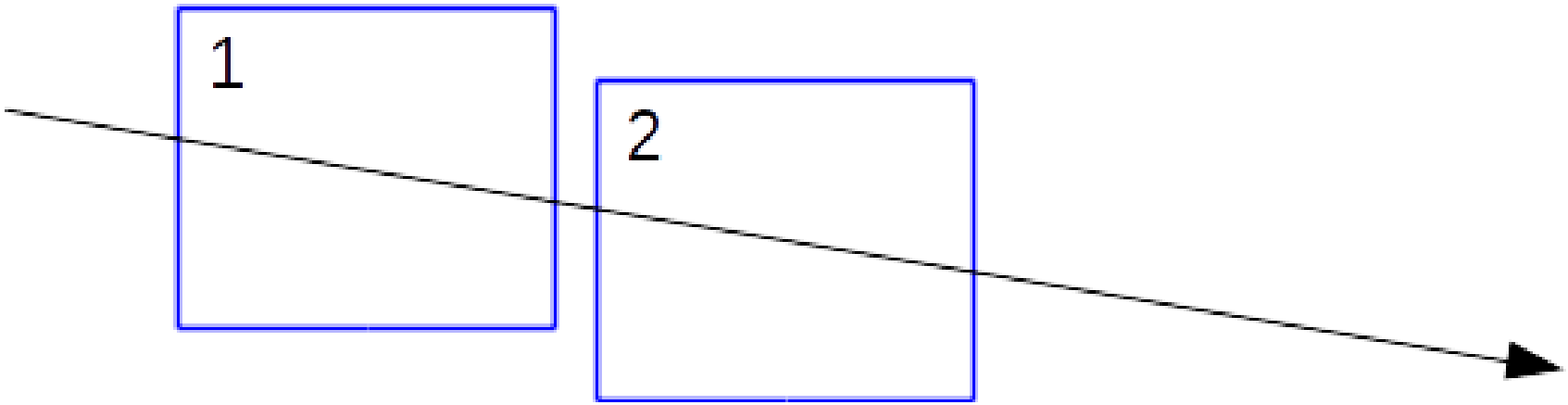
fils "proche" : $min(t_{gauche}, t_{droite})$,
fils "loin" : $max(t_{gauche}, t_{droite})$.

Rappels

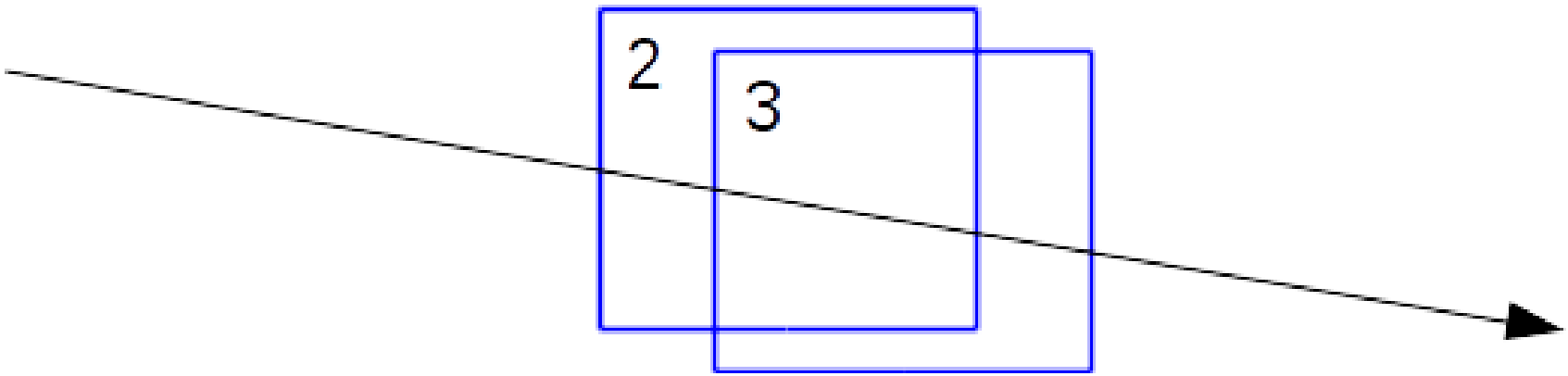
est ce que :

- cet algorithme permet de visiter les feuilles dans le bon ordre ?
- (en s'éloignant de l'origine du rayon)

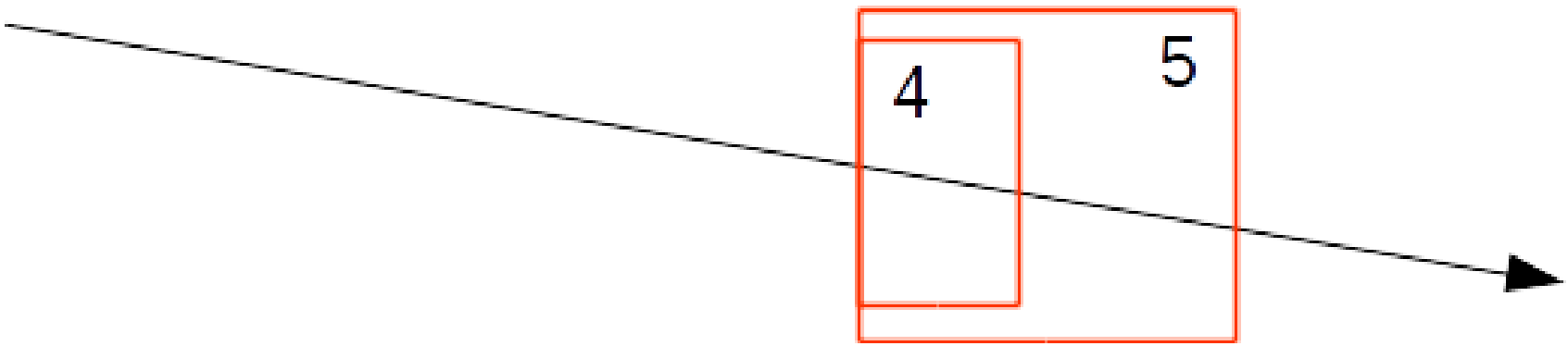
Ordre des feuilles, cas 1



Ordre des feuilles, cas 2



Ordre des feuilles, cas 3



Ordre des fils

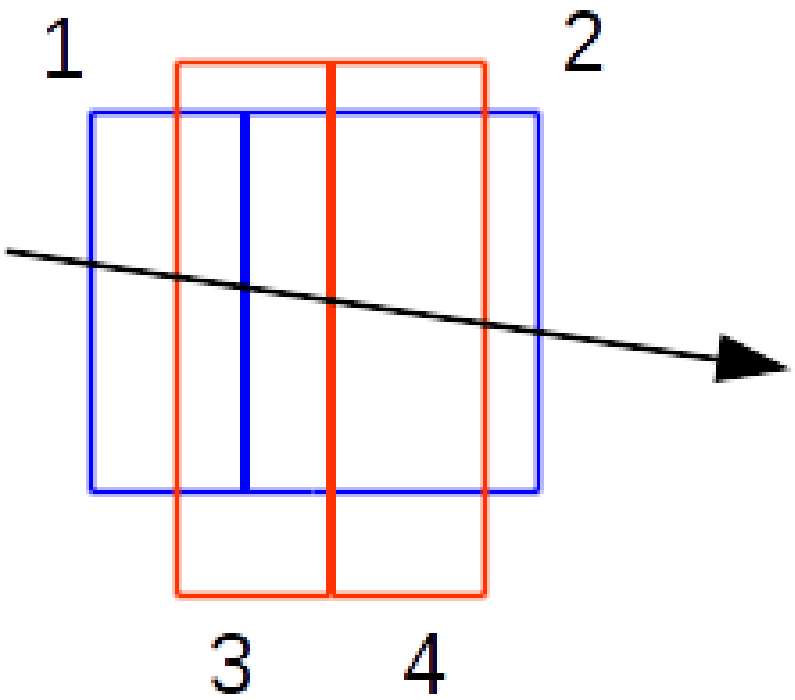
intersection rayon / boite :

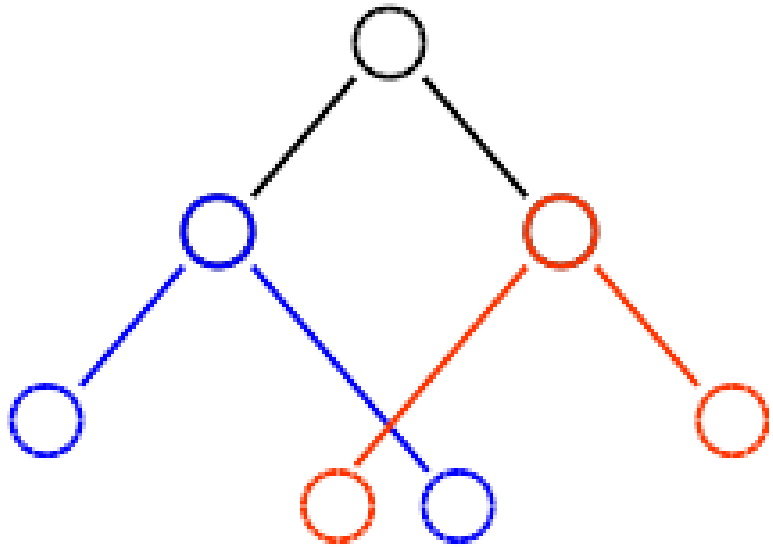
- 2 intersections, entrée et sortie du rayon : t_{min}, t_{max}

fils "proche" / "loin" :

- comparer les entrées, si elles sont égales, comparer les sorties :
- si $t_{min}^{gauche} < t_{min}^{droit}$: fils gauche
- si $t_{min}^{gauche} == t_{min}^{droit}$ et $t_{max}^{gauche} < t_{max}^{droit}$: fils gauche
- sinon fils droit

Ordre des fils : exemple





dans quel ordre sont parcourus les (petits) fils 1-2-3-4, 1-3-2-4 ?

Ordre des fils

??

- la décision (locale) de parcours des fils,
- ne permet pas toujours de parcourir les feuilles dans le bon ordre...

quel algorithme de parcours permettrait de prendre la bonne décision ?

Ordre global des fils

quel est le prochain noeud visité ?

- un des fils du noeud courant,
- ou le prochain noeud dans l'ordre définit le long du rayon ?

c'est équivalent à l'algorithme de Dijkstra sur les graphes, le prochain noeud visité n'est pas nécessairement un voisin direct / fils...

"Ray Tracing Complex Scenes"
T.L. Kay, J.T. Kajiya, SIGGRAPH 86

Arbres / BVH : intuition

comment construire un arbre ?

- mêmes idées que pour les arbres binaires de recherche...
- mais : arbres d'intervalles
- mais : 3d

trier des objets 3d ?

- trier sur X, partitionner sur X == 2 sous ensembles,
- puis, trier sur Y, partitionner sur Y == 4 sous ensembles,
- puis, trier sur Z, partitionner sur Z == 8 sous ensembles.

trier des objets 3d ? ordre des englobants sur X, Y et Z...

Arbres / BVH : intuition

et ça marche ?

- oui...
- construire un noeud à chaque étape,
- simplification courante : ne trier que l'axe le plus étiré de l'englobant

Arbres / BVH : construction

```
int build_node( const int begin, const int end ) {
    Point bmin, bmax;
    bounds(begin, end, bmin, bmax); // englobant des triangles

    if(end - begin < 2) { // 1 triangle, construire une feuille
        nodes.push_back( {bmin, bmax, -begin, -end} );
        return int(nodes.size()) -1;
    }

    // axe le plus etire de l'englobant
    int axis= bounds_max(bmin, bmax);

    // trier les triangles sur l'axe
    std::sort(triangles.data() +begin, triangles.data() +end,
         triangle_less(axis));

    // repartir les triangles et construire les fils du noeud
    int left= build_node(begin, (begin+end) / 2);
    int right= build_node((begin+end) / 2, end);

    // construire le noeud
    nodes.push_back( {bmin, bmax, left, right} );
    return int(nodes.size()) -1;
}
// int root= build_node( 0, int(triangles.size() );
```

Arbres / BVH : construction

trier des objets 3d ?

- plusieurs solutions
- solution 1 : trier les objets sur un axe,
  (construit un arbre équilibré)
- solution 2 : couper l'englobant en 2 parties,
  déterminer quels triangles se trouvent dans la partie 1,
  idem pour la partie 2
- et pour les triangles à cheval ?
  (dépend de la relation d'ordre utilisée pour le tri)

plusieurs constructions : comment les comparer ?

Arbres / BVH : construction

```
int build_node_centroids( const int begin, const int end ) {
    Point bmin, bmax;
    bounds(begin, end, bmin, bmax); // englobant

    Point cmin, cmax;
    centroid_bounds(begin, end, cmin, cmax); // englobant des centres

    if(end - begin < 2) { // 1 triangle, construire une feuille
        nodes.push_back( {bmin, bmax, -begin, -end} );
        return int(nodes.size()) -1;
    }

    int axis= bounds_max(cmin, cmax);

    int m= std::distance(triangles.data(), // repartir les triangles
        std::partition(triangles.data() +begin, triangles.data() +end,
            centroid_less(axis, (cmax(axis) + cmin(axis)) / 2));

    int left= build_node(begin, m); // construire les fils du noeud
    int right= build_node(m, end);

    // construire le noeud
    nodes.push_back( {bmin, bmax, left, right} );
    return int(nodes.size()) -1;
}
// int root= build_node_centroids(0, int(triangles.size()));
```

Exemples

Exemples

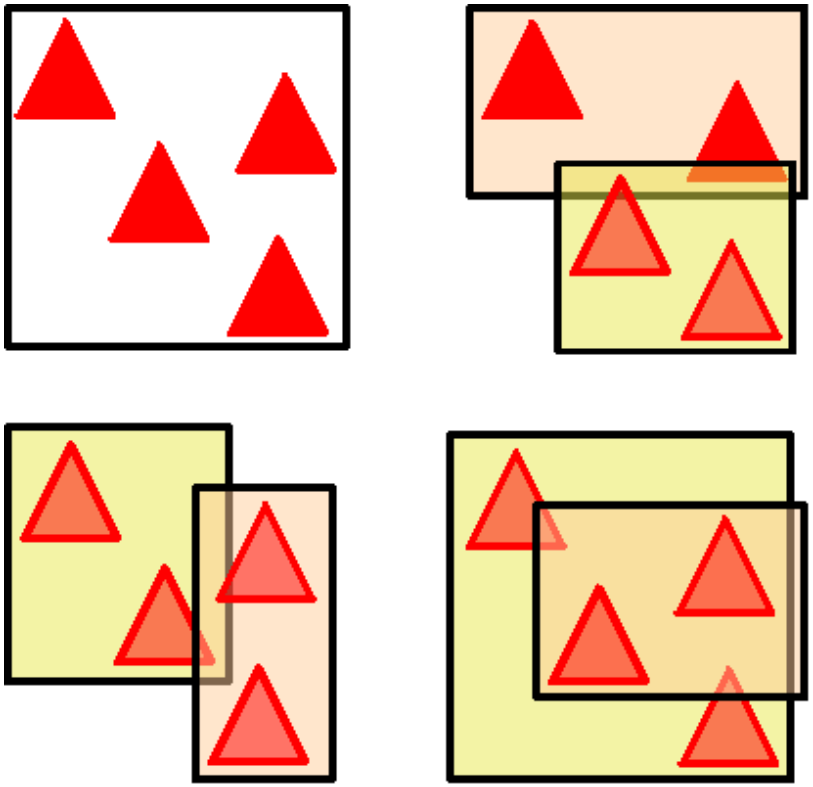
museumHall.obj

- solution 1 : construction 4s, rendu 2s500
- solution 2 : construction 0s700, rendu 1s000

plusieurs constructions : comment les comparer ?

solution xx : construction 8s, rendu 0s800

Exemples



Comparer les constructions

solution 1 : construit un arbre équilibré

- solution 2 construit un arbre plus efficace à parcourir... (et déséquilibré...)
- pourquoi ?

visite d'un noeud :

- noeud interne : tester l'englobant,
- feuille : tester les triangles.

combien de rayons visitent chaque noeud / feuille ?

Comparer les constructions

géométrie et probabilités :

- N rayons passent dans un englobant (convexe) A :
- combien de rayons passent dans $B \in A$?
- $\beta = \frac{Aire(B)}{Aire(A)}$
- pour N rayons, il y en a βN qui passent par $B \in A$...

"Some Integral Geometry Tools to Estimate the Complexity of 3D Scenes"
F. Cazals, M. Sbert, 1997

Comparer les constructions

séparer les triangles en 2 sous-ensembles :

- B englobant de T triangles,
- après répartition des triangles :
- B_1 englobe T_1 triangles,
- B_2 englobe les autres triangles, T_2...
- si N rayons visitent B :
- $\beta_1 N$ rayons visitent B_1 et testent T_1 triangles,
- $\beta_2 N$ rayons visitent B_2 et testent T_2 triangles.

Comparer les constructions

au final :

- $\beta_1 T_1 + \beta_2 T_2$ tests de triangles,
- $+1$ test d'englobant,
- il suffit de comparer ces valeurs pour plusieurs répartitions,
- et de garder la meilleure...

- solution 1 : $T_1 = N/2$ et $T_2 = N/2$, B_1, B_2 ?
- solution 2 : $B_1 \approx B/2$ et $B_2 \approx B/2$, T_1, T_2 ?

et pour T triangles, combien de répartitions possibles ?

Construction exhaustive en $O(T^2)$

```
int build_node_sah( const int begin, const int end ) {
    { ... } // trier sur l'axe le plus etire de l'englobant
    // tester les repartitions
    // 1|n-1, 2|n-2, ... n-1|1
    for(int i= begin +1; i < end; i++) {
        Point left_min, left_max;        // a gauche de la coupe
        bounds(begin, i, left_min, left_max);
        int left_n= i - begin;

        Point right_min, right_max;      // a droite
        bounds(i, end, right_min, right_max);
        int right_n= end - i;

        float cost= 1   // cout de la repartition
            + bounds_area(left_min, left_max) / area * left_n
            + bounds_area(right_min, right_max) / area * right_n;
        if(cost < min_cost) {   // garder la meilleure repartition
            min_cost= cost; min_index= i;
        }
    }

    // repartir les triangles entre les fils du noeud
    int left= build_node_sah(begin, min_index);
    int right= build_node_sah(min_index, end);
    { ... } // construire le noeud
}
// O(n^2), pas vraiment utilisable...
```

exemples

museumHall.obj

- solution 1 : construction 4s, rendu 2s500
- solution 2 : construction 0s700, rendu 1s000
- solution 3 : construction 8s, rendu 0s800

quelle est la meilleure solution ? temps total ? temps de rendu ? pour combien de rayons ?

comment construire l'arbre optimal ?

Modèle de coût

à lire sur le sujet :

- "Efficient Divide-And-Conquer Ray Tracing using Ray Sampling"
  K. Nabata, K. Iwasaki, Y. Dobashi, T. Nishita, 2013
- "Naive Ray-Tracing: A Divide-And-Conquer Approach"
  B. Mora, 2011
- formalisation correcte du problème :
  "Cost prediction for ray shooting in octrees"
  B. Aronov, H. Brönnimann, A.Y. Chang, Y.J. Chiang, 2005

Construction guidée par un modèle de cout

comment ?

rappel : fraction des rayons passant par un noeud qui passent aussi par un fils :

$$P(fils|noeud) = \beta = \frac{A(fils)}{A(noeud)}$$

estimations :

- nombre de noeuds internes intersectés : $\sum P(noeud_i|racine)$,
- nombre de feuilles intersectées : $\sum P(feuille_i|racine)$,
- nombre d'objets intersectés : $\sum P(feuille_i|racine) \times N_i$

Construction guidée par un modèle de cout

pour tout l'arbre :

$$C = \sum P(noeud_i|racine)C_{englobant} + \sum P(feuille_i|racine) \times T_i \times C_{triangle}$$

remarque : plus il y a de noeuds, plus le cout augmente...

trouver l'arbre qui minimise C !
trop de solutions, utiliser un algo glouton,
cf optimisation / minimisation...

"Heuristics for ray tracing using space subdivison"
J. D. MacDonald, K.S. Booth, 1990

et ça marche ?

cout estimé / rendu:

- solution 1 : construction 4s, rendu 2s900, cout 233,
- solution 2 : construction 0s600, rendu 1s400, cout 117
- solution 3 : constuction 6s800, rendu 0s900, cout 97,
- solution x : construction 0s500, rendu 1s500, cout 123,
- solution y : construction 2s900, rendu 0s700, cout 63,
- solution z : construction 0s600, rendu 2s200, cout 86

solution z ??

et ça marche ?

pourquoi ?

- le modèle de cout est correct pour une partition spatiale, pas pour un BVH...
- et pour une distribution uniforme de rayons, pas pour des rayons générés par la camera,
- "On Quality Metrics of Bounding Volume Hierarchies"
  T. Aila, T. Karras, S. Laine, HPG 2013
- rappel : dérivation du modèle
  "Cost prediction for ray shooting in octrees"
  B. Aronov, H. Brönnimann, A.Y. Chang, Y.J. Chiang, 2005

BVH : les détails

cas simple : 2 fils, volumes englobants : cubes alignés sur les axes.

construction :

en fonction du volume occupé par les fils, et du temps de calcul de l'intersection du rayon avec les objets associés aux fils.

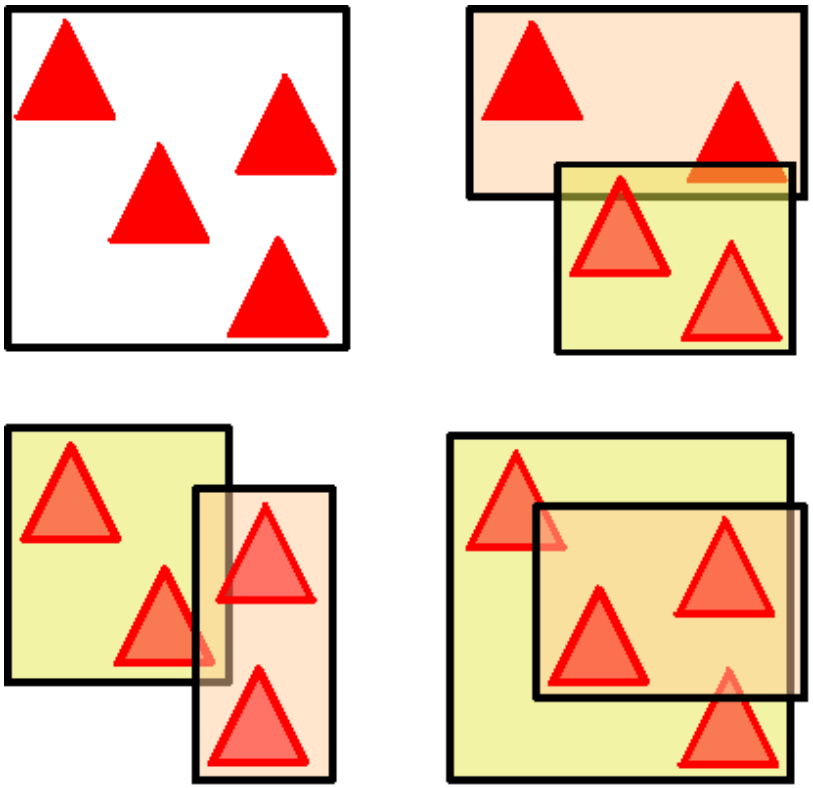
parcours ordonné :

pas obligatoire, mais beaucoup plus efficace (au moins pour les rayons primaires)

Construction

trouver la meilleure répartition des objets pour chaque noeud :

choisir un plan candidat et répartir les objets.



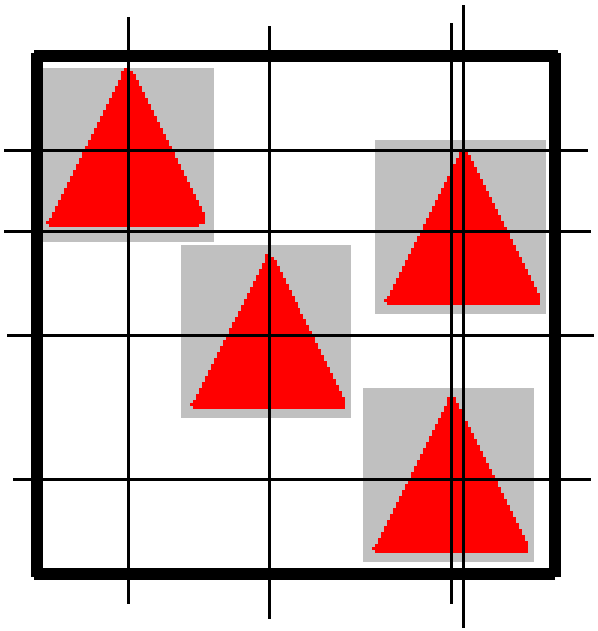
Construction SAH d'un BVH

minimiser C : trouver le meilleur candidat

$$
\begin{aligned}
C &= C_{fils\_gauche} + C_{fils\_droit} \\
C_{fils} &= C_{englobant} + \frac{A(fils)}{A(noeud)} T(fils) \times C_{triangle}
\end{aligned}
$$

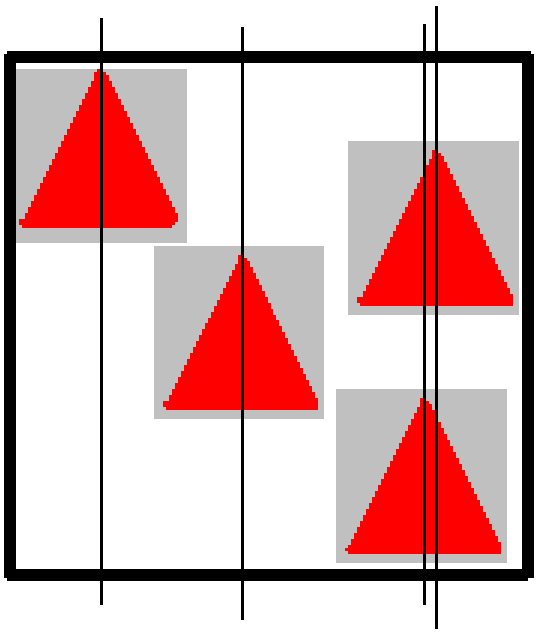
Construction : algorithme

- travaille sur les englobants de chaque objet,
- les candidats sont les 3 plans passants par le centre des englobants.



Construction : algorithme

- travaille sur un axe à la fois,
- teste tous les centres,
- évalue C à chaque fois et garde le meilleur (sur les 3 axes).



Construction : algorithme

- construction récursive :
- critère d'arret ? lorsqu'il n'est plus interessant de continuer :
- $C > T \times C_{triangle}$
  (subdiviser est plus couteux que de tester tous les triangles)

Evaluer C : algorithme

$$C_{fils} = C_{englobant} + \frac{A(fils)}{A(noeud)} T(fils) \times C_{triangle}$$

déterminer les 2 sous ensembles d'objets + cubes englobants :

- naif : re-trier à chaque fois, cf solution 3
  (trop long, $O(n \log n)$ par noeud)
- incrémental : trier une seule fois par axe, puis exploiter l'ordre pour contruire les cubes englobants
  (correct, $O(n)$ par noeud),
- il est facile de calculer le min et le max d'un ensemble lorsqu'on insère un élement,
- mais pas le contraire (lorsqu'un supprime un élément) ?

Evaluer C : algorithme

- parcourir de min vers max et construire la partie gauche et son cube englobant :
- $T(fils_{gauche})$, $A(fils_{gauche})$
- parcourir de max vers min pour la partie droite :
- $T(fils_{droit})$, $A(fils_{droit})$
- tous les termes de C sont connus pour tous les candidats,
- finir l'évaluation de C pour chaque candidat,
- garder le meilleur, répartir les objets en 2 sous-ensembles,
- recommencer

BVH : les détails

construction, parcours et mise à jour :
"Ray tracing deformable scenes using dynamic bounding volume hierarchies"
I. Wald, S. Boulos, P. Shirley, 2007

version approchée et efficace :
"On fast construction of SAH based bounding volume hierarchies"
I. Wald, 2007

construction guidée par la "vraie" distribution de rayons :
"Efficient Divide-And-Conquer Ray Tracing using Ray Sampling"
K. Nabata, K. Iwasaki, Y. Dobashi, T. Nishita, 2013

Encore plus vite ?

quel autre type d'arbre permet de gagner facilement du temps lors du parcours ?

idée :

- quelle est la hauteur d'un arbre binaire ?
- quelle est la hauteur d'un arbre dont les noeuds internes ont k fils ?
- combien de noeuds internes ? est ce que le cout de l'arbre dépend du nombre de noeuds internes ?

"Shallow bounding volume hierarchies for fast SIMD ray tracing of incoherent Rays"
H. Dammertz, J. Hanika, A. Keller, 2008

Encore plus vite ?

avec une carte graphique :

- "Understanding the efficiency of ray traversal on GPUs"
  T. Aila, S. Laine, 2009
- "Architecture considerations for tracing incoherent rays"
  T. Aila, T. Karras. 2010

Encore plus vite ?

avec une hiérarchie de meilleure qualité :

- découper les triangles : cf solution y
  "Spatial Splits in Bounding Volume Hierarchies"
  M. Stich, H. Friedrich, A. Dietrich, 2009

et alors ?

quelle est la meilleure solution ?

- dépend de l'utilisation : du nombre de rayons,
- et du temps total : construction + rendu...
- les constructions longues ne sont interressantes que pour des rendus longs...
- construction parallèle, sur cpu ou gpu ?

cpu : solution 2 et x pour peu de rayons, solution y sinon

gpu : solution x...

"Maximizing Parallelism in the Construction of BVHs, Octrees, and k-d Trees"

T. Karras, 2012