IFT3355: Infographie
Sujet 4: Visibilité 2 (partition de l'espace)

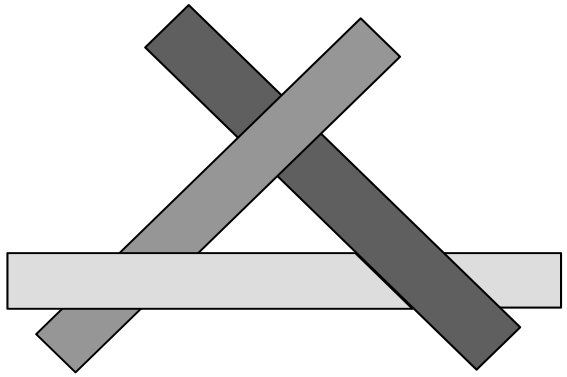
Derek Nowrouzezahrai
Département d'informatique et de recherche opérationelle
Université de Montréal

Université de Montréal

Partition de l'espace

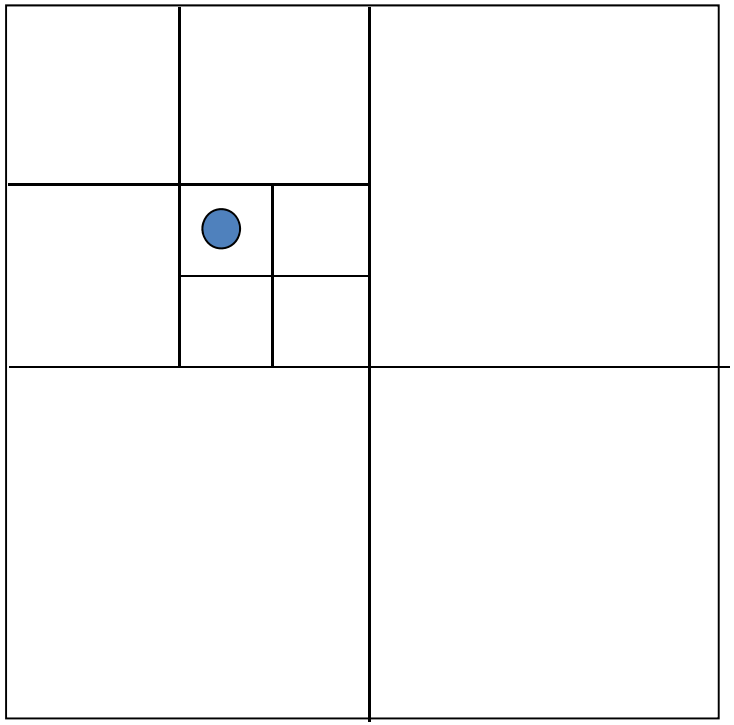
- Dans l'algorithme du peintre, les objets sont ordonnés en fonction de leur distance à la fenêtre, et ils sont affichés du plus éloigné au plus rapproché

- Dans certains cas, cet ordre est impossible sans découpage au préalable



Quadtree

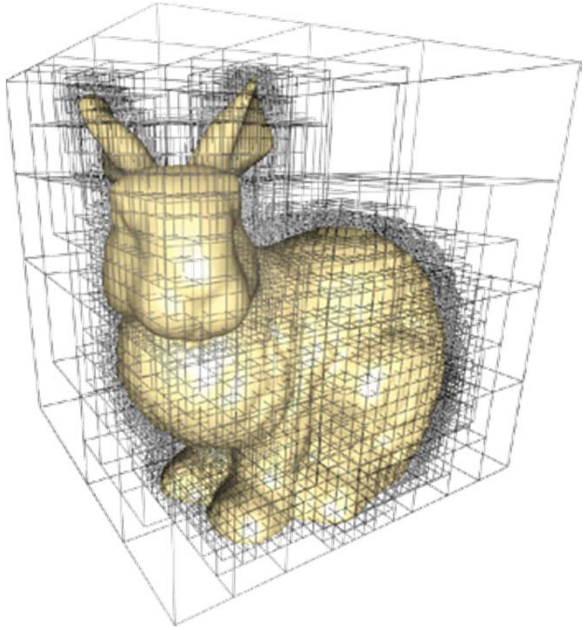
- Pour diviser l'espace 2D hiéarchiquement
  - Divise chaque region carré (ou rectangulaire) en 4 region 'égale' récursivement

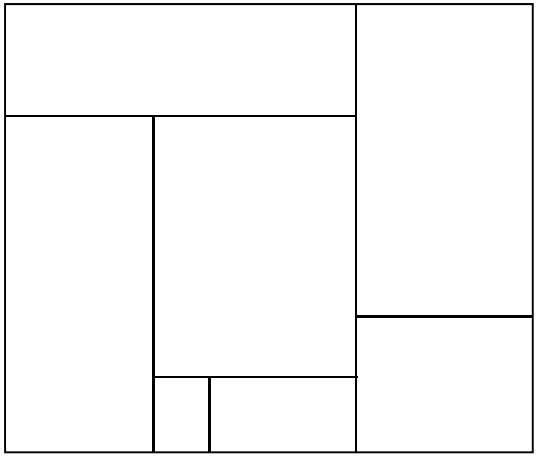


Université de Montréal

Octree

- Cette approche se généralise à l'*octree* en 3D





- En alternant selon *X, Y, Z* et glissant le plan de subdivision, on obtient un *kd-tree (ou selon X et Y en 2D)*

Université de Montréal

Partition binaire de l'espace *(BSP tree)*

- L'*octree* se généralise à son tour à la partition binaire de l'espace en se servant des plans de support des polygones comme éléments de séparation

+ Très utile lorsque la scène est statique mais que le point de vue peut être n'importe où dans la scène

Université de Montréal

Création d'une partition binaire

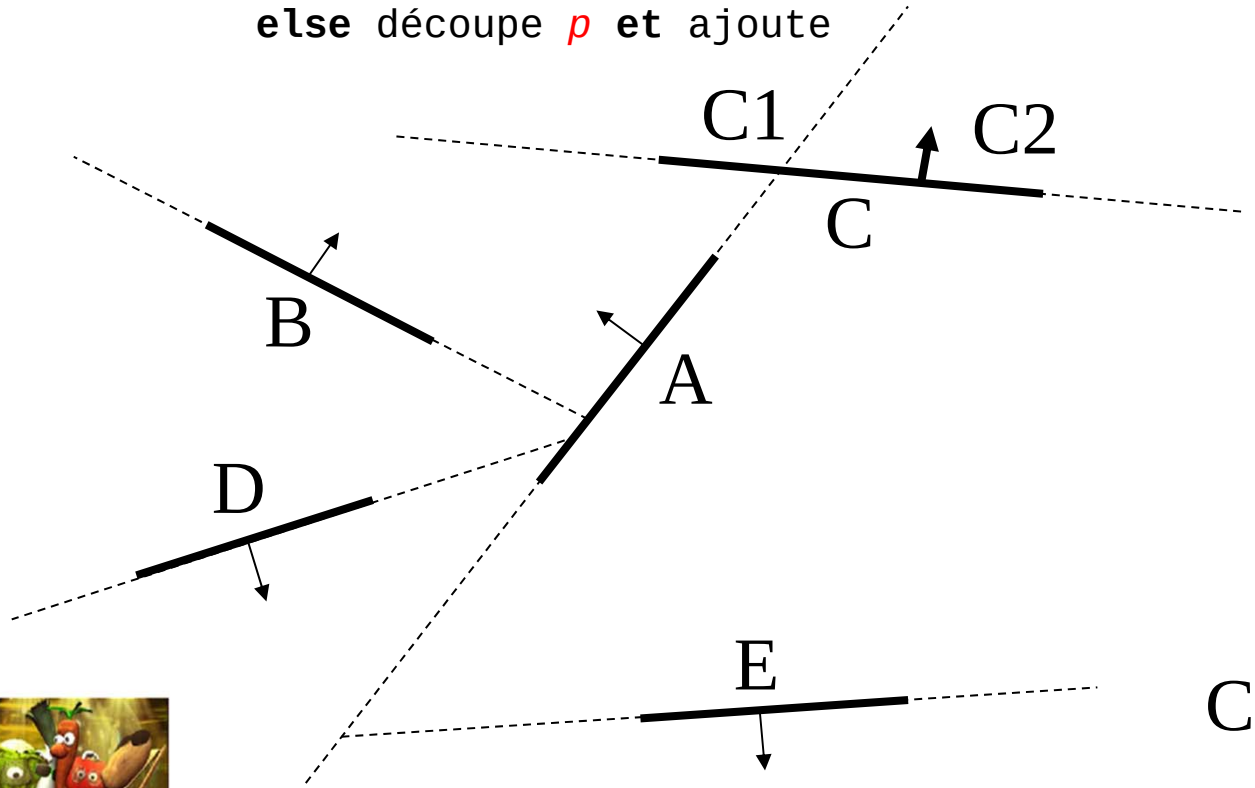
```
BSPTree * BSPMakeTree( liste de polygones )
   choisit un polygone comme racine
   for( tous les autres polygones p )
      if( p est devant )
        ajoute p à la liste de devant
      else if( p est derrière )
        ajoute p à la liste de derrière
      else
        découpe p en deux par rapport au plan de la racine
        ajoute chaque moitié à sa liste respective

   BSPTree *node = new BSPTree()
   node->element = racine
   node->devant = BSPMakeTree( liste de devant )
   node->derrière = BSPMakeTree( liste de derrière )
   return node
```

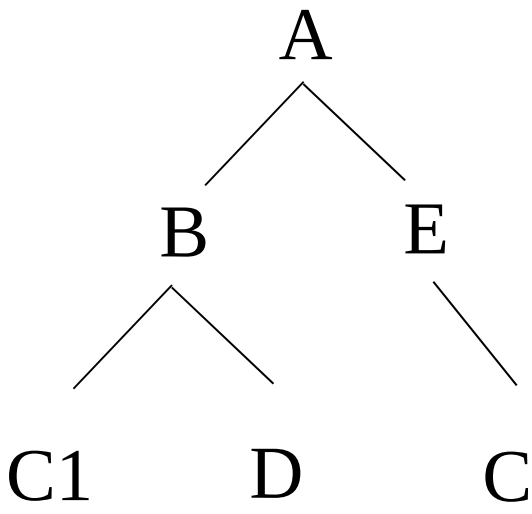
Université de Montréal

Exemple de construction d'arbre *BSP*

```
BSPTree * BSPMakeTree( liste )
    choisit racine
    for( tous autres polygones p )
        if( p est devant )
          ajoute p à devant
        elif( p est derrière )
          ajoute p à derrière
        else découpe p et ajoute
```

```
BSPTree *n = new BSPTree()
n->element = racine
n->devant = BSPMakeTree(devant )
n->derrière = BSPMakeTree(derrière )
return n
```





Université de Montréal

Ordonnancement dans un arbre *BSP*

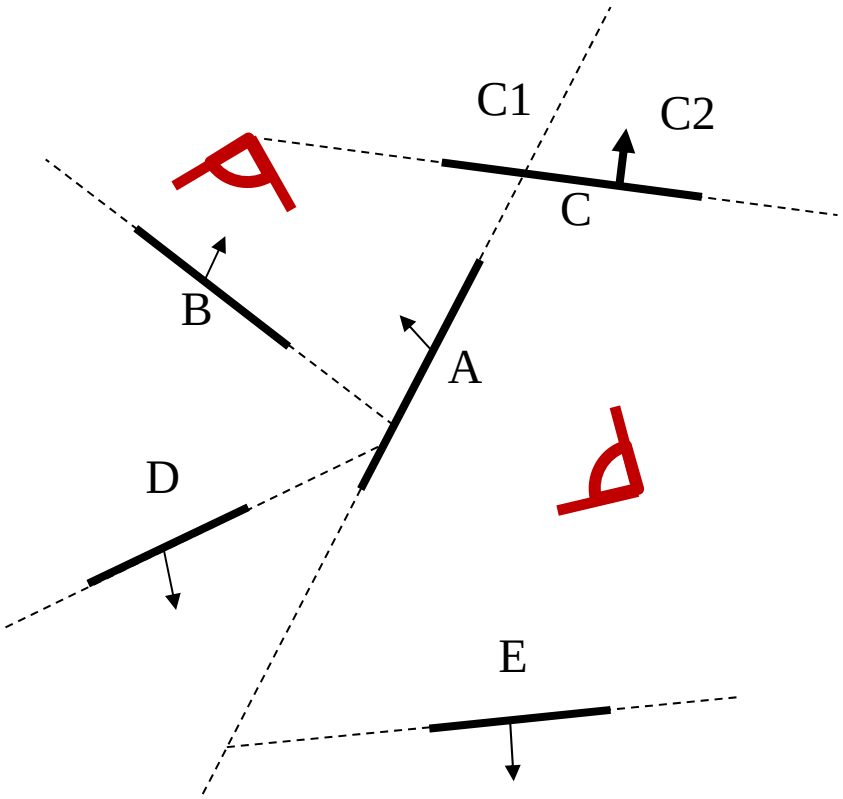
- Si l'oeil est d'un côté d'un plan, les objets de l'autre côté ne pourront jamais être devant un objet du même côté du plan que l'oeil
- On peut donc traverser l'arbre de derrière à devant pour tout point de vue en affichant simplement avec un algorithme du peintre

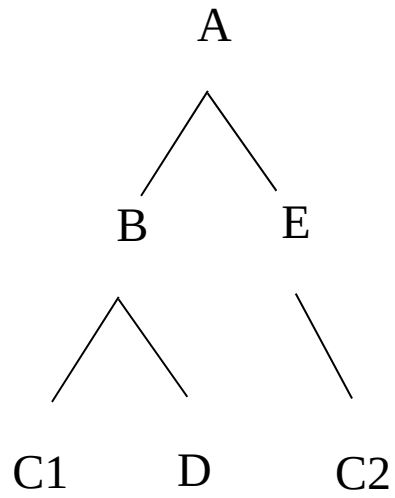
Algorithme d'affichage d'un arbre *BSP*

```
AfficheArbre(sousArbre)
   if(  l'oeil est devant la racine  )
      AfficheArbre(sousArbre->arrière)
      AfficheArbre(sousArbre->racine)
      AfficheArbre(sousArbre->devant)
   else
      AfficheArbre(sousArbre->devant)
      AfficheArbre(sousArbre->racine)    // si aucun
      AfficheArbre(sousArbre->arrière)   backface culling
```

Université de Montréal

Algorithme d'affichage d'un arbre *BSP*





Université de Montréal

Avantages et inconvénients du *BSP*

+ Les plans de la pyramide de vue peuvent être utilisés pour le clippage avec *BSP*
+ Si l'illumination (*shading*) est coûteuse, on peut aussi afficher de devant à derrière en n'affichant un pixel que s'il n'a pas déjà été affiché
+ On peut aussi traiter les ombres de sources de lumière ponctuelles
- Obtenir un arbre *BSP* avec le moins de polygones découpés est une tâche complexe
- Problèmes de robustesse en créant des polygones allongés et étroits