

# Moteur de rendu 3D Graphique : Collision et Optimisation

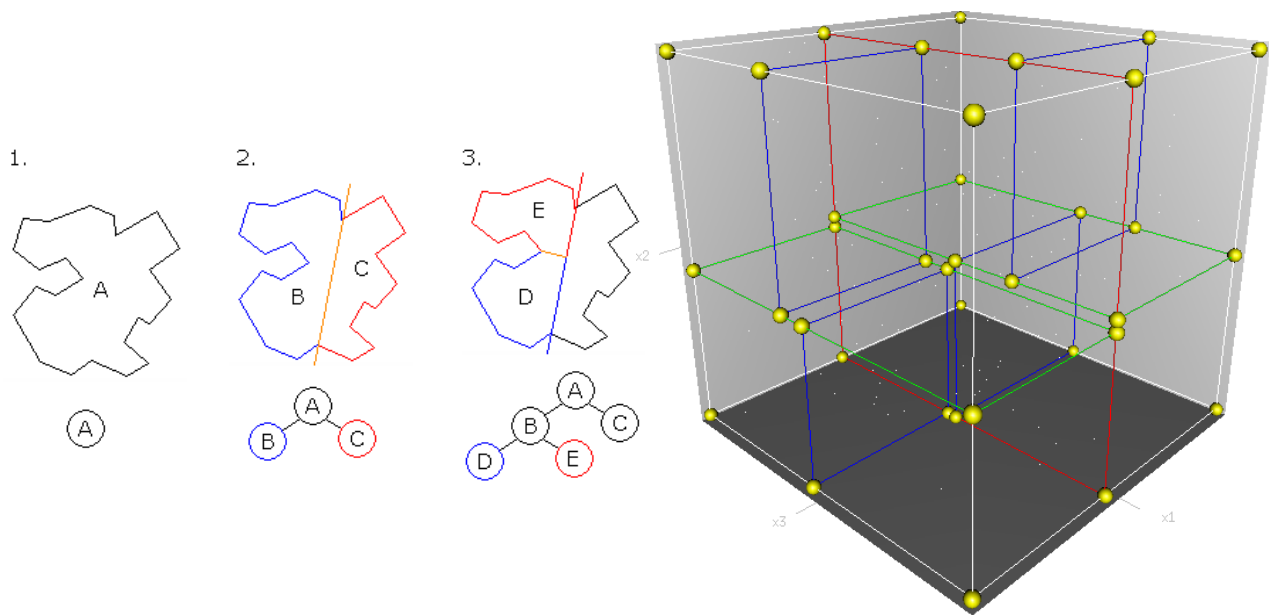


Figure 1: Exemple de gestion de partitionnement d'une zone par \*BSP.  
À gauche: Schéma technique. À droite: Rendue de partitionnement 3d.

## Contexte et but du projet

Le but de ce projet est d'implémenter un système de collision, premièrement basé sur la méthode de partitionnement de l'espace (\*BSP) utilisée par de nombreux moteurs 3D de l'époque. Ensuite, je souhaite ajouter d'autres algorithmes pour renforcer les performances et augmenter l'optimisation de la première méthode et du programme.

Mon objectif final est de pouvoir proposer une gestion de collision avec n'importe quel objet importé, tout en le rendant optimisé afin de conserver des déplacements fluides en temps réel.

## Porteurs du projet

Actuellement en troisième année, je serais le seul étudiant sur ce projet. Je suis intéressé par le domaine de l'infographie et l'algorithmie en général. Je souhaite donc en apprendre plus sur les techniques utilisées dans ces domaines.

## Environnement technique / technologique

Mon projet se basera sur le travail déjà effectué lors du projet "Moteur de rendu 3D Graphique : Rastérisation", permettant un test rapide du développement du système de collision par la fenêtre de rendu ainsi que les inputs clavier/souris. Ainsi l'implémentation restera en C.

## Description du livrable

Le projet sera disponible dans sa nouvelle version sur Linux uniquement.  
La CSFML est un prérequis pour la compilation (via Makefile) ainsi que le lancement du programme. De plus, le projet sera toujours open-source et est toujours disponible sur la plateforme Github à ce lien: <https://github.com/MasterLaplace/Engine-3D>.  
Des fichier .obj d'exemple seront mis à disposition dans le dossier assets/obj\_exemples avec les textures associés dans le dossier assets/textures.

## Organisation et temporalité

J'ai divisé mon projet en 7 parties distinctes:

Nom de la tâche	Commentaire	Estimation  Chaque couleur = 25% du projet
Recherche, documentation et ajout dans le pipeline		2 j/h
Implémentation de la méthode de Partition binaire de l'espace	pour créer un arbre pour chaque mesh et y partitionner les triangles	4 j/h
Recherche et Documentation		1 j/h
Conversion du binary tree en Octree	pour améliorer la vitesse de traitement des collisions, soit le temps de recherche dans l'arbre	3 j/h
Recherche, documentation et mise à jour du pipeline		2 j/h
Mise en place d'un World Partitionning au niveau des meshes	pour charger les meshes uniquement dans le chunk où se situe la camera	5 j/h
Mise en place d'un R-Tree par chunk	pour organiser les meshes au sein du chunk et permettre une amélioration dans la sélection des meshes à surveiller	3 j/h
		Total: 20 J/H