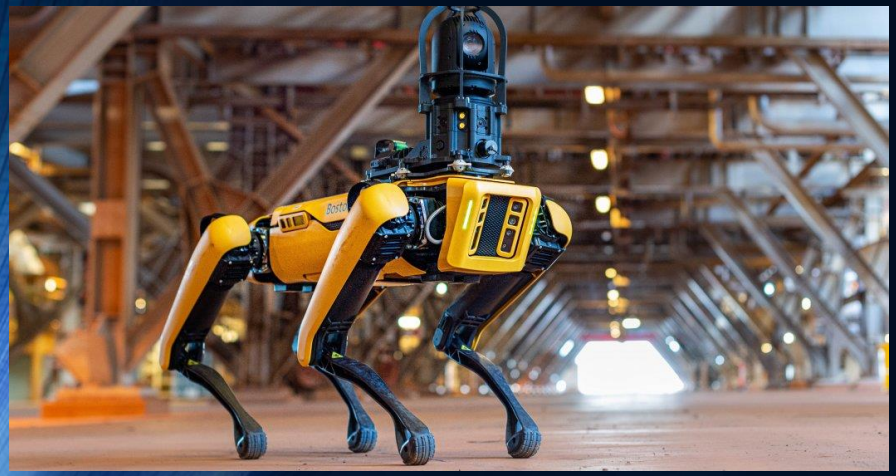
Modélisation d'un Robot capable de cartographier son environnement à l'aide d'un simulateur informatique

- INTRODUCTION DU SIMULATEUR ET IMPLÉMENTATIONS
- MÉTHODES DE CARTOGRAPHIES ÉTUDIÉES
- APPLICATIONS CONCRÈTES

JOLY Julien – N°18311 – TIPE 2021

Présentation du sujet :

- A quel point est-ce possible d'utiliser un simulateur pour modéliser un environnement correspondant à une situation précise ?
- Peut-on utiliser un tel simulateur pour faire créer à un robot une cartographie de son environnement.



bostondynamics.com

shark-robotics.com

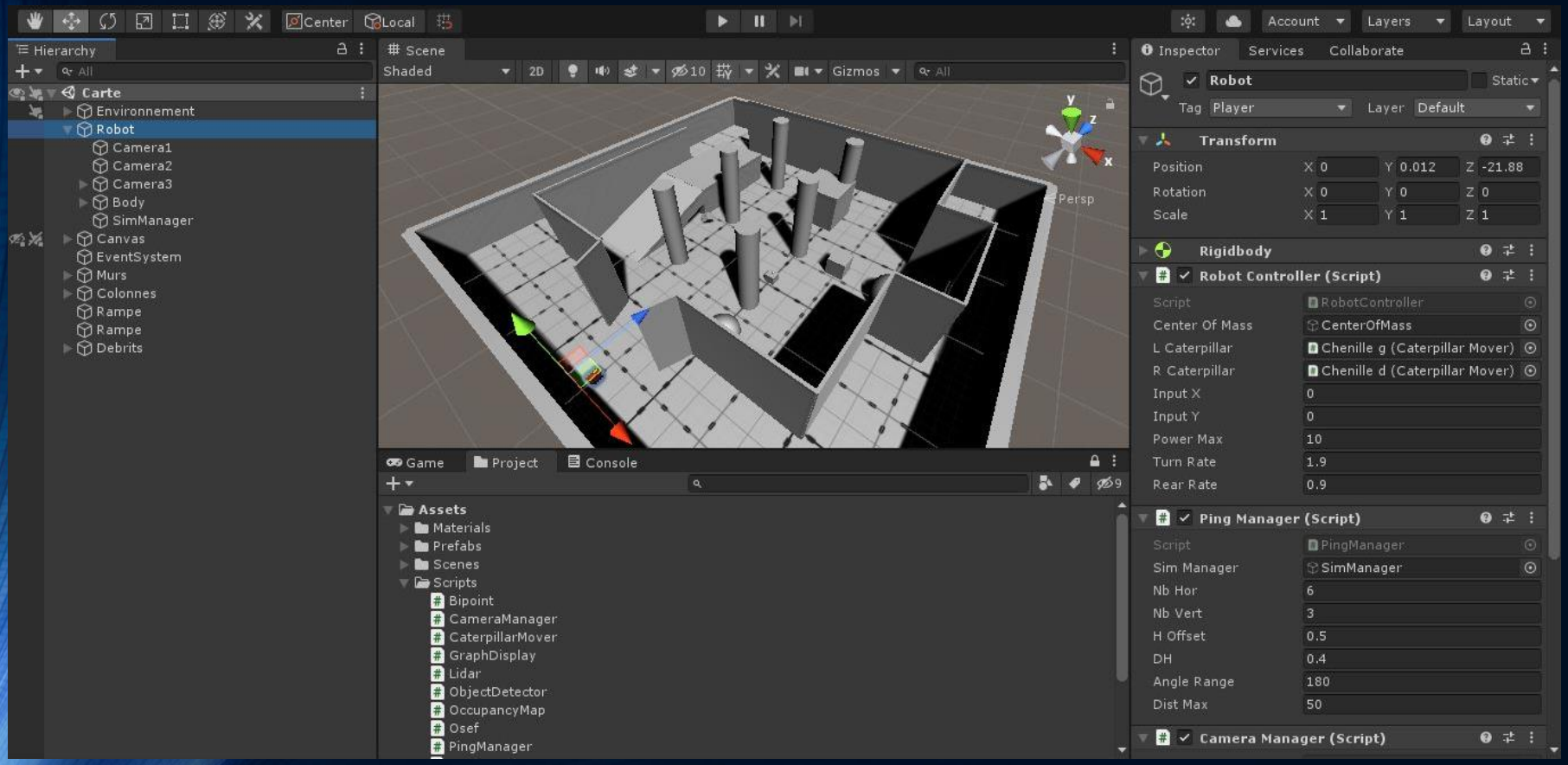
Introduction au simulateur

- PRÉSENTATION
- CRÉATION DU ROBOT
- CRÉATION DES DIFFÉRENTES CARTES
- PRÉSENTATION DU MODULE LIDAR

Moteur physique et graphique Unity

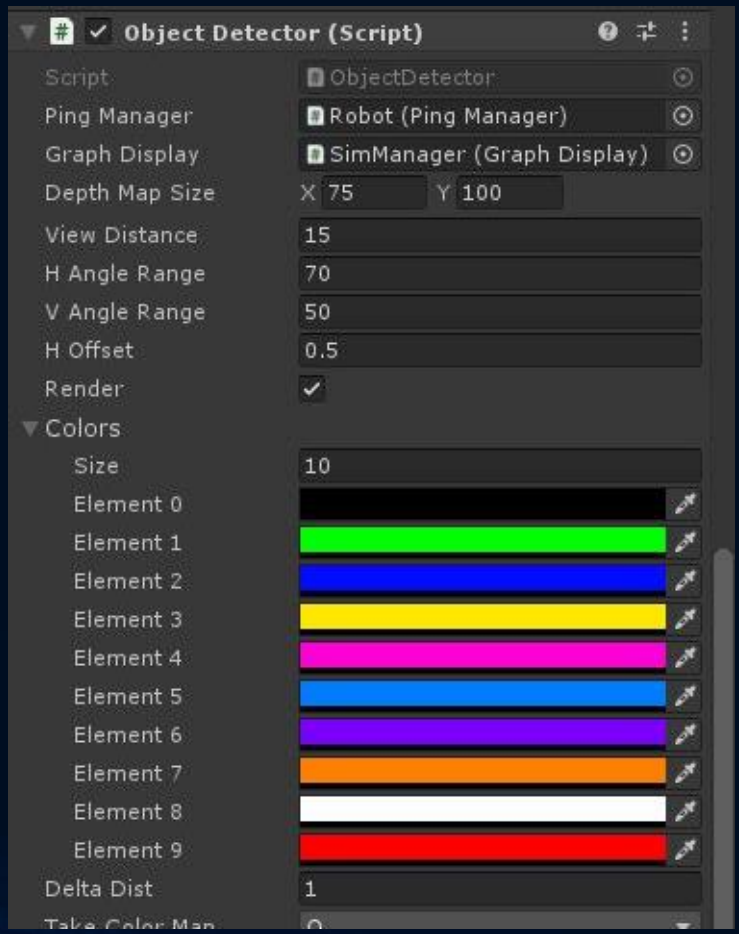
- Développé par une entreprise privée qui met à disposition son moteur et ses scripts
- Moteur 3D : possibilité de créer des objets dans un espace 3D
- Moteur physique : possibilité d'ajouter à ces objets des collisions (Collider) et des calculs de physiques (Rigidbody)
- Le développement se fait en C#

Présentation de l'éditeur



Capture d'écran

Explication du fonctionnement de l'inspecteur



Captures d'écran

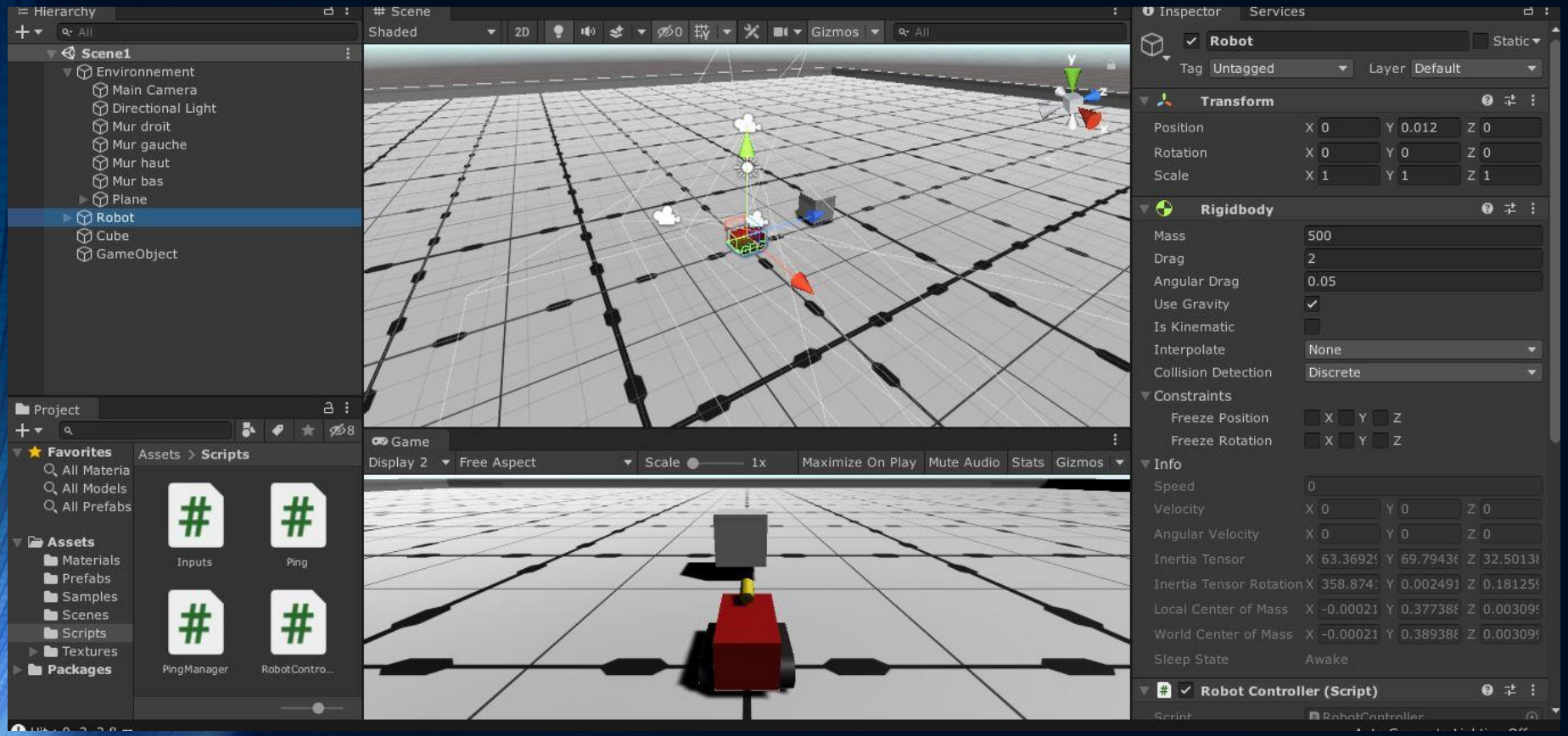
Présentation des robots de références :

- Colossus de Shark Robotics
- TC800-FF de Tecdron
- Le robot développé dans le simulateur

	Colossus	TC800-FF	Simulateur
Masse	380kg	500kg	N/a
Dimensions (L * h * l)m	1,5 x 0,8 x 0,8	1,6 x 0,7 x 0,8	1,16 x 0.9 x 0,9
Pente max	45°	35°	25°
Vitesse	3 $km.h^{-1}$	10 $km.h^{-1}$	15$km.h^{-1}$
Charge utile	550kg	800kg	N/a
Portée	5000m	1000m	N/a

shark-robotics.com // tecdron.com

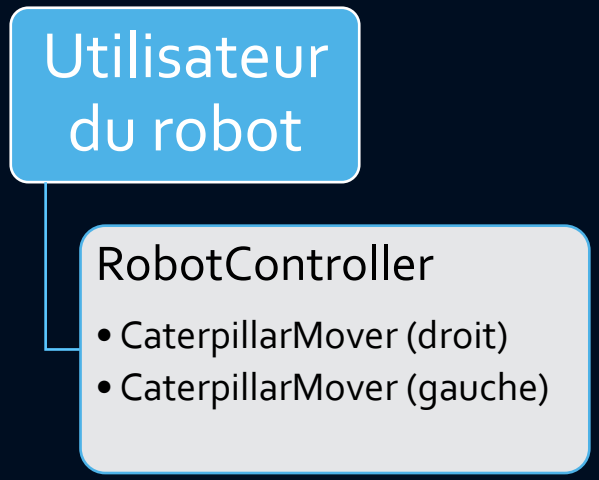
L'implémentation du robot dans l'éditeur



Capture d'écran

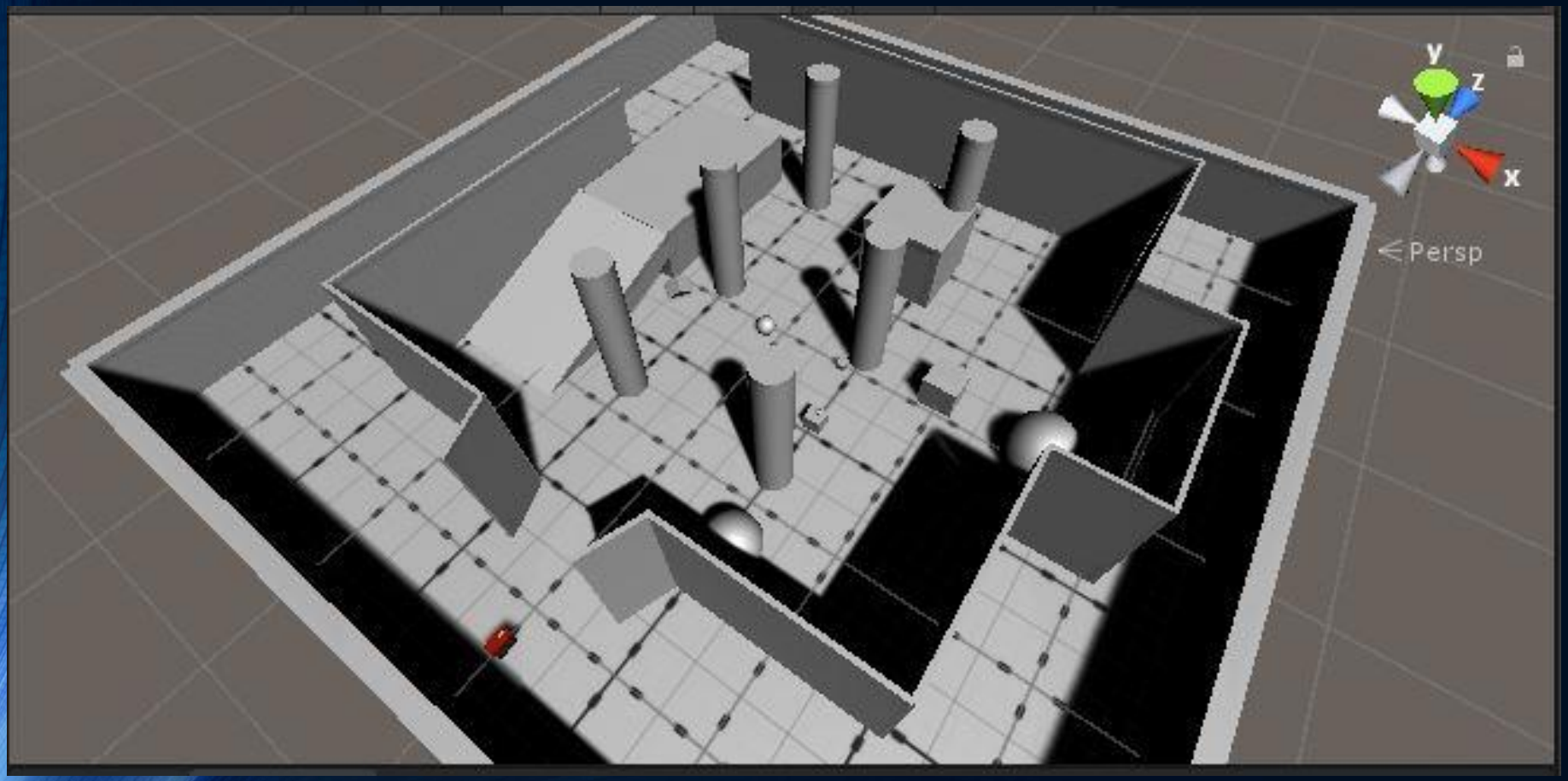
Fonctionnement des Scripts contrôlant le robot

- RobotController
  - Gère les entrées utilisateur
  - Appelle CaterpillarMover pour chaque chenille



- CaterpillarMover :
  - On calcule :
    - $wheelRotation = normal \wedge forward$
    - $power = powerMax \times (inputY + turnRate \times side \times inputX)$
      - Avec $side \in \{-1,1\}$
  - Pour chaque point de contact, on applique :
    - $tractionForceAtContact = wheelsRotation \wedge contact.normal * power$

L'implémentation de la carte dans l'éditeur



Capture d'écran

Fonctionnement et implémentation d'un Lidar

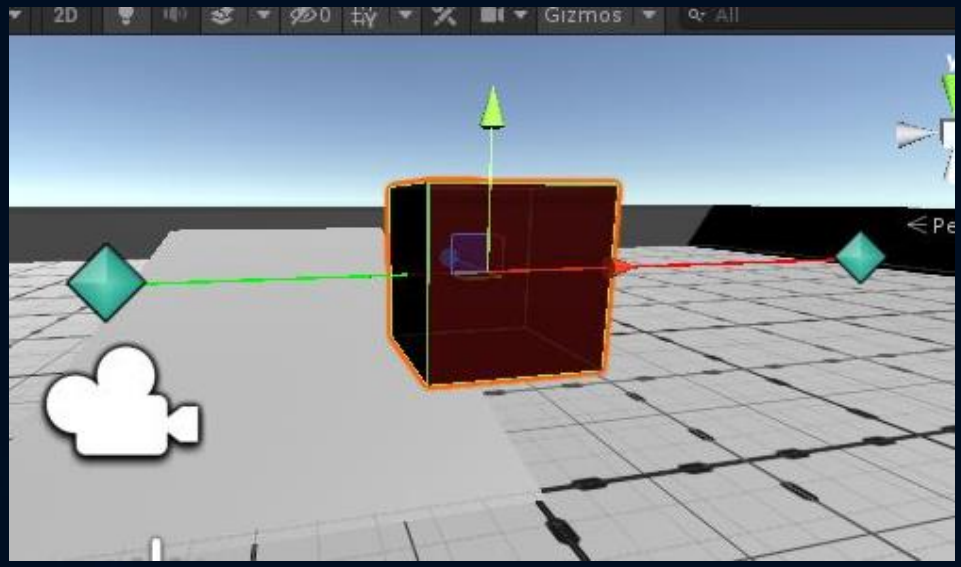
LIDAR POUR LIDAR POUR LASER IMAGING DETECTION AND RANGING

Présentation de la classe Lidar

SendRay	• A partir d'une onde, donne l'onde réfléchie sur les objets.
SendNewWaveHor	• Emet une vague d'onde en forme de prisme et stocke dans une matrice les rayons réfléchis.
SendNewWaveCone	• Emet une vague d'onde en forme de cône et stocke dans une matrice les rayons réfléchis.
EncodeDepthMap	• A partir de la matrice de rayons, construit une carte de profondeur en niveau de gris.

Fonctionnement de SendRay

- Entrées :
  - Rayon envoyé
- Utilise une fonction native pour trouver une liste de RaycastHit
- Par recherche linéaire sur la norme des vecteurs des points de contact, trouve le premier point de contact
- Retourne le rayon correspondant
- Complexité en $\Theta(n)$ avec n le nombre de points de contact

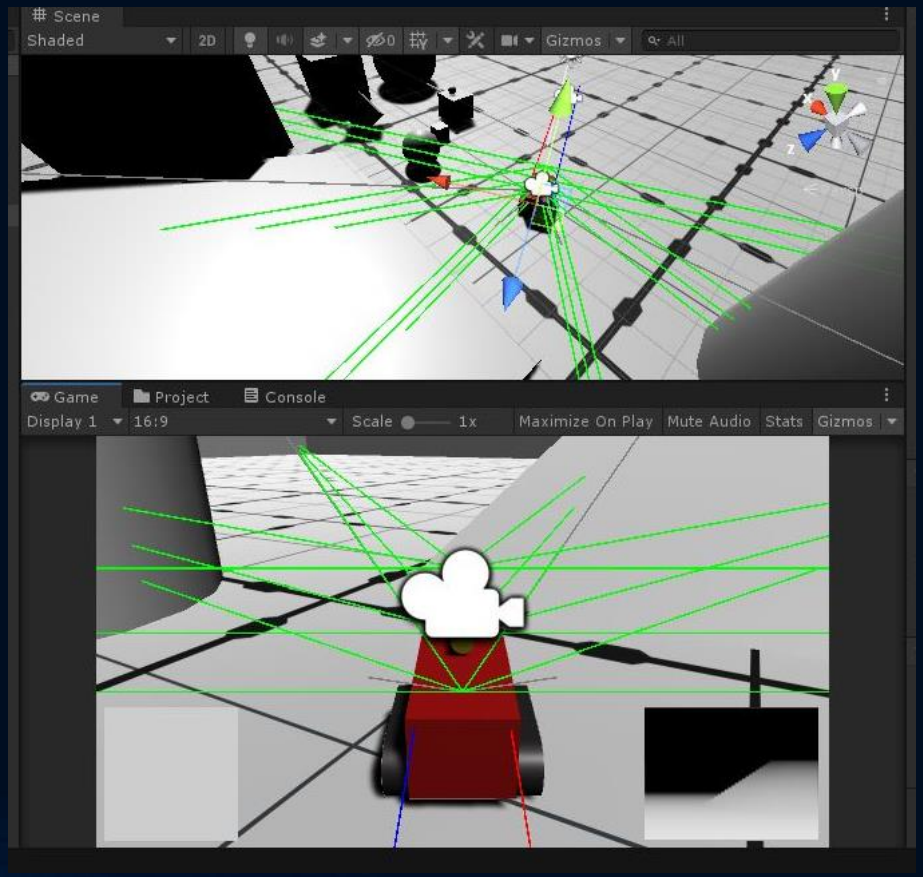


Ci-dessus le rayon vert est celui retourné, et le rouge correspond au rayon envoyé

Capture d'écran

Fonctionnements de SendNewWaveHor

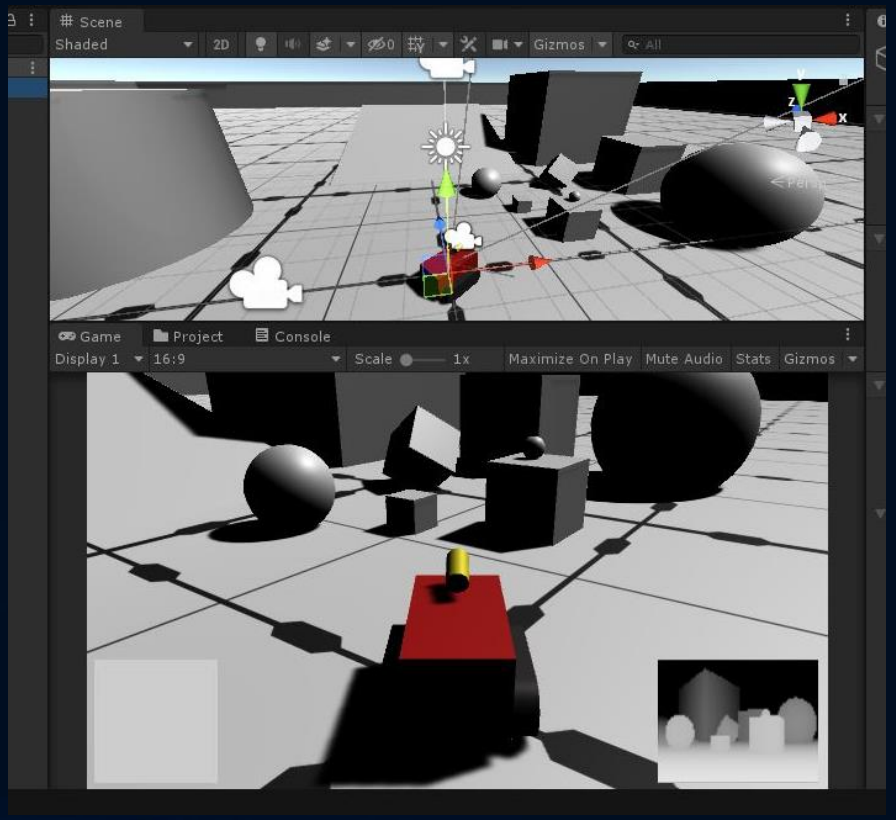
- Entrées :
  - Nombre de rayons (H*L)
  - Taille des rayons
  - Paramètres d'espacement des rayons
  - Coordonnées de l'émetteur
- Calculs des espacements :
  - $horAngle = \frac{angleRange}{L-1}$
- Pour $(i,j) \in [\![H]\!] \times [\![L]\!]$:
  - $angle = j \times horAngle$
  - $direction = (sin(angle), 0, cos(angle))$
- Ajoute le rayon reçu dans une matrice Data
- Retourne Data



Capture d'écran

Fonctionnement de CreateDepthTable :

- Entrées :
  - Matrice Data des rayons à traiter de taille H*L
- Crée une Matrice depthTable de la même taille que Data
- Pour $(i, j) \in [\![H]\!] \times [\![L]\!]$:
  - $depthTable[i, j] = \|Data[i, j].direction\|$
- Retourne depthTable
- Complexité en $\Theta(H \times L)$, le calcul de la norme2 est cependant plutôt lent à cause de la racine



Capture d'écran

Obtention d'une carte de profondeur

- Crée une Texture2D à l'aide de EncodeDepthMap
- L'affiche à l'écran
- Cette action n'est pas très optimale car on doit créer à chaque image une nouvelle Texture

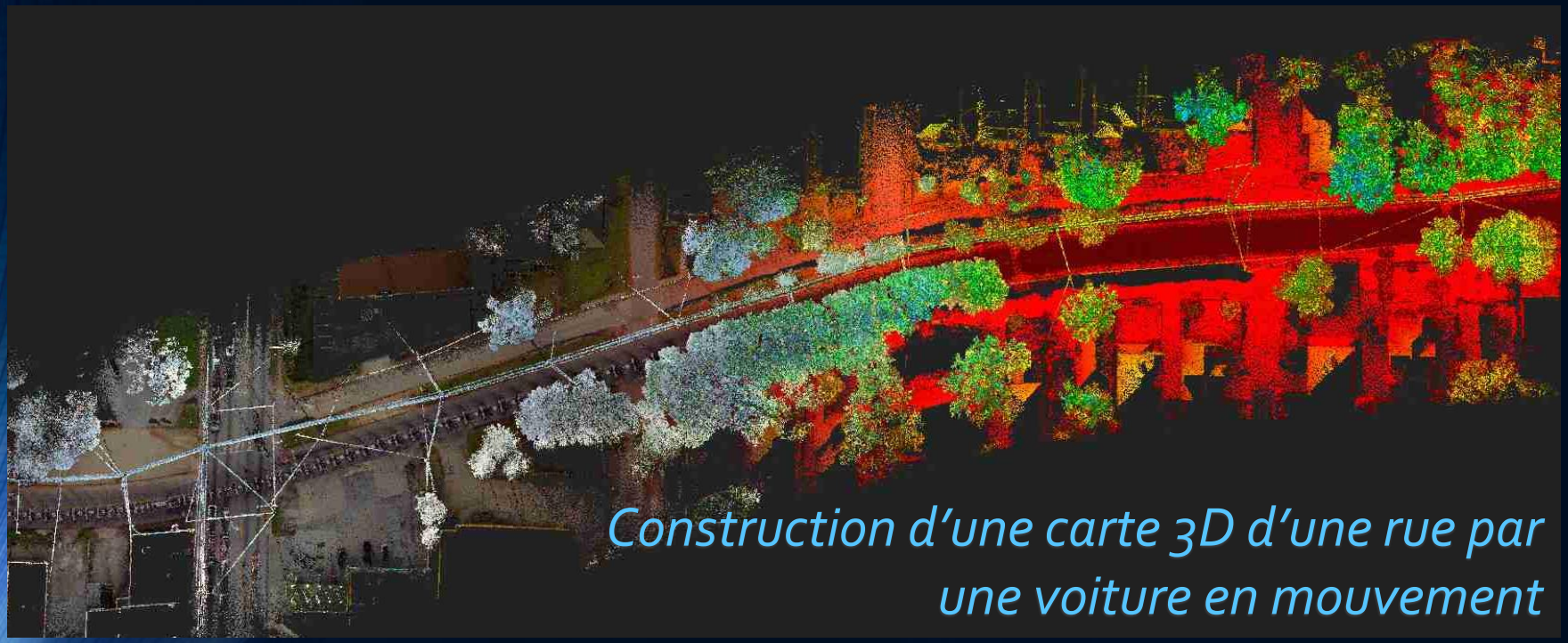
Capture d'écran

Méthodes de cartographie au Lidar

- CARTOGRAPHIE AVEC DES CARTES D'OCCUPATION
- RECONNAISSANCE DE FORMES AVEC UNE CARTE DE PROFONDEUR
- DRONES

Le concept du SLAM

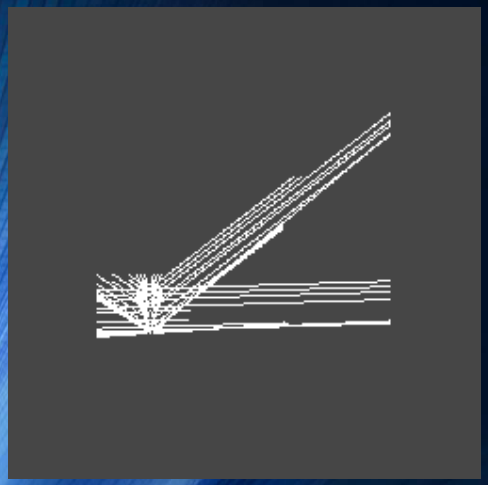
(Simultaneous Localization and Mapping)

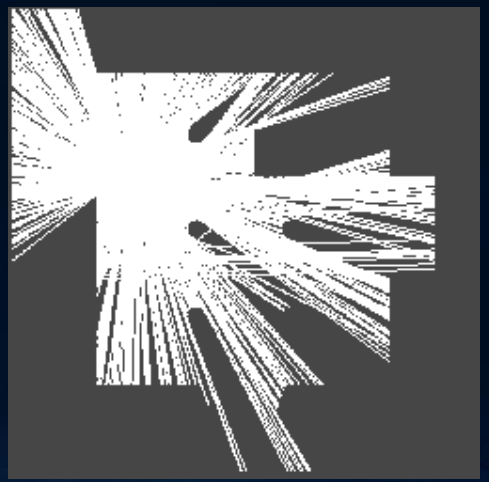


Construction d'une carte 3D d'une rue par une voiture en mouvement

numerisation3d.construction

Carte d'occupation

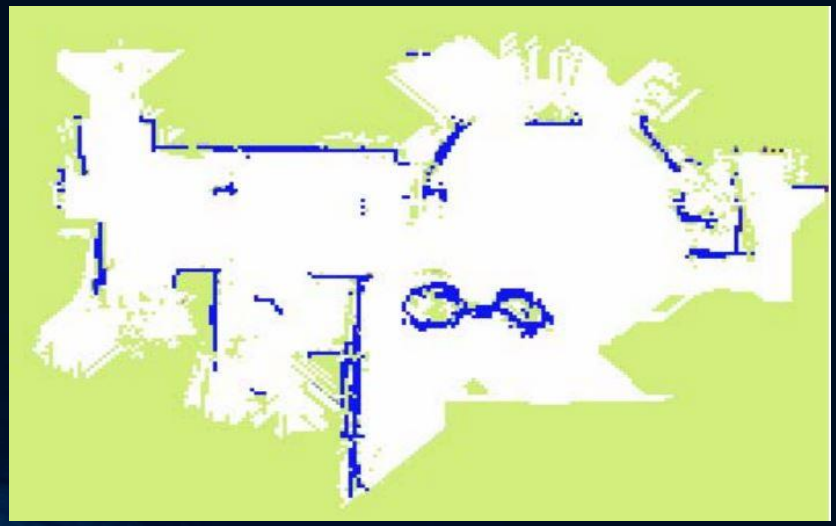




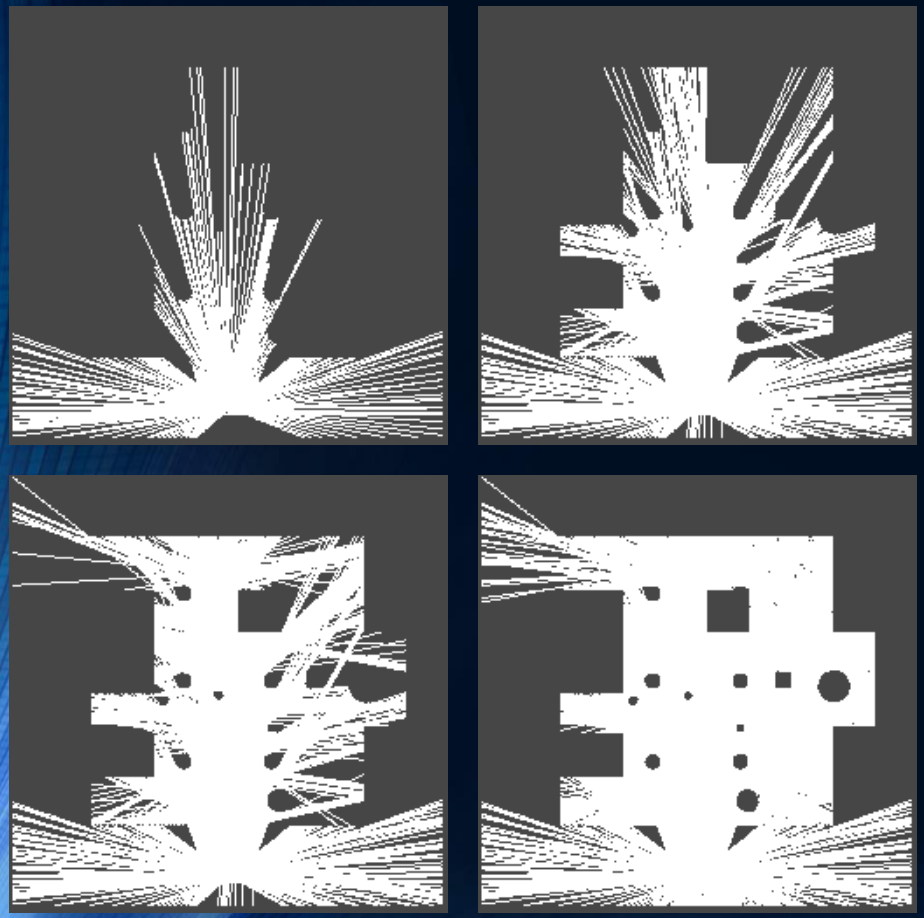
- Exemples de cartes d'occupation :

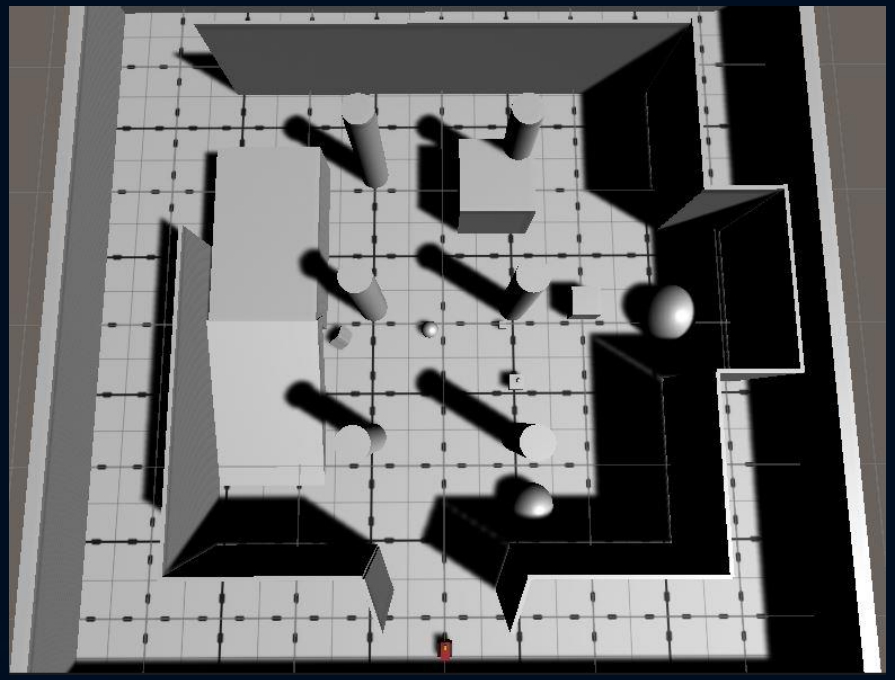
Ci-contre un exemple réalisé dans le simulateur (les couches 0,1,2 et 4)

Ci-dessous un exemple tiré de « Cartographie de l'environnement et suivi simultané de cibles dynamiques par un robot mobile »



Exemple de construction de la carte





Etats de la carte d'occupation à plusieurs moments, comparé à la carte réelle

Méthode de calcul utilisée ici :

Envoi et réception
- Création et envoi de rayons
- Calcul des rayons réfléchis

Calcul de la carte
- Calcul du parcours de chaque rayon

Mise à jour de la Carte
- Lecture de chaque parcours
- Changement de statut des cases de la matrice où passent les rayons

Affichage
- Transformation de carte en Texture2D
- Affichage de cette Texture2D

Présentation des principales fonctions utilisées

Parcours (Quadrillage)

- Donne une liste de bipoints correspondant aux cases où passe le bipoint d'entrée.

UpdateMap (OccupancyMap)

- Pour chaque bipoint obtenu par Lidar, crée la liste parcours correspondante
- Pour chaque case dans chaque parcours, changer l'état de la carte d'occupation de la case correspondante

UpdateOccupationMap (GraphDisplay)

- Pour chaque case, changer le pixel correspondant sur les textures en blanc si vide et en gris si inconnu
- Exporter chaque texture en PNG

Fonctionnement de Parcours

- Entrées :
  - Bipoint(origine, direction), quadrillage
- Calcule :
  - $a = |fleche.x - origine.x|$
  - $b = |fleche.y - origine.y|$
  - $n = \left\lfloor \sqrt{a^2 + b^2} \right\rfloor$
- Si $n \neq 0, \forall k \in [\![1, n]\!]$:
  - $coord_k = Point\left(\frac{k}{n} \times direction\right)$
  - Point() transforme les coordonnées flottantes en entiers Retourne la liste des $coord_k$
- Complexité de l'ordre de la taille du rayon

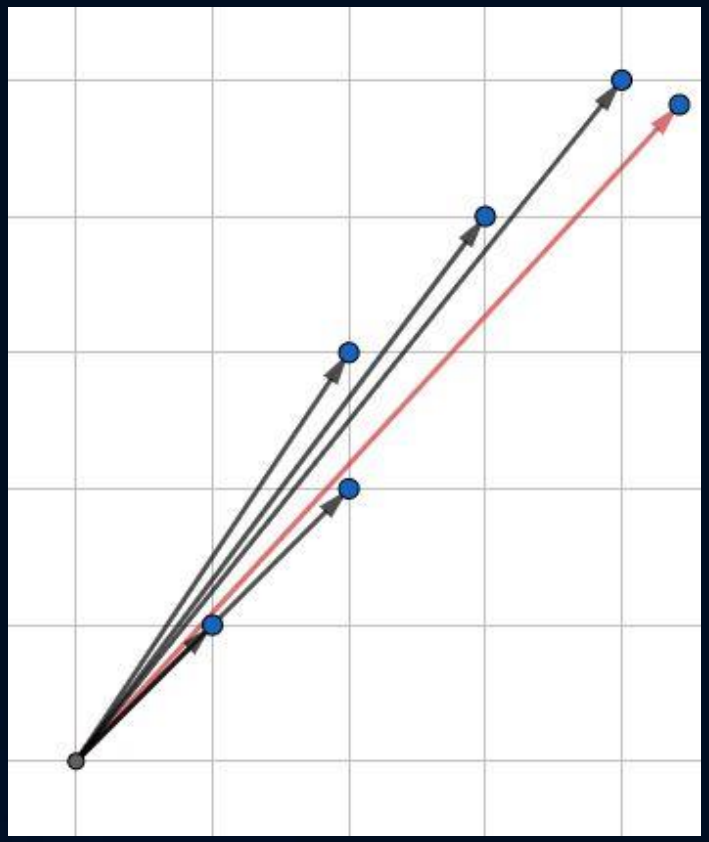


Illustration du découpage du vecteur rouge dans le quadrillage

Fonctionnement de UpdateMap et UpdateOccupationMap

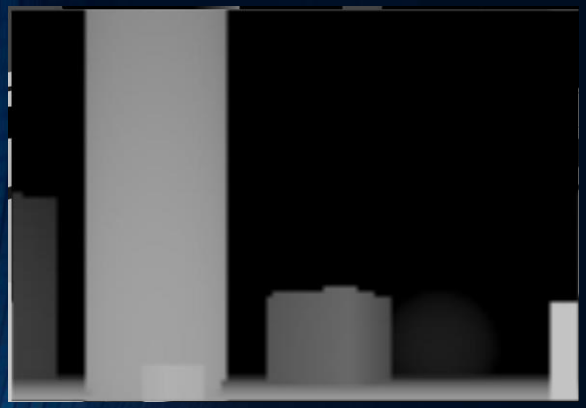
UPDATEMAP

- Récupère la liste parcours de chaque rayon
- Pour chaque vecteur de chaque parcours, colorie la case correspondante dans carte en Vide
- La complexité est alors en $\Theta(h \times l \times t)$

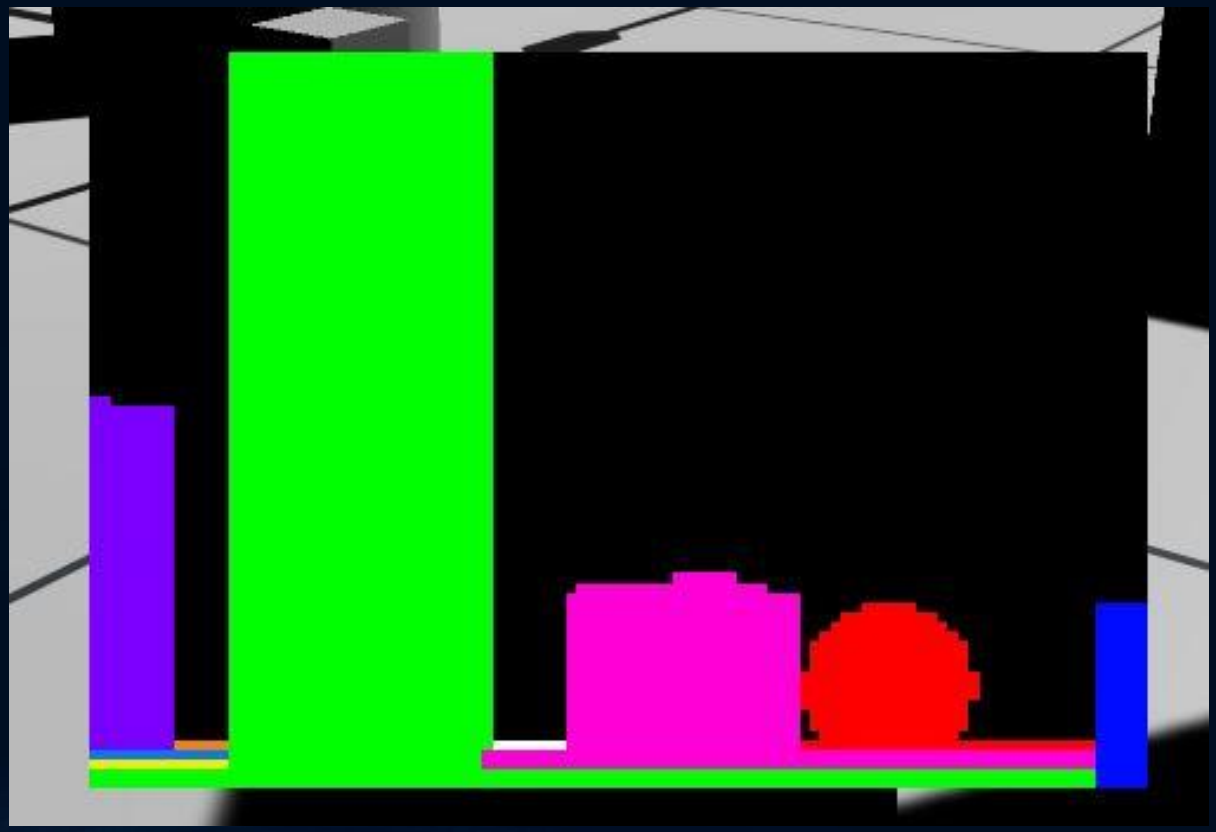
UPDATEOCCUPANCYMAP

- Pour chaque couche k de carte, lit la Texture associée
- $\forall (i,j) \in [\![1,n]\!]^2$:
  - Colorie le pixel (i,j) de la texture2D de la couleur correspondante de $carte[i,j,k]$
- Applique les changements et affiche la Texture sélectionnée
- Complexité en $\Theta(H \times L \times c)$

Détection de différents objets

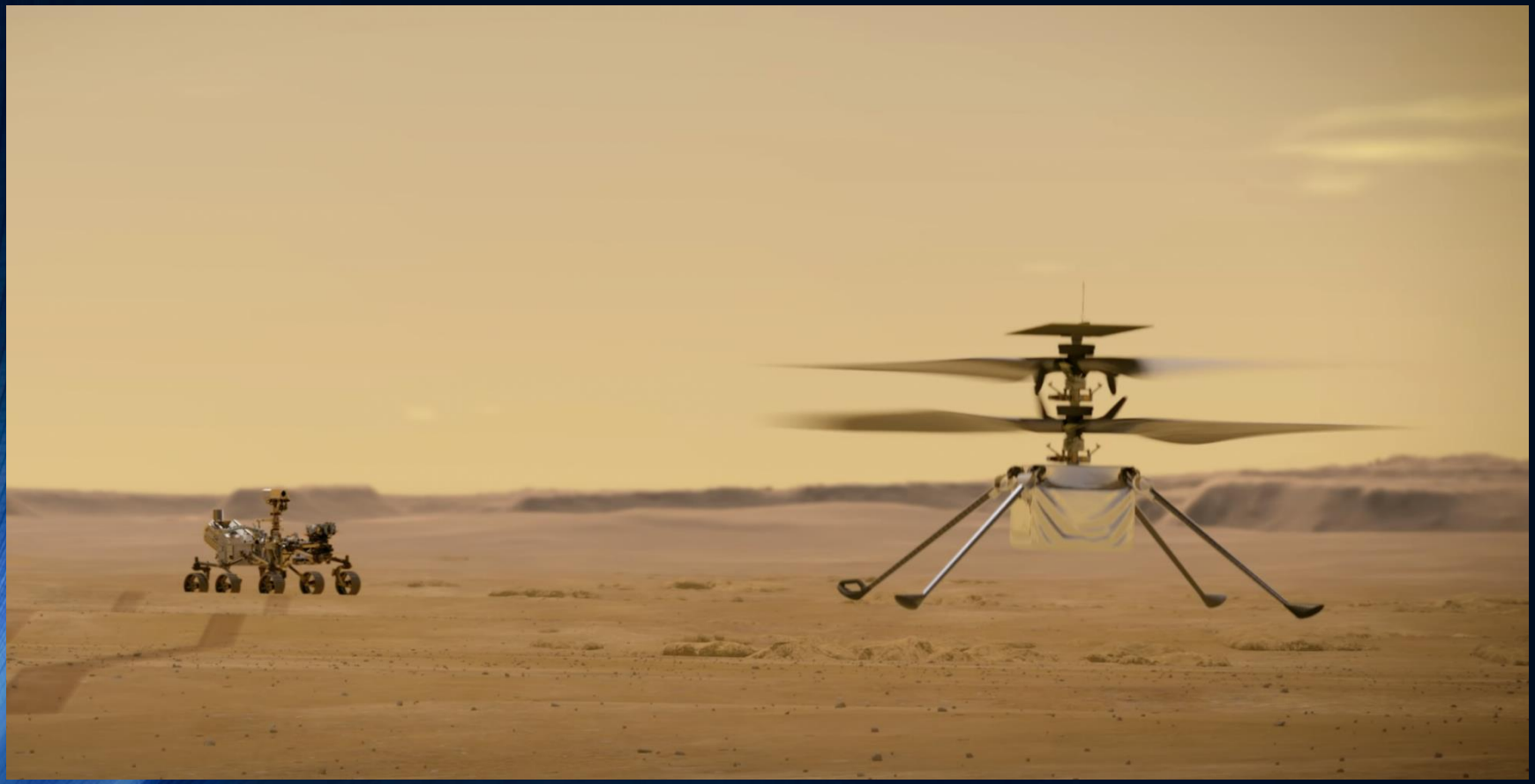


Carte de profondeur coloriée en fonction de la profondeur par ObjectDetector.



Capture d'écran

Drone Ingenuity sur Mars



https://fr.sputniknews.com/sci_tech/...

Applications concrètes

- OBJECTIFS DES RECHERCHES EN CARTOGRAPHIE

Enjeux de la recherche actuelle en cartographie

- Optimisation
  - Programmes plus complexes
  - Très petits processeurs
- Communications
  - Objets de plus en plus connectés
  - Interactions entre systèmes complexes
- Autonomie
  - Confort pour l'homme
  - Endroits dangereux

Conclusion

- Enjeux importants autour de ce domaine
- Utiliser un simulateur informatique est très efficace et très important avant de tester un robot en conditions réelles
- Un tel simulateur peut en effet permettre de créer les programmes en se passant des contraintes physiques
- Les cartes obtenues dans notre cas sont tout de même assez précises mais très incomplètes