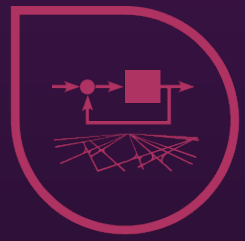
gipsa-lab

Grenoble | images | parole | signal | automatique | laboratoire

Perception embarquée sur GPU pour la robotique



École thématique "signal images : architecture et programmation des GPU"

Amaury Nègre

CNRS, GIPSA-lab

10 Novembre 2017

cnrs Grenoble INP UNIVERSITÉ Grenoble Alpes UMR 5216

Perception embarquée

Traitement de données par un calculateur directement connecté aux capteurs en mouvement, pour la prise de décision.

Exemples

- Vision pour la robotique (reconnaissance de personnes, de visages, de couleurs)
- Localisation, reconstruction 3D
- Détection d'obstacles

Problématiques

- Complexité des traitements (capteurs en mouvement, quantité des données)
- Temps de calcul, réactivité
- Consommation énergétique
- Encombrement
- Fiabilité

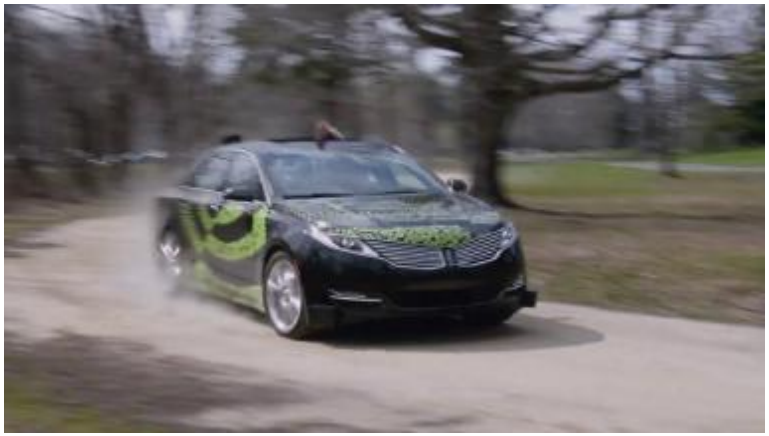
Intérêt du GPU

- Natures des données adaptées (images, nuages de points, son)
- Bon rapport puissance/consommation
- Gamme de produits dédié à l'embarqué (Jetson Tegra K1, X1, X2)
- Facilité de programmation, outils de développement (par rapport aux autres solutions embarquées)

Cas d'études

1. Conduite autonome
2. Navigation de drône

Conduite autonome



Application étudiée

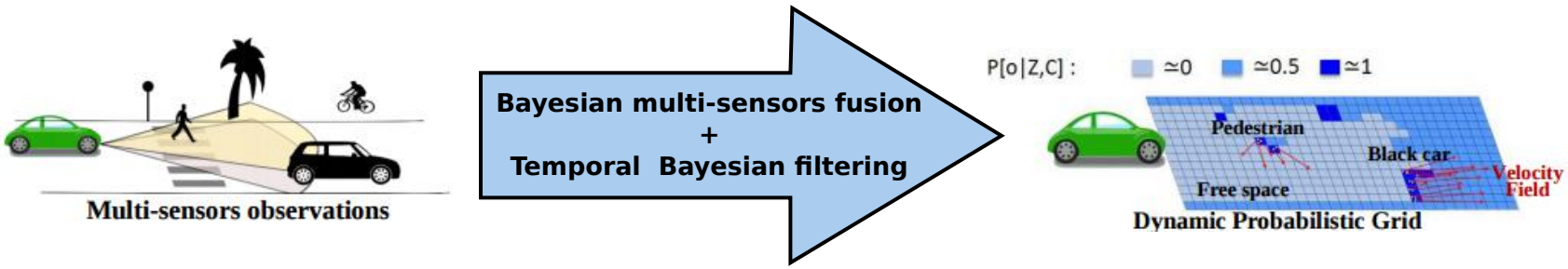
Prédiction de collision

- Environnement ouvert et dynamique
  → calcul en temps réel
- Incertitude
  → approches probabilistes
- Limitation des capteurs
  → fusion multi-capteurs
- Intégration matérielle
  → contraintes embarquée (encombrement, energie)

Application étudiée

Prédiction de collision

- Environnement ouvert et dynamique
  → calcul en temps réel
- Incertitude
  → approches probabilistes
- Limitation des capteurs
  → fusion multi-capteurs
- Intégration matérielle
  → contraintes embarquée (encombrement, energie)



plateformes

Toyota Lexus

Renault Zoé

ROS

Nvidia GTX Titan X
Generation Maxwell

Nvidia GTX Jetson TK1
Generation Maxwell

Nvidia GTX Jetson TX1
Generation Maxwell

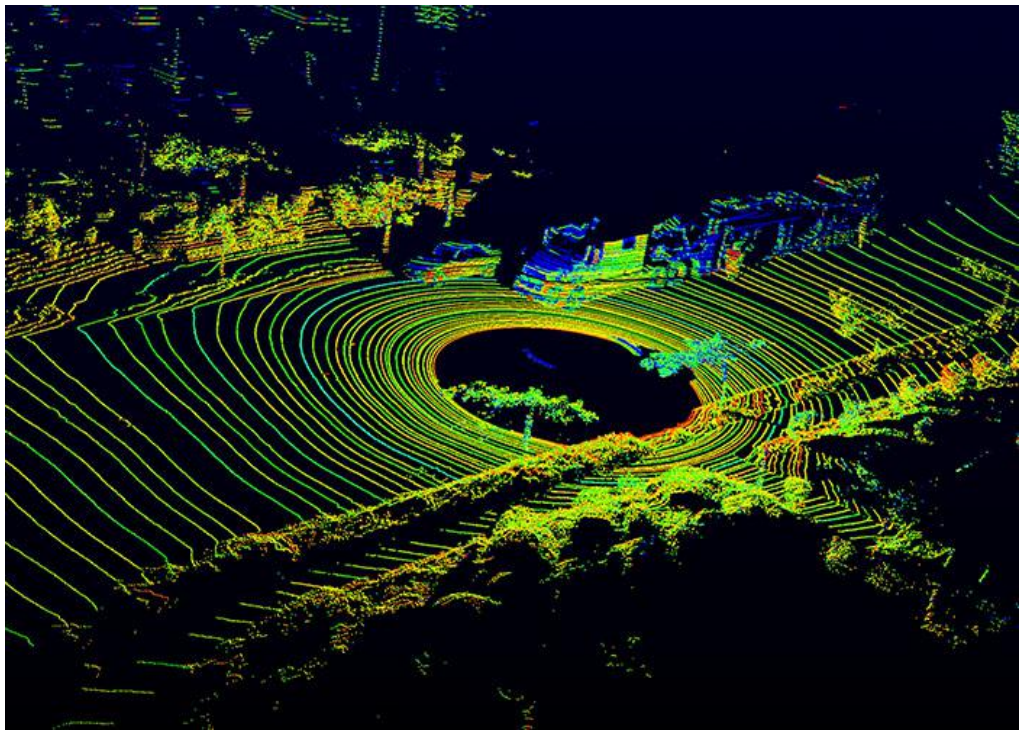
Connected Perception Unit

Données capteur

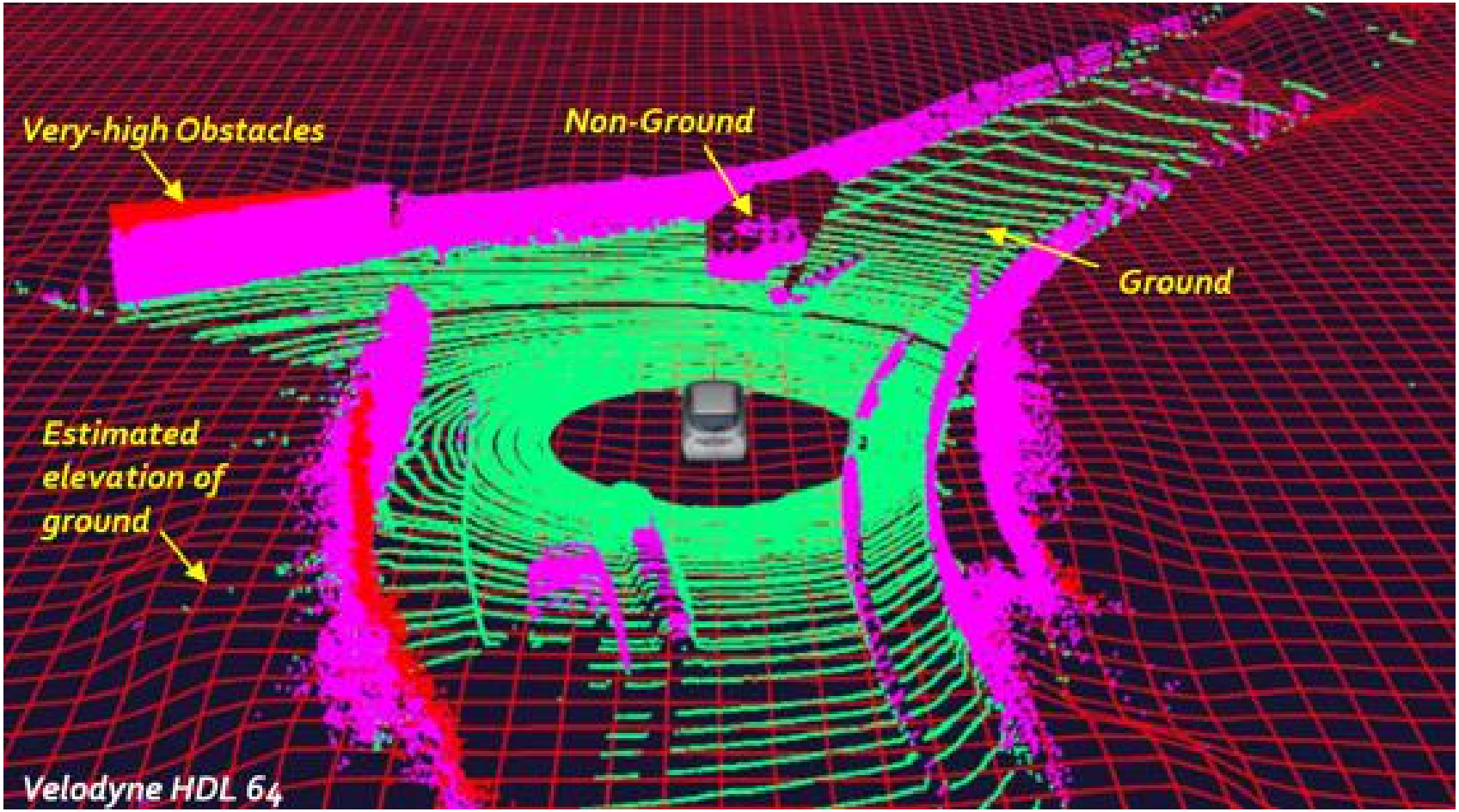
Télémètre laser Velodyne HDL-64E

- angle de vue 360°
- résolution angulaire 0.08°
- 64 canaux

- 2.2 Millions de points par seconde

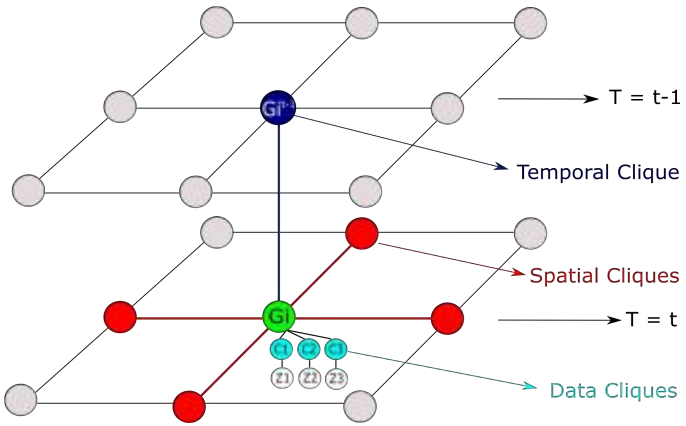


Classification des points et estimation du sol



Modèle mathématique

Champ de Markov Conditionnel Gaussien avec 3 potentiels



- **Potentiel de mesure** : dépend des points observés
- **Potentiel spatial** : dépend du voisinage spatial
- **Potentiel temporel** : dépend des estimations passées

Résolution

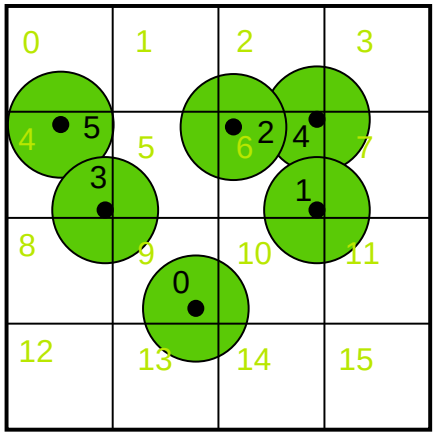
- Discrétisation du plan horizontal sous forme de **maillage uniforme**
- Méthode itérative de type "**Expectation-Maximization**"
  - **Expectation** : calcul des probabilité d'appartenance au sol pour chaque point
  - **Maximization** : mise à jour de l'état de chaque noeud

$$X_i^{k+1} = \alpha \sum_{j \in \mathcal{M}_i} c_j z_j H_{ij}^T + \beta \sum_{j \in \mathcal{N}_i} F_{ij}^T X_j^k + \gamma Q_i^T X_i^{t-1}$$

$$P_i^{k+1} = \alpha \sum_{j \in \mathcal{M}_i} c_j H_{ij}^T H_{ij} + \beta \sum_{j \in \mathcal{N}_i} F_{ij}^T P_j^k F_{ij} + \gamma Q_i^T P_i^{t-1} Q_i$$

Implémentation CUDA

1. Affectation de chaque point laser au noeud correspondant :
   - 1.1 Calcul des indices pour chaque point en parallèle ("hash")
   - 1.2 Tris des points et des indices (`thrust::sort_by_key`)
   - 1.3 Récupération du premier/dernier point par noeud (en parallèle pour chaque point)



– Uniform Grid using Sorting

Index	Unsorted list (cell id, particle id)	List sorted by cell id	Cell start
0	(9, 0)	(4, 3)	
1	(6, 1)	(4, 5)	
2	(6, 2)	(6, 1)	
3	(4, 3)	(6, 2)	
4	(6, 4)	(6, 4)	0
5	(4, 5)	(9, 0)	
6			2
7			
8			
9			5
10			
11			
12			
13			
14			
15			

2. Itérer k fois :
   - 2.1 Calcul des probabilité des points pour chaque noeud
   - 2.2 Mise à jour de l'état de chaque noeud

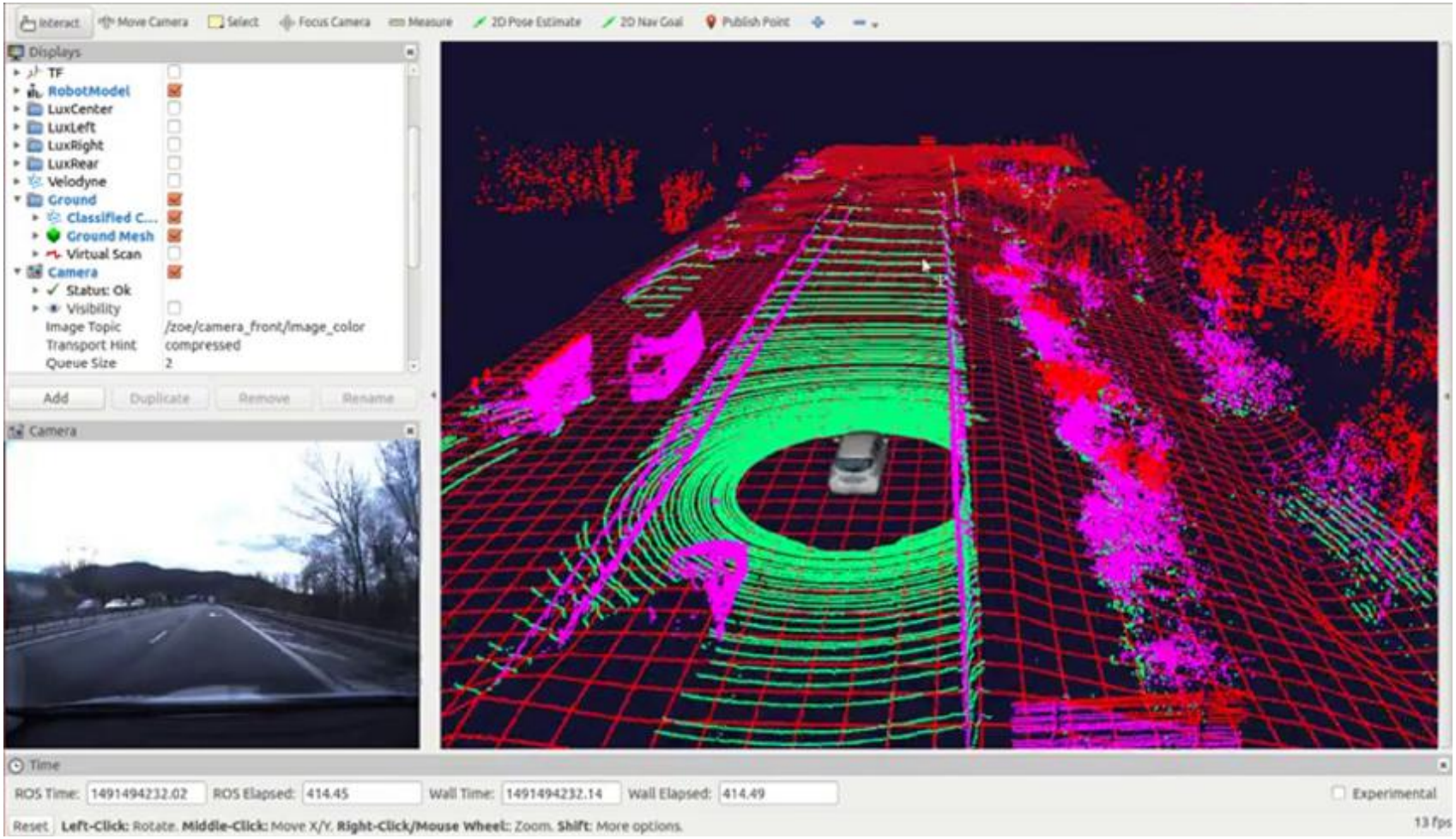
Optimisations

- Utilisation de la mémoire partagée (copie des états précédant au voisinage de chaque noeud)
- Synchronisation globale (pour éviter de lancer k fois le kernel)

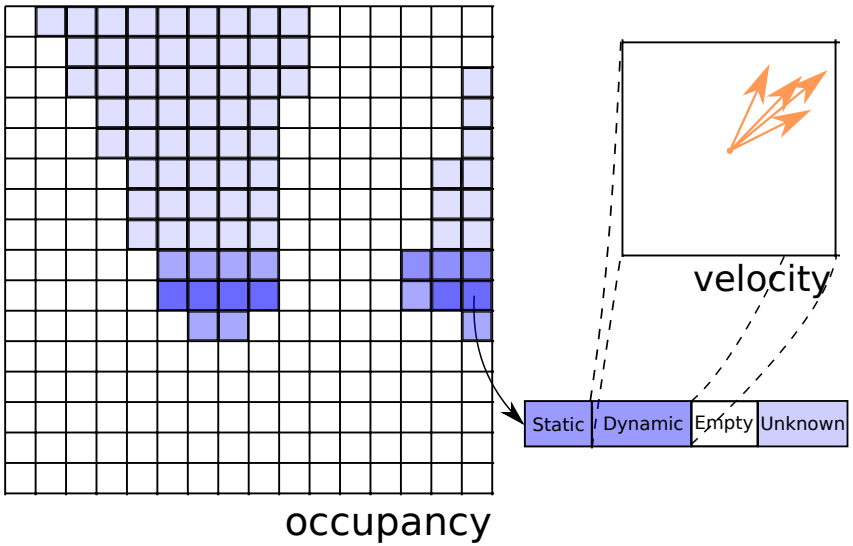
  $\rightarrow$ possible uniquement si le nombre de block $<$ nombre de multiprocesseurs

  $\rightarrow$ impossible sur Tegra X1 (2 multi-processeurs)
- Real-time processing (>10Hz on Jetson X1)

Résultats


Interact
Move Camera
Select
Focus Camera
Measure
2D Pose Estimate
2D Nav Goal
Publish Point
Displays
TF
RobotModel
LuxCenter
LuxLeft
LuxRight
LuxRear
Velodyne
Ground
Classified C...
Ground Mesh
Virtual Scan
Camera
Status: Ok
Visibility
Image Topic /zoe/camera_front/image_color
Transport Hint compressed
Queue Size 2
Add
Duplicate
Remove
Rename
Camera
Time
ROS Time: 1491494232.02
ROS Elapsed: 414.45
Wall Time: 1491494232.14
Wall Elapsed: 414.49
Experimental
Reset
Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click/Mouse Wheel: Zoom. Shift: More options.
13 fps

Construction de grilles d'occupation dynamiques



Représentation Hybride

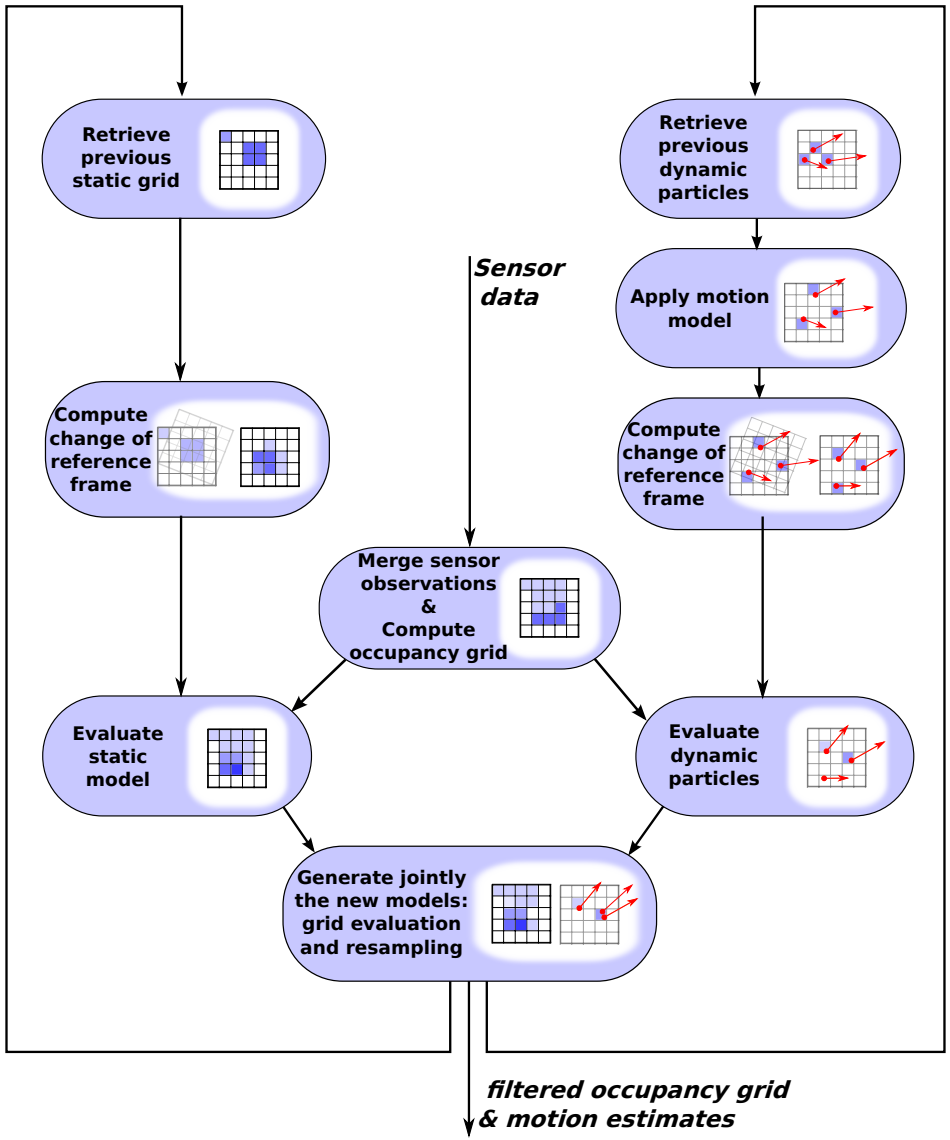
- Grille pour représenter l'état (vide, statique, dynamique, inconnu)
- Particules pour représenter la distribution de vitesse des cellules dynamiques

Résolution

Pour chaque cellule, il faut estimer la distribution de l'état et la vitesse :

$$P(SV|ZC) = \lambda \sum_{C^{-1}S^{-1}V^{-1}} P(C^{-1})P(S^{-1}V^{-1}|C^{-1}) P(SV|S^{-1}V^{-1})P(C|C^{-1}V)P(Z|S,V,C)$$

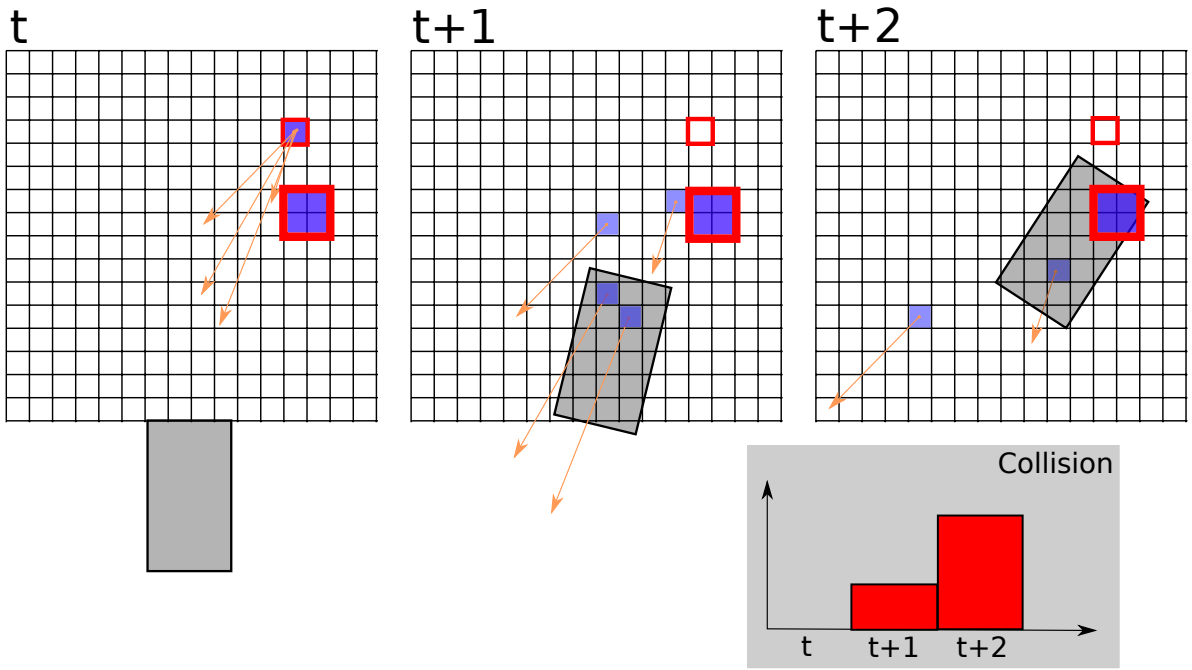
Construction de grilles d'occupation dynamiques



Implémentation Cuda

- Correction du mouvement propre : Transformation affine sur la grille et les particules
- Propagation des particules : parallélisation sur l'ensemble des particules
- Collection des particules par cellule : Hash + Sort
- Modèle capteur : parallélisation sur les cellules
- Prédiction et mise-à-jour : parallélisation sur les cellules
- Ré-échantillonnage des particules : Prefix sum + binary searches (parallélisation par particule)

Estimation du risque de collision



- Simulation du déplacement du véhicule et des particules
- Calcul du temps avant collision (TTC) pour chaque cellule et chaque particule
- Probabilité de collision pour 3 horizons de temps
- Utilisation des grilles de risque ou du risque max (réduction)

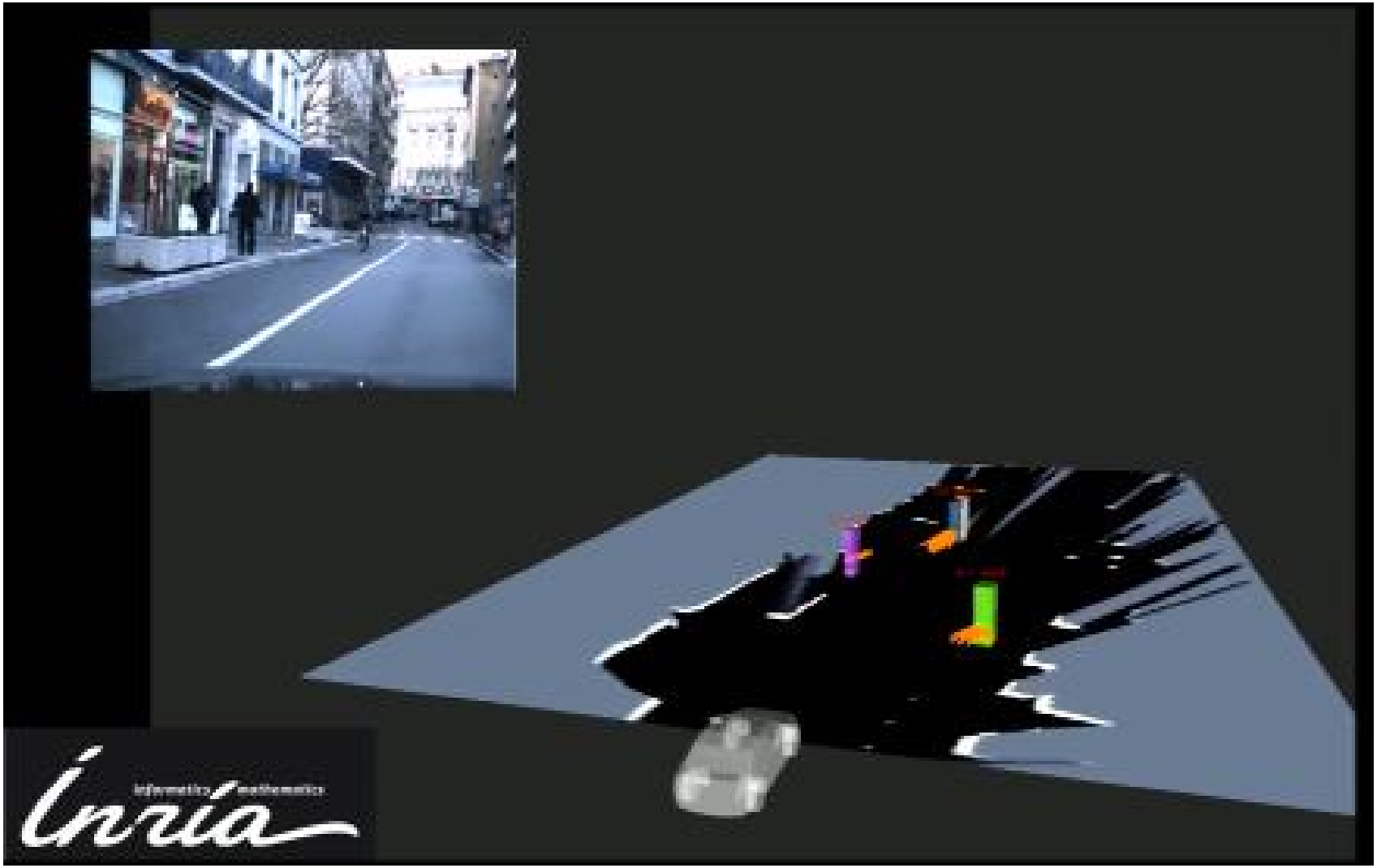
Résultats

- Configuration avec 8 nappes lasers (2x4)
- Taille de grille 1400x600 (840000 cellules) et 65536 particules

Jetson TK1 : Grid fusion **17ms**, grid filtering + object clustering **70ms**

Jetson TX1 : Grid fusion **0.7ms**, grid filtering + object clustering **17ms**

Résultats



Navigation de drones : problématiques

- Réalisation de tâches de vision en temps réel en vol
- Avec des contraintes forte en poids...
- Et en énergie

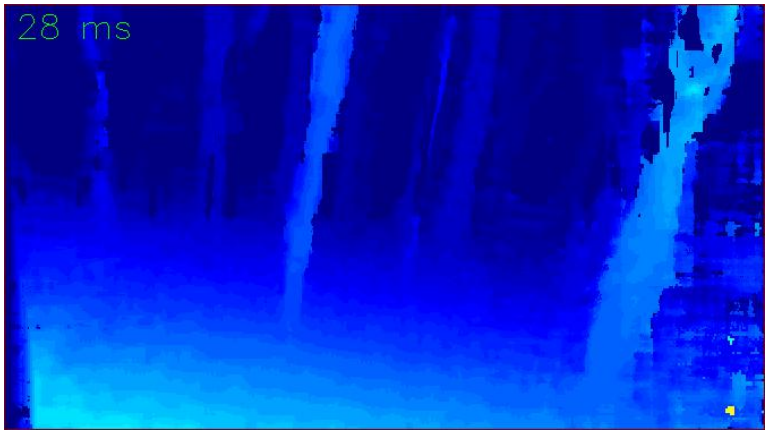
Navigation de drones : problématiques

- Réalisation de tâches de vision en temps réel en vol
- Avec des contraintes forte en poids...
- Et en énergie

Solution :

- Implémentation Cuda sur plateforme jetson TX1/TX2 avec carte fille légère (auvidea J1XX, connect Tech ASG00X)

Carte de profondeur stéréo



Evaluation des implémentations VisionWorks (Nvidia) et OpenCV

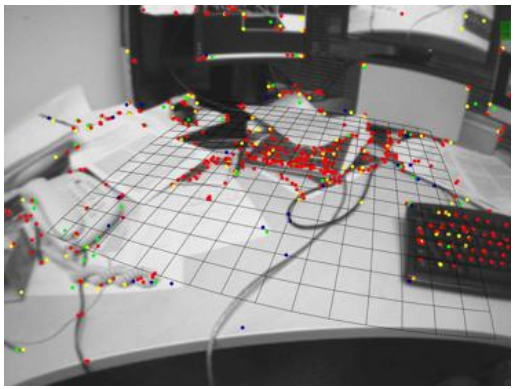
	NVBM	SGM	CVBM	BP	CSBP
1280 x 720 (128)	19 fps	4 fps	14 fps	erreur	0,7 fps
640 x 360 (64)	90 fps	28 fps	62 fps	0,9 fps	3 fps
320 x 180 (32)	250 fps	125 fps	125 fps	6 fps	11 fps

SLAM

Localisation et construction de carte simultanée

- Evolution des algorithmes depuis une dizaine d'année

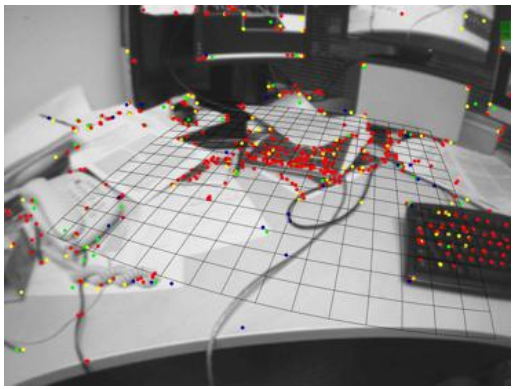
1. Méthodes "sparse" (basée sur des points d'intérêts)

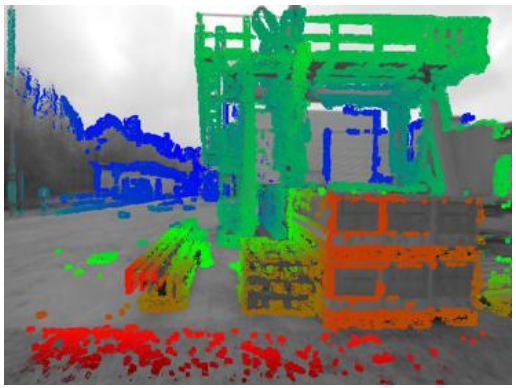


Localisation et construction de carte simultanée

- Evolution des algorithmes depuis une dizaine d'année

1. Méthodes "sparse" (basée sur des points d'intérêts)
2. Méthodes "semi-denses" (patchs centrés sur des points d'intérêts)

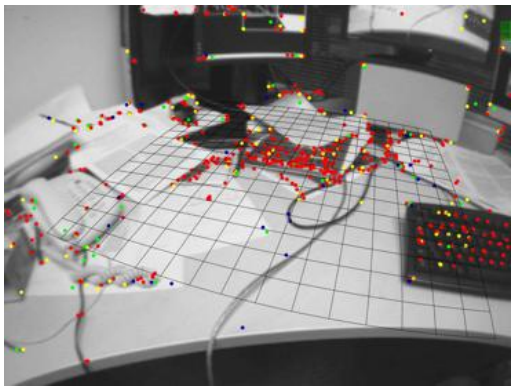
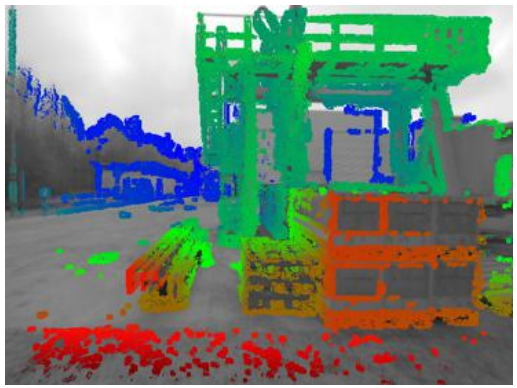
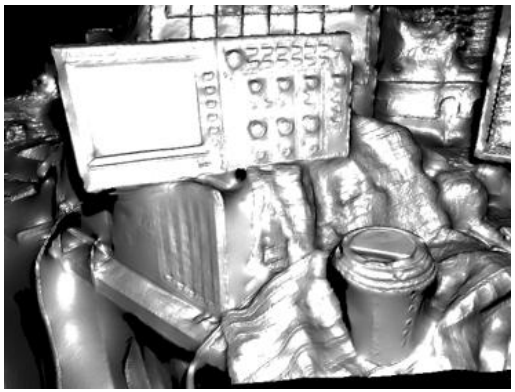




Localisation et construction de carte simultanée

- Evolution des algorithmes depuis une dizaine d'année

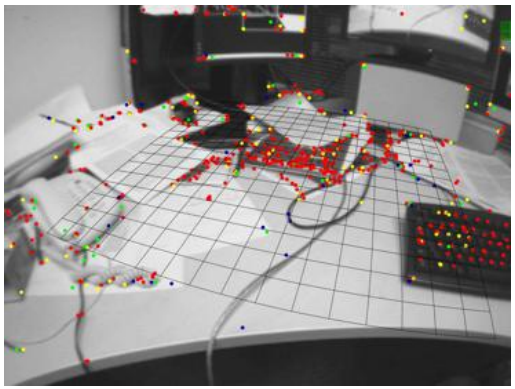
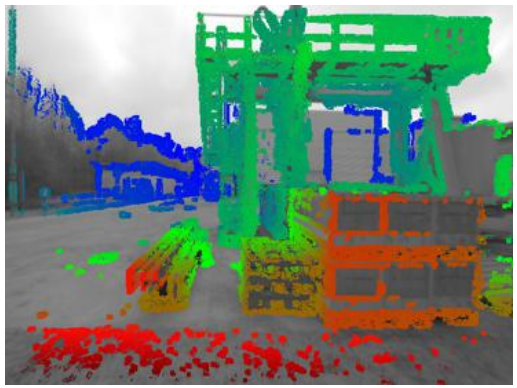
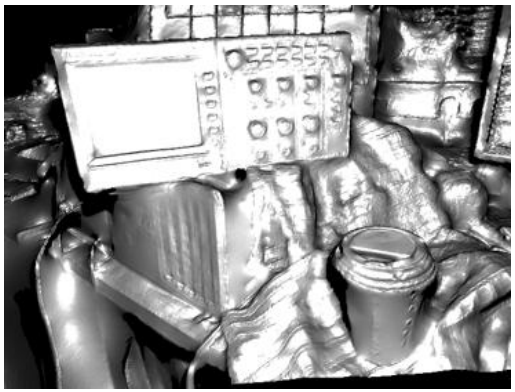
1. Méthodes "sparse" (basée sur des points d'intérêts)
2. Méthodes "semi-denses" (patchs centrés sur des points d'intérêts)
3. Méthodes "denses" (cartes complètes)

Localisation et construction de carte simultanée

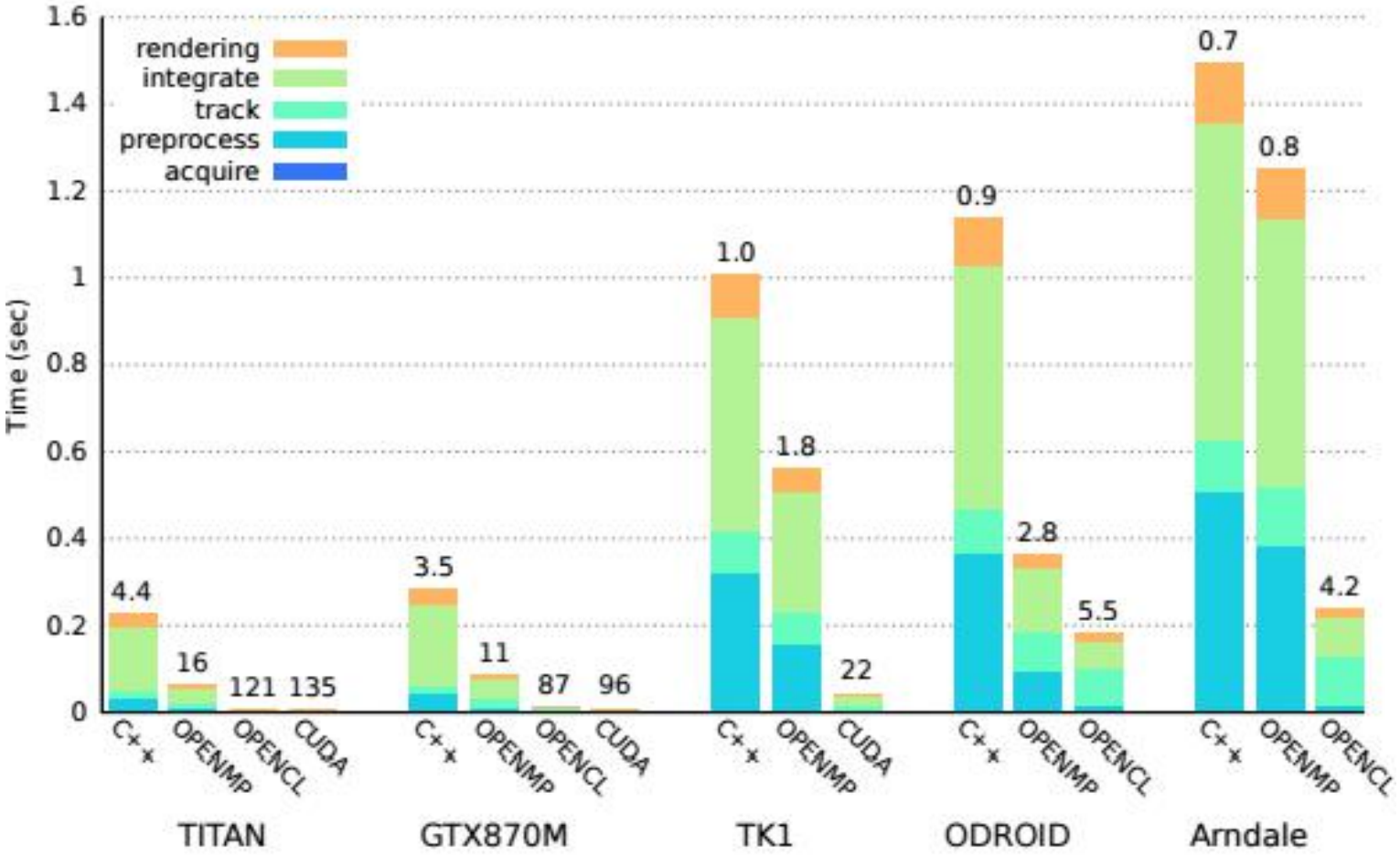
- Evolution des algorithmes depuis une dizaine d'année

1. Méthodes "sparse" (basée sur des points d'intérêts)
2. Méthodes "semi-denses" (patchs centrés sur des points d'intérêts)
3. Méthodes "denses" (cartes complètes)

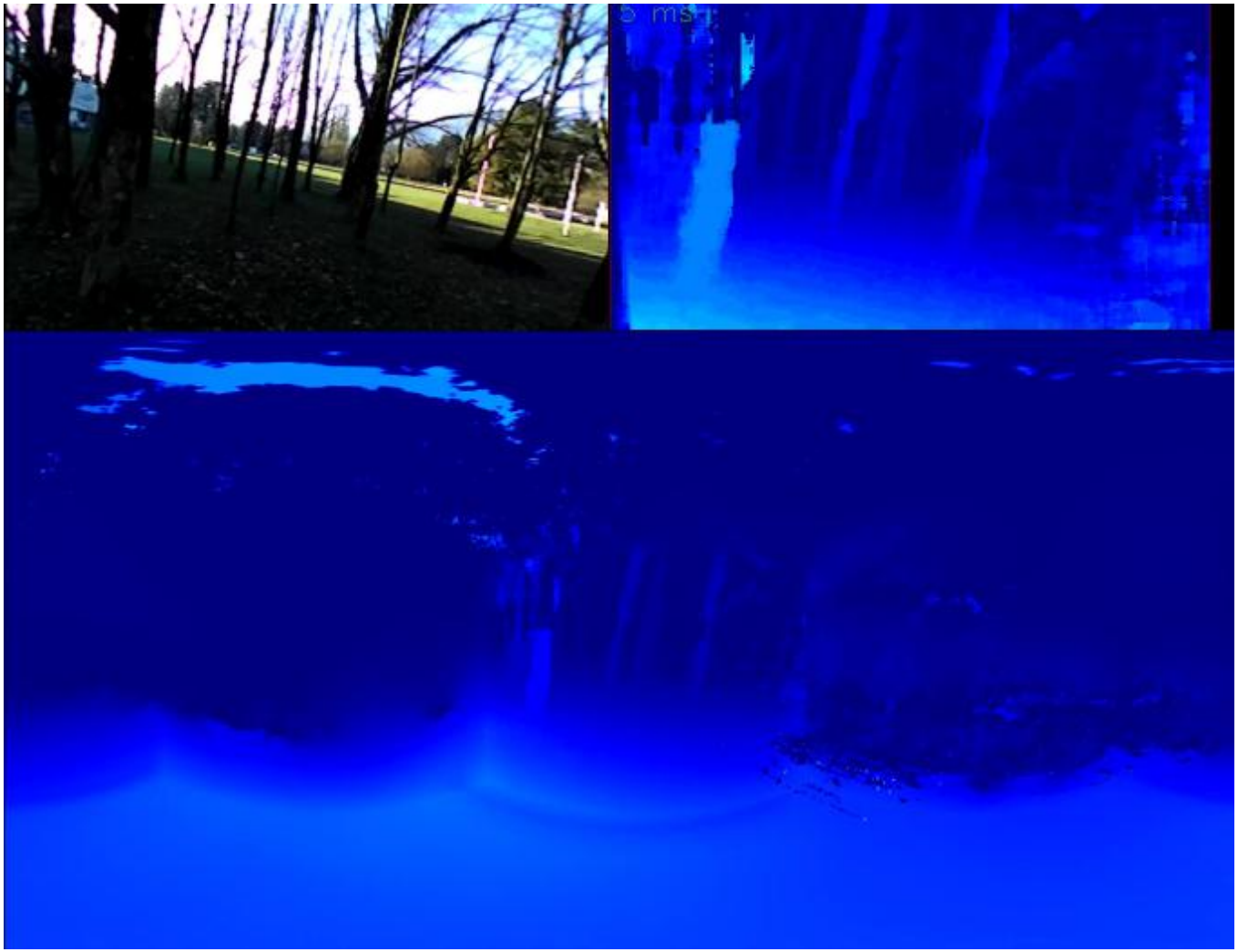
- Architectures GPU très adaptées aux méthodes denses

SLAM



Temps de calculs de KinectFusion,
d'après SLAMBench [Nardi *et al.* 2015]

Odométrie visuelle + cubemap



Synthèse

- GPU bien adapté aux tâches de perception denses (sous forme de grilles, nuages de points ou les deux)
- Performance temps réel avec gros flux de donnée (Lidar3D, camera RGB-D, etc.)
- Cartes embarquées très performantes
- Principaux facteurs limitants : Allocations, Bandes-passante mémoire, synchronisation

Remarques de développeur

- Environnement embarqué très pratique (identique PC ubuntu)
- Utilisation de la librairie **Thrust**
  - Allocateurs par défaut à redéfinir !
- **Débugage** pas toujours simple
- Penser à utiliser le **profiler** Nvidia
- Nécessité de repenser tous les algorithmes !

Merci de votre attention !

Questions ?