

UNIVERSITÉ DE BORDEAUX - REALITY TECH

---

**Mémoire de stage  
Master 2 Informatique  
Image et son**

---

*Auteur :*  
Nicolas PALARD

*Client :*  
Jérémy LAVIOLE  
*Référent :*  
Vincent LEPESTIT





## Résumé

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed non risus. Suspendisse lectus tortor, dignissim sit amet, adipiscing nec, ultricies sed, dolor. Cras elementum ultrices diam. Maecenas ligula massa, varius a, semper congue, euismod non, mi. Proin porttitor, orci nec nonummy molestie, enim est eleifend mi, non fermentum diam nisl sit amet erat. Duis semper. Duis arcu massa, scelerisque vitae, consequat in, pretium a, enim. Pellentesque congue. Ut in risus volutpat libero pharetra tempor. Cras vestibulum bibendum augue. Praesent egestas leo in pede. Praesent blandit odio eu enim. Pellentesque sed dui ut augue blandit sodales. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Aliquam nibh. Mauris ac mauris sed pede pellentesque fermentum. Maecenas adipiscing ante non diam sodales hendrerit. Ut velit mauris, egestas sed, gravida nec, ornare ut, mi. Aenean ut orci vel massa suscipit pulvinar. Nulla sollicitudin. Fusce varius, ligula non tempus aliquam, nunc turpis ullamcorper nibh, in tempus sapien eros vitae ligula. Pellentesque rhoncus nunc et augue. Integer id felis. Curabitur aliquet pellentesque diam. Integer quis metus vitae elit lobortis egestas. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Morbi vel erat non mauris convallis vehicula. Nulla et sapien. Integer tortor tellus, aliquam faucibus, convallis id, congue eu, quam. Mauris ullamcorper felis vitae erat. Proin feugiat, augue non elementum posuere, metus purus iaculis lectus, et tristique ligula justo vitae magna. Aliquam convallis sollicitudin purus. Praesent aliquam, enim at fermentum mollis, ligula massa adipiscing nisl, ac euismod nibh nisl eu lectus. Fusce vulputate sem at sapien. Vivamus leo. Aliquam euismod libero eu enim. Nulla nec felis sed leo placerat imperdiet. Aenean suscipit nulla in justo. Suspendisse cursus rutrum augue. Nulla tincidunt tincidunt mi. Curabitur iaculis, lorem vel rhoncus faucibus, felis magna fermentum augue, et ultricies lacus lorem varius purus. Curabitur eu amet.

## Remerciements

Je souhaite remercier toutes les personnes que j'ai eu l'occasion de rencontrer pendant mon stage pour l'enrichissement aussi bien professionnel que personnel qu'elles m'ont apportées. Je souhaite également remercier toutes les personnes qui m'ont aider que ce soit pendant mon stage ou pour écrire ce mémoire et qui ont contribuer de près ou de loin au succès de celui ci.

J'adresse tous mes remerciements à mon maître et encadrant de stage **M. Laviole Jeremy**, pour m'avoir fait confiance en m'ouvrant les portes de son entreprise mais aussi pour sa présence, ses conseils, et pour toute l'aide et l'accompagnement qu'il a pu me fournir au quotidien durant mon stage.

Je souhaite également remercier **Mme Lauren Thevin**, chercheuse en réalité augmentée et mixte pour les déficients visuels chez Inria et collaboratrice de M. Laviole pour tous les échanges que nous avons eu l'occasion d'avoir durant ce stage qui ont toujours été très enrichissants.

Je souhaite aussi sincèrement remercier **M. Bénard Pierre**, un de mes anciens professeurs à l'Université de Bordeaux pour ses cours, mais surtout pour son extrême disponibilité et sa réactivité lorsque j'avais des questions durant mon stage.

Pour finir je souhaite remercier **M. Lepetit Vincent**, mon enseignant référent qui m'a encadré durant toute la durée de ce stage.

# Table des matières

|  |           |
|--|-----------|
| <b>Remerciements</b>                                 | <b>1</b>  |
| <b>1 Introduction</b>                                | <b>1</b>  |
| 1.1 Contexte et cadre . . . . .                      | 1         |
| 1.2 Objectifs . . . . .                              | 2         |
| <b>2 Notions</b>                                     | <b>3</b>  |
| <b>3 État de l'art</b>                               | <b>7</b>  |
| 3.1 PapARt . . . . .                                 | 7         |
| 3.2 Systèmes de réalité augmentée spatiale . . . . . | 8         |
| <b>4 Développement d'application</b>                 | <b>10</b> |
| 4.1 ReARTable . . . . .                              | 10        |
| 4.1.1 Besoins de l'application . . . . .             | 11        |
| 4.1.2 Choix et implémentation . . . . .              | 11        |
| 4.2 Extraction de document . . . . .                 | 14        |
| 4.2.1 Besoins de l'application . . . . .             | 14        |
| 4.2.2 Choix et implémentation . . . . .              | 14        |
| <b>5 Prototype haute performance</b>                 | <b>19</b> |
| 5.1 Amélioration logiciel . . . . .                  | 19        |
| 5.1.1 Convolution - Théorie . . . . .                | 19        |
| 5.1.2 Convolution - Optimisation . . . . .           | 21        |
| 5.1.3 Tests de performances . . . . .                | 25        |
| 5.1.4 Conclusion . . . . .                           | 26        |
| 5.2 Amélioration matérielle . . . . .                | 27        |
| 5.3 Nectar - Architecture micro services . . . . .   | 29        |
| <b>6 Unity</b>                                       | <b>31</b> |
| 6.1 Module Unity . . . . .                           | 31        |
| 6.2 Application Unity . . . . .                      | 31        |
| <b>Annexe 1</b>                                      | <b>32</b> |
| <b>Annexe 2</b>                                      | <b>34</b> |



# Introduction

Ce mémoire retracera les missions réalisées durant mon stage de Master 2 Informatique pour l'Image et le Son à l'Université de Bordeaux 1 effectué entre Avril et Septembre 2018 (6 mois) dans la société **RealityTech**<sup>1</sup>. Ce rapport ne couvrira cependant que les 5 premiers mois du stage car la date de rendu de ce dernier précède d'un moi la date de fin du stage.

Le stage a donc été effectué chez **RealityTech** une jeune start-up de réalité augmentée spatiale. Issue de l'Inria de Bordeaux, l'institut national de la recherche en information et en automatique, cette dernière est la continuité d'un projet de recherche mené par M. Laviole, ex ingénieur de recherche à l'Inria. Ce projet, PapARt<sup>2</sup> Paper Augmented Reality ToolKit, est un kit de développement (SDK) Processing permettant de créer des expériences de réalité augmentée sous forme d'applications de projection interactive dans des feuilles de papier. Pour réaliser ce type d'application du matériel spécifique est nécessaire tel que des caméra couleurs, des caméra de profondeur qui sont utilisées pour capter le monde réel, un projecteur utiliser pour pouvoir visualiser le contenu numérique dans l'espace et un ordinateur pour effectuer tous les calculs nécessaire au bon fonctionnement des applications. C'est dans un premier temps du côté matériel que **RealityTech** intervient. En effet, celle ci propose des systèmes pré calibré et prêt à l'emploi résolvant donc tous les problèmes de calibration et d'installation de l'environnement nécessaire à l'utilisation de PapARt. En parallèle de la partie matériel, la société développe et améliore activement le kit et créer aussi diverses applications de démonstration, afin de montrer et d'étendre les possibilités de la technologie. Le but est de pouvoir ouvrir la technologie à d'autres domaines d'activités que celui de la recherche pour mettre les outils collaboratifs et systèmes interactifs aux services de l'éducation, du vidéo ludique, de la vulgarisation etc....

## 1.1 Contexte et cadre

Actuellement, **RealityTech** se développe dans un incubateur de start-up appelé **La Banquiz**<sup>3</sup> situé 4 rue Eugène et Marc Dulout, à Pessac Centre. L'objectif de **La Banquiz** est de promouvoir des start-ups Open Source<sup>4</sup> et innovantes en apportant à leurs dirigeants des formations, du coaching individuel et collectif, de l'aide pour la recherche de financement et tout ce qui gravite autour de l'accompagnement de jeunes entreprises.

À ce jour **RealityTech** ne travail qu'avec des laboratoire de recherche tel que l'Inria et cherche à étendre son secteur d'activité. Les systèmes et services proposés par la société fournissent les résultats espérés et la dynamique de celle ci s'oriente donc vers une industrialisation du produit. Ainsi l'entreprise souhaite passer de prototype FabLab créé à l'unité basé sur des technologies de recherche, à prototype industriel basé sur des technologies populaires dans le monde de l'industrie. **RealityTech** se lance donc dans la création d'une plateforme haute performance appelé Nectar et dans le développement d'un nouvel SDK Unity<sup>5</sup> utilisant cette plateforme. Le SDK Processing actuel est très accessible et permet de prototyper bon nombre d'applications rapidement mais il ne permet cependant pas de répondre aux besoins de l'industrie plus spécialement quand il s'agit de développer des applications client ayant besoin d'un moteur 3D, ou d'un moteur physique performant.

Pour débuter ce stage, j'ai eu l'occasion de partir à Laval pendant une semaine pour assister au Laval Virtual, le plus grand salon international sur la réalité augmentée et virtuelle, en tant qu'exposant. Grâce a ce salon, j'ai pu développer une bonne connaissance du produit et ai pu observer de nombreuses

---

1. <http://rea.lity.tech/>

2. <https://project.inria.fr/papart/fr/>

3. <http://labanquiz.com>

4. Open Source - Wikipédia

5. <https://unity3d.com/fr>

technologies à l'état de l'art dans ces domaines. C'est aussi ce salon qui m'a permis de bien comprendre les besoins auxquels pouvait répondre une technologie comme celle de **RealityTech** et par la même occasion les enjeux et les apports de celle ci. Cela a aussi été une très bonne expérience au niveau relationnel car elle m'a permis de créer rapidement une relation avec M. Laviole, mon tuteur de stage.

Le reste de mon stage a été réalisé à **La Banquiz** où j'ai été au contact de toutes les entreprises officiant en son sein. J'ai notamment pu rencontrer d'autres stagiaires tel que Rémi Kressmann, développeur, et Gabin Andrieux, commercial chez **LockEmail**<sup>6</sup>, une entreprise de cyber-sécurité, mais aussi des dirigeants comme Jean François Schaff, docteur en physique quantique ayant créer **Postelo**<sup>7</sup> une plateforme de télé expertise pour les professionnels de santé, avec qui j'ai énormément échangé aussi bien sur des concepts de programmation, que sur la culture de l'informatique en générale.

### Problématique du sujet

## 1.2 Objectifs

Durant ce stage, j'ai eu l'occasion de devoir remplir bon nombre de tâches diverses et variées guidées par les besoins de la société. Voici les objectifs principaux défini pendant mon stage.

**Applications de démonstration** Un des objectifs du stage était le développement d'applications de démonstration en utilisant le système de l'entreprise. Le but était de comprendre l'essence, le fonctionnement global du système et ce qu'il était possible/impossible de réaliser avec celui ci. Cet objectif avait pour but d'acquérir à la fois une vision globale de l'architecture logiciel, du fonctionnement interne du kit de développement, et de l'architecture matérielle nécessaire à son utilisation mais aussi de développer une certaine autonomie pour la suite du stage.

**Plateforme haute performance** Durant ce stage, j'ai été amené a réaliser une preuve de concept visant à présenter une version haute performance du produit de l'entreprise. Cette preuve de concept devait proposer des méthodes et des pistes d'améliorations de la solution actuelle aussi bien sur le plan logiciel avec par exemple l'optimisation d'algorithmes existant, sur le plan matériel avec la réalisation de tests de performances des dispositifs caméra projecteur etc que sur l'architecture globale de la solution, ou il a fallut développer une toute nouvelle architecture en micro services avec un tout nouvel environnement de travail se basant sur celle ci avec un gestionnaire de processus et un serveur web.

**Kit de développement** Comme il a été expliqué dans le contexte du stage, les besoins auxquels répond le kit de développement Processing actuellement en place ne sont plus suffisants lorsqu'il est question d'applications devant faire le rendu de grosse scène 3D, ou des simulation physique,. Le développement d'une version **Unity** du kit permettant de créer des applications de projection interactive était donc nécessaire pour la suite du développement de **RealityTech**. Pour que le développement de ce module soit efficace, l'objectif était d'intégrer et d'utiliser les micro services du prototype haute performance Nectar pour gérer tout ce qui concerne le monde physique (caméras, projecteurs, suivi d'objet, événements de toucher, etc) et d'utiliser Unity dans son rôle de moteur 3D, moteur physique pour gérer le rendu de la scène, la physique des objets dans la scène, la lumière et créer des expériences de projection encore plus évolués. Le module devait aussi permettre d'ouvrir la technologie de **RealityTech** à un plus grand nombre d'utilisateur du fait de la notoriété l'appréciation d'**Unity** dans son domaine.

---

6. <http://www.lockemail.fr/>

7. <https://www.postelo.fr/>

# Notions

Le domaine d'activité qui entoure ce stage est très riche en termes de notions et de vocabulaire. Afin de mieux comprendre de quoi il va être question tout au long de ce rapport, il est nécessaire d'en définir les notions de base.

**Réalité virtuelle** La réalité virtuelle plus communément appelé *Virtual Reality (VR)* désigne l'ensemble des environnements purement numériques (fig 2.1), qu'ils soient réalistes ou non, dans lesquels aucune interaction avec l'environnement réel n'est possible et inversement. Cette réalité se base très généralement sur un casque *Head Mounted Display (HMD)* dont l'utilisateur doit se munir afin d'être immergé dans un monde numérique avec lequel il peut interagir. Dans la réalité virtuelle, l'immersion est une notion importante lorsqu'il s'agit de la différenciée d'un simple programme informatique.



FIGURE 2.1 – Représentation de continuum de la virtualité par Milgram et Kishino, 1995[8]

**Réalité augmentée** La réalité augmentée plus communément appelé *Augmented Reality (AR)* quant à elle est un sous domaine de la réalité virtuelle. L'idée de la réalité augmentée est de venir superposer à l'environnement réel des éléments virtuels. Ces éléments vont alors venir "augmenter" notre monde en apportant le plus souvent des compléments d'informations. Elle est donc qualifiée de sous domaine de la réalité virtuelle car l'utilisateur n'est plus immergé dans un environnement complètement numérique mais du contenu virtuel est ajouté en contexte à la vision réelle. Par abus de langage le terme de réalité augmentée est souvent utilisé pour parler de réalité mixte dont la notion est détaillé dans cette partie. Il faut noter que ce type de réalité ne se base pas uniquement sur des *HMD* mais peut être aussi apprécié à l'aide d'un téléphone par exemple (fig 2.2).



FIGURE 2.2 – Réalité augmentée vu au travers d'un téléphone<sup>1</sup>

**Réalité mixte** La réalité mixte, ou hybride, plus communément appelé *Mixed Reality (MR)*, ou *Crossed Reality (XR)*, est la fusion parfaite de l'environnement numérique et de l'environnement physique (fig 2.1). Dans ce "nouvel" environnement, les objets physiques et numériques coexistent et peuvent interagir entre eux et par exemple une table peut devenir une plateforme pour un personnage virtuel (fig 2.3). Souvent confondu avec la réalité augmentée, cette dernière se différencie car elle ne propose pas seulement une visualisation des objets numériques, elle propose aussi des méthodes d'interactions avec ce contenu et c'est cette notion d'interaction qui permet de la différencier. A l'heure actuelle la réalité mixte nécessite un dispositif de type *HMD* pour être appréciée comme par exemple l'*HoloLens* de Microsoft.

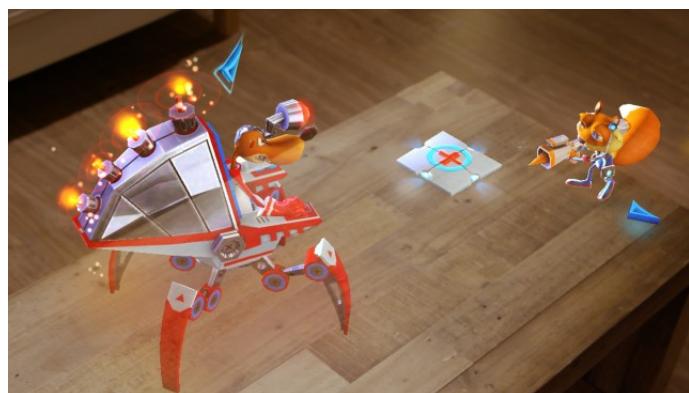


FIGURE 2.3 – Asobo Studio™- Young Conker©

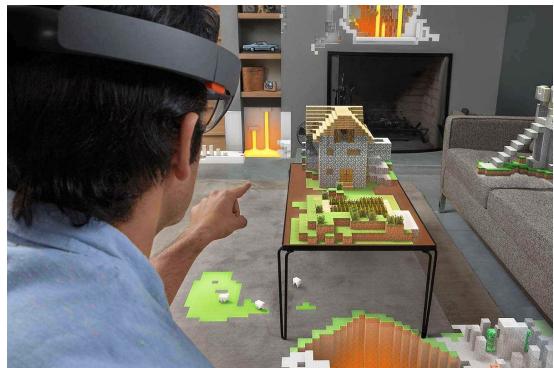
**Réalité augmentée vue au travers** La réalité augmentée vue au travers, plus communément appelé *See Through Augmented Reality (STAR)* est une technique de visualisation de la réalité augmentée où les éléments numériques sont vu au travers d'un écran (fig 2.4a) ou d'un *HMD* (fig 2.4b). C'est le type de visualisation le plus utilisé actuellement. L'un des défauts majeur de ce type de visualisation est que la plus part du temps, chaque utilisateur a besoin de son propre écran ou casque pour pouvoir en profiter pleinement ce qui limite grandement les expériences collaboratives. Aussi les principaux défauts liés aux écrans s'appliquent aussi, a savoir fatigue visuel etc.

---

1. Source : <https://www.engadget.com>



(a) Pokémon GO - Vue au travers téléphone<sup>2</sup>



(b) Microsoft HoloLens - Vue au travers casque<sup>3</sup>

FIGURE 2.4 – Réalité augmentée vue au travers

**Réalité augmentée spatiale** La réalité augmentée spatiale, plus communément appelé *Spatial Augmented Reality (SAR)* est une technique de visualisation de la réalité augmentée se basant sur un dispositif de projection. Les éléments virtuels qui viennent "augmenter" le monde réel sont alors projetés dans l'espace (fig 2.5), d'où le terme spatial. Cette notion d'augmentation de l'espace tend à rendre cette technologie naturellement collaborative car les projections ne dépendent pas d'un dispositif visuel personnel et sont obligatoirement partagées. La SAR permet aussi de favoriser le développement d'interface tangible, en effet, la visualisation se faisant directement sur les objets physiques, la tendance à développer des interfaces en communion avec ceux ci est très forte car très naturel.



FIGURE 2.5 – Présentation d'une voiture en utilisant la réalité augmentée spatiale<sup>4</sup>

**Interface tangible** Une interface utilisateur tangible ou *Tangible User Interface (TUI)* est une interface utilisateur via laquelle des objets physiques, ou encore le toucher, permettent de manipuler des données numériques (fig 2.6b). Les interfaces utilisateurs tangibles remplacent très souvent les interfaces utilisateur graphiques (fig 2.6a) où *Graphical User Interface (GUI)* dans la plupart des application de réalité augmentée car elles fournissent un contrôle direct à l'utilisateur sur ce qu'il souhaite manipuler (par opposition au contrôle indirect, comme la souris, nécessaire à la manipulation des GUI).

- 
- 3. Source : Pokemon GO
  - 3. Source : Microsoft HoloLens
  - 4. Source : Google Image

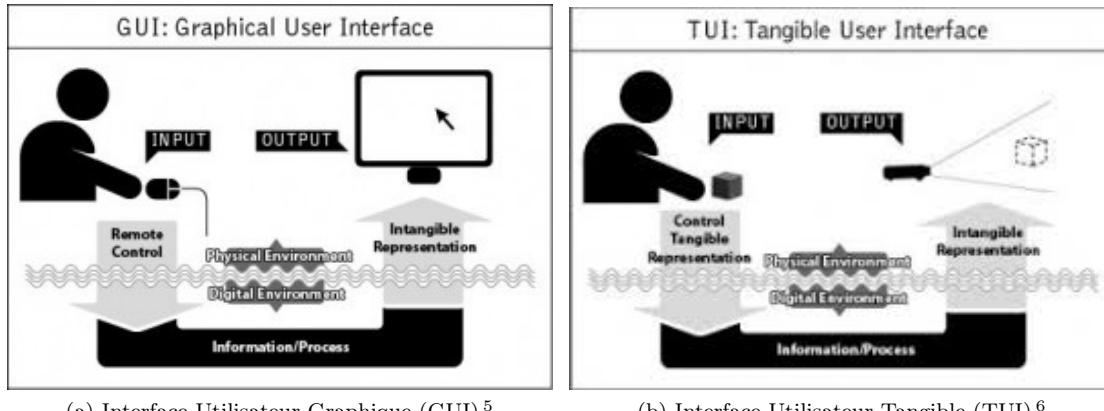


FIGURE 2.6 – Différences des interfaces utilisateurs

**Calcul haute performance** Le calcul haute performance ou *General-Purpose computing on Graphics Processing Units (GPGPU)* désigne une méthode de calcul utilisant la carte graphique (GPU) plutôt que le processeur (CPU). Cette technique permet de bénéficier de la puissance de la carte graphique afin de réaliser du calcul en parallèle et est très souvent utilisée pour la plupart des traitement lourd comme par exemple le rendu d'une scène 3D, l'encodage de vidéo, les simulations physiques (particules) etc. Cette technique repose sur le grand nombre de coeurs présent dans les cartes graphiques (contrairement aux processeurs) et sur la capacité de chacun de ces coeurs à effectuer des opérations simples de manière très efficace. Le calcul haute performance ne peut cependant pas se passer du CPU qui va être principalement utilisé pour récolter et transférer les données traitées ou à traiter.

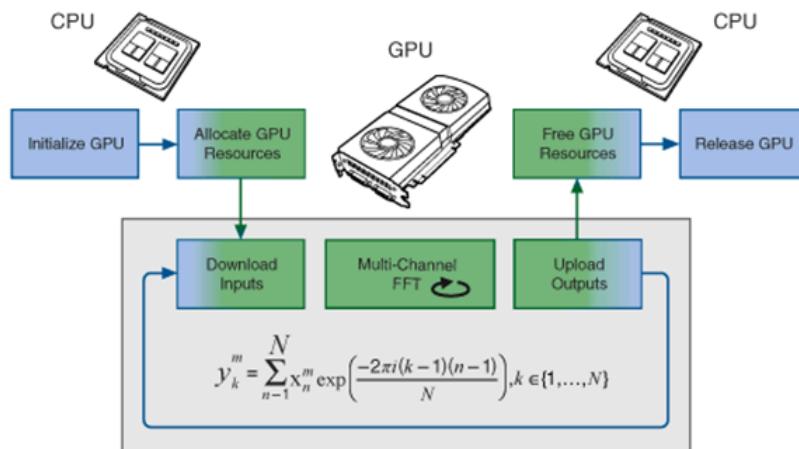


FIGURE 2.7 – Exemple de calcul de la FFT sur GPU<sup>7</sup>

6. Source : Icon Library - From GUI to TUI

6. Source : Icon Library - From GUI to TUI

7. Source : National Instruments

# État de l'art

De nos jours, les technologies de réalité augmentée en vue au travers avec, ou non, un HMD se développent très rapidement mais souffrent encore de problèmes technologiques et ergonomiques majeurs ce qui ne leur permet pas d'être utilisées dans tous les domaines d'application de la réalité augmentée[?]. Récemment une nouvelle technologie de réalité augmentée, se détachant de l'approche traditionnelle liés aux écrans, jusqu'à alors inexploitée dans le domaine de l'informatique est entrain d'émerger sous le nom de réalité augmentée spatiale. Cette technologie apporte des réponses aux problèmes technologiques et ergonomiques établies et permet d'ouvrir un champ des possibles bien plus large[?].

## 3.1 PapART

Comme décrit dans l'introduction chapitre 1, **PapART** ou Paper Augmented Reality Toolkit se présente sous la forme d'un kit de développement permettant de créer des applications interactives en réalité augmentée de création de dessins ou de peinture (fig 3.1). L'idée est de proposer une technique numérique non intrusive pour faciliter une tâche complexe, tel que le dessin tout en permettant à l'utilisateur de s'exprimer[?].



FIGURE 3.1 – Jeremy Laviole utilisant **PapART** pour dessiner en réalité augmentée<sup>1</sup>

Le système interactif (fig 3.3) permettant d'utiliser tout le potentiel de **PapART** est très spécifique et se compose de, 2 dispositifs d'acquisitions.

Le premier est une caméra couleur observant la zone de travail dont le but est de détecter les feuilles de papier servant de base à la projection. Ces feuilles sont ornées de marqueurs ARToolKitPlus<sup>2</sup>, qui est une bibliothèque de détection de marqueurs fiduciaires<sup>3</sup>, qui ont deux utilisations. Ils permettent à la fois la détection de la feuille de papier, en effet, avec un nombre suffisants de marqueurs détectés et une connaissance a priori du modèle de la feuille il est possible d'en estimer la pose<sup>4</sup> (position et rotation dans l'espace), et à la fois la détermination du contenu de la feuille. Chaque marqueur étant unique,

1. Source : Inria - **PapART**

2. <https://github.com/paroij/ar toolkitplus>

3. Un marqueur fiduciaire est un objet placé dans le champ de vision le plus souvent de système d'imagerie qui apparaît sur l'image produite et qui va servir de point de repère ou de référence.

4. Pose : Wikipédia

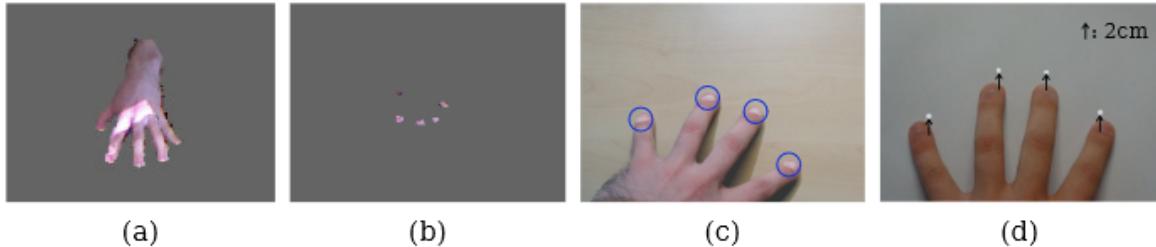


FIGURE 3.2 – PapARt - Détection des interactions de "touch". (a) Détection de la main après opération de seuillage sur le plan de la table. (b) Sélection d'une zone de seulement 1-2cm au dessus de la table. (c) Projection sur les éléments détectés. (d) Décalage de la détection pour indiquer à l'utilisateur où l'interaction a été détectée.<sup>5</sup>

lorsque PapARt détecte une feuille de papier, il récupère également les informations des marqueurs associés permettant de projeter du contenu différents en fonction des marqueurs présents.

Le deuxième dispositif d'acquisition est une caméra de profondeur dont rôle est de détecter les différents utilisateurs, les différents objets et les potentielles interactions. Grâce aux informations de profondeur, les interactions peuvent être détectées soit sur le plan de la zone de travail, ce sont des interactions qualifiées de "touch", soit dans l'espace au dessus de la zone de travail, qu'on qualifiera de "pointage 3D".

Pour finir, en plus des deux dispositifs d'acquisition, un dispositif de projection est présent pour permettre la visualisation du contenu. Son rôle est de projeter dans les zones adéquates (i.e détectées via des feuilles de marqueur) le contenu numérique désiré.

Pour que le projecteur soit capable de projeter précisément des informations dans les feuilles détectées par la caméra, une calibration très précise ainsi qu'une représentation à l'échelle est nécessaire. C'est aussi cette calibration qui permet au système d'avoir des capacités d'interaction (toucher simple, toucher multiple, balayage et autre) sensiblement similaires à celle d'une tablette tactile (fig. 3.2).

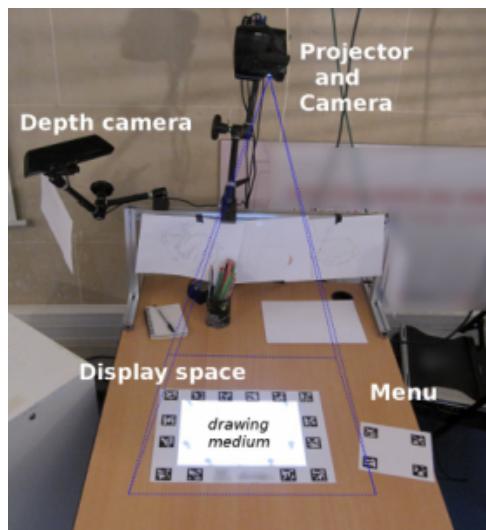


FIGURE 3.3 – Système interactif utilisant PapARt<sup>6</sup>

Au-delà des applications d'aide au dessin qui sont extrêmement nombreuses, PapARt ouvre un champ des possibles assez large. Le rôle de RealityTech est d'explorer ce champ des possibles en améliorant PapARt et en fournissant de nouveaux cas d'utilisation toujours plus innovants.

## 3.2 Systèmes de réalité augmentée spatiale

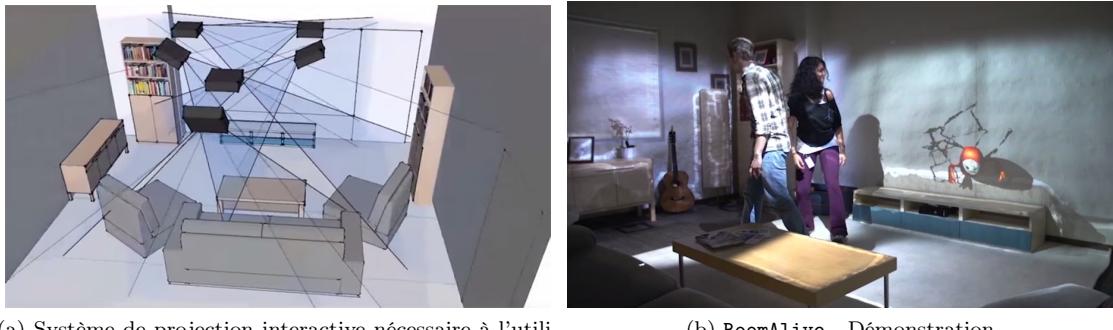
**RoomAliveToolKit** RoomAliveToolKit[7] est un projet tout droit sorti des laboratoires de recherche de Microsoft. RoomAliveToolKit est un kit de développement créé en 2013 par Nikunj Raghuvanshi, Eyal

5. Source : Interaction en Réalité Augmentée Spatiale pour le Dessin Physique - 2.3 SAR Système Multi-touch with Kinect

6. Source : Inria - PapARt

Ofek et Andy Wilson qui permet, à l'instar de PapART, de créer des expériences de projection interactive. La principale différence réside dans le fait que RoomAliveToolKit a pour but de donner vie à des pièces entière en utilisant plusieurs projecteurs et plus caméra qui fonctionne à l'unisson.

RoomAliveToolKit a permis entre autre de développer de nombreux projets basés sur de la projection interactive tel que RoomAlive, Room2Room, IllumiRoom et bien d'autre.



(a) Système de projection interactif nécessaire à l'utilisation de RoomAliveToolkit

(b) RoomAlive - Démonstration

FIGURE 3.4 – Microsoft Research : RoomAliveToolkit et RoomAlive<sup>7</sup>

**Lampix** Lampix est un système de projection interactif dont le but est de rendre n'importe quelle surface horizontal "intelligente" (fig. 3.5) développé par Goerge Popescu et Mihai Dumitrescu. Le fonctionnement aussi bien logiciel que matériel de Lampix est très similaire à celui de PapART. Lampix utilise de l'apprentissage et des techniques de visions par ordinateur pour proposer ces expériences. Étant donné que Lampix fonctionne via apprentissage, pour que le système fonctionne correctement et puisse évoluer, les créateurs de Lampix ont aussi créer la toute première blockchain<sup>8</sup> dédiée à la vision par ordinateur contenant d'énormes jeux de données d'image labellisées (image et description).

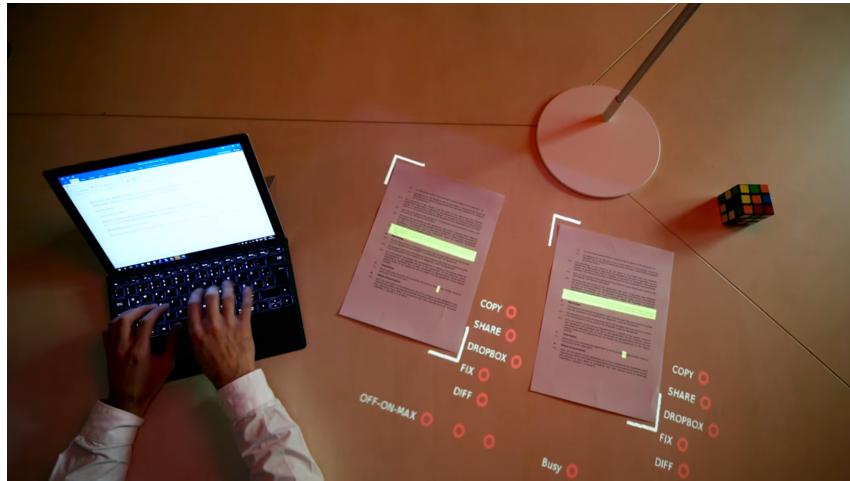


FIGURE 3.5 – Lampix - Tabletop Augmented Reality Platform

7. Source : RoomAliveToolkit

8. Wikipédia: Blockchain

# Développement d'application

Comme expliqué dans l'introduction chapitre 1 le premier objectif du stage était le développement d'applications de réalité augmentée spatiale. Il était important pour commencer, d'évaluer les possibilités mais aussi les contraintes qu'offrait le kit de développement. Ainsi un travail d'analyse et de critique de l'API à été effectué en parallèle du développement d'applications.

## 4.1 ReARTable

D'après le contexte et le public ciblé par l'entreprise, il m'a paru intéressant de développer une démonstration a but à la fois ludique et éducatif. J'ai donc choisi de recréer une Reactable[11] proposé par la société du même nom en réalité augmentée spatiale.

La Reactable est un instrument de musique électronique permettant la génération de son en direct développé depuis 2003. Présenté sous forme d'une table interactive, le son est généré via des éléments tangibles (fig. 4.1) placés a sa surface.



FIGURE 4.1 – Élément tangible utilisé pour la génération d'un élément de synthétiseur sur la Reactable<sup>1</sup>

Chaque élément tangible représente un élément de synthétiseur qu'il est possible de contrôler de plusieurs façon :

- La distance de l'élément par rapport a un autre élément. Cette propriété peut être utilisée pour contrôler, par exemple, l'interaction entre deux éléments.
- L'orientation de l'élément sur la table. Cette propriété peut être utilisée pour contrôler, par exemple, la fréquence de l'élément ce qui va avoir pour effet par exemple pour un battement de ralentir ou d'accélérer ce dernier.
- La disposition de l'élément. Cette propriété permet entre autre de combiner des éléments pour créer des nouveaux son plus riches et plus complexes.
- La position du doigt de l'utilisateur par rapport a un élément. On peut venir contrôler divers paramètre comme l'amplitude par exemple en venant faire graviter son doigt autour d'un élément. Ainsi, c'est en combinant plusieurs éléments entre eux avec différentes orientation et différentes dispositions que l'utilisateur va pouvoir peu a peu "construire" sa musique. Au delà de la détection des éléments tangible, la table est rétro éclairée et permet donc la visualisation en directe de la musique générée (fig. 4.2).

1. Source : Reactable : Elements tangibles



FIGURE 4.2 – Visualisation du son sur la Reactable<sup>2</sup>

#### 4.1.1 Besoins de l'application

Le but de l'application était de proposer une démonstration ce de qu'est capable de faire le système proposé par RealityTech et non pas de créer une simulateur de musique en direct fini reprenant tous les points de la Reactable. Un tel développement pourrai faire l'objet d'un stage entier et ce n'était pas le cas ici.

Pour être en adéquation avec l'idéologie de l'entreprise, l'interface tangible et les modes d'interactions avec la musique était le point le plus cruciale. En gardant ça en tête nous avons défini les besoins fonctionnels principaux :

- Générer du son en direct.
- Créer une représentation physique du son. Chaque son ou élément sonore devait avoir une représentation physique qui lui était associé, c'est à dire, un élément ou groupement d'éléments tangibles représentant ce son.
- Déetecter des éléments physiques représentant les éléments sonores dans une image. L'application devait pouvoir détecter dans une image de caméra les divers éléments physiques présents de façon a ce qu'ils soient utilisés pour identifier les éléments sonores.
- Identifier les représentation physique des sons. Chaque élément sonore étant représenter par un ou plusieurs éléments physique, l'application devait être capable, à partir des résultats de la détection, d'identifier et de différencier des éléments sonores entre eux.
- Modifier un élément sonore. L'application devait pouvoir contrôler certains paramètre défini a l'avance de chaque élément sonore généré. Ces paramètres ont pour but d'apporter a l'utilisateur a un niveau de contrôle supérieur lors de la création de musique en direct.
- Déetecter des événements liés au toucher. Dans le cas du contrôle d'un son, l'utilisateur peut être amener a toucher des zones interactives pour déclencher divers effets.
- Crée une visualisation basique d'un son. L'application devait proposer une visualisation du son généré pour guider l'utilisateur dans son expérimentation.

#### 4.1.2 Choix et implémentation

L'application a donc été développé avec Processing en utilisant PapARt pour la partie visualisation, détection et projection et Sonic Pi[1] pour la génération de musique en direct. Sonic Pi est un synthétiseur temps réel qui permet très facilement de générer des sons de manière cohérente. Le gros avantage de Sonic Pi est qu'il résout tout seul énormément de problèmes posé par la génération dynamique de musique comme par exemple la synchronisation des boucles, les effets d'entrée et de sortie des instruments et bien d'autre ce qui dé-complexifie énormément le processus.

Comme on peut le voir sur le schéma explicatif (fig 4.3), les éléments tangibles représentants des sons se présentent sous forme de regroupement d'éléments rond de petite taille (des aimants dans notre cas). L'idée derrière ce choix est d'encourager la manipulation d'élément physique pour garder le contenu

---

2. Source : Reactable

numérique en contexte et favoriser la création. On peut différencier deux sons en fonction du contenu du regroupement (nombre, position et couleur des éléments regroupés).

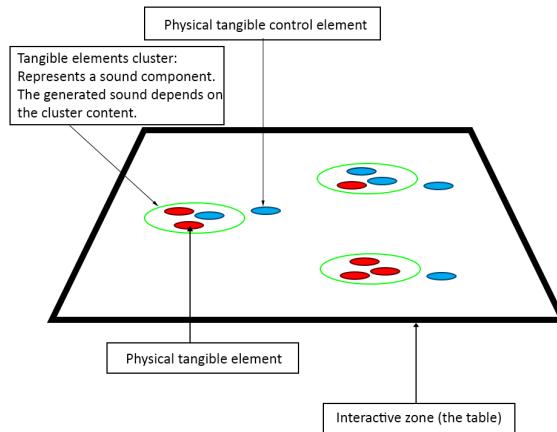


FIGURE 4.3 – ReARtable : Schéma général

Une fois les éléments détectés, regroupés et identifiés, l'élément sonore associé peut être créé. La création d'élément sonore se fait simplement via la transmission d'un message OSC<sup>3</sup> à un serveur Sonic Pi préalablement démarré. Ce message contient l'identifiant unique de la boucle que Sonic Pi doit démarrer. Pour chaque éléments sonores que l'application peut créer Sonic Pi possède une fonction à exécuter que nous avons préalablement créé. Toutes les communications entre l'application et Sonic Pi utilisent ce protocole ce qui permet de démarrer/arrêter/modifier certaines parties du son en directe.

Pour ce qui est du contrôle du son, une zone autour du composant est défini dans laquelle soit un élément tangible, soit une interaction physique (avec le doigt) vont être détecté et converti en interaction avec le contenu numérique (fig. 4.4).

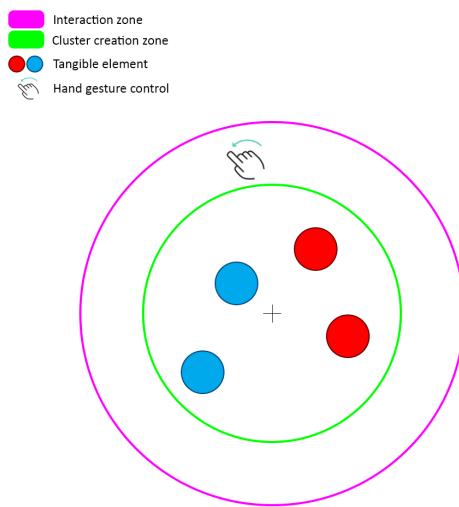


FIGURE 4.4 – Schéma représentant la création d'un son avec zone d'interaction

La dernière étape du développement de l'application était la visualisation de la musique générée (section 4.1.1). Cette étape n'a finalement pas été aboutie par manque de temps. L'idée était d'utiliser le spectre du son et les différentes fréquences qui le compose récupérable à l'aide d'une transformée

3. Le protocole OSC où OpenSoundControl est un format de transmission de donnée conçu pour le contrôle en temps réel

de fourrier<sup>4</sup> pour créer une visualisation globale basée sur les fréquences avec des variations visuels en fonction de la hauteur, du tempo du son et tous les autres paramètre du son qu'on peu extraire.

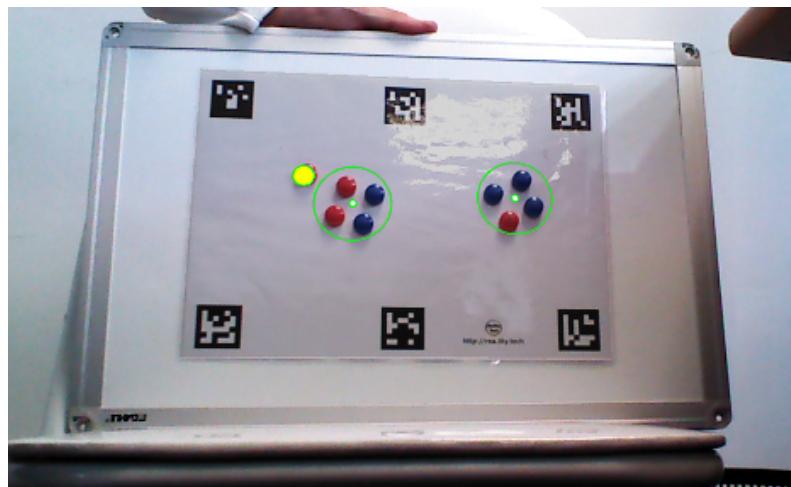


FIGURE 4.5 – Démonstration de l'application

---

4. Opération mathématique permettant de décomposer un signal en la somme des signaux qui le compose Wikipédia - Transformation de Fourier.

## 4.2 Extraction de document

Plus tard au cours de mon stage des cas d'utilisation où l'extraction et la numérisation de document ont été abordé comme par exemple l'écriture d'une lettre en réalité augmentée et ainsi il était judicieux de développer une preuve de concept de cette fonctionnalité sous la forme d'une application de SAR.

Le but de ce développement était d'expérimenter diverses techniques de détection de document en temps réel se basant ou non sur des connaissances à priori comme la taille du document, sa couleur, la couleur du fond (duquel il falloir extraire le document), la présence éléments distinctifs (comme des marqueurs fiduciaires ou des ronds colorés de petite taille).

### 4.2.1 Besoins de l'application

- Accéder au flux vidéo d'une caméra. L'application devait avoir access au flux vidéo d'une caméra filmant le document à détecter.
- Détecter un document dans une image. Des images extraites du flux vidéo, l'application devait être capable, avec ou sans connaissance a priori, de détecter un document se trouvant dans cette image.
- Extraire un document d'une image. Grâce au résultat de la détection, l'application devait être capable d'extraire ce document de l'image afin d'obtenir une image ne contenant que le dit document.

### 4.2.2 Choix et implémentation

La détection de document est un problème connu en traitement d'image sur lequel j'avais déjà eu l'occasion de travailler lors de mon projet de fin d'études durant le deuxième semestre de mon année de Master 2.

Dans cette application nous avons mis en place plusieurs détection différentes pour essayer de trouver une solution a ce problème.

**Détection de document basé sur des marqueurs colorés** Le premier prototype de détection utilise une connaissance a priori sur le document : Le document cible est muni de lignes d'éléments ronds colorés de petite taille dans un ou plusieurs de ses coins (fig 4.6a). Les éléments rond de petit taille sont détectés grâce a PapARt en réalisant une convolution de l'image par un filtre permettant la détection de ces derniers.

Une fois les éléments détectés, ils sont regroupés en différentes lignes (fig 4.7b). Une ligne est défini par un regroupement d'éléments dont l'écart entre chaque éléments ne dépasse pas une certaine distance verticale ou horizontale. L'angle de la ligne est défini par les  $x$  premiers éléments qui la compose,  $x$  étant le nombre minimum d'éléments composant une ligne (fig 4.7). Si un autre élément "dérive" il est rejeté et la ligne est créée. Cette ligne est ensuite utilisé pour calculer deux vecteurs, dont un est confondu avec la ligne et le deuxième est perpendiculaire au premier (fig 4.6c).

La détection finale du document se fait en calculant l'intersection des différents vecteurs verticaux et horizontaux (fig 4.6d)

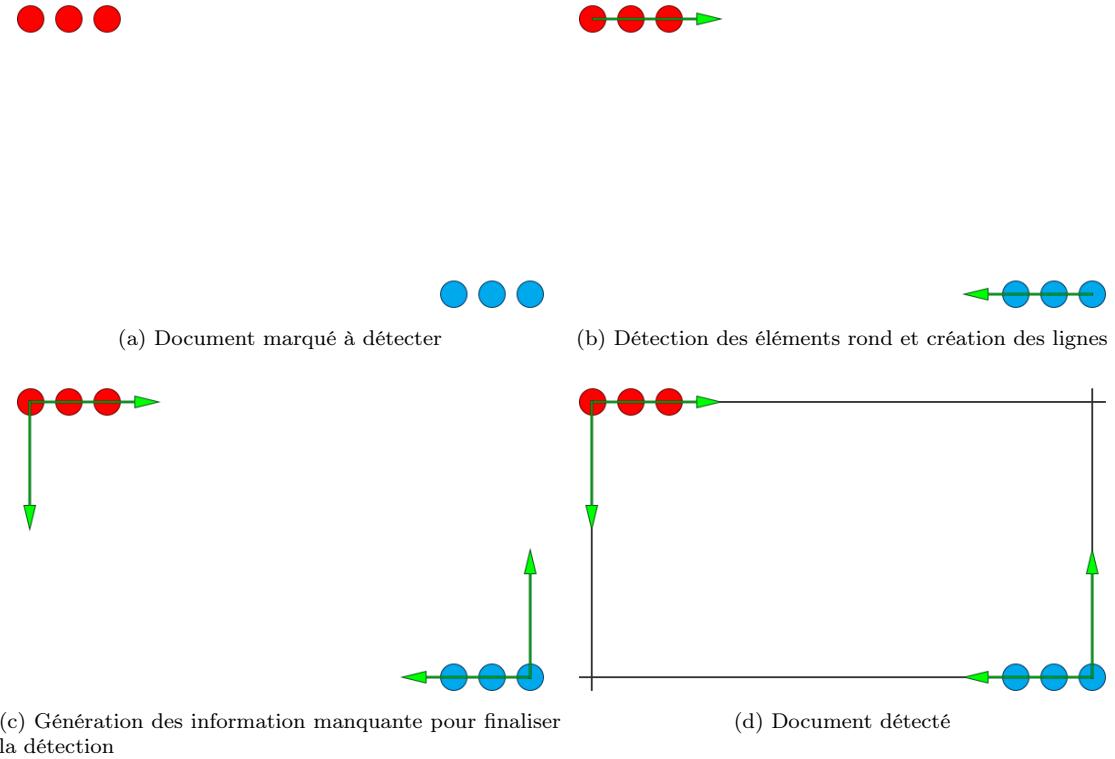


FIGURE 4.6 – Détection de document étape par étape

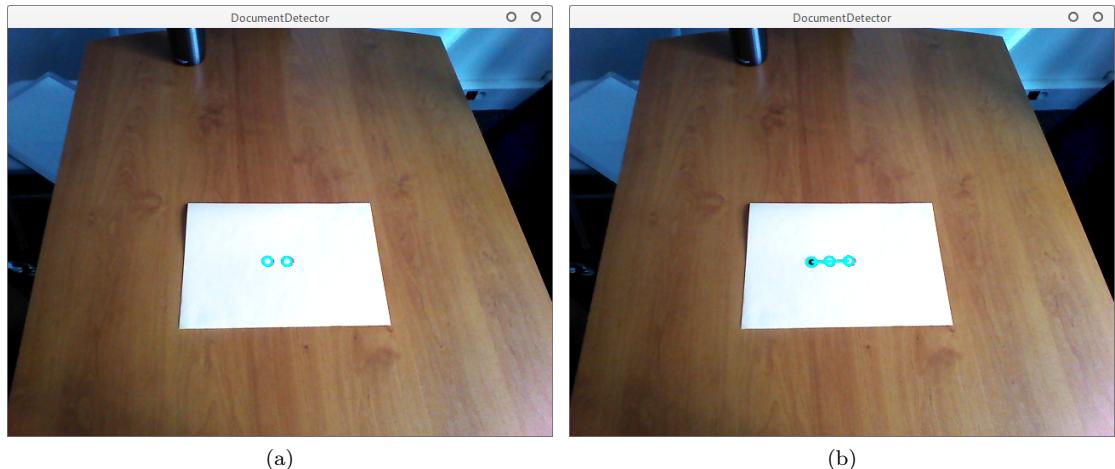


FIGURE 4.7 – Détection et création d'une ligne à partir d'éléments colorés

Comme on peut le voir figure 4.6d, lors de cette détection les bords du document sont rognés. Généralement ces zones ne contiennent aucune information car elles correspondent généralement aux marges verticales et horizontales présentes dans la plus part des documents. Cependant comme évoqué dans les cas d'utilisation, cette détection ne sera pas utiliser seulement pour des documents de type A4, on pourra s'en servir comme outil pour suivre une feuille de papier à la manière des marqueurs ARToolkit, ou pour détecter des post-it par exemple ainsi ces marges ne peuvent pas être ignorées car elle peuvent contenir du contenu potentiellement critique du document.

Une simple connaissance a priori de la distance des éléments rond par rapport au coin permet de résoudre ce problème ou alors pour avoir une détection bien plus précise et robuste, il est possible d'utiliser des algorithmes de détection de contour couplé a des algorithmes d'extraction de lignes pour retrouver le vrai coin du document. C'est ce dont nous allons parler dans la deuxième partie.

**Détection de document - Canny et transformée de Hough** Sans connaissance a priori la détection de document devient un problème compliqué et bien connu du monde de l'informatique surtout lorsqu'il

un besoin de temps réel vient s'ajouter la tâche, comme par exemple, dans une application de scan de document.

L'algorithme de Canny[2] est un filtre de détection de contour permettant d'extraire d'une image des contours (fig. 4.8b) très précis respectant trois critères : La bonne détection, la bonne localisation et la clarté de la réponse. Ces trois critères en font un très bon choix dans le cadre de la détection de document ou la qualité et surtout la précision de la détection est importante pour ne pas rogner des bouts de document par exemple. Cet algorithme a cependant tendance à laisser beaucoup de bruit issue de faible contour tout de même détectés c'est pourquoi il faut effectuer une étape de floutage préalable afin de lisser les zones à faible contour.

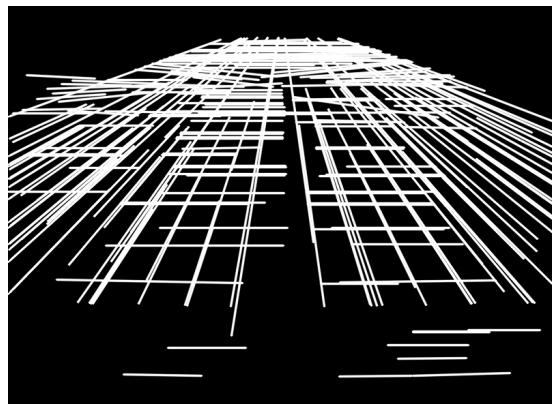
Une fois les contours détectés, il est possible d'essayer d'extraire directement le document mais c'est une tâche plus complexe, car elle requiert d'analyser les contours, qu'il est possible de faciliter en utilisant la transformée de Hough[3]. La transformée de Hough permet d'extraire n'importe quelle forme à partir d'une image contour en utilisant les propriétés mathématiques de la forme. Dans notre cas où nous souhaitons extraire des lignes droites les propriétés mathématiques utilisées correspondent aux coordonnées polaires considérés comme plus robuste que l'équation de la droite (fig. 4.8c)



(a) Image originale<sup>5</sup>



(b) Canny - Détection de contours<sup>6</sup>



(c) Transformée de Hough - Détection de lignes<sup>7</sup>

FIGURE 4.8 – Détection de lignes : Canny + Hough

Il ne reste qu'à filtrer les lignes pour trouver des potentiels documents dans une image.

Cette succession de traitement est cependant lourde et peut difficilement être effectué en temps réel sur des images de haute résolution. Dans notre cas, nous nous sommes servis de ces algorithmes seulement sur des parties d'image (de petite résolution) de façon à améliorer une première détection grossière effectuée par exemple, à l'aide de marqueurs colorés. Une fois la première détection effectuée, nous obtenons une position plus ou moins précise des quatre coins nécessaires à l'extraction du document. Nous utilisons cette information sur la position potentiel des coins pour extraire dans des sous-images centrées sur ces coins (fig. 4.9a). Ensuite nous appliquons les deux algorithmes mentionnés plus tôt, à savoir, la détection de contour (fig. 4.9c) puis la détection de ligne (fig. 4.9d). Nous filtrons ensuite le résultat de la détection de ligne pour obtenir exactement une ligne verticale et une ligne horizontale. Une fois ces deux lignes trouvées, nous calculons les équations de droite (pente, et constante) associées pour pouvoir en calculer l'intersection et finalement trouver le coin dans cette sous-image (fig. 4.9e).

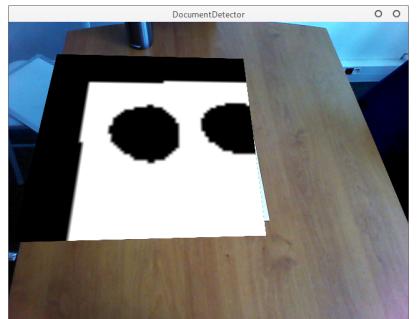
7. Source : <http://funvision.blogspot.com/2016/01/hough-lines-and-canny-edge-sobel.html>

7. Source : <http://funvision.blogspot.com/2016/01/hough-lines-and-canny-edge-sobel.html>

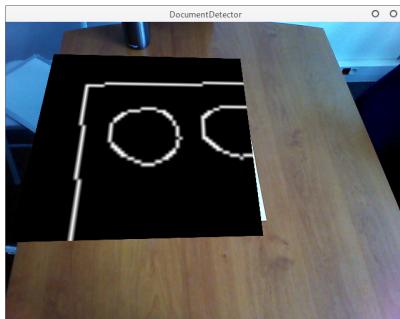
7. Source : <http://funvision.blogspot.com/2016/01/hough-lines-and-canny-edge-sobel.html>



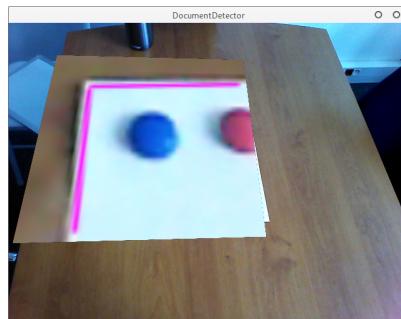
(a) Sous image autour d'un coin.



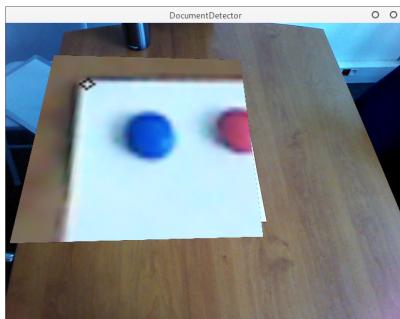
(b) Seuillage de l'image originale



(c) Canny - Détection de contour basée sur un seuillage de l'image originale



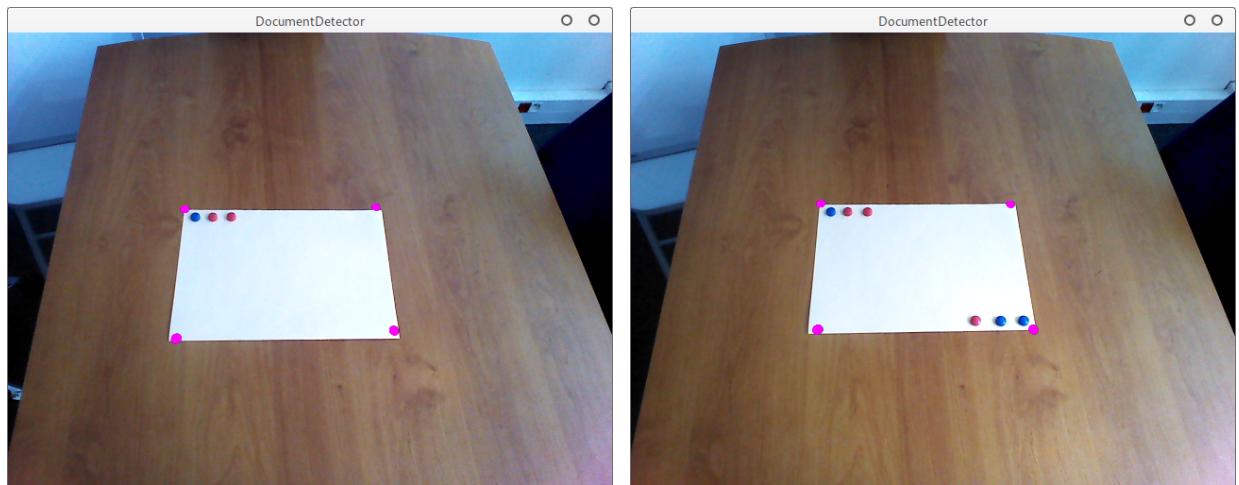
(d) Hough - Détection de lignes basée sur le résultat de Canny



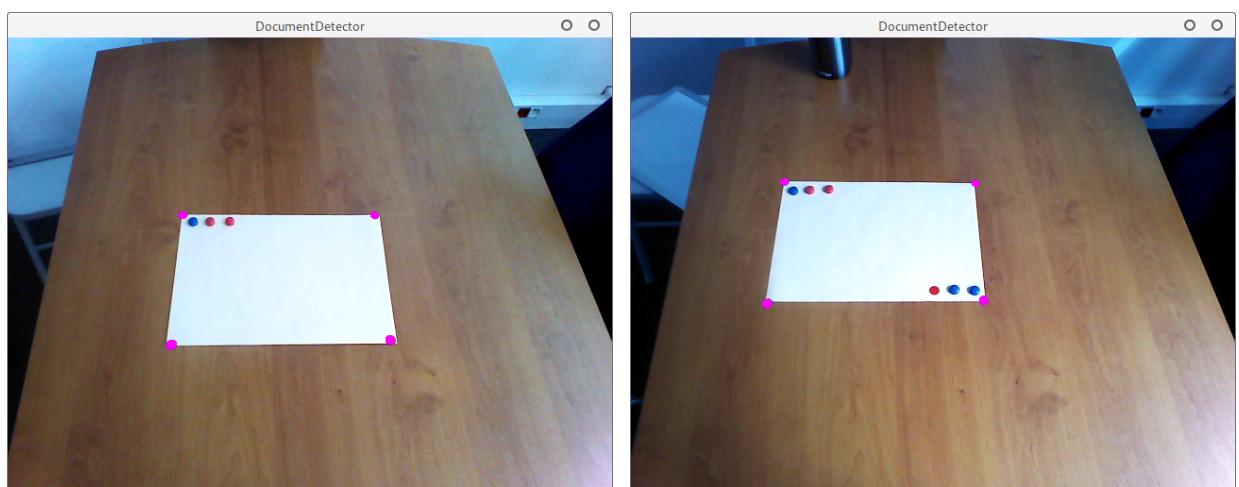
(e) Détection finale du coin.

FIGURE 4.9 – Affinage de la détection d'un coin du document

**Bilan** Pour finir, vous trouverez Figure un rapide comparatif des différents résultats obtenus. On peut observer quelques points notables : En utilisant Hough et Canny pour améliorer la détection d'un coin, l'extraction est plus fidèle au document réel. En effet, les coins détectés sont plus précis que des hypothèses effectuées à priori. Cette méthode est cependant moins rapide car elle requiert de nombreux calculs en plus. Lorsque utilisé en temps réel, on peut cependant observer que la détection avec connaissance à priori du modèle est bien plus robuste, car elle ne dépend pas d'une deuxième détection qui a beaucoup de chance d'échouer (seuillage, détection de contour, détection de ligne, intersection entre deux droites puis obtention finale du coin) ainsi la méthode a utiliser variera avec les cas d'utilisations. Par exemple dans le cadre d'une application de scan de document, une méthode d'extraction de document plus précise sera envisagée, mais dans le cadre d'une estimation de pose il sera préférable d'utiliser une détection rapide et robuste.



(a) Avec connaissance à priori : Taille du document et distance élément ↔ coin      (b) Avec connaissance à priori : Taille du document et distance élément ↔ coin



(c) Avec connaissance à priori : Taille du document

(d) Sans connaissance à priori

FIGURE 4.10 – Résultats des différentes version de la détection de document.

# Prototype haute performance

Dans le cadre d'application comme celui de la réalité augmentée spatiale où les interactions jouent un rôle majeur dans l'expérience de l'utilisateur la réactivité, la fluidité de l'expérience et la latence générale du système sont des points cruciaux qu'il est impossible de négliger. Pour pouvoir atteindre ces objectifs et pouvoir pousser les applications encore plus loin aussi bien dans l'interaction, dans le rendu ou dans le contenu, sans avoir besoin d'une puissance de calcul dépassant l'entendement, une optimisation aussi bien logicielle, matérielle ou architecturale est nécessaire. Cette optimisation a fait l'objet d'une grande partie de mon stage qui m'a amener à développer un prototype dit "haute performance" des outils que propose RealityTech. L'optimisation logicielle a été porté sur l'amélioration des performances des algorithmes de traitement d'image bien connus pour être extrêmement consommateur des ressources. Pour l'optimisation matérielle, la tâche a était un peu différente, nous nous sommes attelé à effectuer des mesures et des calculs sur la puissance théorique du matériel, la latence réel des caméra ou encore la rapidité de l'encodage des flux vidéos pour arriver à établir les performances réelles qu'il nous était possible d'atteindre avec différentes combinaison de matériel. Le développement du prototype s'est achevé avec la création d'une nouvelle architecture logiciel en micro services[?] dans le but de créer un environnement modulaire réactif où les services peuvent mourir sans mettre en péril tout le système et ainsi améliorer grandement la qualité général des outils fournis.

## 5.1 Amélioration logiciel

De nos jour, les optimisations font l'objet de développements ciblés et très spécifique, se concentrant la plupart du temps sur l'amélioration d'un unique point cruciale d'un algorithme ou d'une application. Dans notre cas l'optimisation logiciel a surtout été effectué au niveau des algorithmes de traitement d'images omniprésents et indispensables a la technologie. La réalité augmentée spatiale a besoin du monde réel pour exister c'est pourquoi le matériel dispose de nombreux capteurs (caméras) pour l'analyser et que de nombreux algorithmes de traitement des données captés (images) sont mis en place. Après une rapide analyse du logiciel, il est indéniable que traitement le plus utilisé est la convolution d'une image par un filtre qui possède un nombre incalculable d'application et c'est pourquoi nous avons choisi de concentrer nos efforts sur l'optimisation de ce dernier.

### 5.1.1 Convolution - Théorie

*En mathématiques, le produit de convolution est un opérateur bilinéaire et un produit commutatif, généralement noté \*, qui, à deux fonctions  $f$  et  $g$  sur un même domaine infini, fait correspondre une autre fonction  $f * g$  sur ce domaine, qui en tout point de celui-ci est égale à l'intégrale sur l'entièreté du domaine (ou la somme si celui-ci est discret) d'une des deux fonctions autour de ce point, pondérée par l'autre fonction autour de l'origine — les deux fonctions étant parcourues en sens contraire l'une de l'autre (nécessaire pour garantir la commutativité).<sup>1</sup>*

Dans le cadre du traitement d'image, le produit de convolution représente une technique de filtrage d'image visant à accentuer ou atténuer certaines caractéristiques ce celle ci comme la netteté, le flou ou les zones de fort gradient (les contours) par exemple (fig 5.1). Étant donné que nous travaillons avec des images définie par un nombre fini de pixels, la convolution d'une image est réalisée dans le domaine discret où  $f$  et  $g$  dans la définition mathématique représentent respectivement une image et le filtre qu'on souhaite lui appliquer. Le résultat de cette convolution est une nouvelle image.

---

1. Source : Produit de convolution - Wikipedia

On appelle filtre, ou noyau de convolution, une image (ou une matrice) généralement de petite taille définie en amont qui va être utilisée pour calculer la nouvelle valeur de chacun des pixels de l'image résultat. C'est la définition de ce dernier qui va décider du traitement appliquer à l'image.

Le calcul de la valeur d'un pixel dans l'image résultat se fait de la manière suivante : Le voisinage autour du pixel dont on souhaite calculé la valeur est pondéré par le filtre de convolution que l'on aura préalablement centré sur ce pixel. La nouvelle valeur du pixel représente la somme de toutes les valeurs précédemment calculées (voir algorithme 1).

Sur la figure 5.2 on souhaite calculer la nouvelle valeur du pixel positionné en 3,3 dans l'image d'origine (I). On sélectionne donc un voisinage de même taille que le filtre (K) centré sur ce pixel dont chaque élément va être multiplié par la valeur du filtre pour calculée la valeur du pixel 3,3 dans la nouvelle image soit :

$$I_{3,3} * K = 88 * 1/9 + 21 * 1/9 + 25 * 1/2 + 68 * 1/9 + 14 * 1/9 + 15 * 1/9 + 35 * 1/9 + 52 * 1/9 + 10 * 1/9 = 36$$

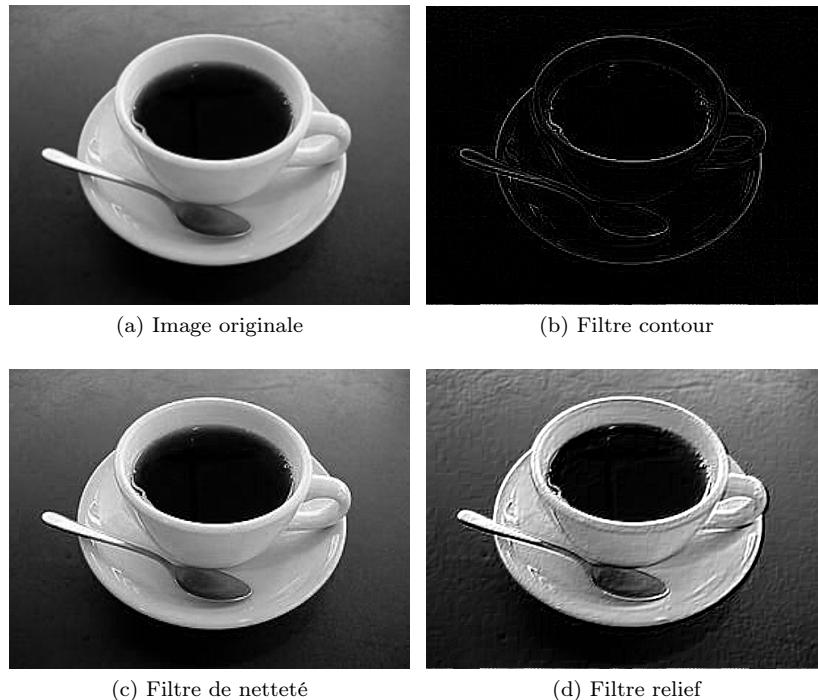


FIGURE 5.1 – Différentes filtres de convolution appliquée à une image.

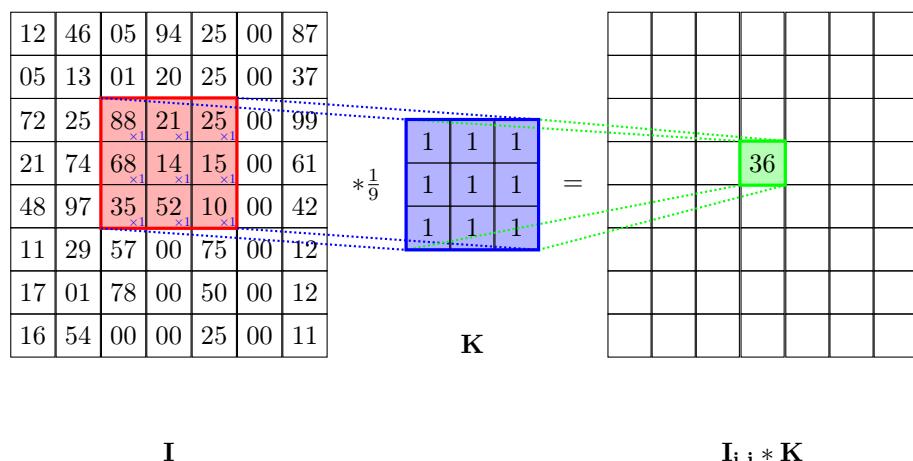


FIGURE 5.2 – Convolution d'une matrice (image) ( $I$ ) par un filtre ( $K$ )

---

**Algorithm 1** Convolution d'image image par un filtre

---

```
procedure CONVOLUTION(I, K, Iw, Ih, Ks)                                ▷ I : image, K : filtre
     $I_{conv} \leftarrow I$ 
     $Khs \leftarrow floor(Ks \div 2)$ 
     $x \leftarrow 0, y \leftarrow 0$ 
     $sum \leftarrow 0$ 
    for  $x \leq Iw; ++x$  do
        for  $y \leq Ih; ++y$  do
            for  $i \leq Ks; ++i$  do
                for  $j \leq Ks; ++j$  do
                     $pos_x \leftarrow x + i - Ksh$                                 ▷ Pour tous les pixels  $x, y$ 
                     $pos_y \leftarrow y + j - Ksh$                                 ▷ Pour chaque éléments dans une fenêtre de taille  $Ks$ 
                    if  $outOf(I, pos_x, pos_y)$  then                                ▷ pos = pos + position dans le voisinage
                        continue
                    end if
                     $sum \leftarrow sum + I_{pos_x, pos_y} * Ki, j$                   ▷ pos = pos + position dans le voisinage
                     $maskSum \leftarrow maskSum + Ki, j$                             ▷ Vérifie que les positions sont dans l'image (bords)
                end for
            end for
             $I_{conv}[x, y] \leftarrow sum \div maskSum$                           ▷ Somme du voisinage par le filtre
        end for
    end for
    return  $I_{conv}$                                                  ▷ Valeur final = somme normalisée
end procedure
```

---

Comme on peut s'en rendre compte dans le pseudo code proposé (algo 1), l'image résultat est une nouvelle image, indépendante de l'image d'origine dont chaque pixel est calculé indépendamment de ces voisins dans cette nouvelle image. Cela signifie que n'importe quel pixel peut être calculé dans n'importe quel ordre. C'est précisément à cette propriété que nous allons nous intéresser car en théorie, avec une puissance de calcul suffisante il est possible de calculer en même temps tous les pixels de l'image résultat. Cet algorithme possède donc un très fort potentiel d'optimisation car il est très largement parallélisable.

### 5.1.2 Convolution - Optimisation

L'optimisation de cet algorithme peut se faire de deux façon bien distinctes. La première se fait en utilisant la puissance de la carte graphique de l'ordinateur pour effectuer énormément de calculs en même temps. C'est l'optimisation sur carte graphique dont nous avons évoqué le principe chapitre 2. La seconde méthode d'optimisation consiste à légèrement changer l'algorithme de convolution : la convolution est séparé en deux filtres distincts[10], un horizontal et un vertical qui sont successivement appliqués à l'image origine (fig 5.3). Ainsi la complexité d'appliquer une convolution de taille  $M \times M$  à une image de taille  $N \times N$  est réduit de  $O(N^2 M^2)$  à  $O(N^2 M)$ .

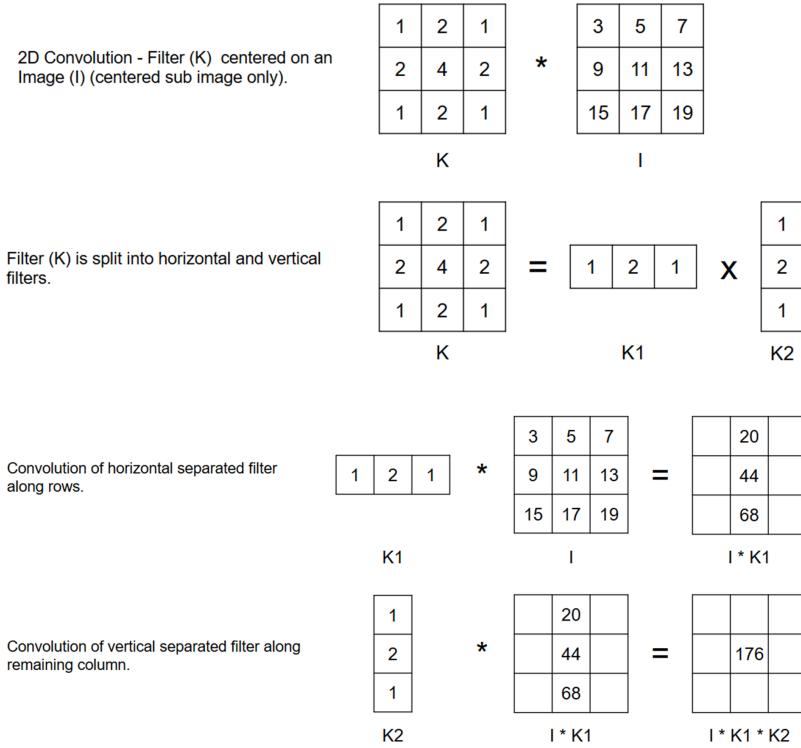


FIGURE 5.3 – Exemple d'un filtre de convolution appliqué successivement horizontalement puis verticalement

Nous avons choisi de n'effectuer que l'optimisation sur carte graphique car la deuxième méthode comporte un gros coût en complexité et en temps de développement que nous n'avons pas jugé nécessaire d'inclure dans cette première version. De plus il n'est pas toujours possible de séparer un filtre  $K$  en 2 sous filtres  $K1, K2$  tel que  $K = K1 \times K2$ .

Pour pouvoir développer la dite optimisation, il a fallut utiliser un langage de programmation sur carte graphique. De nos jours, il en existe plusieurs et ils possèdent tous leurs spécificités cependant pendant la phase de recherche, 3 langages (ou sous langages) se sont démarqués : OpenCL[4], OpenGL ES[5] et CUDA[9]. Nous avons donc choisi d'implémenter 3 version de l'algorithme de convolution naïf (non séparé) utilisant chaque de ces langages et d'en évaluer les performances.

**OpenCL** OpenCL ou *Open Computing Language* est un langage de programmation basé sur le C créé par Khronos Group en 2009. Un programme OpenCL s'écrit en deux parties : La partie **code hôte** et la partie **noyau ou code périphérique** qui représentent respectivement la partie application se chargeant d'orchestrer les différentes tâches, la gestion mémoire, la gestion des périphériques s'exécutant sur l'hôte et la partie calcul permettant de compléter les dites tâches s'exécutant sur les périphériques. La partie hôte est écrite en C tandis que la partie noyau est écrit en OpenCL-C. Il faut donc aussi différencier hôte et périphérique (fig 5.4). Dans notre cas d'utilisation l'hôte représente le processeur et permet de transmettre les données au périphérique qui dans notre cas correspond à une ou plusieurs cartes graphiques.

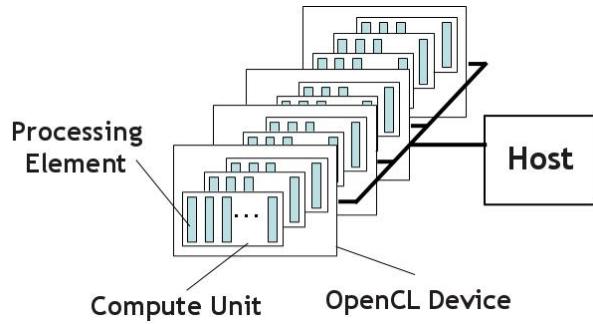


FIGURE 5.4 – Schéma OpenCL - Hôte et périphériques<sup>2</sup>

Nous nous sommes intéressé à OpenCL car il est compatible avec la plupart des systèmes et des architectures aujourd’hui présents sur le marché sans aucune modification de code nécessaire. Cet avantage est aussi l’un de ses plus gros inconvénients car il ne permet pas d’exploiter au mieux chaque architecture comme peut le faire CUDA avec NVIDIA, et les performances de ce dernier ne sont donc pas équivalentes sur chaque architecture.

**OpenGL (ES)** OpenGL est une interface de programmation multi plateforme et multi langage permettant faire le rendu de scène 2D/3D. En tant qu’interface il est possible de l’implémenter de façon logiciel mais elle a été conçue pour être implémentée de manière matérielle avec de profiter au mieux des accélérations matérielles disponibles. Ainsi c’est grâce à ces implémentations qu’OpenGL fournit un *pipeline* programmable de rendu ultra performant. C’est via ce pipeline programmable et plus spécifiquement via le code hôte et les shaders qu’il est possible de transmettre des instructions et des données à la carte graphique (fig. 5.5)

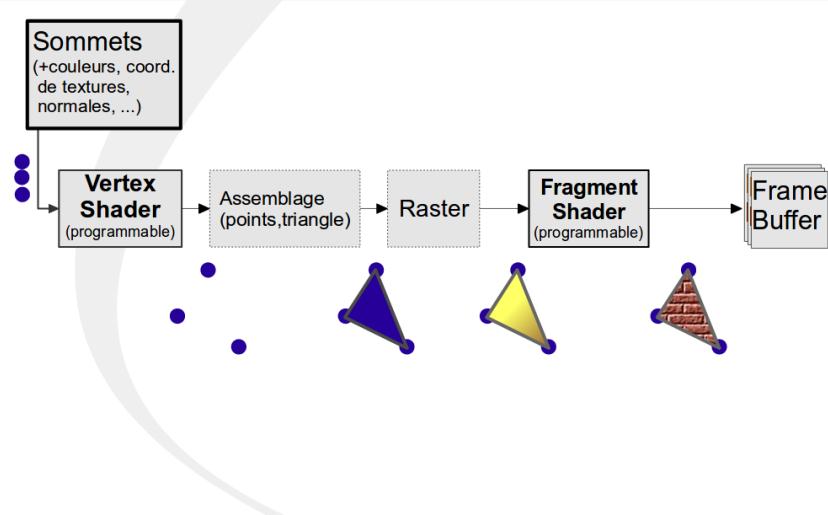


FIGURE 5.5 – Pipeline de rendu - OpenGL<sup>3</sup>

Le pipeline OpenGL reçoit en entrée :

- Des informations sur la géométrie de la scène.
- Des paramètres nécessaires pour effectuer le rendu de la scène. Point de vue de la caméra, lumières, textures, matériaux.

et donne en sortie une image de la scène.

Pour pouvoir utiliser ce pipeline dans le but d’opérer des traitements sur des images 2D, il est nécessaire d’en détourner l’utilisation. Sans géométrie à fournir au vertex shader, le pipeline de rendu ne se déclenche pas. L’idée pour passer outre est de créer un bout de géométrie recouvrant l’écran, le plus souvent un quad, afin d’activer le pipeline. Une fois le pipeline activé, le vertex shader est programmé pour ne rien faire et ainsi les étapes d’assemblage et de rastérisation sont très rapidement passées pour arriver à l’étape du rendu par fragment. C’est dans ce shader que se compose l’image de sortie du rendu et c’est ici que nous avons accès à tous les pixels de l’image. Le calcul de la convolution se fait donc pour chaque

2. Source : <https://www.anandtech.com/show/7334/a-look-at-alteras-opencl-sdk-for-fpgas/>

3. Source : Cours M1 Informatique - Mondes 3D - Pierre Benard

pixel grâce au code présent dans le fragment shader. Une fois le traitement par fragment effectué, l'image résultat est stockée dans le *Frame Buffer Object ou FBO* et peut être récupérée depuis l'hôte.

L'avantage de cette technique est que comme OpenCL, OpenGL (ES2) est largement compatible sur toutes les plateformes et très largement utilisé. Cependant, contrairement à OpenCL, les performances d'OpenGL ne dépendent que du matériel et ne varieront pas ou que très peu d'un système à un autre.

**CUDA** CUDA, contrairement à OpenCL et OpenGL n'est pas seulement un langage de programmation mais bel et bien une architecture de traitement parallèle développée par NVIDIA dont l'unique but est d'exploiter la carte graphique au maximum pour offrir une énorme puissance de calcul au système l'utilisant. Pour ce qui est de la partie programmation, NVIDIA fournit une API permettant d'utiliser cette architecture, CUDA C, et qui fonctionne de façon similaire à OpenCL avec du code hôte et du code périphérique qui seront les noyaux CUDA à exécuter sur la carte graphique. La différence CUDA se démarque c'est dans le modèle qu'il propose, les tâches ou *threads* sont regroupés en blocs ou *blocks* à l'intérieur desquels la mémoire est partagée et où chaque bloc s'exécute sur exactement une unité de calcul (fig. 5.6), la mémoire est unifiée (fig. 5.7), les CPUs et les GPUs ont accès à la même mémoire sans besoin des copies ce qui permet de réduire énormément les temps de transfert des données.

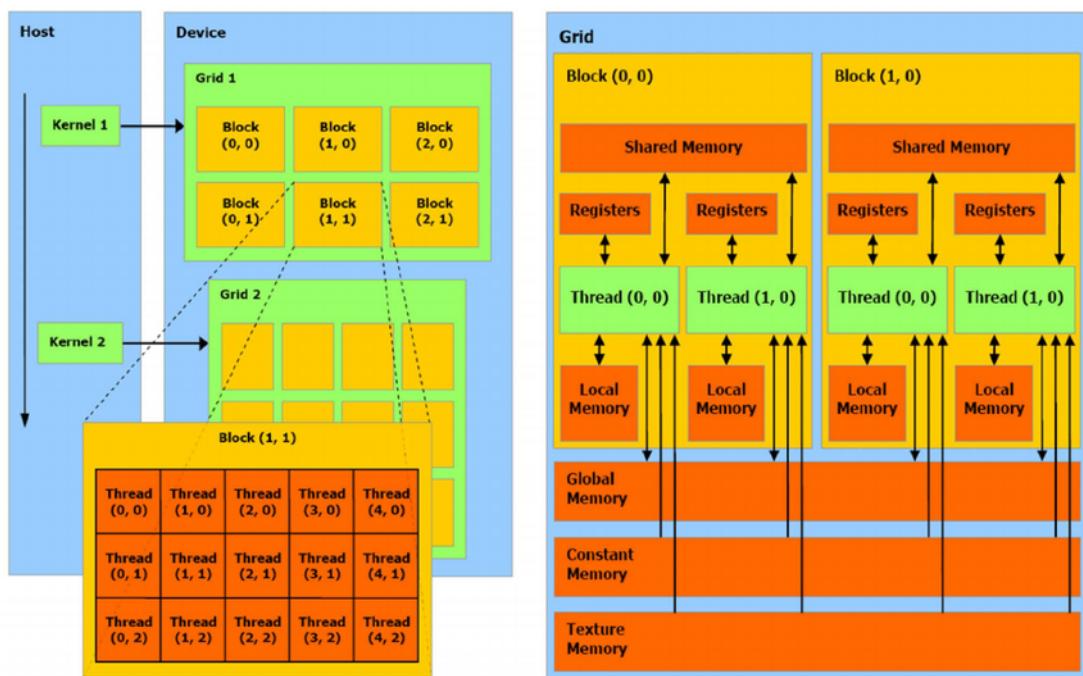


FIGURE 5.6 – Représentation schématique de l'architecture CUDA<sup>4</sup>

## UNIFIED MEMORY

Dramatically lower developer effort

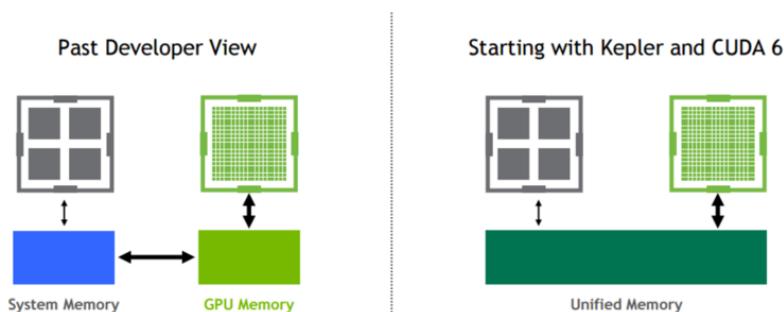


FIGURE 5.7 – Mémoire séparée vs mémoire unifiée<sup>5</sup>

4. Source : NVIDIA CUDA - Unified Device Architecture

5. Source : NVIDIA CUDA - Unified Memory for beginners

CUDA étant une architecture matériel seul les cartes graphique NVIDIA récentes en sont équipé ce qui contrairement aux deux autres ne le rend absolument pas multiplateforme.

### 5.1.3 Tests de performances

Afin de comparer les différentes solutions nous avons réaliser des tests de performances du même algorithme de convolution que nous avons implémenté dans les différents langages cités et sur différentes machines avec des configuration bien différentes. Le but de ces tests était dans un premier temps d'observer l'impacte de l'optimisation et dans un second temps d'orienter le choix d'ordinateur à inclure dans le système de RealityTech en se basant sur les résultats obtenues en fonction des différentes plateforme.

A noter que l'algorithme de convolution implémenté est la version non séparé ou le filtre de convolution est considéré comme une matrice.

Dans ce test de performances, nous avons mesurer plusieurs choses :

- **Le temps de transfert** des données de l'hôte au périphérique. Cette mesure est importante car elle permet d'évaluer l'impact de la mémoire unifiée CUDA par rapport aux autres méthodes ne possédant pas cette fonctionnalité.
- **Le temps de calcul** brut de la convolution de l'image. Afin d'évaluer les performances brut de l'algorithme par langages, nous avons mesurer les temps de calcul de chacun.
- **La "bande passante"** du traitement entier, comprenant transfert calcul et re transfert. Cette mesure donne une bonne idée de la rapidité des algorithme car elle exprime le nombre de mégaoctet qu'il est possible de traiter en une seconde avec chaque implémentation.

**Note :** Les résultats donnés Tableau 5.1, 5.2, 5.3 ont été mesuré sur le kit de développement NVIDIA Tegra Jetson TX2 dont les spécificité sont les suivantes : CPU ARM Cortex-A57 (quad-core) @ 2GHz + NVIDIA Denver2 (dual-core) @ 2GHz, GPU 256-core Pascal @ 1300MHz, RAM 8GB 128-bit LPDDR4 @ 1866Mhz | 59.7 GB/s Nous avons choisi de montrer ici seulement les résultats obtenues sur le kit de développement NVIDIA car il permet d'observer les performances de CUDA dans un environnement ultra optimisé pour ce dernier cependant vous trouverez Annexe 6.2 les résultats de ces mêmes test sur plusieurs ordinateurs différents.

TABLE 5.1 – CUDA 9.0 - Convolution d'une image en niveau de gris par un filtre de taille 5x5 - float 32bits

| Size             | Size (MB) | Compute Time (ms) | Transfer Time (ms) | Total Time (ms) | Bandwidth (MB/s) |
|------------------|-----------|-------------------|--------------------|-----------------|------------------|
| <b>128x128</b>   | 0,0655    | 0,0964            | 0,7737             | 0,8701          | 75,3224          |
| <b>256x256</b>   | 0,2621    | 0,2222            | 1,8457             | 2,0679          | 126,7661         |
| <b>512x512</b>   | 1,0486    | 0,8764            | 3,5266             | 4,4030          | 238,1503         |
| <b>1024x1024</b> | 4,1943    | 3,2107            | 9,1959             | 12,4066         | 338,0703         |
| <b>2048x2048</b> | 16,7772   | 12,7048           | 35,0284            | 47,7332         | 351,4786         |
| <b>4096x4096</b> | 67,1089   | 51,0330           | 139,0710           | 190,1040        | 353,0115         |
| <b>8192x8192</b> | 268,4350  | 210,2130          | 553,8050           | 764,0180        | 351,3464         |

TABLE 5.2 – OpenGL ES 2.0 - Convolution d'une image en niveau de gris par un filtre de taille 5x5 - float 32bits

| Size             | Size (MB) | Compute Time (ms) | Transfer Time (ms) | Total Time (ms) | Bandwidth (MB/s) |
|------------------|-----------|-------------------|--------------------|-----------------|------------------|
| <b>128x128</b>   | 0,0655    | 0,1386            | 0,9038             | 1,0424          | 62,8703          |
| <b>256x256</b>   | 0,2621    | 0,0202            | 1,3858             | 1,4060          | 186,4475         |
| <b>512x512</b>   | 1,0486    | 0,0194            | 4,2613             | 4,2807          | 244,9576         |
| <b>1024x1024</b> | 4,1943    | 0,0212            | 15,6039            | 15,6251         | 268,4337         |
| <b>2048x2048</b> | 16,7772   | 0,0215            | 60,8093            | 60,8309         | 275,8008         |
| <b>4096x4096</b> | 67,1089   | 0,0215            | 241,5410           | 241,5625        | 277,8117         |
| <b>8192x8192</b> | 268,4350  | 0,0267            | 1163,1867          | 1163,2133       | 230,7702         |

TABLE 5.3 – CPU - Convolution d'une image en niveau de gris par un filtre de taille 5x5 - float 32bits

| Size             | Size (MB) | Compute Time (ms) | Transfer Time (ms) | Total Time (ms) | Bandwidth (MB/s) |
|------------------|-----------|-------------------|--------------------|-----------------|------------------|
| <b>128x128</b>   | 0,0655    | 8,3513            | 0,0000             | 8,3569          | 7,8421           |
| <b>256x256</b>   | 0,2621    | 33,8604           | 0,0000             | 33,8698         | 7,7398           |
| <b>512x512</b>   | 1,0486    | 150,1860          | 0,0000             | 150,2010        | 6,9812           |
| <b>1024x1024</b> | 4,1943    | 721,2130          | 0,0000             | 721,2310        | 5,8155           |
| <b>2048x2048</b> | 16,7772   | 3196,6300         | 0,0000             | 3196,6500       | 5,2484           |
| <b>4096x4096</b> | 67,1089   | 13130,9000        | 0,0000             | 13130,9000      | 5,1108           |
| <b>8192x8192</b> | 268,4350  | 53591,6000        | 0,0000             | 53591,7000      | 5,0089           |

Comme on peut s'en rendre compte les gains de performances des deux version optimisées de l'algorithme sont non négligeable par rapport a la version CPU naïve (fig 5.3). On observer que les algorithmes s'exécutant sur la carte graphique sont jusqu'à 55 fois plus rapide pour OpenGL ES (fig 5.2) et jusqu'à 70 fois pour la version CUDA (fig 5.1). Les cases vertes dans les tableaux indiquent que l'image a pu être traité en pseudo temps réel avec une fréquence de rafraîchissement de 25fps qui signifie que pour chaque image à afficher, nous disposons d'un temps de  $1/25 * 1000 = 40$  millisecondes pour en faire le rendu. Au delà de la constat d'optimisation, on peut voir que la version CUDA et la version OpenGL affichent des résultats plutôt similaire, ils sont tout deux capable de traiter en temps réel des images de taille 1024x1024 pixels sans difficulté. On peut cependant noter une différence flagrante entre ces deux versions, en effet, on peut observer le gain apportée par la mémoire unifiée CUDA lorsque l'on compare les temps de transfert avec ceux d'OpenGL. En moyenne les temps de transfert en CUDA sont 2 fois plus rapide que leur équivalent OpenGL ES ce qui a un impact significatif sur les performances car ils correspondent à la majeur partie du temps d'exécution du programme.

#### 5.1.4 Conclusion

Au vu des résultats obtenues section 5.1.3 on peut noter deux choses :

- L'optimisation de l'algorithme utilisant un filtre de convolution séparé n'aura quasiment aucun impact sur les performances en OpenGL car les temps de calcul sont négligeables par rapport aux temps de transfert ainsi seules les version CUDA et CPU bénéficieront des amélioration qu'il peut potentiellement apporter.
- CUDA et OpenGL fournissent tout deux des résultats plutôt similaires (avec une potentielle amélioration du côté CUDA) mais n'offre tout les deux pas les mêmes possibilités. Avec CUDA, l'algorithme ne peut tourner que sur des machines supportant l'architecture. Nous avons jugé que le gain apporté par rapport a OpenGL ES, qui lui est totalement, n'était pas suffisant pour contrebalancer ce coût, c'est pourquoi nous avons choisi de continuer à utiliser et développer la version OpenGL ES.

## 5.2 Amélioration matérielle

Comme évoqué dans l'introduction, le deuxième axe d'amélioration de la technologie de RealityTech se base sur l'aspect purement matériel du système qu'elle propose. Dans cette partie, nous avons essayé d'observer et de mesurer la "puissance" du matériel utilisé afin de déterminer les parties cruciale a améliorer.

En premier lieu, nous nous sommes intéressé à mesurer la latence des dispositifs d'acquisition. La latence est définie comme le temps écoulé entre l'acquisition et l'affichage d'une information. Nous nous sommes donc procuré de nombreuses caméras différentes dont nous avons mesurer la latences sur plusieurs ordinateur. Certains ordinateurs possèdent des capacités d'encodage vidéo matériel (comme sur le NVIDIA Jetson TX2 obtenue pour l'occasion qui possède un module MSENC, un encodeur matériel<sup>6</sup>) ce qui nous a permis d'en évaluer l'impacte sur la latence lors de l'obtention du flux vidéo. Mis a part le Jetson TX2 possédant une caméra embarquée, les mesures de la latence ont toutes été effectué en utilisant GStreamer[6] avec la même commande d'obtention du flux afin d'éviter au maximum les différences de mesure.

Pour mesurer cette latence, nous avons utiliser la méthode dite "Glass to glass" qui est pratiquement la seule méthode actuellement. Pour effectuer une telle mesure il faut afficher sur un écran un chronomètre haute résolution, pointer la caméra sur l'écran, afficher le flux vidéo de la caméra, puis prendre une photo de l'écran avec le compteur et le flux vidéo de la caméra filmant se compteur côté à côté (fig 5.8). La latence est finalement obtenue en faisant la soustraction des deux temps affichés par les compteurs. Cette méthode comporte un bon nombre de défauts dont le plus critique est la résolution du chronomètre utilisé. En effet la latence d'une caméra s'exprime en millisecondes, ainsi pour avoir une mesure assez précise, le chronomètre doit avoir un taux de rafraîchissement inférieur a la milliseconde ce qui est extrêmement rare. Ensuite le taux de rafraîchissement ainsi que la latence de l'écran utilisé viennent aussi perturber les mesures. Dans notre cas nous avons utilisé un chronomètre avec une résolution de l'ordre de 1 a 5 millisecondes<sup>7</sup> et un écran 120Hz, avec 1 milliseconde de temps de latence ce qui devrait réduire les imprécisions introduite dans les mesures. Aussi au lieu de prendre une photographie, nous avons décidé de réaliser des vidéos ralenti en 240 fps et d'afficher en plus du chronomètre une vidéo ou 12 couleurs se succèdent a une fréquence 1Hz. Ainsi en plus de la mesure du chronomètre nous pouvons calculer grâce a la vidéo ralenti le nombre d'images qu'il faut pour qu'un changement de couleur dans la vidéo se refléter dans l'affichage du flux vidéo de la caméra. Étant donné que nous filmons a 240 fps, chaque images de la vidéo ou le changement de couleur n'est pas reflété corresponds pas a  $1/240 * 1000 = 4,16$  millisecondes. Par exemple, si sur la vidéo ralenti, un changement de couleur met 3 images a être reflété, alors la latence est de  $3 \times 4,16 = 12.48$  millisecondes à plus ou moins 4.16ms.

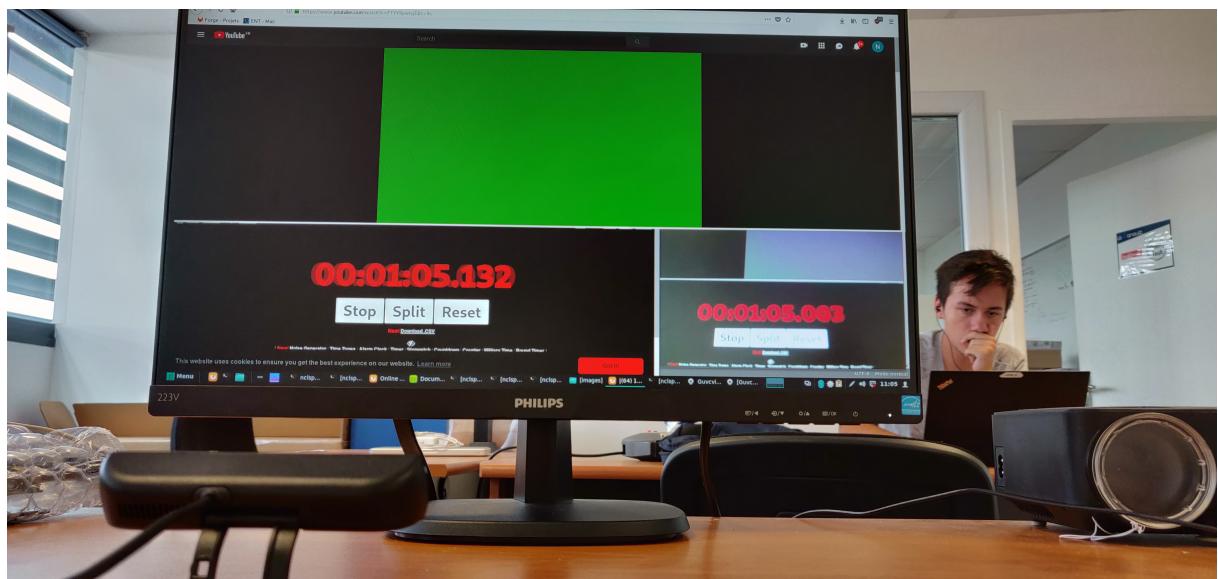


FIGURE 5.8 – Exemple de mesure "glass to glass" de la latence d'une caméra

Dans un soucis de cohérence avec la partie optimisation logicielle vous trouverez Tableau /reffig :latency :camera un comparatif des différentes latence de caméra obtenues sur le kit de développement NVIDIA Jetson TX2. Vous trouverez plus de résultats sur différents ordinateur Annexe .

6. NVIDIA Jetson TX2 - Charactéristiques des modules
7. Online stopwatch

TABLE 5.4 – Latence de plusieurs caméra mesuré en Glass to glass - NVIDIA Jetson TX2

|                  | <b>Onboard (TX2)</b> | <b>Logitech</b> | <b>SR300</b> | <b>PSEye</b> | <b>Aukay</b> | <b>ELP</b> |
|------------------|----------------------|-----------------|--------------|--------------|--------------|------------|
| <b>640x480</b>   | 75                   | 120             | 70           | 120          | 85           | 80         |
| <b>1280x1020</b> | 80                   | 80              | 85           | /            | 85           | 85         |
| <b>1920x1080</b> | 80                   | 130             | 300          | /            | 170          | 95         |

On s'aperçoit très rapidement que les tests de latence sont plutôt décevant, en effet les latences sont toutes plus ou moins similaire même si il y a quelques variation notamment pour la résolution 1920x1080 ce qui ne permet pas d'émettre beaucoup d'hypothèse d'amélioration. On observe que même la caméra embarqué disposant de son propre circuit intégré sur la carte mère du TX2 n'apporte aucun gain significatif par rapport aux autres caméra. N'étant pas satisfait des résultats, nous avons décidé de mesurer la latence d'une caméra professionnelle point gray et la latence observé a été de seulement 8-10 millisecondes en résolution 1280x1020.

### 5.3 Nectar - Architecture micro services

Pourachever le développement du prototype haute performance nous avons choisi de réétudié l'architecture logicielle de PapARt. Actuellement, PapARt est un gros kit de développement proposant une multitude de services regroupés en son sein comme par exemple l'acquisition du flux vidéo d'une caméra, le traitement des images, la détection de marqueurs, la visualisation, l'estimation de pose, etc.... . Avec la centralisation des services une panne peut être dramatique et rendre tout le système inutilisable. L'idée était donc de développer une nouvelle architecture pour pallier à ce défaut et permettre au système de gagner en réactivité, stabilité, performance, modularité et temps de maintenance. Une architecture en micro services s'est imposé comme une solution de choix car elle répond à tous les besoins énoncés.

Une architecture en micro service consiste à décomposer un logiciel en une multitude de services indépendants effectuant chacun une tâche bien précise. Ces services peuvent ensuite communiquer les uns avec les autres par le biais d'une API.

**Performance** Contrairement à une bibliothèque classique, avec une telle architecture il est possible d'allouer des ressources à la demande aux services en ayant besoin. Cela permet, par exemple, d'allouer beaucoup de ressources aux services qui les demandent lorsqu'un faible nombre d'entre eux est entrain de fonctionner, là où dans le cas d'une bibliothèque classique, les ressources supplémentaires allouées l'auraient été pour tous les services. Ce gain de performance n'est pas négligeable car il permet d'améliorer considérablement la gestion des ressources, qui peut devenir critique en cas de saturation etc....

**Réactivité** Dans le cas d'une panne d'un service critique, comme par exemple le service récupérant le flux vidéo de la caméra, avec une architecture centralisée la gestion de cette panne est très complexe et nécessite souvent de redémarrer tout le système, ce qui prends un certain temps, la ou une architecture en micro services couplé a un gestionnaire de processus se chargera uniquement de relancer le service en panne et, si il faut, les services associés, de manière quasiment transparente pour l'utilisateur.

**Modularité** L'architecture en micro services offre une très grande modularité, chaque service peut être écrit dans n'importe quel langage et peut être intégré au système sans cout tant qu'il respecte l'API de communication inter processus si il n'est pas totalement indépendant. Il est ainsi très facile pour n'importe qui de rajouter des modules qui peuvent venir soit compléter un service existant soit tout simplement rajouter une fonctionnalité au système. Par exemple, un service d'estimation de pose peut être complété par un service de filtrage de façon totalement transparente a l'utilisateur finale. L'utilisation ou non du filtrage peut être contrôlé de façon automatique par un autre module gérant par exemple la qualité générale voulu par l'utilisateur.

**Maintenabilité** Lorsqu'un service est défectueux, le problème peut être très rapidement identifier car les services sont très faiblement couplés et ainsi il n'est souvent pas nécessaire de devoir parcourir une grande quantité de code pour pouvoir identifier un bug. De plus grâce a la modularité de l'architecture, un service en maintenance n'affecte pas la stabilité générale du système.

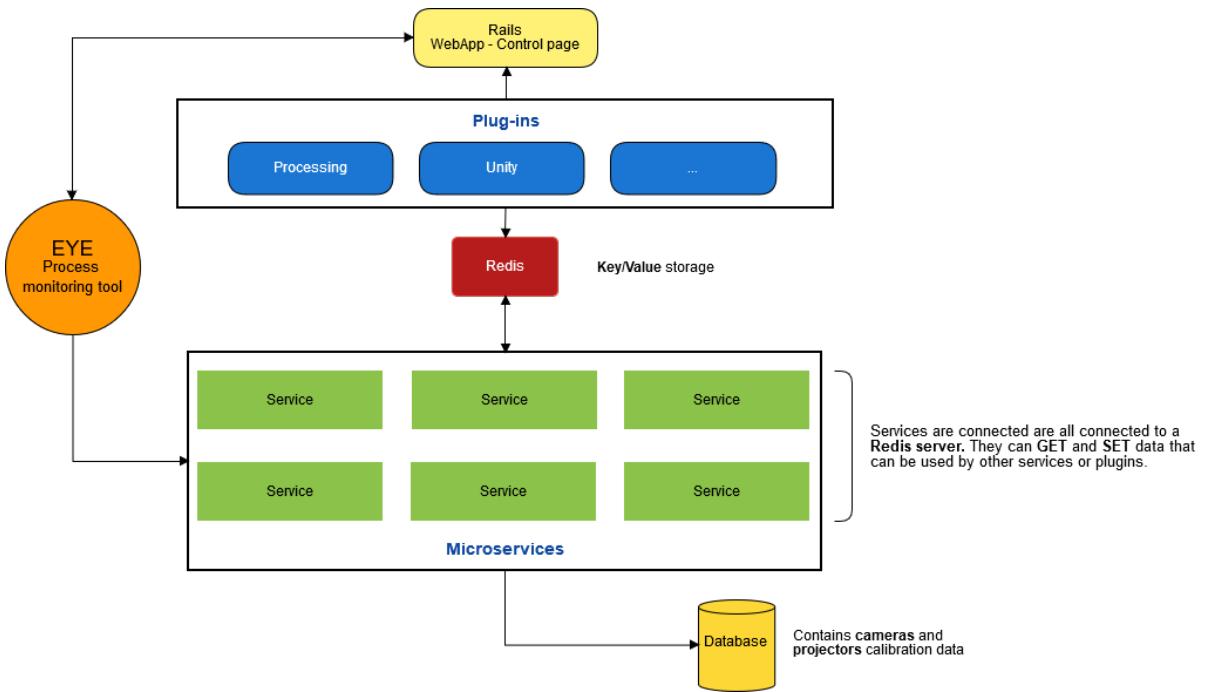


FIGURE 5.9 – Schéma de l'architecture en micro services développée

Pour mettre en place cette nouvelle architecture (fig 5.9) nous nous sommes basés sur 3 technologies principales :

- **Redis[?]** ou *REmote DIctionary Server* est un système de stockage clef-valeur qui contrairement aux bases de données plus conventionnelle stock les données directement dans la mémoire vive (RAM) de l'ordinateur ou dans la mémoire virtuelle et non pas sur le disque. Cette spécificité permet à Redis d'atteindre des performances inégalées par des bases de données classiques et il s'impose donc comme un choix de qualité pour nos besoins.
- **Eye[?]** est un outil de gestion de processus qui permet de s'assurer du cycle de vie des processus qu'il gère. Eye offre la possibilité lancer/redémarrer/couper automatiquement des processus en fonction de leur état, de leur dépendances et de leur consommation en ressources (mémoire, CPU). Il est ainsi possible de définir pour chaque service de quoi il dépend et quels sont les ressources maximales auxquels qui lui sont autorisés ce qui permet d'éviter la mise en péril de tout le système lorsqu'un un processus commence à consommer 99% de la mémoire disponible par exemple.
- **Ruby on Rails[?]** est un *framework* destiné à la création d'applications web modernes rapidement et simplement. L'idée est d'utiliser Rails pour générer une page web de contrôle du système permettant de lancer / couper des processus, effectuer des tests et tout autre action permettant d'observer l'état général du système.

Dans notre architecture, tous les micro services sont connectés à Redis. Ils peuvent chacun manipuler les données présentes et en écrire de nouvelle. C'est uniquement par ce biais que se fait la communication inter services. Ainsi, par exemple, un service accédant à la caméra ne transfèrent pas directement ses données à un service analysant l'image, mais envoi l'image courante de la caméra dans Redis qui est ensuite récupéré par cet autre service. Redis possède un système de "notifications", il est possible d'ordonner au un client Redis (un service) d'écouter sur une clef, de ce fait des que des nouvelles données arrivent le service est directement au courant. C'est le pipeline événementiel que nous avons utiliser pour éviter l'attente active des services et ainsi améliorer les performances générale du système. Couplé à Redis, c'est Eye qui s'assure a tous moments que tous les services demandé par l'utilisateur sont entrain d'être exécuté. Pour cela, il vérifie que le service est en marche mais aussi que toutes les dépendances le sont aussi. Eye peut recevoir des commandes de l'application web qui est elle même manipulée soit par l'utilisateur soit par un module de développement. Les modules de développement sont des API permettant aux utilisateurs développeurs de créer leurs propres applications de réalité augmentée spatiale utilisant le système de RealityTech.

# Unity

Dans cette partie on parlera finalement du développement du module Unity utilisant la nouvelle plate-forme mise en place au cours du stage dont il est question Chapitre 5. Ce développement a été la parfaite occasion de tester et mettre à l'épreuve cette dernière et avoir un retour réel sur son utilisabilité. Il sera discuté, dans un premier temps, des enjeux, des apports, de la cible et de la conception du module en abordant rapidement les difficultés rencontrées liées à Unity ou à la nouvelle plate-forme puis dans un second temps du développement d'application avec ce module.

Pour finir, nous effectuerons une rapide comparaison entre la version actuelle et la version en développement des kits de développement (Unity et Processing) afin d'avoir une évaluation dans des conditions réelles d'utilisation.

## 6.1 Module Unity

## 6.2 Application Unity

# Annexe 1

TABLE 6.1 – OpenGL ES 2.0 - Convolution d'une image en niveau de gris par un filtre de convolution de taille 5x5 - float 32 bits. HP Pavillion X2 - 4GB RAM - Intel Celeron CPU N2910 @ 1,60GHz x 4 - GPU @ Intel Bay Trail

| Size             | Size (MB) | Compute Time (ms) | Transfer Time (ms) | Total Time (ms) | Bandwidth (MB/s) |
|------------------|-----------|-------------------|--------------------|-----------------|------------------|
| <b>128x128</b>   | 0,0655    | 9,9136            | 11,7581            | 21,6717         | 3,0240           |
| <b>256x256</b>   | 0,2621    | 9,8709            | 12,6144            | 22,4853         | 11,6584          |
| <b>512x512</b>   | 1,0486    | 9,6095            | 19,6039            | 29,2134         | 35,8938          |
| <b>1024x1024</b> | 4,1943    | 9,5579            | 48,5473            | 58,1052         | 72,1845          |
| <b>2048x2048</b> | 16,7772   | 9,5562            | 165,5163           | 175,0726        | 95,8300          |
| <b>4096x4096</b> | 67,1089   | 9,6677            | 502,2200           | 511,8877        | 131,1008         |
| <b>8192x8192</b> | 268,4350  | 9,6540            | 2273,5233          | 2283,1773       | 117,5708         |

TABLE 6.2 – CPU - Convolution d'une image en niveau de gris par un filtre de convolution de taille 5x5 - float 32 bits. HP Pavillion X2 - 4GB RAM - Intel Celeron CPU N2910 @ 1,60GHz x 4 - GPU @ Intel Bay Trail

| Size             | Size (MB) | Compute Time (ms) | Transfer Time (ms) | Total Time (ms) | Bandwidth (MB/s) |
|------------------|-----------|-------------------|--------------------|-----------------|------------------|
| <b>128x128</b>   | 0,0655    | 15,3841           | 0,0000             | 15,3953         | 4,2569           |
| <b>256x256</b>   | 0,2621    | 61,9902           | 0,0000             | 62,0079         | 4,2276           |
| <b>512x512</b>   | 1,0486    | 287,5590          | 0,0000             | 287,5890        | 3,6461           |
| <b>1024x1024</b> | 4,1943    | 1161,5000         | 0,0000             | 1161,5300       | 3,6110           |
| <b>2048x2048</b> | 16,7772   | 4677,9100         | 0,0000             | 4677,9400       | 3,5865           |
| <b>4096x4096</b> | 67,1089   | 18764,3000        | 0,0000             | 18764,3000      | 3,5764           |
| <b>8192x8192</b> | 268,4350  | 76909,9000        | 0,0000             | 76909,9000      | 3,4903           |

TABLE 6.3 – CUDA - Convolution d'une image en niveau de gris par un filtre de convolution de taille 5x5 - float 32 bits. PC Custom - 8GB RAM - Intel(R) Core(TM) i5-7400 CPU @ 3.00GHz - NVIDIA GTX 1060 GPU @ 1506 Mhz 3Go GDDR5

| Size             | Size (MB) | Compute Time (ms) | Transfer Time (ms) | Total Time (ms) | Bandwidth (MB/s) |
|------------------|-----------|-------------------|--------------------|-----------------|------------------|
| <b>128x128</b>   | 0,0655    | 0,0211            | 0,4548             | 0,4759          | 137,7067         |
| <b>256x256</b>   | 0,2621    | 0,0492            | 0,6379             | 0,9179          | 285,5966         |
| <b>512x512</b>   | 1,0486    | 0,1635            | 1,4086             | 1,6951          | 618,6057         |
| <b>1024x1024</b> | 4,1943    | 0,6200            | 3,3843             | 3,7492          | 1118,7067        |
| <b>2048x2048</b> | 16,7772   | 2,4485            | 10,2548            | 10,5924         | 1583,8903        |
| <b>4096x4096</b> | 67,1089   | 9,6682            | 40,9823            | 41,3963         | 1621,1328        |
| <b>8192x8192</b> | 268,4350  | 38,3053           | 198,3800           | 201,6840        | 1330,9682        |

TABLE 6.4 – OpenGL ES 2.0 - Convolution d'une image en niveau de gris par un filtre de convolution de taille 5x5 - float 32 bits. PC Custom - 8GB RAM - Intel(R) Core(TM) i5-7400 CPU @ 3.00GHz - NVIDIA GTX 1060 GPU @ 1506 Mhz 3Go GDDR5

| Size             | Size (MB) | Compute Time (ms) | Transfer Time (ms) | Total Time (ms) | Bandwidth (MB/s) |
|------------------|-----------|-------------------|--------------------|-----------------|------------------|
| <b>128x128</b>   | 0,0655    | 0,2222            | 1,2313             | 1,4535          | 45,0887          |
| <b>256x256</b>   | 0,2621    | 0,0149            | 1,4875             | 1,5024          | 174,4831         |
| <b>512x512</b>   | 1,0486    | 0,0166            | 5,8531             | 5,8697          | 178,6425         |
| <b>1024x1024</b> | 4,1943    | 0,0061            | 5,7249             | 5,7310          | 731,8617         |
| <b>2048x2048</b> | 16,7772   | 0,0062            | 20,4904            | 20,4965         | 818,5378         |
| <b>4096x4096</b> | 67,1089   | 0,0138            | 78,3060            | 78,3198         | 856,8579         |
| <b>8192x8192</b> | 268,4350  | 0,0115            | 370,6600           | 370,6715        | 724,1857         |

TABLE 6.5 – CPU - Convolution d'une image en niveau de gris par un filtre de convolution de taille 5x5 - float 32 bits. PC Custom - 8GB RAM - Intel(R) Core(TM) i5-7400 CPU @ 3.00GHz - NVIDIA GTX 1060 GPU @ 1506 Mhz 3Go GDDR5

| Size             | Size (MB) | Compute Time (ms) | Transfer Time (ms) | Total Time (ms) | Bandwidth (MB/s) |
|------------------|-----------|-------------------|--------------------|-----------------|------------------|
| <b>128x128</b>   | 0,0655    | 4,1572            | 0,0000             | 4,1608          | 15,7510          |
| <b>256x256</b>   | 0,2621    | 10,2566           | 0,0000             | 10,2596         | 25,5511          |
| <b>512x512</b>   | 1,0486    | 42,7127           | 0,0000             | 42,7166         | 24,5474          |
| <b>1024x1024</b> | 4,1943    | 197,4300          | 0,0000             | 197,4370        | 21,2437          |
| <b>2048x2048</b> | 16,7772   | 898,1140          | 0,0000             | 898,1210        | 18,6803          |
| <b>4096x4096</b> | 67,1089   | 3733,9500         | 0,0000             | 3733,9600       | 17,9726          |
| <b>8192x8192</b> | 268,4350  | 13622,6846        | 0,0000             | 13622,6846      | 19,7050          |

## Annexe 2

TABLE 6.6 – Latence caméras (millisecondes) - Résolution 640x480

|                          | <b>On board</b> | <b>logitech</b> | <b>SR300</b> | <b>PSEye</b> | <b>Aukay</b> | <b>ELP</b> |
|--------------------------|-----------------|-----------------|--------------|--------------|--------------|------------|
| <b>NVIDIA Jetson TX2</b> | 75              | 120             | 70           | 90           | 85           | 80         |
| <b>HP Pavilion X2</b>    | /               | 90              | /            | 130          | /            | /          |
| <b>PC Custom</b>         | /               | 85              | /            | 85           | 80           | 80         |

TABLE 6.7 – Latence caméras (millisecondes) - Résolution 1280x720 (HD)

|                          | <b>On board</b> | <b>logitech</b> | <b>SR300</b> | <b>PSEye</b> | <b>Aukay</b> | <b>ELP</b> |
|--------------------------|-----------------|-----------------|--------------|--------------|--------------|------------|
| <b>NVIDIA Jetson TX2</b> | 80              | 80              | 85           | /            | 85           | 85         |
| <b>HP Pavilion X2</b>    | /               | 120             | /            | /            | /            | /          |
| <b>PC Custom</b>         | /               | 80              | /            | /            | 90           | 80         |

TABLE 6.8 – Latence caméras (millisecondes) - Résolution 1920x1080 (Full HD)

|                          | <b>On board</b> | <b>logitech</b> | <b>SR300</b> | <b>PSEye</b> | <b>Aukay</b> | <b>ELP</b> |
|--------------------------|-----------------|-----------------|--------------|--------------|--------------|------------|
| <b>NVIDIA Jetson TX2</b> | 80              | 130             | 300          | /            | 170          | 95         |
| <b>HP Pavilion X2</b>    | /               | /               | /            | /            | /            | /          |
| <b>PC Custom</b>         | /               | 130             | /            | /            | 90           | 120        |

# Bibliographie

- [1] Sam Aaron. Sonic pi - the live coding music synth for everyone. <https://sonic-pi.net/>. Last accessed 27 July 2018.
- [2] John Canny. A computational approach to edge detection. *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE*, 1986.
- [3] Richard O Duda and Peter E Hart. Use of the hough transformation to detect lines and curves in pictures. *Communications of the ACM*, 15(1) :11–15, 1972.
- [4] Khronos Group. Opencl overview. <https://www.khronos.org/opencl/>. Last accessed 28 July 2018.
- [5] Khronos Group. Opengl es. <https://www.khronos.org/opengles/>. Last accessed 28 July 2018.
- [6] GStreamer. Gstreamer. <https://gstreamer.freedesktop.org/>. Last accessed 13 August 2018.
- [7] Brett Jones, Rajinder Sodhi, Michael Murdock, Ravish Mehra, Hrvoje Benko, Andrew Wilson, Eyal Ofek, Blair MacIntyre, Nikunj Raghuvanshi, and Lior Shapira. Roomalive : Magical experiences enabled by scalable, adaptive projector-camera units. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology*, UIST ’14, pages 637–644, New York, NY, USA, 2014. ACM.
- [8] Paul Milgram, Haruo Takemura, Akira Utsumi, and Fumio Kishino. Augmented reality : A class of displays on the reality-virtuality continuum. In *Telemanipulator and telepresence technologies*, volume 2351, pages 282–293. International Society for Optics and Photonics, 1995.
- [9] NVIDIA Corporation. NVIDIA CUDA C programming guide, 2010. Version 3.2.
- [10] Victor Podlozhnyuk. Image convolution with cuda. *NVIDIA Corporation white paper, June*, 2097(3), 2007.
- [11] Reactable. Reactable experience - music knowledge technology. <http://reactable.com/>. Last accessed 27 July 2018.
- [12] Unity Technologies. Unity. <https://unity3d.com/>. Last accessed 25 August 2018.