

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

Auteur :
Nicolas PALARD

Client :
Jérémie LAVIOLE
Référent :
Vincent LEPETIT



Résumé

Dans le cadre de mon stage de fin d'étude, j'ai eu l'occasion de travailler chez **RealityTech**[17], une jeune start-up dans le domaine de la réalité augmentée spatiale[2]. Durant le temps que j'y ai passé, il m'a été demandé d'étudier les objectifs, les intérêts et les apports de cette technologie. J'ai tout d'abord pu l'apprivoiser par le biais de **PapART**[16], le système de projection interactif que développe la société, avec lequel j'ai développé mes premières applications. Tout au long de ces développements, j'ai découvert les enjeux mais aussi les difficultés inhérentes à de tels systèmes. Ces diverses difficultés ainsi que la rigidité de **PapART** m'ont amené à développer un prototype d'une nouvelle plateforme pour la société, dont le but était d'offrir de meilleures performances et d'être plus robuste que le système actuel, tout en conservant les fonctionnalités. De l'optimisation matérielle à la refonte complète de l'architecture, en passant par l'optimisation algorithmique, il m'a été demandé d'opérer sur tous les fronts afin de créer un prototype viable. Un prototype n'allant pas sans tests, j'ai du m'atteler à réaliser une batterie de tests de performance qui ont permis de mettre en lumière les avancées mais aussi les défauts de ce dernier. Les performances n'étant pas le seul critère de validation du prototype, j'ai poursuivi mon stage en développant un module **Unity**[28] permettant de le mettre à profit. En plus de permettre l'évaluation de l'utilisabilité du prototype, le module devait aussi fournir aux utilisateurs développeurs de **RealityTech** la possibilité de créer des applications de réalité augmentée spatiale en résolvant, pour ces derniers, les nombreuses problématiques liées au domaine telle que la calibration caméra projecteur, la modélisation du monde réel etc. Ainsi, pour terminer mon stage, j'ai endossé le costume de l'utilisateur développeur et ai réalisé une application de démonstration en utilisant le module nouvellement créé afin d'en réaliser l'évaluation.

Mot-clés : Réalité augmentée, réalité augmentée spatiale, interface tangible, **PapART**, programmation haute performance, microservices, unity, processing, traitement d'image, vision par ordinateur.

Remerciements

Je tiens à remercier toutes les personnes que j'ai eu l'occasion de rencontrer durant mon stage, pour l'enrichissement aussi bien professionnel que personnel qu'elles m'ont apporté. Je souhaite également remercier toutes les personnes qui m'ont conseillé, que ce soit pendant mon stage ou pour écrire ce mémoire, et qui ont contribué de près ou de loin au succès de celui-ci.

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

Je remercie également **Mme Thevin Lauren**, chercheuse en réalité mixte (réalité augmentée / réalité virtuelle) pour les déficients visuels chez **Inria**, et collaboratrice de M. Laviole, pour tous les échanges que nous avons eu durant ce stage et qui se sont toujours révélés 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.

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

Table des matières

1	Introduction	1
1.1	Contexte et cadre	1
1.2	Objectifs	2
2	Notions	4
2.1	Réalité virtuelle	4
2.2	Réalité augmentée	4
2.3	Réalité mixte	5
2.4	Réalité augmentée vue au travers	5
2.5	Réalité augmentée spatiale	6
2.6	Interface tangible	6
2.7	Calcul haute performance	7
3	État de l'art	8
3.1	PapARt	8
3.2	Autre systèmes de réalité augmentée spatiale	10
3.2.1	RoomAliveToolKit	10
3.2.2	Lampix	10
4	Développement d'application	12
4.1	ReARTable	12
4.1.1	Besoins de l'application	13
4.1.2	Choix et implémentation	13
4.2	Extraction de document	16
4.2.1	Besoins de l'application	16
4.2.2	Choix et implémentation	16
4.2.3	Bilan	19
5	Prototype haute performance	21
5.1	Amélioration logicielle	21
5.1.1	Convolution - Théorie	21
5.1.2	Convolution - Optimisation	23
5.1.3	OpenCL	24
5.1.4	OpenGL (ES)	25
5.1.5	CUDA	26
5.1.6	Tests de performance	27
5.1.7	Bilan	28
5.2	Amélioration matérielle	30
5.3	Nectar - Architecture microservices	32
5.4	Bilan	34
6	Unity	35
6.1	Module Unity	35
6.2	Illusion de projection	38
6.3	Bilan	40
7	Conclusion	42

Annexes	44
Annexe 1 - Tests de performance convolution	44
Annexe 2 - Tests de latence caméra	46
Annexe 3 - Unity diagramme d'exécution des fonctions	47
Annexe 4 - Gantt effectif	48

Introduction

Ce mémoire a pour but de retracer 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 (six mois), dans la société **RealityTech**¹. Ce rapport ne couvrira cependant que les cinq premiers mois du stage car la date de rendu de ce dernier précède d'un mois la date de fin du stage.

Le stage a donc été effectué chez **RealityTech** une jeune start-up de réalité augmentée spatiale créée fin 2016, ne comptant actuellement aucun employé. 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**² *Paper Augmented Reality Toolkit*, est un kit de développement ou *Software Development Kit (SDK)* **Processing** permettant de créer des expériences de réalité augmentée sous forme d'applications de projection interactive sur des feuilles de papier. Pour réaliser ce type d'application, du matériel spécifique est nécessaire tel que des caméras couleur et des caméras de profondeur qui sont utilisées pour capter le monde réel, un projecteur utilisé pour pouvoir visualiser le contenu numérique dans l'espace ainsi qu'un ordinateur pour effectuer tous les calculs nécessaires 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-calibrés et prêts à l'emploi résolvant ainsi tous les problèmes de calibration et d'installation de l'environnement nécessaire à l'utilisation de **PapART**. En parallèle de la partie matérielle, la société développe et améliore activement le kit et créé également diverses applications de démonstration, afin de présenter et d'étendre les possibilités de la technologie. Le but est de pouvoir ouvrir la technologie à d'autres domaines d'activité que celui de la recherche, pour mettre les outils collaboratifs et les systèmes interactifs au service 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**³, situé 4 rue Eugène et Marc Dulout, à Pessac Centre. L'objectif de **La Banquiz** est de promouvoir des start-up Open Source⁴ 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 des jeunes entreprises.

À ce jour, **RealityTech** ne travaille qu'avec des laboratoires 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 prototypes FabLab créés à l'unité basé sur des technologies de recherche, à des prototypes industriels basés sur des technologies populaires dans le monde de l'industrie. **RealityTech** se lance donc dans la création d'une plateforme haute performance appelée **Nectar** et dans le développement d'un nouvel *SDK Unity*⁵ 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.

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>

Pour débuter ce stage, j'ai eu l'occasion de partir à Laval durant une semaine pour participer au **Laval Virtual**⁶, le plus grand salon international sur la réalité augmentée et virtuelle, en tant qu'exposant. Grâce à ce salon, j'ai pu développer une bonne connaissance du produit et ai pu observer de nombreuses technologies à l'état de l'art dans ces domaines. Ce salon m'a également 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 sur le plan relationnel car elle m'a aidé à créer rapidement une relation avec M. Laviole, mon tuteur de stage.

Le reste de mon stage s'est déroulé 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 tels que Rémi Kressmann, développeur, et Gabin Andrieux, commercial chez **LockEmail**⁷, une entreprise de cyber-sécurité, mais aussi des dirigeants comme Jean François Schaff, docteur en physique quantique ayant créé **Postelo**⁸ 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 général.

1.2 Objectifs

Au cours de ce stage, j'ai eu l'occasion de remplir bon nombre de tâches diverses et variées, guidées par les besoins de la société. Les tâches principales sont décrites dans le diagramme de Gantt effectif (Figure 1.1) des cinq premiers mois de ce stage. Une version plus détaillée du diagramme et des tâches réalisées est disponible Annexe 4.

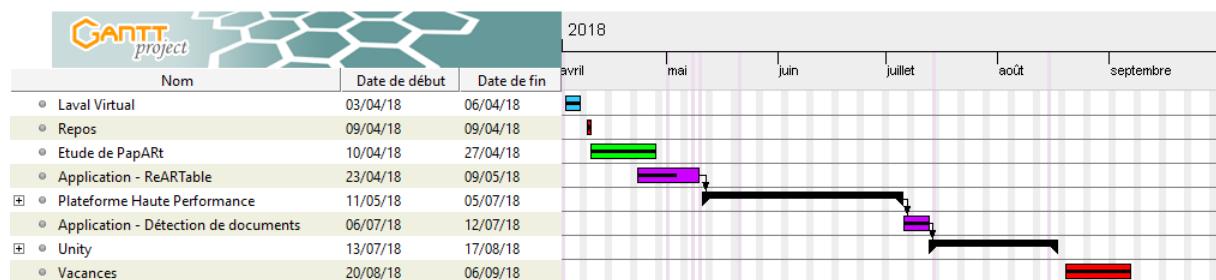


FIGURE 1.1 – Diagramme de Gantt effectif des tâches réalisées

Applications de démonstration Le stage a débuté par le développement d'applications de démonstration en utilisant le système de l'entreprise (Chapitre 4). Le but était de comprendre l'essence, le fonctionnement global du système et ce qu'il était possible/impossible de réaliser avec celui-ci. Ainsi, en réalisant les tâches qui m'ont été confiées, j'ai pu acquérir à la fois une vision globale de l'architecture logicielle, du fonctionnement interne du kit de développement et de l'architecture matérielle nécessaire à son utilisation. Cela m'a également aidé à développer une certaine autonomie pour la suite du stage.

Plateforme haute performance J'ai été amené à réaliser une preuve de concept visant à présenter une version haute performance du produit de l'entreprise (Chapitre 5). Cette preuve de concept devait proposer des méthodes et des pistes d'amélioration de la solution actuelle aussi bien sur le plan logiciel, avec par exemple l'optimisation d'algorithmes existants (section 5.1), que sur le plan matériel, avec la réalisation de tests de performance des différents dispositifs d'acquisitions (section 5.2) que sont la caméra, le projecteur ainsi que sur l'architecture globale de la solution, où il a fallu développer une toute nouvelle architecture en microservices incluant un tout nouvel environnement de travail, ainsi qu'un gestionnaire de processus et un serveur web (section 5.3).

Kit de développement Comme précédemment 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 grosses scènes 3D ou des simulations physiques par exemple. Le développement d'une version **Unity** du kit, permettant de créer des applications de projection interactive, s'est donc avéré nécessaire pour la suite du développement de **RealityTech** (Chapitre 6).

6. <https://www.laval-virtual.org/>

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

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

Pour que le développement de ce module soit efficace, l'objectif était d'intégrer et d'utiliser les micro-services de la plateforme 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 et de moteur physique pour gérer le rendu, la physique des objets, la lumière ainsi créer des expériences de projection encore plus évoluées. Le module devait également permettre d'ouvrir la technologie de **RealityTech** à un plus grand nombre d'utilisateur du fait de la notoriété d'**Unity** dans son domaine.

Au vu de ces objectifs, il en découle que la problématique générale du stage est le développement d'applications et de technologies de réalité augmentée spatiale.

La présentation de ce mémoire sera divisée en plusieurs parties. Dans un premier temps, nous présenterons le domaine d'activité dans lequel s'insère ce stage en définissant les concepts clés nécessaire à la bonne compréhension de notre sujet. Ceci débouchera sur une revue des technologies existantes en la matière. Par la suite, les objectifs susmentionnés seront développés dans des parties spécifiques. Ainsi, le développement d'application, la création du prototype haute performance et du module **Unity** feront l'objet d'une présentation approfondie, retracant les étapes de leur création ainsi que les difficultés méthodologiques auxquelles nous avons été confrontés. Enfin, nous conclurons en exposant les bénéfices aussi bien personnels que professionnels que ce stage m'a apporté.

Notions

Le domaine d'activité entourant ce stage est très riche en matière de notion et de vocabulaire. Ainsi, 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.

2.1 Réalité virtuelle

La réalité virtuelle[20], plus communément appelée *Virtual Reality (VR)*, désigne l'ensemble des environnements purement numérique (Figure 2.1), qu'ils soient réalistes ou non, dans lesquels aucune interaction avec l'environnement réel n'est possible et inversement. Cette réalité requiert l'utilisation d'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.



FIGURE 2.1 – Représentation de continuum de la virtualité par Milgram et Kishino, 1995[20]

2.2 Réalité augmentée

La réalité augmentée[20], plus communément appelée *Augmented Reality (AR)*, quant à elle, est un sous-domaine de la réalité virtuelle. L'idée de la réalité augmentée est de superposer à l'environnement réel des éléments virtuels qui vont alors venir "augmenter" cet environnement, apportant, le plus souvent, des compléments d'information. 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 détaillée section 2.3. Il faut noter que ce type de réalité ne se base pas uniquement sur l'utilisation de *HMD* mais peut également être appréciée à l'aide d'un téléphone par exemple Figure 2.2.



FIGURE 2.2 – Réalité augmentée vue au travers d'un téléphone¹

2.3 Réalité mixte

La réalité mixte[22], ou hybride, plus communément appelée *Mixed Reality (MR)*, ou *Crossed Reality (XR)*, est la fusion parfaite de l'environnement numérique et de l'environnement physique (Figure 2.1). Dans ce "nouvel" environnement, les objets physiques et numériques coexistent et peuvent interagir entre eux (par exemple, une table peut devenir une plateforme pour un personnage virtuel (Figure 2.3). Souvent confondue avec la réalité augmentée, la différence se situe principalement dans le fait que la réalité mixte ne propose pas seulement une visualisation des objets numériques, elle propose aussi des méthodes d'interaction avec ce contenu. C'est donc la notion d'interaction avec les éléments virtuels qui permet de les différencier.

A l'heure actuelle, la réalité mixte nécessite un dispositif de type *HMD* pour être appréciée. C'est le cas notamment de l'*HoloLens* de Microsoft, un casque permettant la superposition d'éléments virtuels au champ de vision de l'utilisateur et permettant également l'interaction entre le monde virtuel et la réalité.

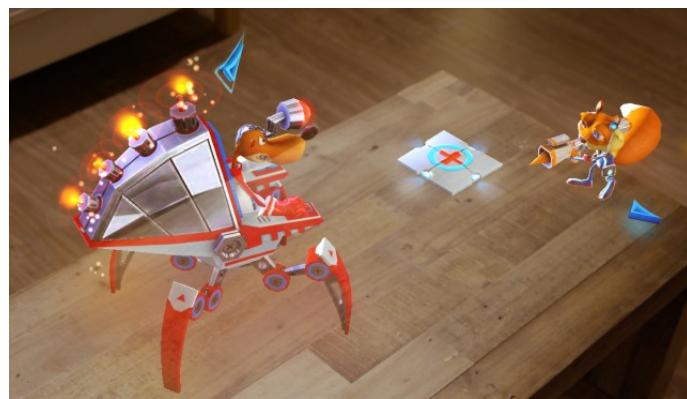


FIGURE 2.3 – Exemple d'interaction environnement physique/environnement virtuel²

2.4 Réalité augmentée vue au travers

La réalité augmentée vue au travers[20], plus communément appelée *See Through Augmented Reality (STAR)*, est une technique de visualisation de la réalité augmentée où les éléments numériques sont vus au travers d'un écran (Figure 2.4a) ou d'un *HMD* (Figure 2.4b). C'est le type de visualisation le plus utilisé actuellement. Toutefois, l'un des défauts majeur de ce type de visualisation est que, la plupart du temps, chaque utilisateur a besoin de son propre écran ou casque pour pouvoir en profiter pleinement ce qui limite grandement les expériences collaboratives. De plus, les principaux inconvénients liés à l'utilisation des écrans s'appliquent également notamment la fatigue visuelle.

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

2. Source : Asobo Studio - Young Counter



(a) Réalité augmentée vue au travers d'un téléphone (b) Réalité augmentée vue au travers d'un casque

FIGURE 2.4 – Réalité augmentée vue au travers³

2.5 Réalité augmentée spatiale

La réalité augmentée spatiale[2], plus communément appelée *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 (Figure 2.5), d'où le terme "spatiale". 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 donc obligatoirement partagées. La SAR permet aussi de favoriser le développement d'interfaces tangibles (section 2.6). En effet, la visualisation se faisant directement sur des objets physiques, la tendance à développer des interfaces en communion avec ceux-ci est très forte car très naturelle.



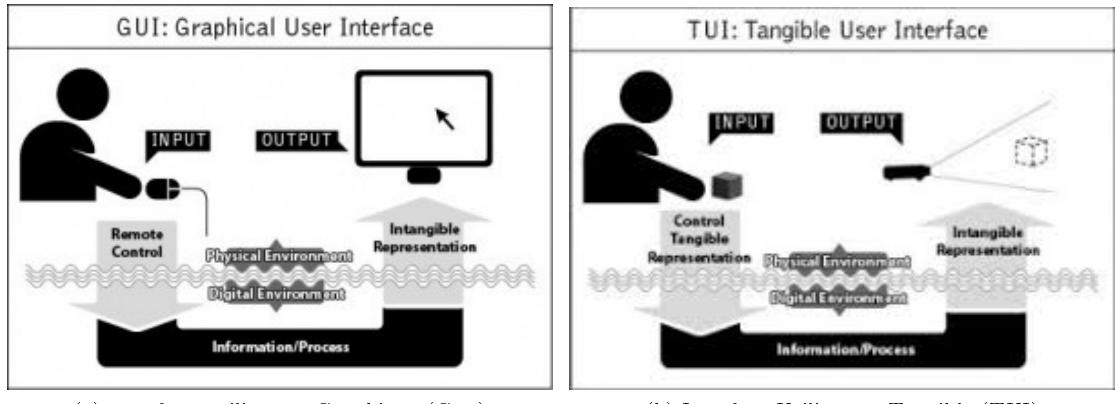
FIGURE 2.5 – Présentation d'une voiture en utilisant de la réalité augmentée spatiale⁴

2.6 Interface tangible

Une interface utilisateur tangible[13] ou *Tangible User Interface (TUI)* est une interface utilisateur via laquelle des objets physiques, ou encore le toucher, permettent de manipuler des données numériques (Figure 2.6b). Les interfaces utilisateurs tangibles remplacent très souvent les interfaces utilisateurs graphiques (Figure 2.6a) ou *Graphical User Interface (GUI)* dans la plupart des applications de réalité augmentée car elles fournissent un contrôle direct à l'utilisateur sur ce qu'il souhaite manipuler (par opposition au contrôle indirect tel que la souris, nécessaire à la manipulation des GUI).

3. Source : Pokemon GO et Microsoft HoloLens

4. Source : Google Image



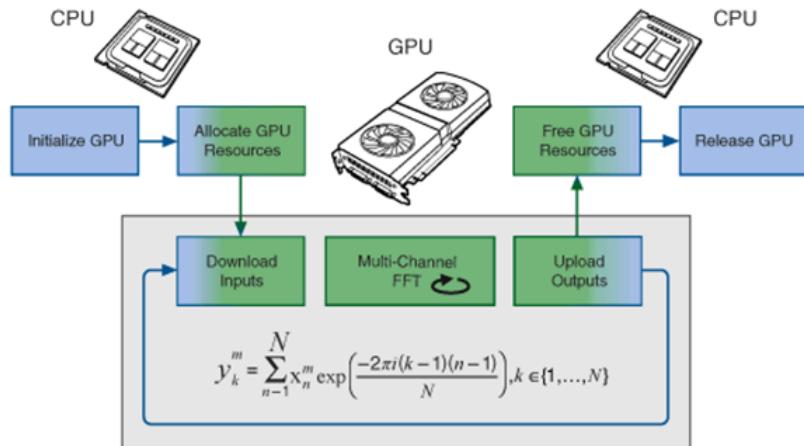
(a) Interface Utilisateur Graphique (GUI)

(b) Interface Utilisateur Tangible (TUI)

FIGURE 2.6 – Différences des interfaces utilisateurs⁵

2.7 Calcul haute performance

Le calcul haute performance[27] ou *General-Purpose computing on Graphics Processing Units (GPGPU)* désigne une méthode de calcul utilisant la carte graphique (GPU) plutôt que le processeur (CPU) (Figure 2.7). 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 traitements lourds tels que le rendu d'une scène 3D, l'encodage de vidéo, les simulations physiques (particules) etc. Cette technique repose sur le grand nombre de cœurs présents dans les cartes graphiques (contrairement aux processeurs) et sur la capacité de chacun de ces cœurs à effectuer des opérations simples de manière très efficace. Le calcul haute performance ne peut cependant pas se passer du CPU qui va principalement être utilisé pour récolter et transférer les données traitées ou à traiter.

FIGURE 2.7 – Exemple de calcul de la FFT sur GPU⁶

5. Source : Icon Library - From GUI to TUI

6. Source : National Instruments

État de l'art

De nos jours, les technologies de réalité augmentée en vue au travers, avec ou sans casque, 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[19]. Récemment, une nouvelle technologie de réalité augmentée se détachant de l'approche traditionnelle liée aux écrans, jusqu'alors inexploitée dans le domaine de l'informatique, est en train d'émerger sous le nom de réalité augmentée spatiale. Cette technologie apporte des réponses aux problèmes technologiques et ergonomiques établis et permet ainsi d'ouvrir un champ des possibles bien plus large[3].

3.1 PapART

Comme évoqué précédemment dans l'introduction (Chapitre 1), PapART ou Paper Augmented Reality Toolkit se présente sous la forme d'un kit de développement permettant de réaliser des applications interactives de création de dessins ou peintures en réalité augmentée (Figure 3.1). L'idée est de proposer une technique numérique non intrusive dont le but est de faciliter la réalisation d'une tâche complexe, telle que le dessin, tout en permettant à l'utilisateur de s'exprimer[18].



FIGURE 3.1 – Jeremy Laviole utilisant PapART pour dessiner en réalité augmentée¹

Le système interactif (Figure 3.2) permettant d'utiliser tout le potentiel de PapART est très spécifique et se compose de deux dispositifs d'acquisition.

1. Source : Inria - PapART

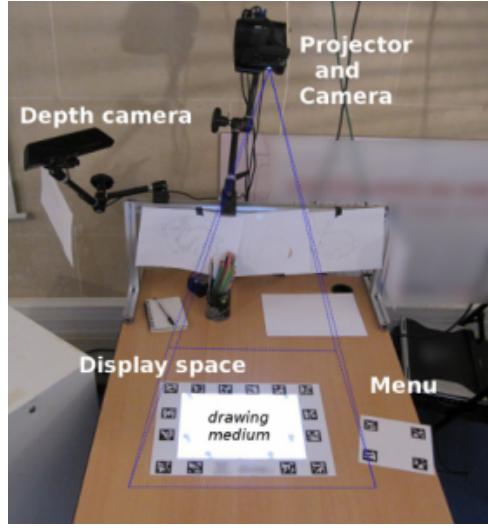


FIGURE 3.2 – Système interactif utilisant PapART²

Le premier est une caméra couleur observant la zone de travail. Son but est de détecter les feuilles de papier servant de base à la projection. Ces feuilles sont ornées de marqueurs ARToolKitPlus³, une bibliothèque de détection de marqueurs fiduciaires⁴, qui ont deux utilisations. Dans un premier temps, ils permettent la détection de la feuille de papier. En effet, avec un nombre suffisant de marqueurs détectés et une connaissance à priori du modèle de la feuille, il est possible d'en estimer la pose⁵ (position et rotation dans l'espace). Ces marqueurs servent également à identifier la feuille pour potentiellement en déterminer le contenu. Chaque marqueur étant unique, lorsque PapART détecte une feuille de papier, il est également possible de récupérer les informations des marqueurs associés permettant de projeter du contenu différent en fonction des marqueurs présents.

Le deuxième dispositif d'acquisition est une caméra de profondeur dont le 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, que l'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 sur 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 (Figure 3.3).

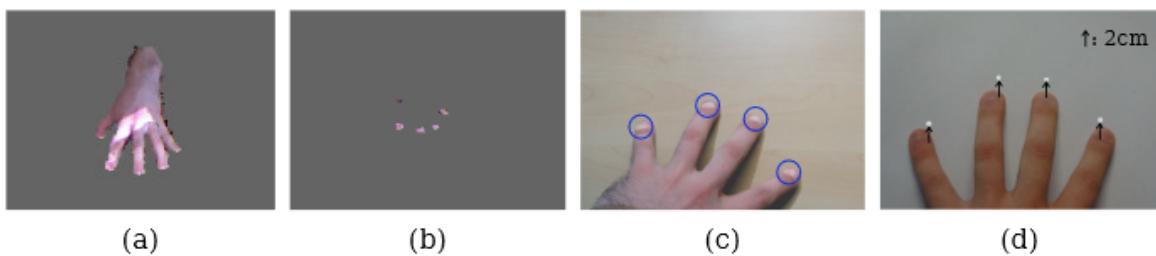


FIGURE 3.3 – PapART - Détection des interactions de "touch". (a) Détection de la main après opération de seuillage sur le plan de la table. (b) Sélection d'une zone de seulement 1-2cm au dessus de la table. (c) Projection sur les éléments détectés. (d) Décalage de la détection pour indiquer à l'utilisateur où l'interaction a été détectée.⁶

2. Source : Inria - PapART

3. <https://github.com/paroj/artoolkitplus>

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

5. Pose : Wikipédia

Au-delà des applications d'aide au dessin qui sont aujourd'hui extrêmement nombreuses, PapARt laisse entrevoir une multitude de possibilités. Ainsi, l'objectif de RealityTech est d'explorer ce champ des possibles en améliorant PapARt et en présentant de nouveaux cas d'utilisation toujours plus innovants. Par exemple, RealityTech a récemment développé un système interactif permettant de vulgariser le fonctionnement d'un réseau de neurones (Figure 3.4).

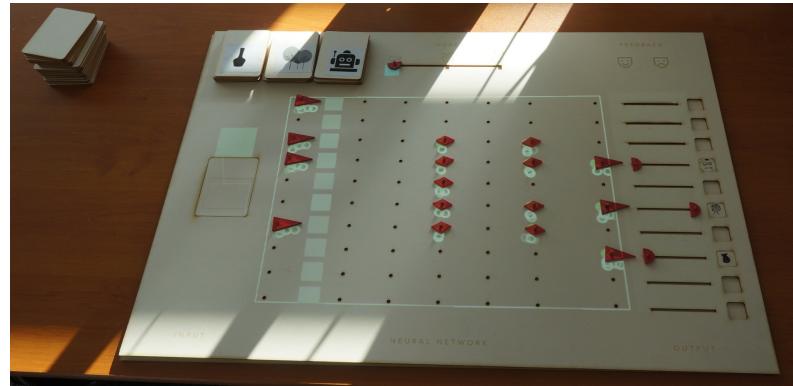


FIGURE 3.4 – Visualisation d'un réseau de neurone en réalité augmentée spatiale⁷

3.2 Autre systèmes de réalité augmentée spatiale

3.2.1 RoomAliveToolKit

RoomAliveToolKit[14] (Figure 3.5a) est un projet tout droit sorti des laboratoires de recherche de Microsoft. Il s'agit d'un kit de développement créé en 2013 par Nikunj Raghuvanshi, Eyal 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 entières en utilisant plusieurs projecteurs et plusieurs caméras fonctionnant à l'unisson.

RoomAliveToolKit a permis, entre autres, de développer de nombreux projets basés sur de la projection interactive tels que RoomAlive (Figure 3.5b), Room2Room, ou encore IllumiRoom pour ne citer qu'eux.



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

(b) RoomAlive - Démonstration

FIGURE 3.5 – Microsoft Research : RoomAliveToolkit et RoomAlive⁸

3.2.2 Lampix

Développé par George Popescu et Mihai Dumitrescu, Lampix est un système de projection interactive dont le but est de rendre n'importe quelle surface horizontale "intelligente" (Figure 3.6). Le fonctionnement aussi bien logiciel que matériel de Lampix est très similaire à celui de PapARt. Lampix utilise de

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

7. Source : Créez votre réseau de neurones artificiel - RealityTech

8. Source : RoomAliveToolkit

l'apprentissage et des techniques de vision 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 créé la toute première blockchain⁹ dédiée à la vision par ordinateur contenant d'énormes jeux de données d'image labellisées (image et description).

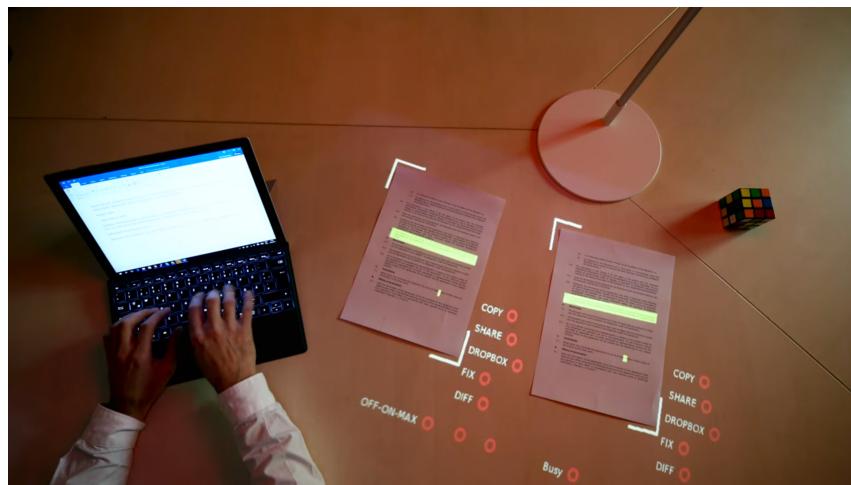


FIGURE 3.6 – Lampix - Tabletop Augmented Reality Platform¹⁰

9. Wikipédia: Blockchain

10. Source : Lampix

Développement d'application

Comme présenté 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'interface de programmation *Application Programming Language (API)* à été effectué en parallèle du développement d'applications.

4.1 ReARTable

En tenant compte du contexte et du public ciblé par l'entreprise, il m'a paru intéressant de développer une démonstration dont le but était à la fois ludique et éducatif. J'ai donc choisi de recréer une **Reactable**[24], proposée par la société du même nom, en réalité augmentée spatiale, que nous avons nommé **ReARTable**.

La **Reactable** est un instrument de musique électronique permettant la génération de sons en direct, développé depuis 2003. Présenté sous la forme d'une table interactive, le son est généré par le biais d'éléments tangibles placés à sa surface (Figure 4.1).

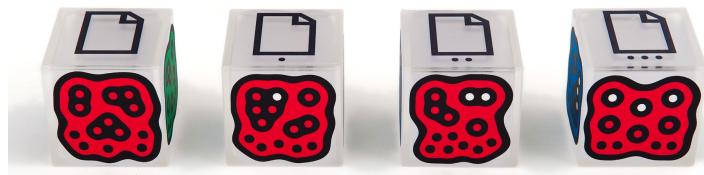


FIGURE 4.1 – Éléments tangibles utilisés pour la génération d'un élément de synthétiseur sur la **Reactable**¹

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

- La distance de l'élément par rapport à 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 si l'on prend l'exemple d'un battement de ralentir ou d'accélérer ce dernier.
 - La disposition de l'élément. Cette propriété permet, entre autres, de combiner des éléments pour créer de nouveaux sons, plus riches et plus complexes.
 - La position du doigt de l'utilisateur par rapport à un élément. On peut venir contrôler divers paramètres comme l'amplitude par exemple, en faisant graviter son doigt autour d'un élément.
- Ainsi, c'est en combinant plusieurs éléments entre eux avec différentes orientations et différentes dispositions que l'utilisateur va pouvoir peu à peu "construire" sa musique. Au-delà de la détection des éléments tangibles, la table est rétro éclairée et permet donc la visualisation, en direct, de la musique générée (Figure 4.2).

1. Source : Reactable : Elements tangibles



FIGURE 4.2 – Visualisation du son sur la Reactable²

4.1.1 Besoins de l'application

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

Pour être en adéquation avec l'idéologie de l'entreprise, l'interface tangible ainsi que les modes d'interaction avec la musique étaient les points les plus importants à considérer. En gardant cela à l'esprit, nous avons défini les besoins fonctionnels principaux que voici :

- Générer du son en direct.
- Créer une représentation physique du son. Chaque son ou élément sonore devait avoir une représentation physique qui lui était associée, c'est à dire, un élément ou groupement d'éléments tangibles le représentant.
- 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 à ce qu'ils soient utilisés pour identifier les éléments sonores.
- Identifier les représentations des sons. Chaque élément sonore étant représenté par un ou plusieurs éléments physiques, 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 paramètres définis à l'avance de chaque élément sonore généré. Ces paramètres ont pour but d'apporter à l'utilisateur 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 amené à toucher des zones interactives pour déclencher divers effets.
- Créer 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 développée avec Processing, en utilisant, d'une part, PapARt pour la partie visualisation, détection et projection et, d'autre part, Sonic Pi[1] pour la génération de musique en direct. Sonic Pi est un synthétiseur temps réel qui permet de générer très facilement des sons de manière cohérente. Le gros avantage de Sonic Pi est qu'il résout tout seul bon nombre de problèmes posés 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'autres, ce qui dé-complexifie énormément le processus.

Comme on peut le voir sur le schéma explicatif Figure 4.3, les éléments tangibles représentant des sons apparaissent sous forme de regroupement d'éléments ronds de petite taille (des aimants dans notre cas). L'idée derrière ce choix est d'encourager la manipulation d'éléments physiques pour garder le contenu

2. Source : Reactable

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

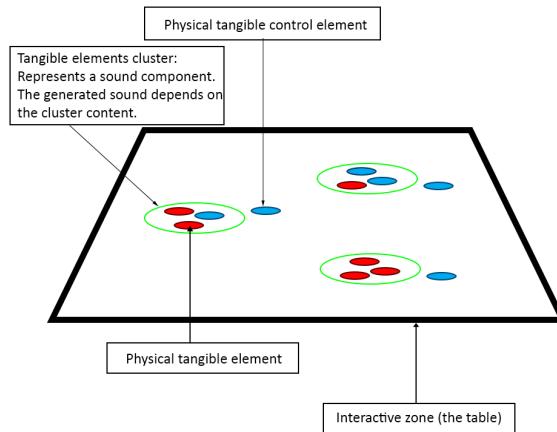


FIGURE 4.3 – ReARTable : Illustration du concept

Une fois les éléments détectés, regroupés et identifiés, l’élément sonore associé peut être créé. La création de l’élément sonore se fait simplement via la transmission d’un message OSC³ à un serveur Sonic Pi préalablement démarré. Ce message contient l’identifiant unique de la boucle que Sonic Pi doit démarrer. Pour chaque élément sonore que l’application peut créer, Sonic Pi possède une fonction à exécuter que nous avons préalablement créée. Toutes les communications entre l’application et Sonic Pi utilisent ce protocole ce qui permet de démarrer/arrêter/modifier certaines parties du son en direct.

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

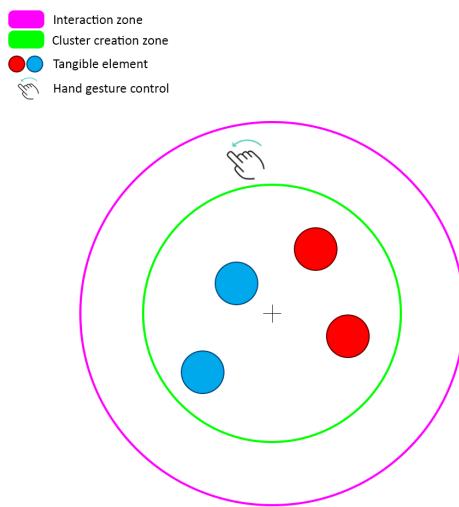


FIGURE 4.4 – Schéma représentant la création d’un son avec zone d’interaction

La dernière étape du développement de l’application concernait la visualisation de la musique générée. Cette étape n’a finalement pas été aboutie par manque de temps. L’idée était d’utiliser le spectre du son

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

et les différentes fréquences qui le compose, récupérables à l'aide d'une transformée de Fourier[4] ⁴, pour créer une visualisation globale basée sur les fréquences avec des variations visuelles en fonction de la hauteur, du tempo et de tous les autres paramètres du son qu'il est possible d'extraire et d'utiliser.

Vous trouverez Figure 4.5 un aperçu de l'état actuel de l'application.



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, nous avons eu l'occasion d'aborder des problématiques comme l'extraction et la numérisation de document (par exemple dans le cas de l'écriture d'une lettre en réalité augmentée) en tant que cas d'utilisation de **PapART**. Il était donc opportun de développer une preuve de concept de cette fonctionnalité sous la forme d'une application de réalité augmentée spatiale.

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 faudrait extraire le document), la présence d'éléments distinctifs (tels que des marqueurs fiduciaires ou des ronds colorés de petite taille) etc.

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.
- Déetecter 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 donc avons explorer plusieurs solutions sollicitant différents procédés dans le but d'essayer de trouver une solution adéquate à ce problème.

Détection de document basée sur des marqueurs colorés

Le premier prototype de détection que nous avons conçu utilise beaucoup de connaissances à priori sur le document afin de faciliter la détection. Ainsi, le document cible est muni de lignes d'éléments ronds, colorés, de petite taille, dans un ou plusieurs de ses coins (Figure 4.6a). Les éléments ronds de petite taille sont détectés grâce à **PapART** en réalisant une convolution de l'image par un filtre permettant la détection de ces derniers dont le principe est détaillé section 5.1.1.

Une fois les éléments détectés, ils sont regroupés en différentes lignes (Figure 4.6b). Une ligne est définie par un regroupement d'éléments dont l'écart entre chaque élément 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 (Figure 4.7). Si un autre élément "dérive" il est rejeté et la ligne est créée. Cette ligne est ensuite utilisée pour calculer deux vecteurs, dont un est confondu avec la ligne et le deuxième est perpendiculaire au premier (Figure 4.6c).

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

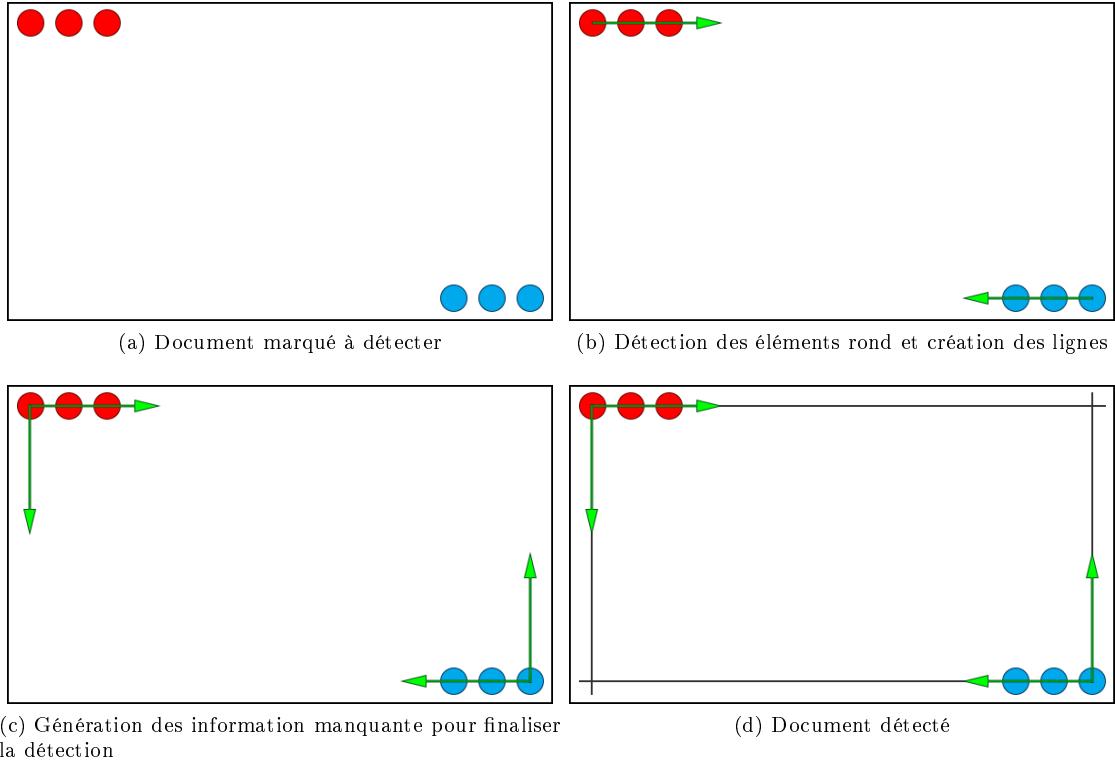


FIGURE 4.6 – Détection de document étape par étape

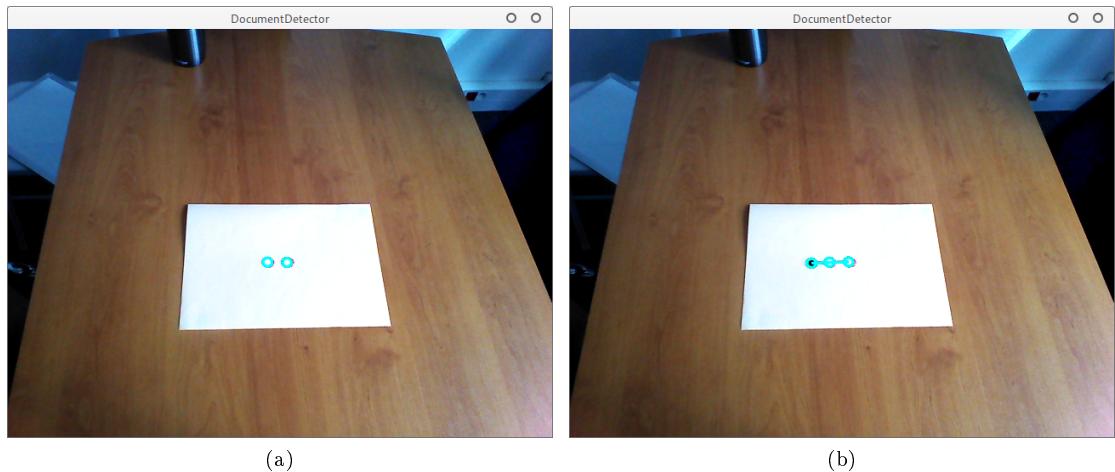


FIGURE 4.7 – Détection et création d'une ligne à partir d'éléments colorés ($x = 3$)

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

Une simple connaissance à priori de la distance des éléments ronds par rapport au coin permet de résoudre ce problème. Toutefois, si l'on souhaite obtenir une détection plus précise, il est conseiller d'utiliser des algorithmes de détection de contour couplés à des algorithmes d'extraction de lignes qui permettront de retrouver de façon plus fidèle le coin du document. Cette re-détection du coin est détaillée section 4.2.2.

Détection de document - Canny et transformée de Hough

Avec peu ou pas de connaissance a priori, la détection de document devient un problème compliqué et bien connu dans le domaine du traitement d'image surtout lorsqu'un besoin de temps réel vient s'ajouter à la tâche.

L'algorithme de Canny[5] est un filtre de détection de contours, permettant d'extraire d'une image (Figure 4.8a) des contours très précis (Figure 4.8b) 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, où 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 issu de faibles contours tout de même détectés. C'est pourquoi il faut effectuer une étape de floutage préalable afin de lisser les zones à faibles contours.

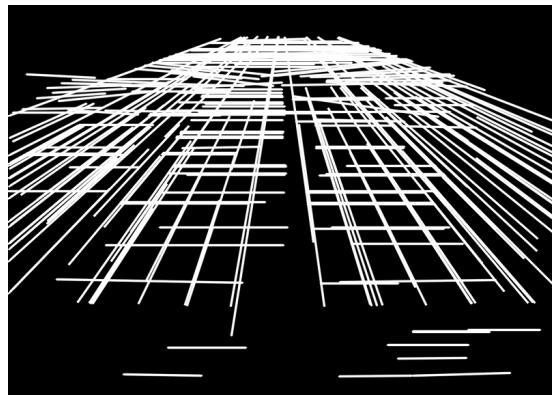
Une fois les contours détectés, il est possible d'essayer d'extraire directement le document mais c'est une tâche complexe car elle requiert d'analyser les contours. Cette action peut être facilitée en utilisant une technique appelée transformée de Hough[8]. En effet, la transformée de Hough permet d'extraire n'importe quelle forme à partir d'une image contour en utilisant les propriétés mathématiques de celle ci. Dans notre cas, où nous souhaitons extraire des lignes droites, les propriétés mathématiques utilisées correspondent aux coordonnées polaires considérées comme plus robuste que l'équation de la droite (Figure 4.8c). Il ne reste qu'à filtrer les lignes afin de trouver des potentiels documents dans une image.



(a) Image originale



(b) Canny - Détection de contours



(c) Transformée de Hough - Détection de lignes

FIGURE 4.8 – Détection de lignes : Canny + Hough⁵

Cette succession de traitements est cependant lourde et peut difficilement être effectuée en temps réel sur des images haute résolution.

Dans notre cas, nous nous sommes servis de ces deux algorithmes mais uniquement sur des parties d'image (de petite résolution) de façon à améliorer une première détection grossière effectuée notamment 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 potentielle des coins pour extraire des sous-images centrées sur ces coins (Figure 4.9a) que nous seuillons afin d'obtenir une image binaire (Figure 4.9b). Ensuite, nous appliquons les deux algorithmes mentionnés plus tôt, à savoir la détection de contours (Figure 4.9c) puis la détection de

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

lignes (Figure 4.9d). Après cela, nous filtrons 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 ainsi trouver le coin dans notre sous image (Figure 4.9e).

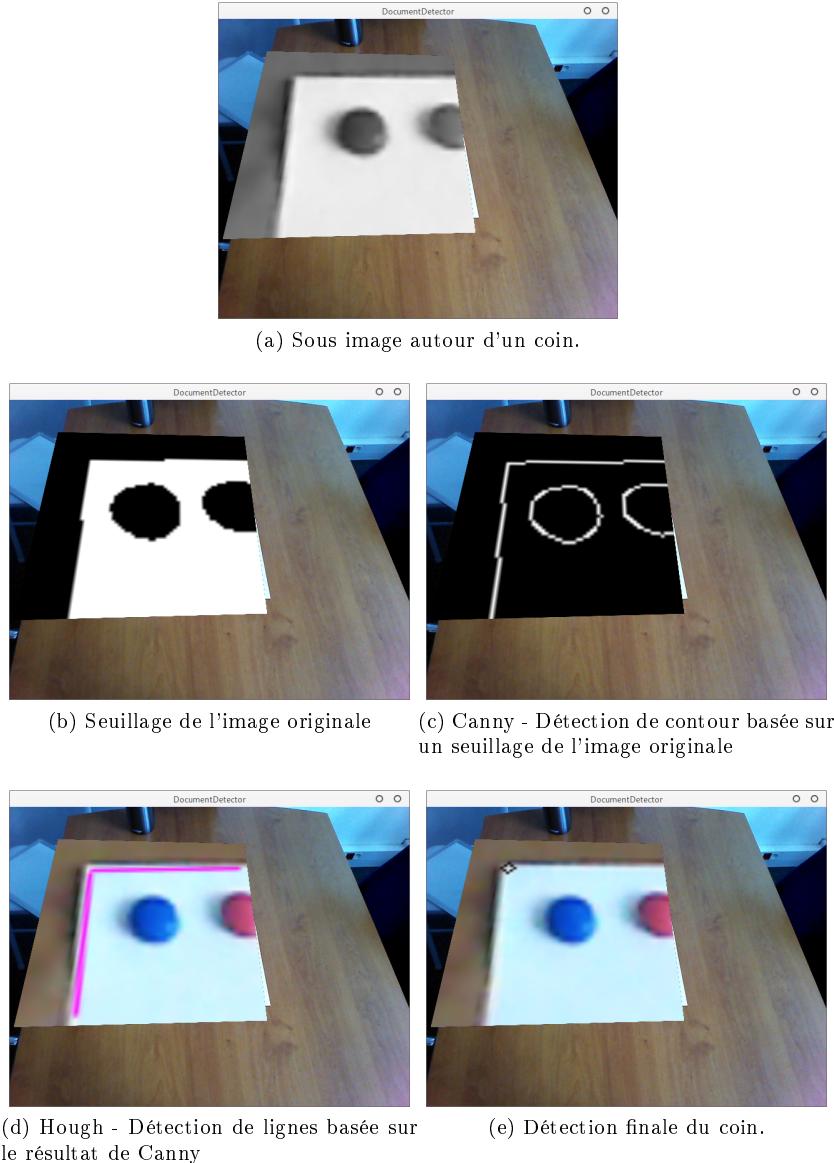


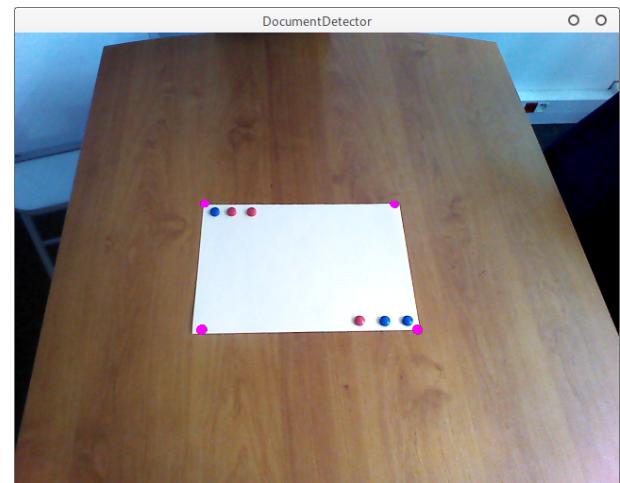
FIGURE 4.9 – Affinage de la détection d'un coin du document

4.2.3 Bilan

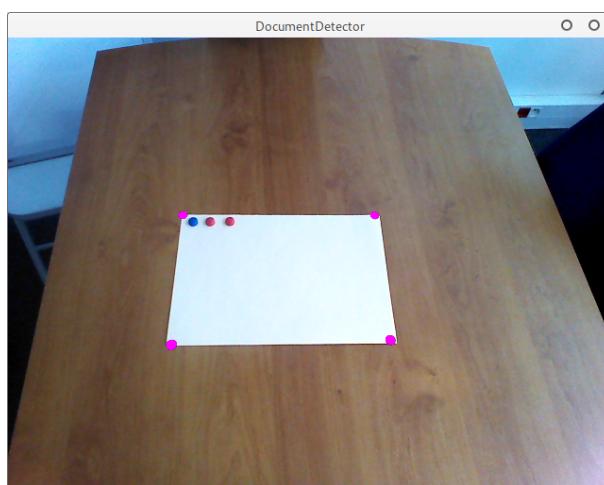
Pour finir, un comparatif des différents résultats obtenus est présenté sur la Figure 4.10. On peut observer quelques points notables : En utilisant Canny puis Hough pour améliorer la détection d'un coin, l'extraction est plus fidèle au document réel (Figure 4.10c et 4.10d). En effet, les coins détectés sont plus précis qu'avec des hypothèses effectuées à priori (Figure 4.10a et 4.10b). Toutefois, cette méthode est moins rapide car elle requiert de nombreux calculs supplémentaires. Lorsque utilisée en temps réel, une détection avec connaissance à priori du modèle sera bien plus robuste, car elle ne dépend pas d'une deuxième détection qui à beaucoup de chance d'échouer (seuillage, détection de contours, détection de lignes, intersection entre deux droites puis obtention finale du coin). Ainsi, la méthode à utiliser variera avec les cas d'utilisation. Par exemple, dans le cadre d'une application de scan de document, une méthode d'extraction de document plus précise sera envisagée, mais pour une estimation de pose et un suivi de document il sera préférable d'utiliser une détection plus rapide et robuste.



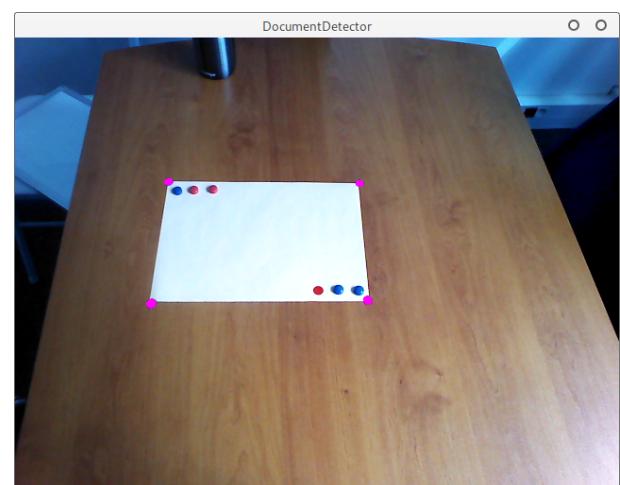
(a) Avec connaissance à priori : Taille du document et distance élément ↔ coin



(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 versions de la détection de document.

Prototype haute performance

Dans le cadre d'application qu'est la réalité augmentée spatiale, les interactions jouent un rôle majeur dans l'expérience de l'utilisateur. Ainsi, 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 que dans le rendu ou dans le contenu, sans avoir besoin d'une puissance de calcul dépassant l'entendement, une optimisation à la fois logicielle, matérielle et architecturale est nécessaire. Cette optimisation a fait l'objet d'une grande partie de mon stage et m'a amené à développer un prototype dit "haute performance" des outils que propose RealityTech. L'optimisation logicielle s'est portée sur l'amélioration des performances des algorithmes de traitement d'image, bien connus pour être extrêmement consommateurs des ressources. Pour l'optimisation matérielle, la tâche fut quelque peu différente. En effet, nous nous sommes attelés à effectuer des mesures et des calculs sur la puissance théorique du matériel, la latence réelle des caméra ainsi que la rapidité de l'encodage des flux vidéos, pour arriver à établir les performances réelles qu'il nous était possible d'atteindre avec différentes combinaisons de matériel. Enfin, le développement du prototype s'est achevé avec la création d'une nouvelle architecture logiciel en microservices[7]. Le but était de créer un environnement modulaire réactif, où les services peuvent mourir sans pour autant mettre en péril tout le système et ainsi améliorer grandement la qualité générale des outils fournis.

5.1 Amélioration logicielle

De nos jours, les optimisations font l'objet de développements ciblés et très spécifiques, se concentrant la plupart du temps sur l'amélioration d'un unique point crucial d'un algorithme ou d'une application. Dans notre cas, l'optimisation logicielle a surtout été effectuée au niveau des algorithmes de traitement d'image, omniprésents et indispensables à la technologie. La réalité augmentée spatiale a besoin du monde réel pour exister, c'est pourquoi le matériel se compose de nombreux capteurs (caméras) pour l'analyser et que de nombreux algorithmes de traitement des données captées (images) sont mis en place. Après une rapide analyse du logiciel existant, il est indéniable que le traitement le plus utilisé est la convolution d'une image par un filtre qui possède un nombre incalculable d'applications. 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).¹*

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 de celle-ci telles que la netteté, le flou ou les zones de fort gradient (les contours) par exemple (Figure 5.1). Étant donné que nous travaillons avec des images définies 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 appliqué 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 calculer 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 (Algorithm 1). Prenons un exemple concret. 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 afin de calculer la valeur du pixel 3,3 dans la nouvelle image soit :

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

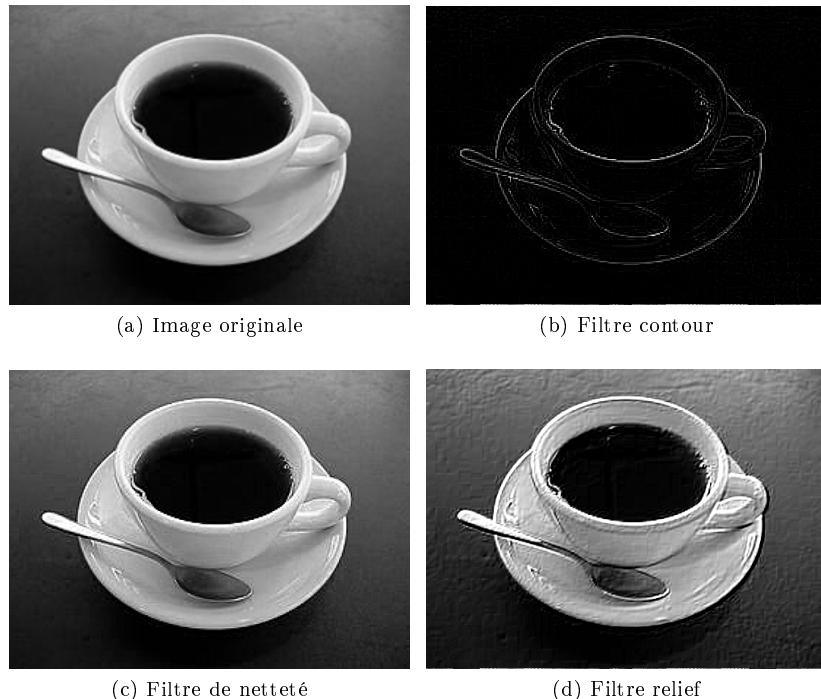


FIGURE 5.1 – Différents filtres de convolution appliqués à une image.

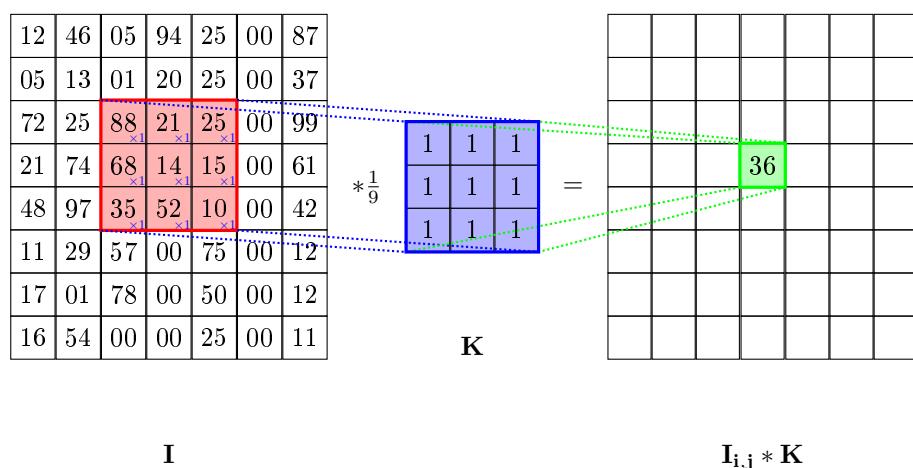


FIGURE 5.2 – Convolution d'une matrice (image) (I) par un filtre (K)

Algorithm 1 Convolution d'une image par un filtre

```

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

```

Comme on peut s'en rendre compte dans le pseudo code proposé ci-dessus (Algorithme 1), l'image résultat est une nouvelle image (indépendante de l'image d'origine) dont chaque pixel a été 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 façons 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 dans le Chapitre 2. La seconde méthode d'optimisation consiste à légèrement changer l'algorithme de convolution : la convolution est séparée en deux filtres distincts[23], un horizontal et un vertical, qui sont successivement appliqués à l'image origine (Figure 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 $O(N^2 M)$ puisqu'au lieu de parcourir pour chaque pixel un voisinage de taille $M \times M = M^2$ il suffit de parcourir $2 \times M$ pixels soit M .

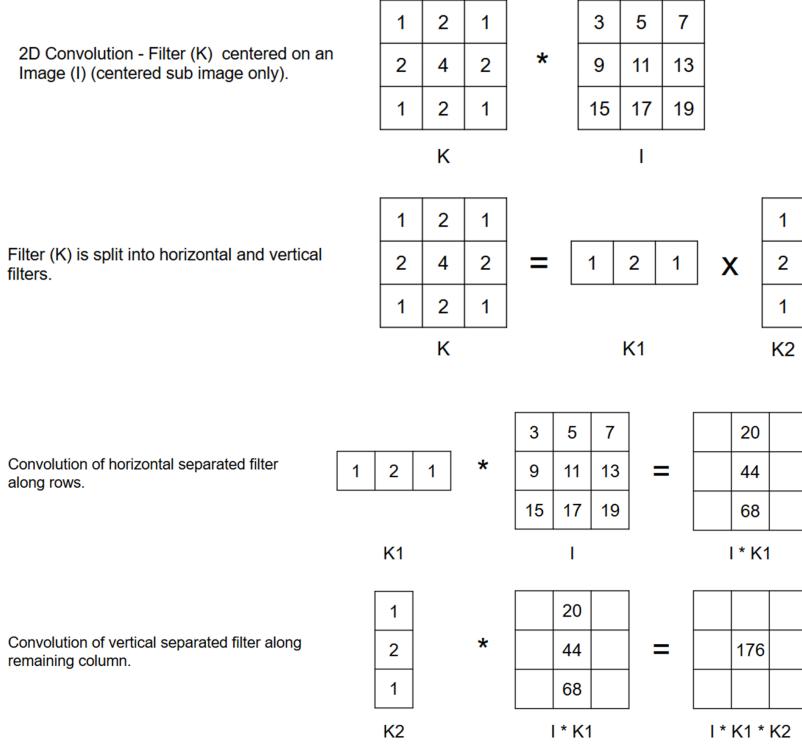


FIGURE 5.3 – Exemple d'un filtre de convolution appliqué successivement horizontalement puis verticalement

Nous avons choisi de n'effectuer que l'optimisation sur carte graphique car la deuxième méthode comporte un gros coût en complexité et 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 deux sous-filtres $K1, K2$ tel que $K = K1 \times K2$.

Pour pouvoir développer ladite 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, trois langages (ou sous-langages) se sont démarqués : OpenCL[9], OpenGL ES[10] et CUDA[21]. Nous avons donc choisi d'implémenter trois versions de l'algorithme de convolution naïf (non séparé) utilisant chacun de ces langages, afin d'en évaluer et comparer les performances.

5.1.3 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 ainsi que la gestion des périphériques s'exécutant sur l'hôte et la partie calcul permettant de compléter lesdites tâches s'exécutant sur les périphériques. La partie hôte est écrite en C tandis que la partie noyau est écrite en OpenCL-C. Il faut donc bien différencier hôte et périphérique (Figure 5.4). Dans notre cas d'utilisation, l'hôte représente le processeur et permet de transmettre les données aux périphériques qui, ici, correspondent à une ou plusieurs cartes graphiques.

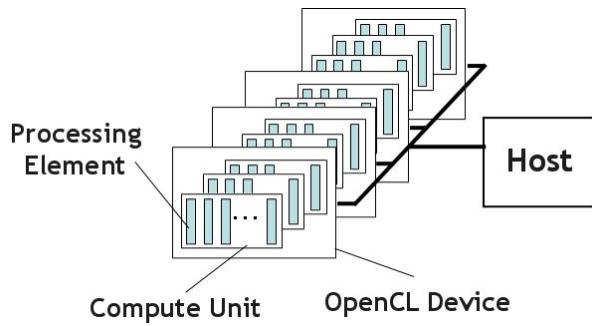


FIGURE 5.4 – Schéma OpenCL - Hôte et périphériques²

Nous nous sommes intéressés 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 puisqu’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.

5.1.4 OpenGL (ES)

OpenGL est une interface de programmation multi-plateforme et multi-langage permettant de faire le rendu de scènes 3D. En tant qu’interface, il est possible de l’implémenter de façon logicielle mais elle a été conçue pour être implantée de manière matérielle afin de profiter au mieux des accélérations dédiées disponibles. Ainsi, c’est grâce à ces implantations qu’OpenGL fournit un *pipeline* programmable de rendu ultra performant. C’est justement par le biais de ce dernier et plus spécifiquement via le code hôte et les *shaders*, des programmes informatiques servant à paramétriser des étapes du rendu, qu’il est possible de transmettre des instructions et des données à la carte graphique (Figure 5.5)

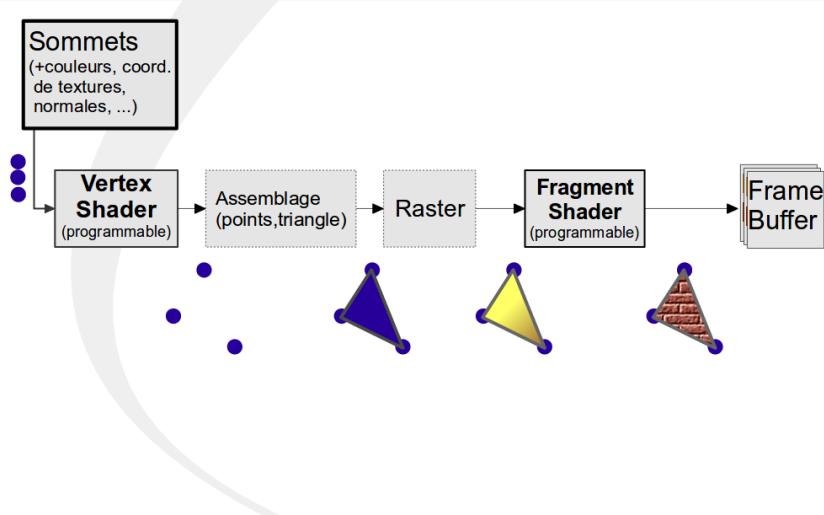


FIGURE 5.5 – Pipeline de rendu - OpenGL³

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 l’optique d’opérer des traitements sur des images 2D, il est nécessaire d’en détourner l’utilisation. En effet, 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* de rendu. Une fois activé, le *vertex shader* est programmé pour ne rien faire. Les étapes d’assemblage et de rastérisation sont, de ce fait, très rapidement

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

achevées et l'étape du rendu par fragment peut alors débuter. L'image de sortie du rendu est composée dans le *fragment shader*, c'est donc ici que nous avons accès à chacun des pixels composant l'image finale. Le code présent dans ce *shader* permet d'effectuer, pour chaque pixel, le calcul de la convolution. 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 (ES) est largement compatible avec toutes les plateformes et est très largement utilisé. Cependant, contrairement à OpenCL, les performances d'OpenGL ne dépendent que du matériel, ainsi, elles ne varieront pas, ou très peu, d'une architecture à un autre. Ainsi pour une puissance de calcul théorique identique et mais une architecture différente les performances seront quasiment équivalentes.

5.1.5 CUDA

A la différence d'OpenCL et d'OpenGL, CUDA n'est pas seulement un langage de programmation mais bel et bien une architecture de traitement parallèle développée par NVIDIA. Son unique but est d'exploiter la carte graphique à son 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. Là où CUDA se démarque c'est dans le modèle qu'il propose : les tâches (*threads*) sont regroupées en blocs (*blocks*) à l'intérieur desquels la mémoire est partagée. De plus, chaque bloc s'exécute sur exactement une unité de calcul (Figure 5.6). Par ailleurs, la mémoire est unifiée (Figure 5.7), de ce fait, les CPUs et les GPUs travaillant ensemble ont accès à la même mémoire, ce qui permet d'éviter bon nombre de copies et ainsi réduire considérablement les temps de transfert de données.

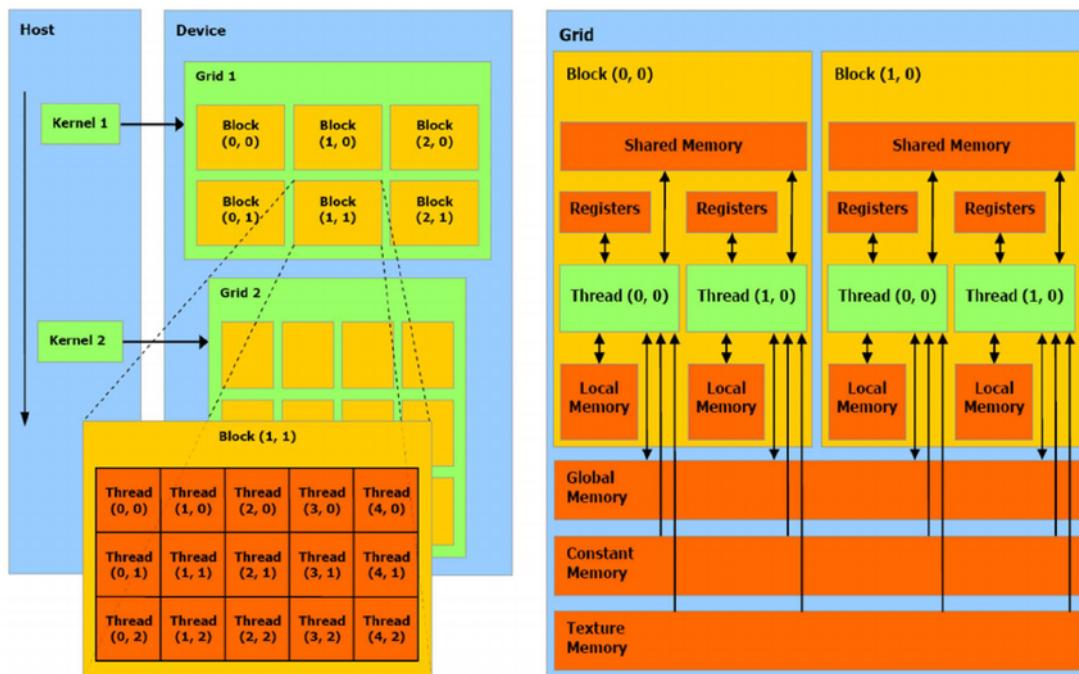


FIGURE 5.6 – Représentation schématique de l'architecture CUDA⁴

4. Source : NVIDIA CUDA - Unified Device Architecture

UNIFIED MEMORY

Dramatically lower developer effort

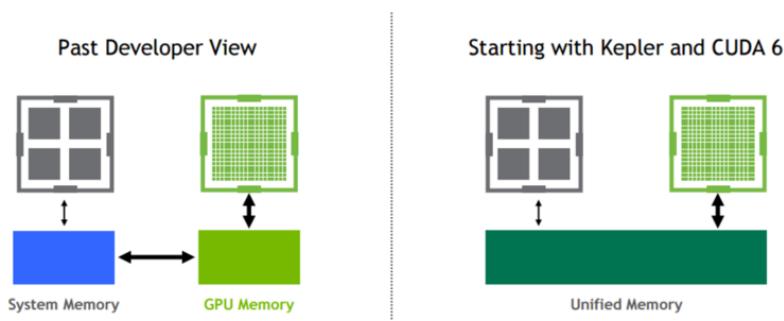


FIGURE 5.7 – Mémoire séparée vs mémoire unifiée⁵

CUDA étant une architecture matérielle, seules les cartes graphiques NVIDIA récentes en sont équipées, ce qui, contrairement aux deux autres, ne la rend absolument pas multi-plateforme.

5.1.6 Tests de performance

Afin de comparer les différentes solutions, nous avons réalisé des tests de performance du même algorithme de convolution, que nous avons implémenté dans les différents langages cités et sur différentes machines avec des configurations bien différentes. Le but de ces tests était, dans un premier temps, d'observer l'impact 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 obtenus en fonction des différentes plateformes.

Note : L'algorithme de convolution implémenté est la version non séparé où le filtre de convolution est considéré comme une matrice.

Note 2 : Nous avons choisi de ne pas réalisé les tests de performance OpenCL car nous avons jugé que cette technologie comportait beaucoup de trop de défauts et n'était donc pas pertinente.

Dans ce test de performance, nous avons mesuré plusieurs choses :

- **Le temps de transfert** des données de l'hôte au périphérique. Cette mesure est importante car elle permet d'évaluer le coût des opérations de transfert et de ce fait 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 brutes de l'algorithme par langage, nous avons mesuré le temps nécessaire au filtrage d'une image avec un noyau donné. Le temps de calcul ne prend en compte que le strict minimum des opérations nécessaires au calcul. Les allocations de variable, de mémoire etc. ne sont donc pas prises en compte dans ce temps.
- **La "bande passante"** du traitement entier, comprenant transfert calcul et re transfert. Cette mesure donne une bonne idée de la rapidité des algorithmes car elle exprime le nombre de mégaoctet qu'il est possible de traiter en une seconde avec chaque implémentation.

Note : Les résultats présentés dans les tableaux 5.1, 5.2 et 5.3 ont été mesurés sur le kit de développement NVIDIA Jetson TX2 dont les spécificités sont les suivantes : CPU ARM 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 d'exposer ici seulement les résultats obtenus sur le kit de développement NVIDIA car il permet d'observer les performances de CUDA dans un environnement optimisé pour ce dernier cependant vous trouverez en Annexe 1 les résultats de ces mêmes tests effectués sur plusieurs ordinateurs avec des configurations différentes.

5. Source : NVIDIA CUDA - Unified Memory for beginners

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)
128x128	0,0655	0,0964	0,7737	0,8701	75,3224
256x256	0,2621	0,2222	1,8457	2,0679	126,7661
512x512	1,0486	0,8764	3,5266	4,4030	238,1503
1024x1024	4,1943	3,2107	9,1959	12,4066	338,0703
2048x2048	16,7772	12,7048	35,0284	47,7332	351,4786
4096x4096	67,1089	51,0330	139,0710	190,1040	353,0115
8192x8192	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)
128x128	0,0655	0,1386	0,9038	1,0424	62,8703
256x256	0,2621	0,0202	1,3858	1,4060	186,4475
512x512	1,0486	0,0194	4,2613	4,2807	244,9576
1024x1024	4,1943	0,0212	15,6039	15,6251	268,4337
2048x2048	16,7772	0,0215	60,8093	60,8309	275,8008
4096x4096	67,1089	0,0215	241,5410	241,5625	277,8117
8192x8192	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)
128x128	0,0655	8,3513	0,0000	8,3569	7,8421
256x256	0,2621	33,8604	0,0000	33,8698	7,7398
512x512	1,0486	150,1860	0,0000	150,2010	6,9812
1024x1024	4,1943	721,2130	0,0000	721,2310	5,8155
2048x2048	16,7772	3196,6300	0,0000	3196,6500	5,2484
4096x4096	67,1089	13130,9000	0,0000	13130,9000	5,1108
8192x8192	268,4350	53591,6000	0,0000	53591,7000	5,0089

Comme on peut s'en rendre compte, les gains de performance des deux versions optimisées de l'algorithme sont non négligeables par rapport à la version CPU naïve (Tableau 5.3). En effet, on observe que les algorithmes s'exécutant sur la carte graphique sont jusqu'à 55 fois plus rapide pour OpenGL ES (Tableau 5.2) et jusqu'à 70 fois pour la version CUDA (Tableau 5.1). Les cases vertes dans les tableaux indiquent que l'image a pu être traitée en pseudo temps réel avec une fréquence de rafraîchissement de 25 images par seconde (IPS) ou *Frames Per Second (FPS)*, ce 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 ce constat d'optimisation, on peut voir que la version CUDA et la version OpenGL affichent des résultats plutôt similaires puisqu'ils sont tout deux capables de traiter en temps réel des images de taille 1024x1024 pixels sans difficulté. Toutefois, on note quand même une différence flagrante entre ces deux versions. En effet, on peut observer le gain apporté 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 (pour des images haute résolution) sont environ deux fois plus rapide que leurs équivalents OpenGL ES, ce qui a un impact significatif sur les performances car ils correspondent à la majeure partie du temps d'exécution du programme.

5.1.7 Bilan

Au vu des résultats obtenus section 5.1.6 nous pouvons souligner 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 versions CUDA et CPU bénéficieront des améliorations que cette optimisation peut potentiellement apporter.
- CUDA et OpenGL fournissent tout deux des résultats semblables (avec une potentielle amélioration du côté CUDA) mais n'offrent pas les mêmes possibilités. Avec CUDA, l'algorithme ne peut tourner que sur des machines possédant une architecture compatible. Nous avons jugé que le gain apporté par rapport à OpenGL ES, qui lui est totalement compatible, 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 concerne 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 cruciales à améliorer.

Nous nous sommes donc intéressés à 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 procurés de nombreuses caméras différentes dont nous avons mesuré les latences sur plusieurs ordinateurs. Certains ordinateurs possédaient des capacités matérielles d'encodage vidéo (comme sur le NVIDIA Jetson TX2, obtenu pour l'occasion, et qui possède un module MSENC, un encodeur matériel⁶) ce qui nous a permis d'en évaluer l'impact sur la latence lors de l'obtention du flux vidéo. Mis à part le Jetson TX2 possédant une caméra embarquée, les mesures de la latence ont toutes été effectuées en utilisant GStreamer[11] avec la même commande d'obtention du flux afin d'éviter au maximum les différences de mesure. Aussi pour une avoir idée plus précise des capacités maximales qu'il était possible d'atteindre, nous avons désactivé toutes les options de compression etc. afin éviter tout traitement du flux pouvant causer une augmentation de la latence.

Pour mesurer la latence, nous avons utilisé la méthode dite *glass to glass*[26] qui est l'une des seules méthodes actuellement employée. 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 ce compteur, côté à côté (Figure 5.8). La latence est finalement obtenue en faisant la soustraction des deux temps affichés par les compteurs.

Cette méthode comporte certains défauts, le plus critique étant 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 à la milliseconde, ce qui est extrêmement rare. Ensuite, le taux de rafraîchissement et la latence de l'écran utilisé viennent également perturber les mesures. Dans notre cas, nous avons utilisé un chronomètre avec une résolution de l'ordre de 1 à 5 millisecondes⁷ et un écran 120Hz avec 1 milliseconde de temps de latence ce qui devait réduire les imprécisions introduites dans nos mesures. Aussi, au lieu de prendre une photographie, nous avons décidé de réaliser des vidéos ralenties en 240fps et d'afficher, en plus du chronomètre, une vidéo où 12 couleurs se succèdent à une fréquence 1Hz. Ainsi, en plus de la mesure du chronomètre, nous pouvons calculer grâce à la vidéo ralentie le nombre d'image qu'il faut pour qu'un changement de couleur dans la vidéo se reflète dans l'affichage du flux vidéo de la caméra. Étant donné que nous filmons à 240fps, chaque image de la vidéo où le changement de couleur n'est pas reflété, correspond à $(1/240) * 1000 = 4,16$ millisecondes. Ainsi, si sur la vidéo ralentie, un changement de couleur met trois images à 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

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

Dans un soucis de cohérence avec la partie optimisation logicielle, vous trouverez dans le tableau 5.4, un comparatif des différentes latences de caméra obtenues sur le kit de développement NVIDIA Jetson TX2. Les résultats obtenus avec différents ordinateurs sont, quant à eux, présenté en Annexe 2.

TABLE 5.4 – Latence (en ms) de plusieurs caméras mesurée en *glass to glass* - NVIDIA Jetson TX2

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

On s'aperçoit très rapidement que les tests de latence sont plutôt décevants. En effet, les latences sont toutes plus ou moins similaires même s'il y a quelques variations notamment en résolution *full HD* 1920x1080, ce qui ne permet pas d'émettre beaucoup d'hypothèse d'amélioration. On observe que même la caméra embarquée dite *Onboard* disposant de son propre circuit intégré sur la carte mère du TX2 n'apporte aucun gain significatif par rapport aux autres caméras. N'étant pas satisfait des résultats, nous avons décidé de mesurer la latence d'une caméra professionnelle Point Gray et la latence observée a été de seulement 8-10 millisecondes avec une résolution de 1280x1020.

Aucune réelle conclusion n'a pu être tirée de ces résultats mais il s'avère que la plupart des constructeurs de caméras non professionnelles n'étant pas dédiées à la vision par ordinateur ne concentrent par leurs efforts sur la réduction de la latence de ces dernières mais plutôt sur l'amélioration de la qualité de l'image, la fidélité des couleurs etc. N'étant pas disposés à nous munir de caméras professionnelles, les améliorations matérielles possibles sont limitées. En effet, seule la configuration de l'ordinateur permettant au système de fonctionner pourrait apporter de réels changements en termes de performances. C'est notamment ce que l'on peut observer dans les résultats des tests de convolution Annexe 1 où un ordinateur avec une puissance de calcul théorique plus élevée génère de meilleurs résultats.

5.3 Nectar - Architecture microservices

Pourachever le développement du prototype haute performance, nous avons choisi de réétudier 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 imposée comme solution de choix car elle répond à tous les besoins énoncés.

Une architecture en microservices 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'attribuer des ressources aux services les nécessitant lorsqu'un faible nombre d'entre eux est entrain de fonctionner. A la différence d'une bibliothèque classique où 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 notamment.

Réactivité Dans le cas d'une panne d'un service critique, tel que 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 prend un certain temps. En revanche, une architecture en microservices couplée à un gestionnaire de processus se chargera uniquement de relancer le service en panne et, s'il le faut, les services associés, de manière quasiment transparente pour l'utilisateur.

Modularité L'architecture en microservices offre une très grande modularité; chaque service peut être écrit dans n'importe quel langage et peut être intégré au système sans coût tant qu'il respecte l'*API* de communication inter processus s'il n'est pas totalement indépendant. Il est ainsi très facile pour n'importe qui de rajouter des modules venant 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 à l'utilisateur final. L'utilisation ou non du filtrage peut être contrôlée de façon automatique par un autre module gérant par exemple la qualité générale des traitements voulue par l'utilisateur.

Maintenabilité Lorsqu'un service est défectueux, le problème peut être très rapidement identifié 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 à la modularité de l'architecture, un service en maintenance n'affecte pas la stabilité générale du système.

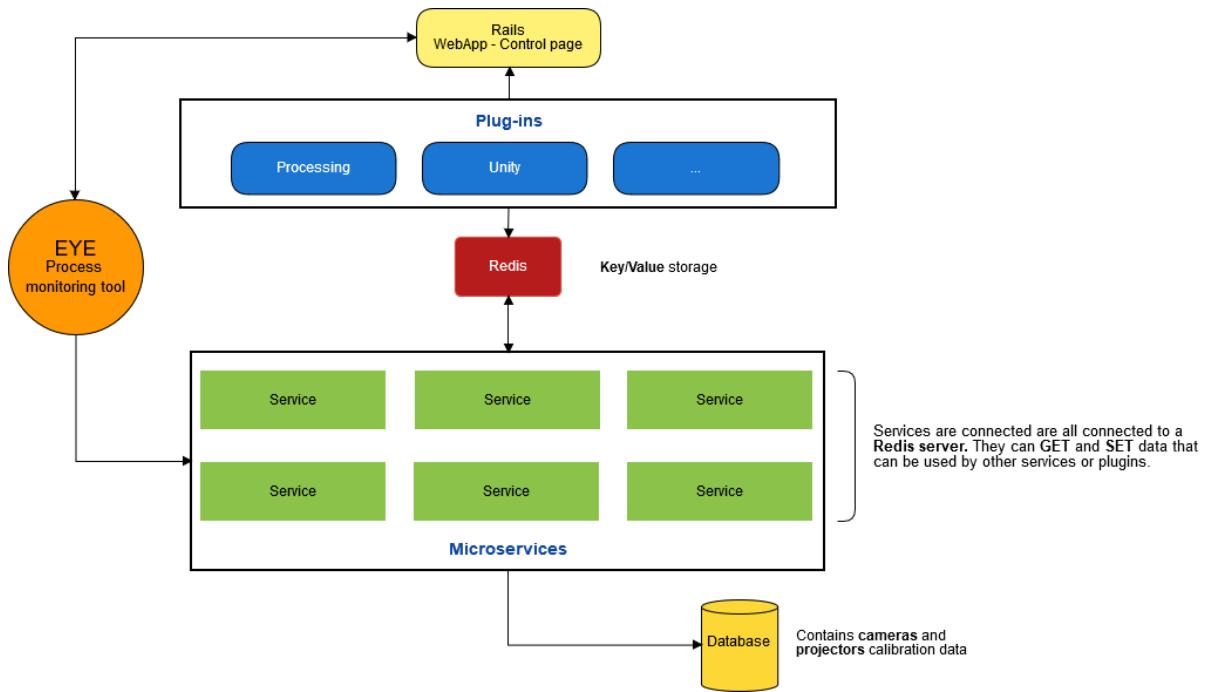


FIGURE 5.9 – Schéma de l'architecture en microservices développée

Pour mettre en place cette nouvelle architecture (Figure 5.9), nous nous sommes basés sur trois technologies principales :

- **Redis**[25] ou *REmote DIctionary Server* est un système de stockage clé-valeur qui, contrairement aux bases de données plus conventionnelles, stocke les données directement dans la mémoire vive (*Random Access Memory (RAM)*) de l'ordinateur ou dans la mémoire virtuelle et non pas directement sur le disque. Cette spécificité permet à **Redis** d'atteindre des performances inégalées par des bases de données classiques du fait du stockage en mémoire vive (bien plus rapide) et il s'impose donc comme un choix de qualité pour nos besoins.
- **Eye**[15] 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 de lancer/redémarrer/couper automatiquement des processus en fonction de leur état, de leur dépendance et de leur consommation en ressource (mémoire, CPU). Il est ainsi possible de définir, pour chaque service, de quoi il dépend et quelles sont les ressources maximales qui lui sont autorisées, ce qui permet d'éviter la mise en péril de tout le système lorsqu'un processus commence à consommer 99% de la mémoire disponible par exemple.
- **Ruby on Rails**[12] 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 microservices sont connectés à **Redis**. Ils peuvent chacun manipuler les données présentes et en écrire de nouvelles. C'est uniquement par ce biais que se fait la communication inter services. Ainsi, par exemple, un service accédant à la caméra ne transfère pas directement ses données à un service analysant l'image, mais envoie l'image courante de la caméra dans **Redis** qui est ensuite récupérée par cet autre service. En outre, **Redis** possède une *pipeline* événementiel. Ce *pipeline* permet aux clients **Redis** (services) écoutant ou publiant sur une clé de notifier ou d'être notifié d'autres / par d'autres services. De ce fait, dès que des nouvelles données seront publiées sur une clé écoutée par des clients, ceux-ci recevront instantanément lesdites données. C'est ce *pipeline* que nous avons choisi d'utiliser pour éviter l'attente active des services et ainsi améliorer les performances générales du système. Le phénomène d'attente active ce présente lorsqu'un processus vérifie constamment si une condition est vraie. Dans notre cas la condition pourrait être "est ce que des nouvelles données sont disponibles ?". Cette attente n'est pas bénéfique pour le système car les ressources sont précieuses et limitées et ainsi un processus ne peut se permettre de monopoliser ces derniers sous prétexte de vérifier si des nouvelles données sont arrivées d'où l'utilisation du *pipeline* événementiel.

Couplé à **Redis**, c'est **Eye** qui s'assure à tout moment que tous les services demandés par l'utilisateur sont en train d'être exécutés. Pour cela, il vérifie que le service est en marche mais aussi que toutes les dépendances sont satisfaites. **Eye** peut recevoir des commandes de l'application web qui est elle-même

7. Wikipédia - Mémoire virtuelle

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**.

5.4 Bilan

Après avoir réalisé puis testé indépendamment chacune des parties composant le prototype haute performance, la dernière étape consistait à tester ce dernier, de bout en bout, dans un cas d'utilisation réelle. Le but de cet ultime test étant d'évaluer si le prototype remplit, ou non, les objectifs que nous nous sommes fixés à savoir : performance, réactivité, modularité et maintenabilité. Pour effectuer ce test, il fallait développer une application utilisant ce prototype. Pour ce faire, nous avions besoin du module **Unity** le mettant à profit. Le développement du module **Unity** ainsi que de l'application est détaillé Chapitre 6. A ce jour, le module et l'application n'étant toujours pas terminés, le test n'a pas encore été réalisé.

Unity

Dans cette partie, il s'agira finalement de parler du développement du module **Unity** utilisant la nouvelle plateforme mise en place, dont il est question Chapitre 5. Ce développement fut la parfaite occasion de tester et mettre à l'épreuve cette dernière et d'avoir, par la même occasion, un retour réel sur son utilisabilité.

Dans un premier temps, nous discuterons des apports, des enjeux, mais aussi de la conception et de la cible du module. Nous en profiterons pour aborder les difficultés rencontrées, qu'elles soient liées à **Unity** ou à la nouvelle plateforme. Dans un second temps, nous nous attarderons sur le développement d'une application avec ce module, de façon à évaluer s'il répond aux besoins et s'il y répond de façon efficace. Pour finir, nous effectuerons une rapide comparaison entre la version actuelle et la version en développement des kits (**Unity** et **Processing**), afin d'avoir une évaluation dans des conditions réelles d'utilisation et peut être faire émerger des pistes d'amélioration de la version **Unity**.

6.1 Module Unity

L'objectif du module **Unity** est de permettre à ses utilisateurs d'exploiter la puissance (logicielle et matérielle) de **Nectar** de façon intuitive, afin qu'ils puissent développer des applications de réalité augmentée spatiale sans jamais avoir besoin de se soucier des problèmes liés à cette technologie, comme la problématique de calibration du couple caméra projecteur par exemple. Pour cela, nous avons créé dans **Unity** les composants et les comportements cruciaux du système tels que les caméras, les caméras de profondeur, les projecteurs, les utilisateurs, la table, et bien d'autres. En plus de résoudre bon nombre de problèmes pour l'utilisateur, ces composants rendent possible la représentation du monde réel (Figure 6.1). Cette représentation est très importante car, dans le domaine de la réalité augmentée spatiale où ce dit monde sert de base aux augmentations et, de ce fait, ne peut pas être négligé, en avoir une représentation virtuelle précise permet aux utilisateurs de concevoir leurs applications dans le même environnement que celui où elles seront projetées.

Le module **Unity** possède actuellement deux types de composants :

- Les composants modélisant les objets matériels du système de projection, correspondant aux différents dispositifs d'acquisition (caméras, projecteurs).
- Les composants modélisant la partie logicielle, correspondant aux divers services de traitement fournis par **Nectar**.

Le fonctionnement d'un composant, peu importe ce qu'il modélise, reste le même. Dans un premier temps, ce dernier crée un client et essaie de se connecter à la base de données **Redis**, base dans laquelle tous les services de **Nectar** si tant est qu'ils produisent des données stockent ces dernières. Si **Redis** est déficient, la connexion échoue et il s'agit alors d'une erreur critique car cela signifie par la même occasion que **Nectar** n'a pas pu démarrer ses services. Dans un tel cas, le composant envoie une requête HTTP au serveur web communiquant avec le gestionnaire de processus **Eye** dans le but de redémarrer **Redis**. Si **Redis** est toujours déconnecté, un message d'erreur critique est remonté à l'utilisateur qui ne pourra utiliser aucun des composants du module jusqu'à ce que **Redis** soit de nouveau opérationnel.

Une fois la connexion avec **Redis** établie, le composant, toujours par le biais d'une requête HTTP, questionne le serveur web sur l'état du service qu'il représente. Par exemple, le composant modélisant la caméra se renseignera sur l'état du service caméra. Si le service est hors ligne, une requête de démarrage peut être envoyée au serveur web qui la transmet instantanément à **Eye** afin de rendre opérationnel le service désiré. Si le service désiré n'existe pas ou ne peut être démarré, un message d'avertissement sur l'indisponibilité de ce dernier est remonté à l'utilisateur qui peut toutefois continuer à utiliser les autres composants du module ne dépendant pas de celui-ci. Lorsque le service est démarré, ou s'il était déjà en ligne, le composant peut finalement récupérer les données qu'il désire utiliser dans **Redis**. Pour ce faire, il est nécessaire que ce dernier en connaisse l'emplacement. **Redis** fonctionnant sur un système de

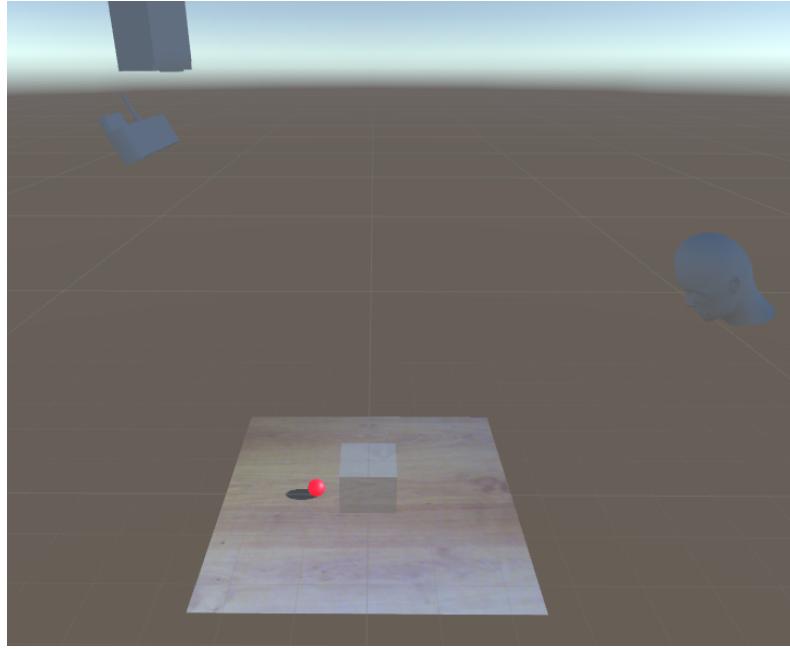


FIGURE 6.1 – Représentation du monde dans **Unity** - A droite la tête de l'utilisateur, au centre la table, en haut à gauche le couple caméra projecteur.

clé/valeur comme expliqué section 5.3, une connaissance a priori de la clé est requise. C'est pourquoi chaque composant possède un ou plusieurs champs configurables par l'utilisateur pour indiquer les clés à utiliser afin de récupérer les données dans **Redis**. Par exemple, le composant caméra qui doit à la fois récupérer les paramètres intrinsèques, extrinsèques et le format des données produites par la caméra, possède trois champs configurables dans **Unity** permettant d'indiquer les clés de stockage dans **Redis**.

On peut observer, encadrés dans la Figure 6.2, les trois points importants du module.

- En vert est représentée la zone où les composants nécessaires à la création d'application sont ajoutés afin d'en utiliser les fonctionnalités. Ici par exemple, nous utilisons une caméra, un projecteur et une caméra de profondeur rangés dans la catégorie *hardware*, un point de vue utilisateur *User-POV (Point Of View)* et différents services comme par exemple le suivi de feuille *Paper Tracker*. Nous avons aussi ajouté à cette scène différents *Renderer* permettant de visualiser les données acquises/générées par les dispositifs d'acquisition.
- En rouge, on retrouve la zone où il va être possible de contrôler l'état des composants et, si besoin, de démarrer les services comme mentionné précédemment. Dans notre exemple, toujours Figure 6.2, on peut observer le script de contrôle du composant projecteur. Ce script permet de spécifier quel type de dispositif on souhaite créer ; ici la valeur est *PROJECTOR*. On peut également voir l'état interne du composant ; ici le composant est dans l'état *WORKING*, ce qui signifie que ce dernier a réussi à se connecter à **Redis** et à récupérer les données dont il avait besoin pour fonctionner. On y retrouve aussi les clés où sont stockées lesdites données à récupérer, et un bouton permettant de démarrer/redémarrer le service en cas d'échec lors de l'initialisation.
- En bleu, on peut voir la zone où les notifications de tout type (erreur, avertissement, fonctionnement) sont remontées à l'utilisateur lui indiquant constamment l'état des services et les problèmes détectés. Chaque notification commence par le nom du composant la générant suivi du message, afin de garder une certaine lisibilité.

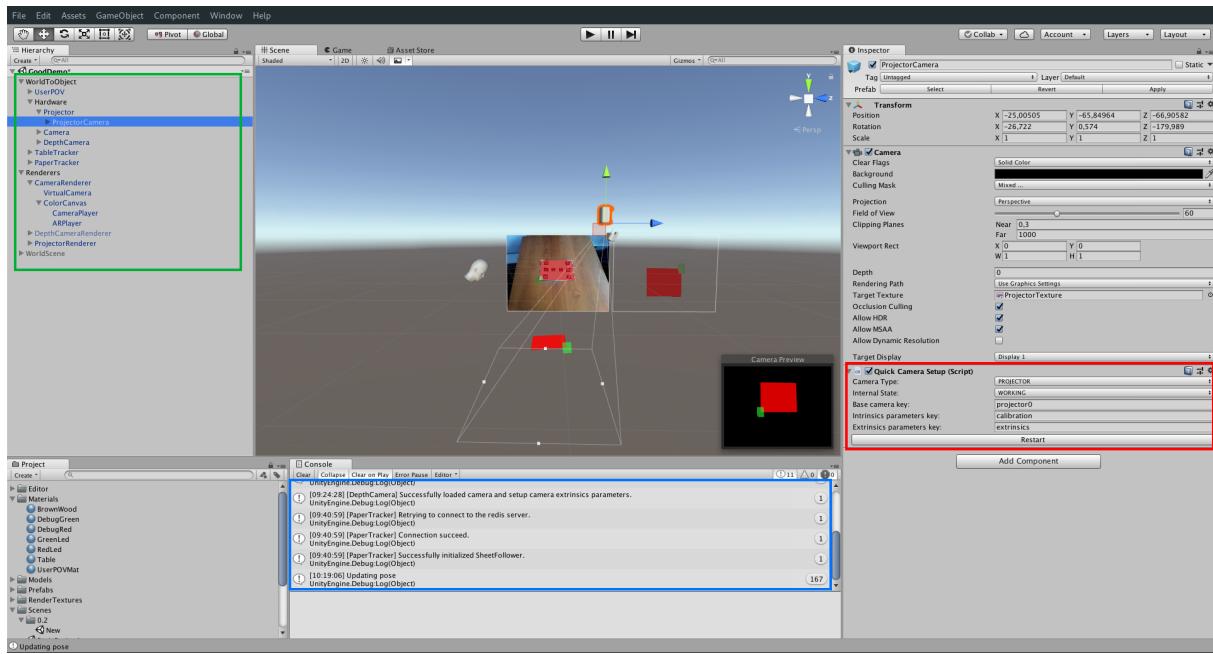


FIGURE 6.2 – Vue globale du module **Unity** et de ses différents composants

Tous les composants et scripts ont été spécialement créés pour s'exécuter directement dans l'éditeur **Unity** afin de faciliter considérablement le développement pour les utilisateurs. Ainsi, il n'est pas nécessaire de construire et d'exécuter l'application pour avoir un retour sur la création en cours. Cela permet donc de gagner un temps précieux et d'accélérer le développement.

Le mode éditeur possède cependant quelques défauts majeurs qui ont requis une modification à la fois du module et à la fois des services **Nectar** afin de fonctionner correctement. En effet, à l'origine les services **Nectar** utilisaient le *pipeline* événementiel de **Redis** pour qu'ils n'aient pas besoin d'effectuer de l'attente active et ainsi consommer 99% des ressources, attendant indéfiniment de nouvelles données. Ce *pipeline* permettait de souscrire à certaines clés, afin d'être notifié lorsque de nouvelles données étaient publiées sur ces dites clés. Ce *pipeline* était très performant mais a posé quelques problèmes avec le fonctionnement en éditeur de **Unity**. En effet, en mode éditeur, les performances sont très amoindries¹ et ce dernier n'était donc pas capable de traiter suffisamment vite les événements reçus. Les données s'accumulaient sans être consommées par les clients. **Redis** stockant ces données directement dans la mémoire vive comme expliqué section 5.3, l'accumulation causait une surcharge et mettait en péril toute la base. Il était donc nécessaire de couper la connexion avec le client du composant se trouvant dans l'incapacité de consommer ces données.

Pour que cette version puisse fonctionner correctement, nous avons décidé d'ajouter à tous les services **Nectar** la possibilité, ou non, d'utiliser le pipeline événementiel. Cette option permet ainsi d'utiliser le pipeline classique où les données ne s'accumulent pas mais écrasent les anciennes déjà existantes. Du côté du module, le même comportement a été implémenté pour tous les composants fonctionnant directement dans l'éditeur afin de pouvoir utiliser les services. Cette solution résout bel et bien le problème de fonctionnement en mode éditeur mais comporte encore quelques défauts.

Le pipeline ne se basant plus sur des événements, la seule façon de mettre à jour les composants est d'utiliser la boucle principale générée par **Unity**. De ce fait, comme on peut le voir dans l'ordre d'exécution des différentes fonctions de **Unity** que vous trouverez en Annexe 3, celles de la boucle que nous pouvons utiliser sont celles de mise à jour (*Update*). Le problème avec ces fonctions réside dans le fait que, lors de l'utilisation dans l'éditeur, celles-ci ne sont pas systématiquement appelées (comme elles le seraient en mode jeu ou quand l'application est construite puis exécutée) afin d'alléger la consommation des ressources et permettre d'avoir un meilleur confort de développement. Ainsi, il en résulte que les composants ne se mettent à jour que lorsque **Unity** reçoit des événements et décide de se mettre à jour, comme c'est le cas si une valeur telle que la position d'un objet est modifiée.

Une des dernières difficultés rencontrées durant le développement du module **Unity** a été la gestion des formats de données envoyées par les différents services et plus spécifiquement les matrices. Chaque service étant indépendant, les données qu'il envoie ne sont pas toujours au même format que celui attendu par **Unity**. En effet, si on prend le cas des matrices, celles du moteur 3D de **Unity** sont dites *row major*

1. Unity : Low performance in editor

ce qui signifie que les données sont indexées en fonction des lignes alors que celles du moteur 3D de Processing sont dites *column major*, et donc indexées en fonction des colonnes. Il a donc fallu appliquer des traitements spécifiques en fonction des données reçues afin d'obtenir une visualisation cohérente de ces dernières.

Une fois la majeure partie des problèmes du module résolue, il était temps d'enfiler le costume d'utilisateur développeur client de RealityTech et d'expérimenter le développement d'une application de réalité augmentée spatiale avec ce module. Vous trouverez les détails de ce développement section 6.2.

6.2 Illusion de projection

Après avoir achevé la première version du module Unity, dans une optique de test de ce dernier mais aussi dans le but de proposer une preuve de concept, j'ai été chargé de développer une application simulant une illusion de projection afin de créer un effet de fausse transparence. Cette preuve de concept avait été demandée par de potentiels clients de la société, rencontrés au Laval Virtual, afin qu'ils puissent effectuer de la présentation de produit interactive et plus spécifiquement de la présentation de parfum. Ceci est typiquement un cas d'utilisation auquel il était difficile de répondre avec la version Processing du kit de développement car elle ne permettait pas facilement de manipuler des objets 3D, de gérer la transparence, de faire le rendu de la scène en plusieurs étapes etc.

Afin de pouvoir créer une illusion de projection, il est nécessaire d'avoir une connaissance a priori de la surface de projection, permettant d'appliquer les déformations nécessaires à la géométrie à projeter. Grâce à la représentation du monde réel directement dans Unity, nous pouvons aisément disposer des éléments de géométrie qui nous sont nécessaires (Figure 6.3). Ainsi, seuls les éléments purement virtuels à projeter sont à rajouter dans la scène 3D.

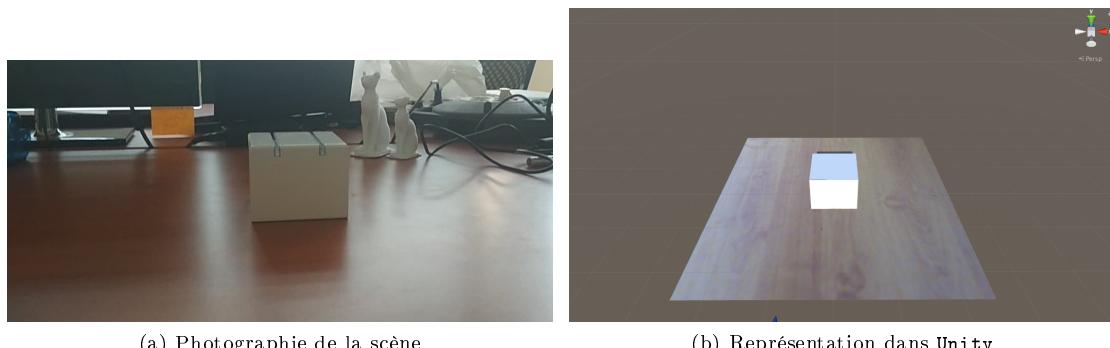


FIGURE 6.3 – Monde réel et sa représentation dans Unity

Aussi, une illusion de projection ne fonctionne que pour un point de vue donné (Figure 6.4) car la perception de la géométrie du monde réel diffère selon ce dernier. Les déformations à appliquer ne seront donc pas les mêmes selon le point de vue.



FIGURE 6.4 – Illusion de projection

Ainsi, les éléments requis pour commencer à créer notre illusion sont : la modélisation du monde réel, ici la table et sa texture (pour que l'effet de transparence ait du sens) ainsi que le flacon de parfum que l'on souhaite faire apparaître transparent, la position de l'utilisateur observant la scène et le point de vue du projecteur. Tous ces éléments font partie du module de développement, il a donc suffit de les ajouter à la scène pour en obtenir la représentation (Figure 6.5). De plus, afin de vérifier la validité de l'effet nous avons choisi d'ajouter une sphère virtuelle rouge derrière le flacon, invisible du point de vue de l'utilisateur, de sorte à ce qu'elle apparaisse lorsque l'effet de transparence sera mis en place.

Note : Dans notre cas, nous n'avions pas le modèle 3D du parfum imprimé à disposition, nous avons donc choisi de le remplacer par un cube cependant cela n'a aucun impact sur la simulation de l'effet de transparence.

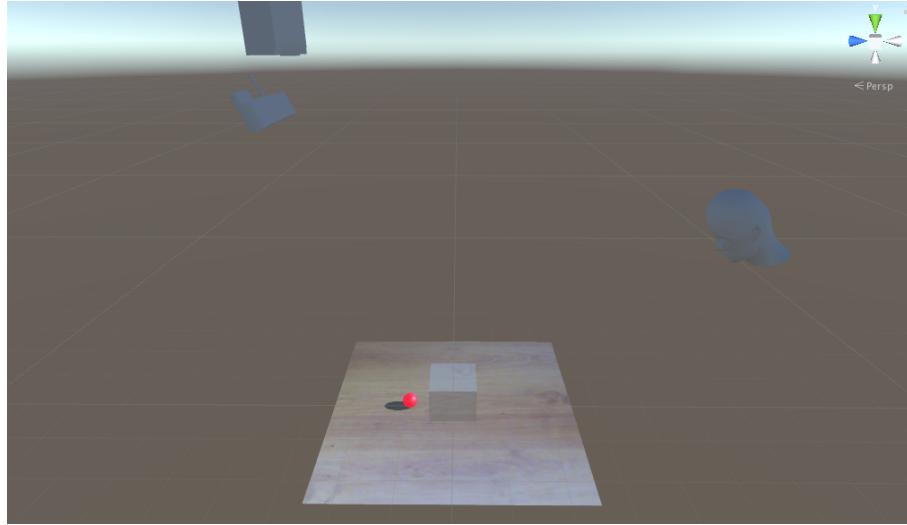


FIGURE 6.5 – Scène modélisée pour tester l'effet de transparence

Pour réaliser un tel effet, le rendu se fait en plusieurs étapes. Dans un premier temps, le rendu de la scène est effectué depuis le point de vue de l'utilisateur avec l'objet d'intérêt transparent et en ignorant sa représentation physique (Figure 6.6a). Ce rendu se fait *offscreen* et est ensuite récupéré dans une texture. La prochaine étape consiste à utiliser la texture ainsi créée et à la projeter sur la géométrie de la scène, comprenant cette fois ci l'objet physique, afin qu'elle subisse les déformations adéquates pour créer l'illusion (Figure 6.6b, 6.6c et 6.6d). On peut observer qu'une ombre est apparue. En effet, pour renforcer l'effet qu'allait produire l'illusion nous avons choisi d'ajouter une lumière afin d'obtenir des ombres cependant nous avons oublié d'activer leurs rendus pour les objets transparents. Pour pouvoir projeter cette texture sur la scène, il a fallu implémenter un principe de vidéo projecteur. Ce vidéo projecteur ne devait projeter la texture que sur le modèle physique et non pas sur sa représentation transparente. Enfin, la dernière étape consiste à faire le rendu de la scène, avec la texture projetée sur la géométrie vue par le projecteur (Figure 6.6e) pour ainsi pouvoir projeter ce résultat dans le monde réel.

Pour des raisons logistiques, je n'ai malheureusement pas eu l'occasion de prendre une photo de ce dispositif en train de fonctionner.

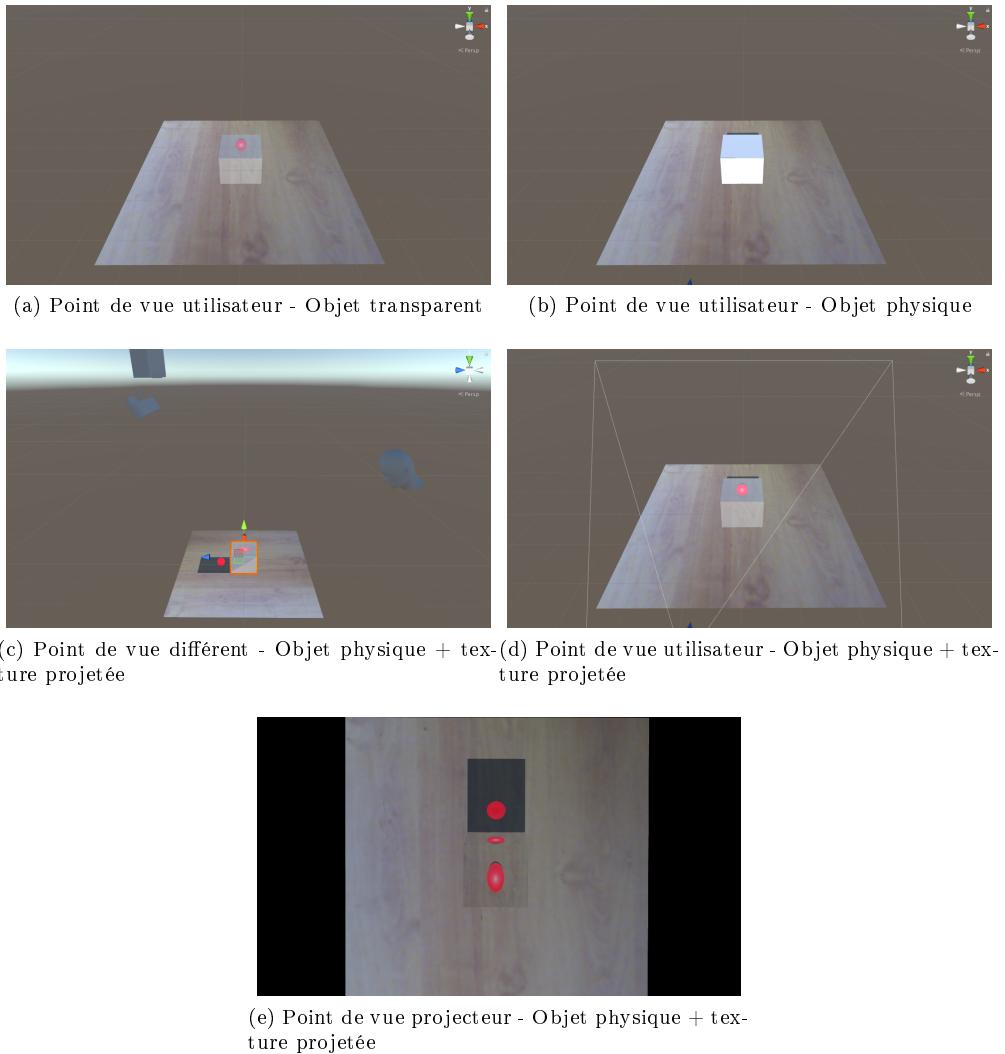


FIGURE 6.6 – Étapes de la création de l’illusion de projection

Dans l’état actuel des choses, l’illusion est encore incomplète pour plusieurs raisons. La première étant que nous utilisons actuellement un système composé d’un seul projecteur positionné au dessus de l’objet. Il est de ce fait impossible de couvrir toute la surface du cube ou de n’importe quel objet en général avec de la projection. Plus précisément, dans notre cas, nous ne pouvons projeter que sur la face supérieure et arrière du cube. Aussi pour renforcer l’illusion, il aurait été intéressant d’utiliser la caméra ainsi que la caméra de profondeur du système pour créer des représentations 3D virtuelles des éléments du monde réel. Ces objets 3D auraient ainsi pu être intégrés à la scène dans **Unity** et donc à l’illusion de transparence. Par exemple, si une personne décidait de passer sa main derrière le cube physique non transparent dans le monde réel, il aurait été possible de lui afficher cette dernière en transparence derrière le cube.

6.3 Bilan

Comme on pouvait s’y attendre, le développement d’applications de réalité augmentée spatiale avec **Unity** est très agréable. Le coût en développement (du module, de l’architecture en microservices etc.) est largement contrebalancé par les multiples possibilités offertes par ce moteur. De plus, le module répond assez bien aux objectifs que nous nous étions fixés que ce soit en termes de performance ou d’expérience pour l’utilisateur développeur.

Le fonctionnement en mode éditeur apporte un vrai plus dont nous avons pu ressentir l’effet lors du développement de l’application dont il est question section 6.2. Toutefois, la nécessité de générer des événements pour qu’**Unity** mettent à jour les composants reste pénible. Pour pallier à ce problème, nous avons imaginé une nouvelle version des microservices utilisant de nouveau le pipeline évènementiel de **Redis**, où ces derniers ne publient plus les données qu’ils génèrent mais plutôt une notification au format **JSON**[6] très légère, ne risquant donc pas de surcharger la RAM et permettant d’indiquer aux services que

des nouvelles données sont disponibles. On peut ainsi résoudre le problème de la surcharge mémoire et de l'attente active. Cette version fera l'objet des tâches à effectuer durant le dernier mois de stage.

Par ailleurs, la gestion des clés pour accéder aux données dans le module est très peu pratique et en complexifie l'utilisation sans qu'il n'y ai aucun bénéfice pour l'utilisateur. Une solution à ce problème aurait été de supprimer la gestion des clés dans Unity et de l'exporter sur le serveur web. Ainsi, lors d'une requête de démarrage d'un service, le serveur web en plus de démarrer ce service, pourrait envoyer à Unity, une réponse contenant la clé où ce dernier publie ses données. La gestion des clés deviendrait alors totalement invisible pour l'utilisateur du module ce qui lui permettrait de concentrer ses efforts sur le contenu de l'application.

Conclusion

Au cours de ce stage, j'ai pu mettre en pratique les nombreux enseignements qui m'ont été dispensés tout au long de mon cursus. En passant de l'architecture logicielle au traitement d'image sans oublier le rendu 3D, j'ai eu l'occasion de tester, d'approfondir et même d'acquérir diverses connaissances comme par exemple la programmation sur carte graphique, l'utilisation d'**Unity** ou encore le développement d'une architecture en microservices. Immérgé dans le domaine de la réalité augmentée spatiale, j'ai pu cerner et comprendre les enjeux, les apports mais aussi les difficultés et les contraintes que représente une telle technologie. En effet, j'ai pu, d'une part, saisir l'importance de la communion entre contenu et interface ainsi que l'importance du contexte lors des développements d'application qui m'ont été confiés (comme la **ReARTable**, l'extraction de documents ou encore l'illusion de projection). D'autre part, la création du prototype haute performance a été l'occasion d'aborder le côté beaucoup plus technique de cette technologie, comme les différents besoins en puissance de calcul, gestion des ressources et gestion du système.

Le stage n'étant pas fini, le dernier mois de celui-ci sera consacré à poursuivre le développement et l'amélioration du module **Unity** ainsi que du prototype haute performance réalisés, afin de pallier aux divers problèmes encore présents et évoqués tout au long de ce mémoire (latence caméra, autres optimisations, nouveau pipeline événementiel, gestion des clés dans **Unity**). En plus de ces développements, il sera aussi question de rédiger davantage de tests afin de permettre à la société de déterminer, en se basant non pas uniquement sur des tests d'utilisation mais également sur des tests de performance générale du système, si elle souhaite continuer à développer cette technologie ou si les contraintes techniques entourant ce domaine sont encore trop fortes.

Outre les enrichissements en termes de connaissances et de compétences j'ai aussi eu l'occasion de bénéficier d'une grande ouverture professionnelle en travaillant dans un environnement totalement différent de tout ce que j'avais connu auparavant où la liberté, l'autonomie et la volonté de créer sont des valeurs fortement mises en avant. J'ai d'ailleurs, grâce à ce stage, pu décrocher un contrat de travail dans le domaine de la réalité augmentée spatiale dont la mission principale sera le développement d'outil de création de contenu pour les déficients visuel à l'**Inria**.

Annexes

Annexe 1 - Tests de performance convolution

TABLE 7.1 – OpenGL ES 2.0 - Convolution d'une image en niveau de gris par un filtre de convolution de taille 5x5 - float 32 bits. HP Pavilion 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)
128x128	0,0655	9,9136	11,7581	21,6717	3,0240
256x256	0,2621	9,8709	12,6144	22,4853	11,6584
512x512	1,0486	9,6095	19,6039	29,2134	35,8938
1024x1024	4,1943	9,5579	48,5473	58,1052	72,1845
2048x2048	16,7772	9,5562	165,5163	175,0726	95,8300
4096x4096	67,1089	9,6677	502,2200	511,8877	131,1008
8192x8192	268,4350	9,6540	2273,5233	2283,1773	117,5708

TABLE 7.2 – CPU - Convolution d'une image en niveau de gris par un filtre de convolution de taille 5x5 - float 32 bits. HP Pavilion 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)
128x128	0,0655	15,3841	0,0000	15,3953	4,2569
256x256	0,2621	61,9902	0,0000	62,0079	4,2276
512x512	1,0486	287,5590	0,0000	287,5890	3,6461
1024x1024	4,1943	1161,5000	0,0000	1161,5300	3,6110
2048x2048	16,7772	4677,9100	0,0000	4677,9400	3,5865
4096x4096	67,1089	18764,3000	0,0000	18764,3000	3,5764
8192x8192	268,4350	76909,9000	0,0000	76909,9000	3,4903

TABLE 7.3 – 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)
128x128	0,0655	0,0211	0,4548	0,4759	137,7067
256x256	0,2621	0,0492	0,6379	0,9179	285,5966
512x512	1,0486	0,1635	1,4086	1,6951	618,6057
1024x1024	4,1943	0,6200	3,3843	3,7492	1118,7067
2048x2048	16,7772	2,4485	10,2548	10,5924	1583,8903
4096x4096	67,1089	9,6682	40,9823	41,3963	1621,1328
8192x8192	268,4350	38,3053	198,3800	201,6840	1330,9682

TABLE 7.4 – 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)
128x128	0,0655	0,2222	1,2313	1,4535	45,0887
256x256	0,2621	0,0149	1,4875	1,5024	174,4831
512x512	1,0486	0,0166	5,8531	5,8697	178,6425
1024x1024	4,1943	0,0061	5,7249	5,7310	731,8617
2048x2048	16,7772	0,0062	20,4904	20,4965	818,5378
4096x4096	67,1089	0,0138	78,3060	78,3198	856,8579
8192x8192	268,4350	0,0115	370,6600	370,6715	724,1857

TABLE 7.5 – 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)
128x128	0,0655	4,1572	0,0000	4,1608	15,7510
256x256	0,2621	10,2566	0,0000	10,2596	25,5511
512x512	1,0486	42,7127	0,0000	42,7166	24,5474
1024x1024	4,1943	197,4300	0,0000	197,4370	21,2437
2048x2048	16,7772	898,1140	0,0000	898,1210	18,6803
4096x4096	67,1089	3733,9500	0,0000	3733,9600	17,9726
8192x8192	268,4350	13622,6846	0,0000	13622,6846	19,7050

Annexe 2 - Tests de latence caméra

TABLE 7.6 – Latence caméras (millisecondes) - Résolution 640x480

	On board	logitech	SR300	PSEye	Aukay	ELP
NVIDIA Jetson TX2	75	120	70	90	85	80
HP Pavillon X2	/	90	/	130	/	/
PC Custom	/	85	/	85	80	80

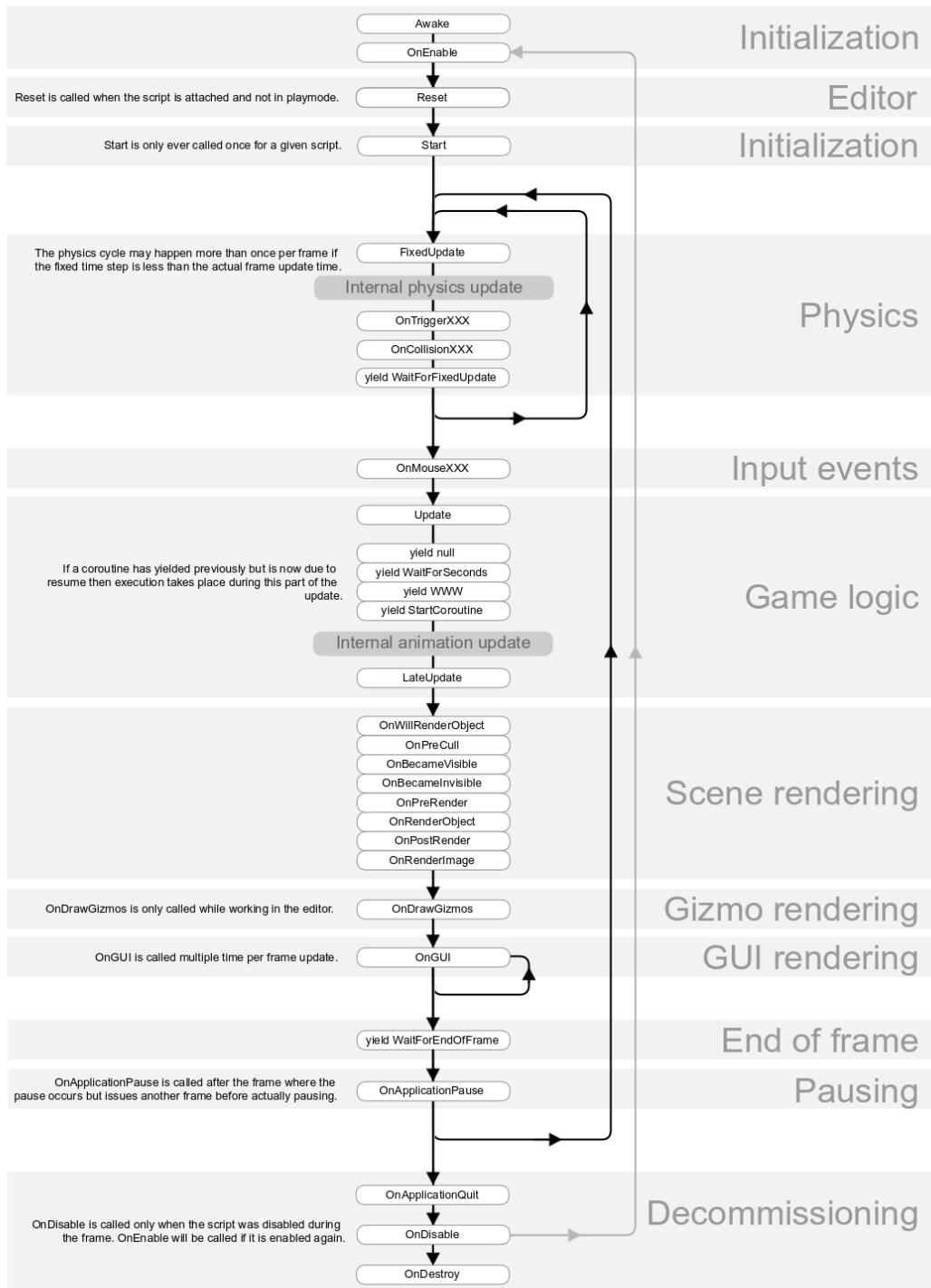
TABLE 7.7 – Latence caméras (millisecondes) - Résolution 1280x720 (HD)

	On board	logitech	SR300	PSEye	Aukay	ELP
NVIDIA Jetson TX2	80	80	85	/	85	85
HP Pavillon X2	/	120	/	/	/	/
PC Custom	/	80	/	/	90	80

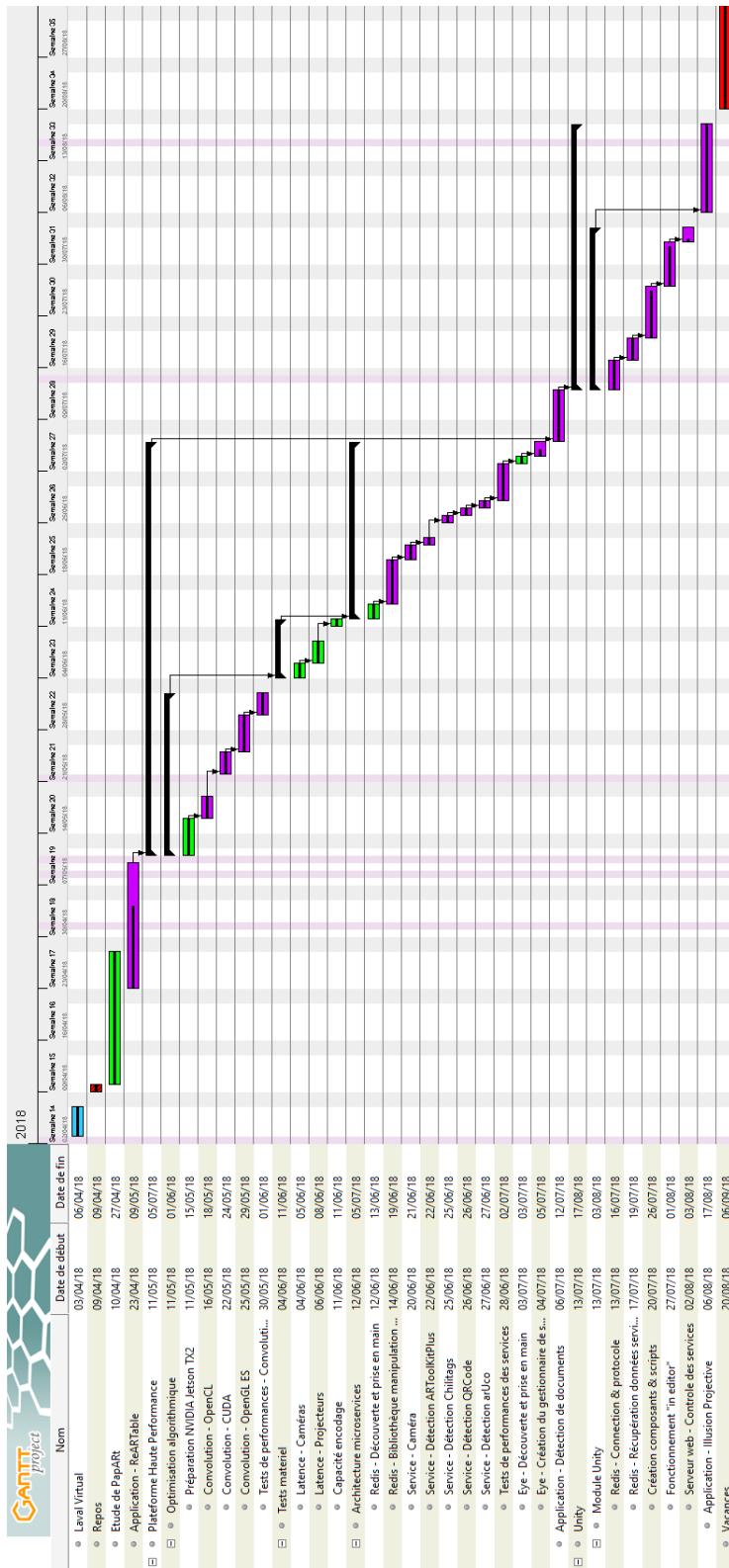
TABLE 7.8 – Latence caméras (millisecondes) - Résolution 1920x1080 (Full HD)

	On board	logitech	SR300	PSEye	Aukay	ELP
NVIDIA Jetson TX2	80	130	300	/	170	95
HP Pavillon X2	/	/	/	/	/	/
PC Custom	/	130	/	/	90	120

Annexe 3 - Unity diagramme d'exécution des fonctions



Annexe 4 - Gantt effectif



Bibliographie

- [1] Sam Aaron. Sonic pi - the live coding music synth for everyone. <https://sonic-pi.net/>. Last accessed 27 July 2018.
- [2] Oliver Bimber and Ramesh Raskar. *Spatial augmented reality : merging real and virtual worlds*. AK Peters/CRC Press, 2005.
- [3] Oliver Bimber and Ramesh Raskar. Modern approaches to augmented reality. In *ACM SIGGRAPH 2006 Courses*, page 1. ACM, 2006.
- [4] Ronald Newbold Bracewell and Ronald N Bracewell. *The Fourier transform and its applications*, volume 31999. McGraw-Hill New York, 1986.
- [5] John Canny. A computational approach to edge detection. *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE*, 1986.
- [6] Douglas Crockford. The application/json media type for javascript object notation (json). Technical report, 2006.
- [7] Namiot Dmitry and Sneps-Sneppen Manfred. On micro-services architecture. *International Journal of Open Information Technologies*, 2(9), 2014.
- [8] 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.
- [9] Khronos Group. Opencl overview. <https://www.khronos.org/opencl/>. Last accessed 28 July 2018.
- [10] Khronos Group. Opengl es. <https://www.khronos.org/opengles/>. Last accessed 28 July 2018.
- [11] GStreamer. Gstreamer. <https://gstreamer.freedesktop.org/>. Last accessed 13 August 2018.
- [12] David Heinemeier Hansson. Ruby on rails. <https://rubyonrails.org/>. Last accessed 15 August 2018.
- [13] Hiroshi Ishii. The tangible user interface and its evolution. *Communications of the ACM*, 51(6) :32–36, 2008.
- [14] 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.
- [15] Kostya. Eye. <https://github.com/kostya/eye>. Last accessed 15 August 2018.
- [16] Jeremy Laviole. Papart - paper augmented reality toolkit - augmented reality for drawing. <https://project.inria.fr/papart/fr/>. Last accessed 1 September 2018.
- [17] Jeremy Laviole. Realitytech - spatial augmented reality. <http://reality.tech/>. Last accessed 1 September 2018.
- [18] Jérémie Laviole and Martin Hachet. Papart : interactive 3d graphics and multi-touch augmented paper for artistic creation. In *3D User Interfaces (3DUI), 2012 IEEE Symposium on*, pages 3–6. IEEE, 2012.
- [19] Wenkai Li, AYC Nee, and SK Ong. A state-of-the-art review of augmented reality in engineering analysis and simulation. *Multimodal Technologies and Interaction*, 1(3) :17, 2017.
- [20] 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.
- [21] NVIDIA Corporation. NVIDIA CUDA C programming guide, 2010. Version 3.2.
- [22] Yuichi Ohta and Hideyuki Tamura. *Mixed reality : merging real and virtual worlds*. Springer Publishing Company, Incorporated, 2014.

- [23] Victor Podlozhnyuk. Image convolution with cuda. *NVIDIA Corporation white paper, June, 2007*(3), 2007.
- [24] Reactable. Reactable experience - music knowledge technology. <http://reactable.com/>. Last accessed 27 July 2018.
- [25] RedisLab. Redis. <https://redis.io/>. Last accessed 15 August 2018.
- [26] RidgeRun. Jetson - glass to glass latency. https://developer.ridgerun.com/wiki/index.php?title=Jetson_glass_to_glass_latency. Last accessed 1 September 2018.
- [27] Jason Sanders and Edward Kandrot. *CUDA by example : an introduction to general-purpose GPU programming*. Addison-Wesley Professional, 2010.
- [28] Unity Technologies. Unity. <https://unity3d.com/>. Last accessed 25 August 2018.