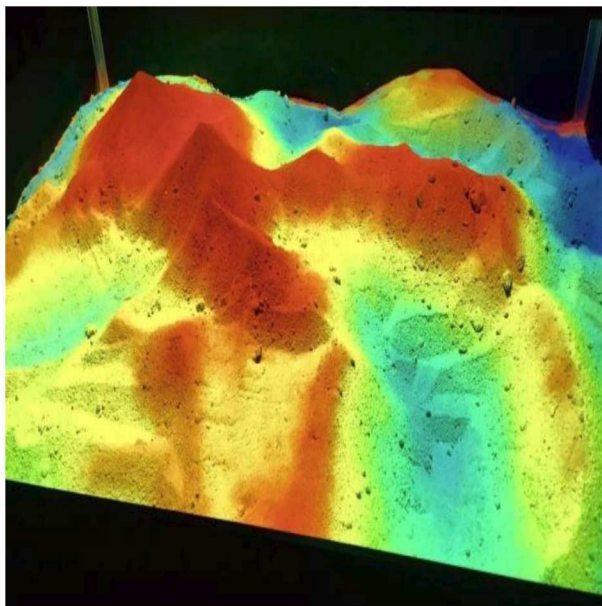


Bac à sable interactif



Thèse de Bachelor présentée par

Simon FANETTI

pour l'obtention du titre Bachelor of Science HES-SO en

**Ingénierie des technologies de l'information avec orientation en
Logiciels et systèmes complexes**

Septembre 2020

Professeur-e HES responsable

**Paul ALBUQUERQUE et Adrien
LESCOURT**

Mandant (si existant)

Aucun

Image de couverture : Bac à sable avec projection de niveaux en couleurs

Source : Sakariye ADOW, 2019, Réalité augmentée dans un bac à sable pour l'aménagement du territoire

TABLE DES MATIÈRES

Remerciements	v
Énoncé du sujet	vi
Résumé	viii
Liste de acronymes	ix
Liste des illustrations	xi
Liste des tableaux	xiii
Liste des annexes	xiv
1 Introduction	1
1.1 Contexte et problématique	1
1.2 Solutions envisagées et objectifs	1
1.3 Résumé du travail réalisé, méthodologie et annonce du plan du rapport	2
2 Etude de l'existant	4
2.1 UC Davis	4
2.2 iSandBOX	7
3 Description du dispositif	9
3.1 Dispositif	9
3.2 Caméra	10
3.3 Applications	11
3.4 Problématiques géométriques	11
4 Besoins et Fonctionnalités	14
4.1 Définition des besoins	14
4.2 Fonctionnalités	14
5 Architecture logicielle	17
5.1 Librairies	17
5.2 Librairie d'utilisation	18
5.3 Librairie de calibration	18
5.4 Application de calibration	18
5.5 Applications utilisateurs	20
6 Technologies et outils	21
6.1 OpenCV	21
6.2 RealSense 2	21
6.3 yaml-cpp	21
6.4 Qt	21
6.5 XrandR	21

6.6	Profiling	21
7	Conception et implémentation	23
7.1	Librairie d'utilisation	23
7.2	Librairie de calibration	33
7.3	Fichier de configuration	40
7.4	Application de calibration	43
8	Tests et résultats	49
8.1	Optimisation	49
8.2	Tests et résultats	51
8.3	Utilisation	52
9	Conclusion	54
9.1	Améliorations	54
9.2	Perspectives	54
	Annexes	55
	Références documentaires	71

REMERCIEMENTS

Je remercie M. Paul Albuquerque, M. Adrien Lescourt et M. Pierre Kunzli de m'avoir suivi tout au long de ce projet.

ÉNONCÉ DU SUJET

h e p i a

département ITI
ingénierie des technologies
de l'information

**Printemps 2020
Session de bachelor**

BAC À SABLE INTERACTIF

ORIENTATION : LOGICIELS ET SYSTÈMES COMPLEXES

Descriptif :

Le bac à sable de réalité augmentée (projet ARSandbox : Augmented Reality Sandbox, voir p.ex. www.youtube.com/watch?v=CE1B7tdGCw0) est un outil pédagogique qui permet de façonner un paysage à la main et d'importer la topographie dans un modèle de terrain 3D. Il est possible de projeter sur le terrain des courbes de niveau ou encore simuler des écoulements de rivières. Le dispositif actuel est constitué d'un bac à sable rectangulaire, d'une caméra de détection de profondeur, d'un projecteur et d'un PC. Cependant le logiciel qui gère la solution ne permet pas d'exploiter correctement le dispositif: architecture client-serveur contraignante, peu configurable, faibles performances...

Dans ce projet, il s'agira dans un premier temps de mettre en place une librairie qui gère l'ensemble du dispositif, couplé à une application de calibration pour paramétrer l'installation. Il faudra ensuite accélérer la réactivité du système en parallélisant le traitement des images, avant de développer une application de démonstration.

Travail demandé :

Dans les limites du temps disponible, les tâches suivantes seront traitées :

- Refonte de l'architecture du projet ARSandbox
- Développement d'une librairie C++ qui est une interface pour toutes les interactions avec le bac à sable
- Réalisation d'une application de calibration du bac à sable avec extraction de la configuration
- Amélioration de la précision et optimisation des performances du traitement
- Implémentation d'une application de démo (p.ex. courbes de niveau, modélisation d'une topologie, etc.)

Candidat-e :
FANETTI SIMON

Filière d'études : ITI

Professeur-e(s) responsable(s) :
**LESCOURT ADRIEN
ALBUQUERQUE PAUL**

En collaboration avec :
Travail de bachelor soumis à une convention de
stage en entreprise : non

Travail de bachelor soumis à un contrat de
confidentialité : non

RÉSUMÉ

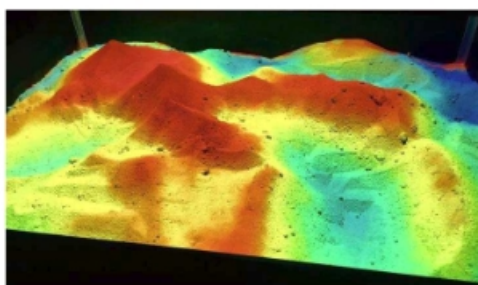
h e p i a

département ITI
ingénierie des technologies
de l'information

**Printemps 2020
Session de bachelor**

RÉSUMÉ

La création de maquettes physiques de terrains par les architectes paysagistes est un processus nécessaire lors de la réalisation de projet d'aménagement de territoire. Cependant, ces maquettes basées sur des matériaux comme le carton ou le bois, ne leur permettent pas de les modifier en cas de changements dans leur projets. C'est pourquoi nous proposons de substituer leurs maquettes par la technologie du bac à sable, qui leur permettra de vivre une expérience de réalité augmentée. Grâce aux projections d'images sur le bac, tenant compte de la topologie de celui-ci, les architectes paysagistes pourront modeler leur terrains dans le sable et avoir un rendu direct des aménagements, laissant place aux changements à tout instant. Toutefois, la création d'applications permettant ceci est pour le moment irréalisable, car aucune interface ne permet d'utiliser cette infrastructure. Après avoir reproduit l'infrastructure nécessaire au projet, constituée d'un bac à sable, d'une caméra de profondeur Intel RealSense, d'un beamer et d'un PC sous distribution Linux, nous avons commencé à développer notre interface C++ permettant son utilisation. Afin de permettre une flexibilité lors de l'installation de l'infrastructure, nous avons développé une API de calibration permettant l'obtention des paramètres nécessaires à la compensation de l'infrastructure permettant une expérience de réalité augmentée cohérente. De plus, une application de calibration basée sur cette API, permet à l'utilisateur de suivre la procédure de manière guidée et adaptée à différents environnements. Elle permet notamment de générer la configuration regroupant nos différents paramètres liés à l'infrastructure et est utilisée par notre API d'utilisation, qui a été développée par la suite. Cette API couplée à la configuration générée, permet l'utilisation du bac et finalement l'interface nécessaire aux développeurs permettant la création d'applications en réalité augmentée. C'est pourquoi finalement, nous avons développé une application affichant les niveaux du bac en couleurs.



Candidat-e :

SIMON FANETTI

Filière d'études : ITI

Professeur-e(s) responsable(s) :

PAUL ALBUQUERQUE ET ADRIEN LES-COURT

En collaboration avec : -

Travail de bachelor soumis à une convention de stage
en entreprise : non

Travail soumis à un contrat de confidentialité : non

LISTE DE ACRONYMES

API Application Programming Interface. x, 17, 54

CPU Central Processing Unit. 49

GPU Graphics Processing Unit. 29, 49

UC Davis Université de Californie à Davis. x, 1, 4, 5, 9, 10, 11

UCLA Université de Californie à Los Angeles. x, 4

LISTE DES ILLUSTRATIONS

1.1	Maquette de terrains en carton. Source : tiré de www.figurasfondo.fr , ref. URL01	1
1.2	Bac à sable de réalité augmentée de Université de Californie à Davis (UC Davis) . Source : tiré de arsandbox.ucdavis.edu , ref. URL02	2
2.1	Université de Californie à Los Angeles (UCLA) démonstration de bac à sable avec simulation d'eau. Source : tiré de arsandbox.ucdavis.edu , ref. URL02	4
2.2	Infrastructure du bac à sable de l' UC Davis . Source : tiré de arsandbox.ucdavis.edu , ref. URL03	5
2.3	Kinect v1 utilisée dans le projet UC Davis. Source : tiré de fr.wikipedia.org , ref. URL04	5
2.4	Projets UC Davis à travers le monde. Source : tiré de arsandbox.ucdavis.edu , ref. URL02	6
2.5	Simulation de volcan avec iSandBOX. Source : tiré de isandbox.co.uk , ref. URL06	7
2.6	Simulation de l'âge de glace avec iSandBOX. Source : tiré de isandbox.co.uk , ref. URL06	7
3.1	Infrastructure du bac à sable de l' UC Davis . Source : tiré de arsandbox.ucdavis.edu , ref. URL03	9
3.2	Dispositif du bac à sable de réalité augmentée à HEPIA	10
3.3	Correction de la projection d'un pixel	12
3.4	Zones de la caméra et de la projection	13
3.5	Correction de la rotation du vidéoprojecteur	13
4.1	Diagramme de séquence de la librairie d'utilisation	15
4.2	Diagramme de séquence de la librairie de calibration	16
5.1	Relations entre les classes de l' Application Programming Interface (API)	17
5.2	Relations entre les éléments principaux de l'application de calibration	19
5.3	Etapas d'initialisation de la librairie utilisateur	20
7.1	Correction de la projection d'un pixel	24

7.2	Fil d'exécution de la correction d'une projection	25
7.3	Module de la RealSense D415. Source : Intel RealSense D400 Series Datasheet June 2020 (p.108)	26
7.4	Point de vue des objectifs de profondeur et de couleur de la caméra RealSense .	27
7.5	Zone de l'image projetée par rapport au point de vue de la caméra	27
7.6	Contenu de l'image de la caméra de profondeur. Source : Intel RealSense D400 Series Datasheet June 2020 (p.17)	28
7.7	Aperçu des buffers et leurs données lors de l'ajustement d'une image projetée .	30
7.8	Géométrie de l'adaptation d'un pixel lors de sa projection	31
7.9	Logique de l'algorithme de deprojection	32
7.10	Rotation nécessaire à la projection pour la redresser	34
7.11	Cible blanche nécessaire à la calibration	35
7.12	Routine de calibration permettant d'approximer la position du beamer	35
7.13	Routine d'exécution détaillé de la calibration permettant d'approximer la posi- tion du beamer	36
7.14	Régression linéaire	39
7.15	Distance la plus courte entre deux droites	40
7.16	Formules déterminant la matrice de rotation. Source : docs.opencv.org, ref. URL08	41
7.17	Formules appliquant la matrice de rotation. Source : docs.opencv.org, ref. URL08	41
7.18	Ordre d'exécution des classes des étapes de calibration	44
7.19	Interface graphique de l'application de calibration, étape d'ajustement du trai- tement d'images	46
8.1	Temps passé dans HoughCircle lors de l'exécution du programme utilisateur . .	50
8.2	Temps passée dans spatial filter lors de l'exécution du programme utilisateur . .	50
8.3	Temps passée dans copyPixelsInto lors de l'exécution du programme utilisateur	51
8.4	Routine d'exécution d'un programme utilisateur	53

Références des URL

- URL01 <https://www.figurasfondo.fr/maquette-darchitecture-en-carton-gris/>
- URL02 <https://arsandbox.ucdavis.edu/>
- URL03 <https://arsandbox.ucdavis.edu/instructions/hardware-2/>

- URL04 <https://fr.wikipedia.org/wiki/Kinect>
- URL06 <https://isandbox.co.uk/software/>
- URL08 https://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/warp_affine/warp_affine.html
- URL09 https://docs.opencv.org/2.4/modules/imgproc/doc/geometric_transformations.html?highlight=warpaffine#warpaffine

LISTE DES TABLEAUX

2.1	Tableau des caractéristiques de la caméra de profondeur Kinect V1. Source : tiré de researchgate.net, ref URL05	6
3.1	Tableau comparatif des caméra de profondeur de Intel RealSense. Source : tiré de www.intelrealsense.com, ref URL07	11
8.1	Résultats des tests d'exécution de l'application de tests	52

Références des URL

- URL05 https://www.researchgate.net/figure/Comparative-specifications-of-Microsoft-Kinect-v1-and-v2_tbl1_313333776
- URL07 <https://www.intelrealsense.com/stereo-depth/>

LISTE DES ANNEXES

Annexe 1 : Arborescence Gperftools	57
Annexe 2 : Interface Valgrind	58
Annexe 3 : Exemple d'utilisation de la librairie	59
Annexe 4 : Exemple de fichier de configuration yaml	60
Annexe 5 : API Sandbox	61
Annexe 6 : API SandboxSetup	64
Annexe 5 : Diagramme UML API Sandbox	68
Annexe 6 : Diagramme UML API SandboxSetup	69
Annexe 7 : Diagramme UML application de calibration	70

INTRODUCTION

1.1. CONTEXTE ET PROBLÉMATIQUE

Les architectes paysagistes doivent produire des maquettes physiques de terrains afin de concrétiser leurs projets d'aménagement de territoire. Malheureusement ces maquettes basées sur divers matériaux comme le bois ou le carton, font de celles-ci des maquettes difficilement ajustables en cas de modifications souhaitées.

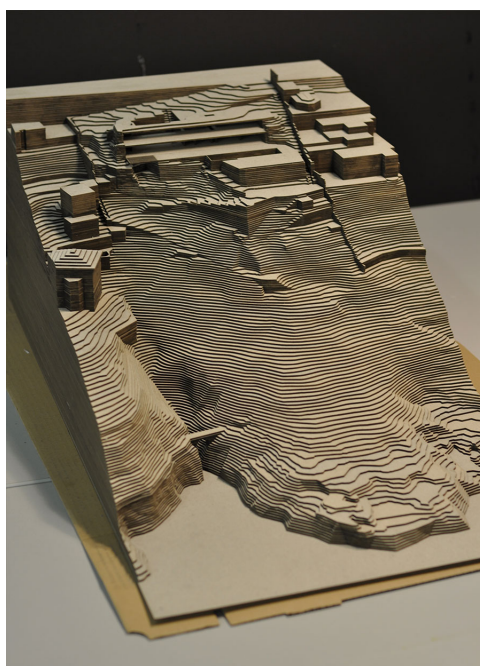


ILLUSTRATION 1.1 – Maquette de terrains en carton. Source : tiré de www.figurasfondo.fr, ref. URL01

En conséquence ils doivent recommencer leur maquette à chaque modification majeure de leur plan. Ce manque de flexibilité lors de la moindre modification ne leur permet pas de visualiser rapidement différents plans et nécessite un travail mûrement réfléchi en amont.

1.2. SOLUTIONS ENVISAGÉES ET OBJECTIFS

Afin de résoudre ce problème d'adaptation de terrains, nous allons réaliser un bac à sable de réalité augmentée qui permettra de modéliser les terrains des architectes paysagistes avec différentes applications de réalité augmentée.

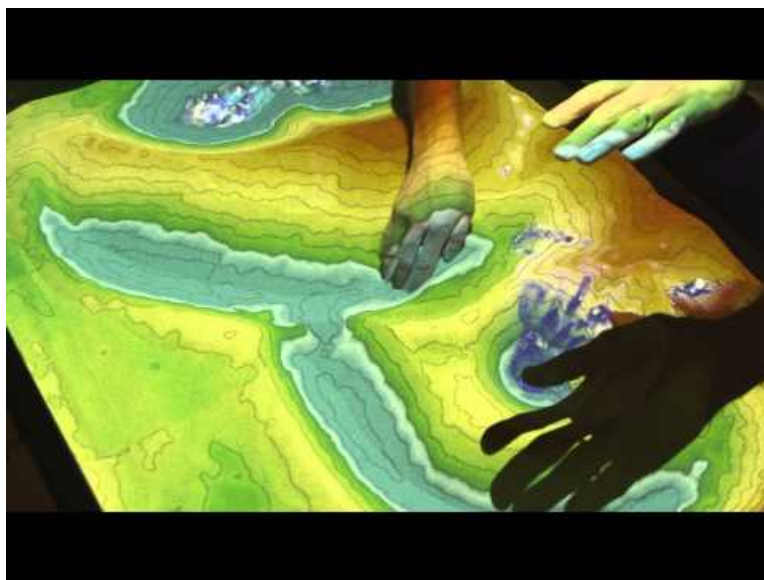


ILLUSTRATION 1.2 – Bac à sable de réalité augmentée de UC Davis. Source : tiré de arsandbox.ucdavis.edu, ref. URL02

Notre bac à sable de réalité augmentée fonctionnant à l'aide d'une caméra de profondeur, ainsi qu'un beamer, devra permettre d'être flexible sur le choix de ces composants et permettra ainsi le suivi de l'évolution de ces technologies. De plus, cette flexibilité implique qu'une adaptation dépendant de l'infrastructure devra être effectuée lors de l'utilisation du bac.

1.3. RÉSUMÉ DU TRAVAIL RÉALISÉ, MÉTHODOLOGIE ET ANNONCE DU PLAN DU RAPPORT

Sur la base de l'infrastructure présente à HEPIA, nous avons produit une librairie permettant d'utiliser le bac à sable de réalité augmentée, ainsi qu'une application de calibration basée sur celle-ci et permettant la sauvegarde de la configuration. Enfin, une application de démonstration a été développée aussi basée sur la librairie.

La librairie et les applications sont disponibles sur les gits suivants :

- Librairie : https://githopia.hesge.ch/ar_sandbox/ar_sandbox_lib
- Applications : https://githopia.hesge.ch/ar_sandbox/ar_sandbox_app

Lors de ce projet, nous avons été surpris par la crise du COVID-19, ce qui nous a forcé à pratiquer le travail à domicile. Or ce projet nécessitant de développer en présence du bac à sable, il nous a fallu trouver une alternative. En bricolant un bac à sable fait maison et à l'aide des réunions hebdomadaires, nous avons pu continuer le développement du projet. Le

développement de notre application de calibration sur ce prototype et les tests effectués sur le bac présent à l'HEPIA en fin de crise ont démontrés la flexibilité qu'apportait l'application de calibration.

Dans ce document, nous verrons au chapitre 2 une analyse de l'existant, puis une description du dispositif et les problèmes géométriques. Ensuite nous verrons les besoins et fonctionnalités au chapitre 4, suivi de l'architecture logicielle au chapitre 5 et les technologies et outils au chapitre 6. Enfin nous verrons la conception et l'implémentation et terminerons avec les tests et résultats au chapitre 8 et la conclusion au chapitre 9.

ETUDE DE L'EXISTANT

2.1. UC DAVIS

L'UC Davis a créé son propre prototype de bac à sable de réalité augmentée. Leur but est de fournir un outil pédagogique interactif basé sur des applications d'affichage de topographie et de simulation de fluide.

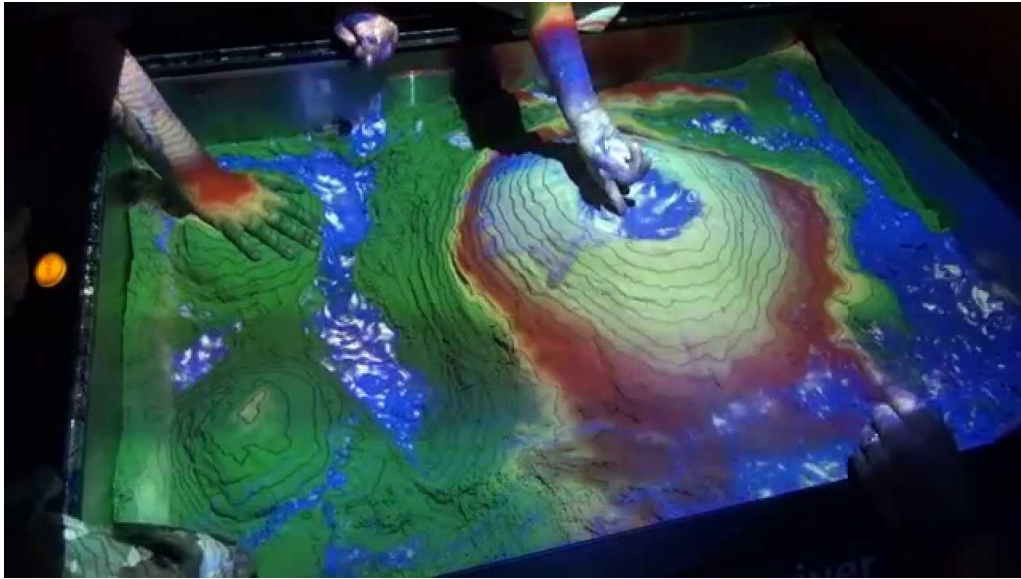


ILLUSTRATION 2.1 – UCLA démonstration de bac à sable avec simulation d'eau. Source : tiré de arsandbox.ucdavis.edu, ref. URL02

Afin de produire ces applications, le projet se base sur cette infrastructure :

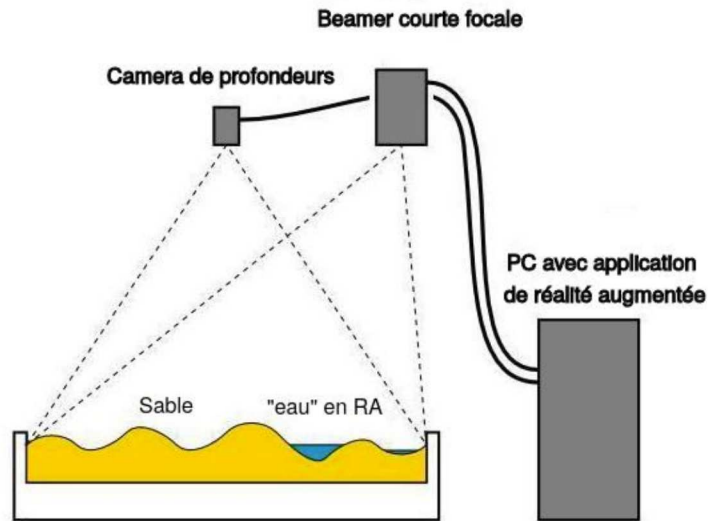


ILLUSTRATION 2.2 – Infrastructure du bac à sable de l'UC Davis. Source : tiré de arsand-box.ucdavis.edu, ref. URL03

Ils utilisent la Kinect v1 de Microsoft (*Kinect*, 2020) comme caméra de profondeur, ainsi qu'un beamer courte portée avec un ratio d'affichage de 4 :3, afin que la zone projetée corresponde à la zone de capture de la Kinect. Le tout interconnecté à l'ordinateur exécutant les applications de réalité augmentée. Grâce à la topologie récupérée du bac à sable par la caméra, l'image peut être ajustée et projetée correctement sur le bac.



ILLUSTRATION 2.3 – Kinect v1 utilisée dans le projet UC Davis. Source : tiré de fr.wikipedia.org, ref URL04

Caméra	Microsoft KinectV1
Couleurs - Résolution	640 × 480 jusqu'à 30 images par seconde
Profondeurs - Résolution	320 × 240 jusqu'à 30 images par seconde
Taille	35.5cm x 17.8cm x 7.6cm

TABLEAU 2.1 – Tableau des caractéristiques de la caméra de profondeur Kinect V1. Source : tiré de researchgate.net, ref URL05

Malheureusement pour les utilisateurs du projet, Microsoft annonce l'arrêt de la production de ces caméras en octobre 2017 (*Kinect*, 2020). Cependant, le projet permet tout de même d'utiliser les autres modèles Kinect-for-Xbox 1414 et 1473, Kinect pour Windows, et Kinect pour Xbox One, ainsi que les caméras RealSense d'Intel (INTEL REALSENSE, 2020c) (DAVIS, 2016). De plus une version permettant d'utiliser la version deux de la Kinect est en développement.

Comme ce projet fait partie des premiers et qu'il est open source, UC Davis ajoute les projets de bac à sable reproduisant ce projet dans le monde.



ILLUSTRATION 2.4 – Projets UC Davis à travers le monde. Source : tiré de arsandbox.ucdavis.edu, ref. URL02

Tout ces projets sont des bacs, qui ont été montés dans le monde en suivant les indications données dans leur guide.

2.2. iSANDBOX

Le projet iSandBOX est un projet de bac à sable en réalité augmentée réalisé par Universal Terminal Systems pour le monde professionnel et le divertissement pour les enfants (*Augmented reality sandbox | Award-winning, certified | Ships internationally*, 2020).

Cette entreprise propose plusieurs applications de réalité augmentée différentes comme :

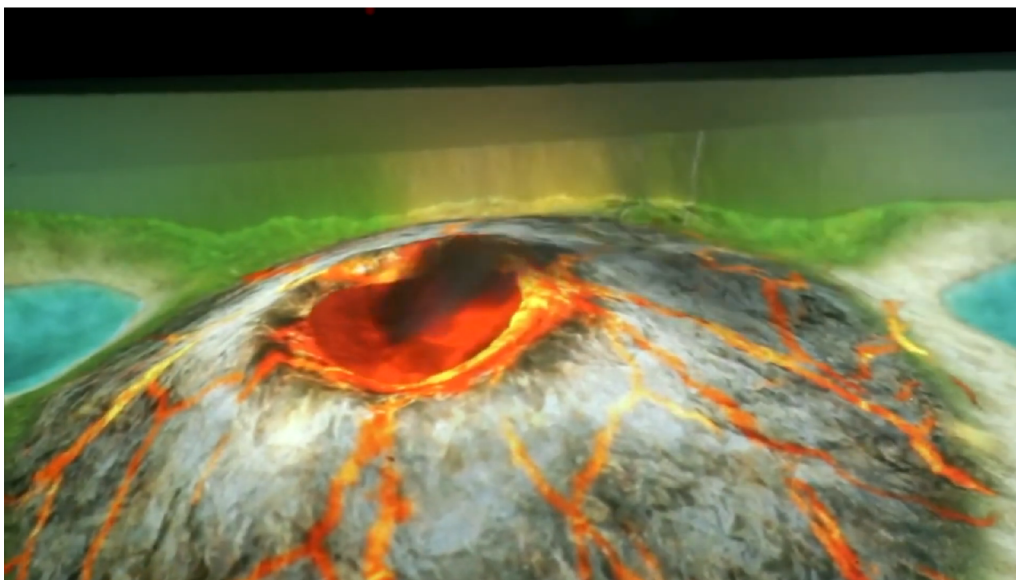


ILLUSTRATION 2.5 – Simulation de volcan avec iSandBOX. Source : tiré de isandbox.co.uk, ref. URL06

Cette simulation de volcan qui brûle tout sur son passage et projette des nuages de cendres.



ILLUSTRATION 2.6 – Simulation de l'âge de glace avec iSandBOX. Source : tiré de isandbox.co.uk, ref. URL06

Ou encore cette simulation de l'âge de glace avec de vraies statuettes de dinosaures.

DESCRIPTION DU DISPOSITIF

3.1. DISPOSITIF

Afin de reproduire cet outil de réalité augmentée, nous nous sommes basés sur le projet open source de l'UC Davis.

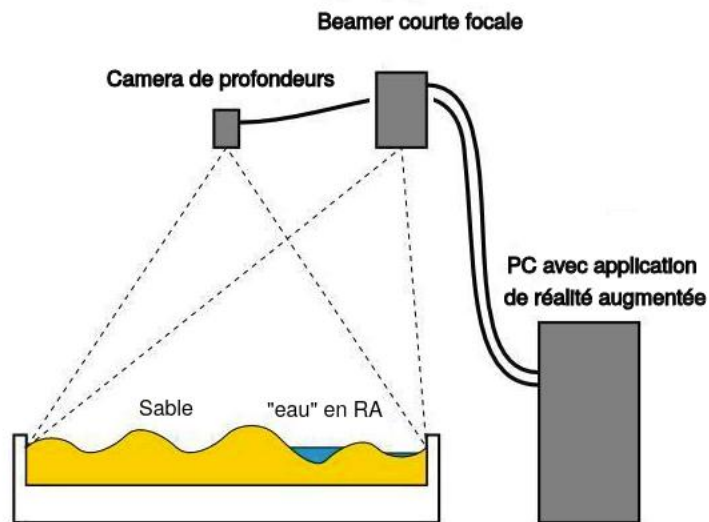


ILLUSTRATION 3.1 – Infrastructure du bac à sable de l'UC Davis. Source : tiré de arsand-box.ucdavis.edu, ref. URL03

Nous avons donc monté notre propre bac à sable (100cm x 75cm x 20cm), posé un beamer à courte focale qui se trouve à environ 1.2 mètres au-dessus du bac afin de recouvrir l'entièreté du bac, ainsi qu'une caméra de profondeur positionnée à un mètre au dessus du bac à sable. De plus nous utilisons un ordinateur standard basé sur une distribution linux.

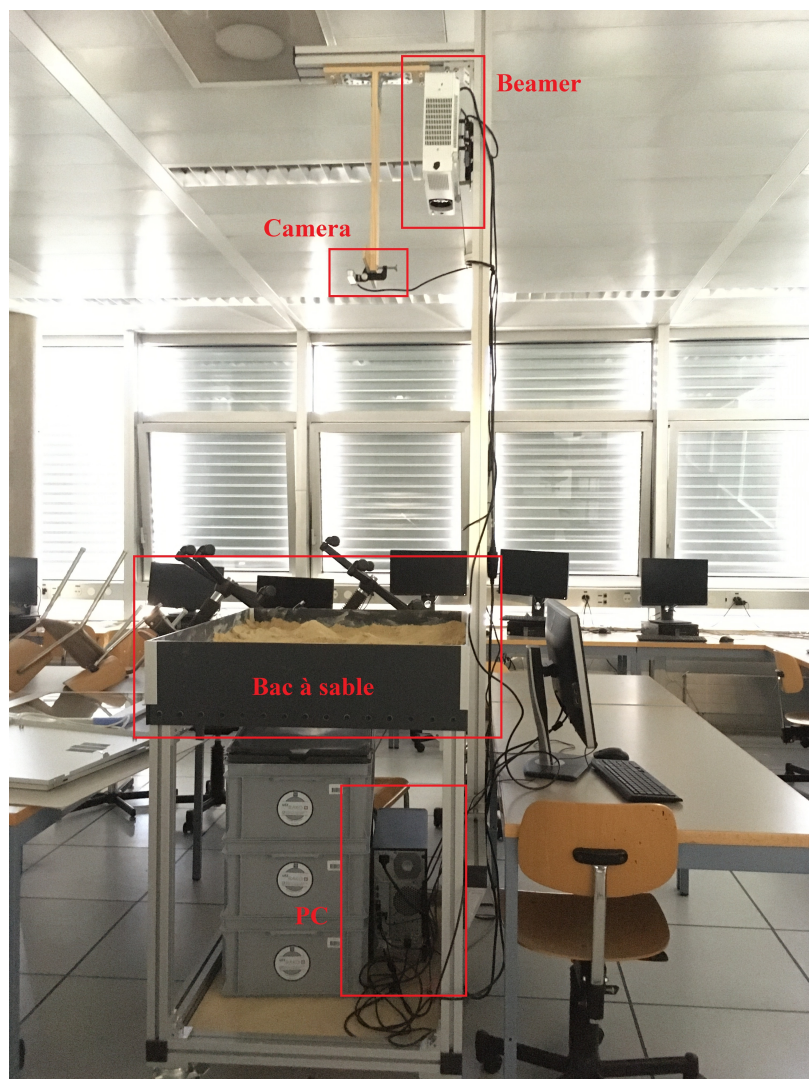


ILLUSTRATION 3.2 – Dispositif du bac à sable de réalité augmentée à HEPIA

Cependant, comme énoncé précédemment dans le projet d'UC Davis, la Kinect utilisée dans leur projet est désormais obsolète. Nous avons donc dû choisir une autre caméra de profondeur pouvant fonctionner correctement à un mètre du bac.

3.2. CAMÉRA

Lors du choix de la caméra, il faut prendre en compte ses performances, ainsi que ses spécificités d'objectif. Le but étant de trouver une caméra pouvant être positionnée à un mètre au-dessus du bac, nous avons comparé les caractéristiques de la Realsense D415 (INTEL REALSENSE, 2020a) et de la D435 (INTEL REALSENSE, 2020b).

Les deux caméras ont globalement les mêmes caractéristiques, avec la D435 qui est plus imposante que la D415. Mais ce qui détermina notre choix, c'est l'ouverture des caméras. La

Caméra	Ouvertures objectif	Distance minimum	Résolution profondeur	Vitesse de rafraîchis- sement
D415	$65^{\circ} \pm 2^{\circ}$ × $40^{\circ} \pm 1^{\circ}$	0.16 m	Up to 1280 × 720	Up to 90 fps
D435	86° × 57° ($\pm 3^{\circ}$)	0.105 m	Up to 1280 × 720	Up to 90 fps

TABLEAU 3.1 – Tableau comparatif des caméra de profondeur de Intel RealSense. Source : tiré de www.intelrealsense.com, ref URL07

D435 a une ouverture plus grande que la D415, ce qui implique que si nous voulons avoir notre bac à sable complètement dans l’objectif, nous devons poser la D435 plus bas que la D415. Mais comme nous voulons poser notre caméra à un mètre et que notre bac à sable est de dimensions 100cm x 75cm, nous avons déduit que la D415 s’approchait le plus de cette hauteur.

3.3. APPLICATIONS

Les applications fournies par le projet de l’UC Davis fonctionnent correctement avec leur projet. Malheureusement, le projet ne propose pas de librairie permettant d’exploiter le bac à sable. Il est donc impossible en l’état de développer des applications avec ce projet et aucune de leurs applications ne peut fonctionner sur notre infrastructure.

3.4. PROBLÉMATIQUES GÉOMÉTRIQUES

Afin de pouvoir exploiter cet outils, nous devons résoudre quelques problématiques géométriques liées à la disposition de l’infrastructure.

Normalement, lorsque nous voulons projeter une image avec un vidéoprojecteur, nous le faisons sur une surface plane. Or dans notre cas, l’image est projetée sur le sable avec une certaine topologie.

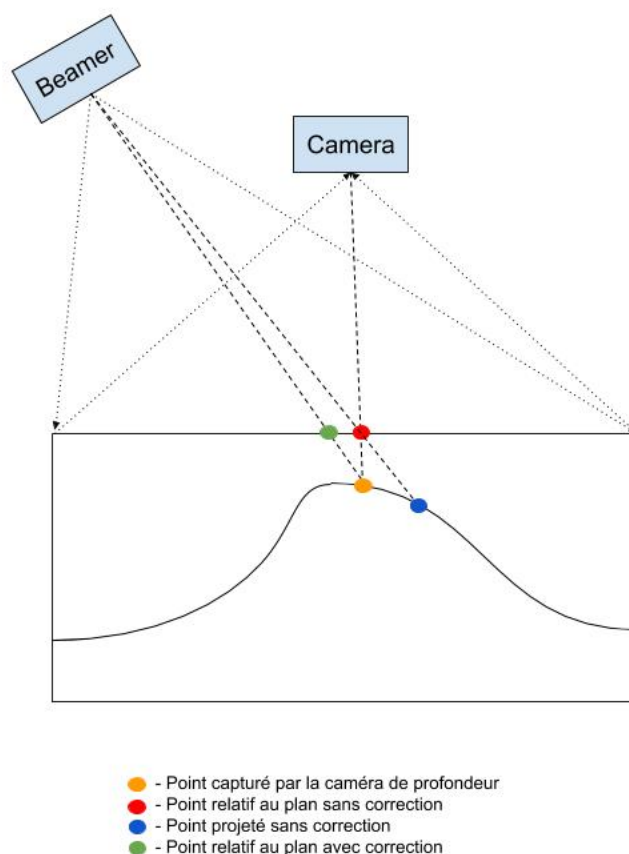


ILLUSTRATION 3.3 – Correction de la projection d'un pixel

Comme notre vidéoprojecteur doit pouvoir projeter sur une surface plane, nous avons défini le sommet du bac à sable comme repère pour projeter notre image. Ainsi le champ de vision de la caméra correspond à la zone de projection du vidéoprojecteur.

Dans ce schéma, nous avons la caméra qui récupère la hauteur à un certain point (orange) et nous avons notre vidéoprojecteur qui projette un pixel. Ce pixel sera projeté à la position bleu sans ajustement, car l'image projetée correspond au champ de vision de la caméra. C'est seulement en corrigeant la position sur notre plan (sommet du bac à sable), que nous pouvons adapter notre affichage à la position du vidéoprojecteur. Donc en ajustant le pixel qui serait projeté à la position bleu, vers la position orange, nous arrivons à projeter une image cohérente sur le sable. Malheureusement, l'infrastructure mise en place ne sera pas toujours aussi exacte que celle sur le schéma. Notamment la position de la caméra et du vidéoprojecteur, ainsi que la taille du bac à sable. C'est pourquoi il faut ajuster la taille de l'image projetée à celle de la capture de la caméra.

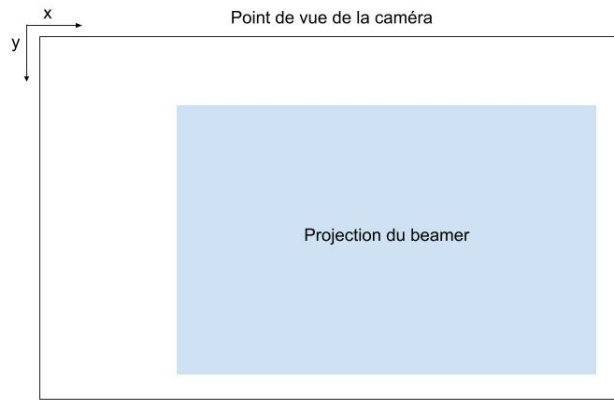


ILLUSTRATION 3.4 – Zones de la caméra et de la projection

Ainsi, en déterminant la zone de projection par rapport au point de vue de la caméra, nous pouvons ajuster seulement les pixels de la projection et ainsi conserver la cohérence entre la zone de projection et la zone capturée par la caméra.

Enfin il se peut que le vidéoprojecteur ne soit pas parfaitement droit et qu'il y ait une rotation de l'image projetée par rapport à la caméra.

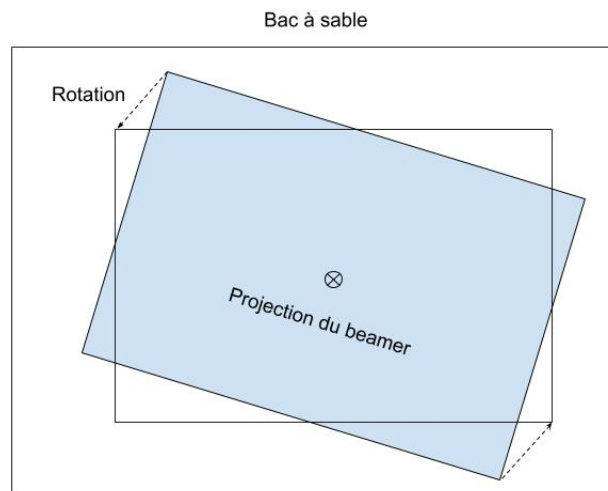


ILLUSTRATION 3.5 – Correction de la rotation du vidéoprojecteur

En corrigeant cette rotation centrée à la projection lors de l'affichage, nous aurons une projection alignée au point de vue de la caméra.

BESOINS ET FONCTIONNALITÉS

4.1. DÉFINITION DES BESOINS

Les utilisateurs du bac sont variés. Il y a tout d'abord les utilisateurs qui voudront développer des applications sur le bac à sable et ensuite les utilisateurs qui voudront utiliser les applications développées pour le bac. Comme il n'existe actuellement pas de librairie permettant aux développeurs de faire des applications, il faudra fournir une librairie d'utilisation du bac à sable le leur permettant. Cette librairie devra permettre de corriger les problématiques indiquées précédemment dans la section *problématiques géométriques*. De plus, elle devra permettre aux applications de réalité augmentée d'être affichées correctement et le plus dynamiquement possible, afin de garder une fluidité dans l'expérience de réalité augmentée.

Lorsqu'un utilisateur d'application voudra utiliser le bac, il devra tout d'abord le construire et passer par l'étape de calibration qui permet de déterminer les facteurs géométriques liés à l'infrastructure, afin de pouvoir les corriger lors de l'utilisation des applications basées sur la librairie d'utilisation. C'est pourquoi une application de calibration permettant de déterminer ces facteurs de manière guidée est nécessaire.

Une fois ces facteurs déterminés, les applications devront en prendre connaissance afin d'être cohérentes avec l'infrastructure présente. De plus, comme ces facteurs sont nécessaire aux applications, il faudra permettre la sauvegarde de cette configuration, afin de ne pas réitérer la procédure de calibration à chaque utilisation des applications. Car cette procédure n'est nécessaire que lors du montage du système ou lors de sa modification.

4.2. FONCTIONNALITÉS

La librairie devra être séparée en deux parties distinctes. Une devra fournir les fonctionnalités de calibration de la librairie, et l'autre devra fournir les fonctionnalités d'utilisation de la librairie. Ces deux librairies devront permettre la création des applications de réalité augmentée et de calibration.

Librairie d'utilisation

La librairie d'utilisation devra répondre à ces attentes :

- capture les photos de profondeurs et couleurs avec la camera
- ajuste une image à projeter sur le bac à sable
- charge la configuration depuis un fichier de configuration
- initialise le bac à sable

De telle manière que nous puissions l'utiliser de la façon suivante :

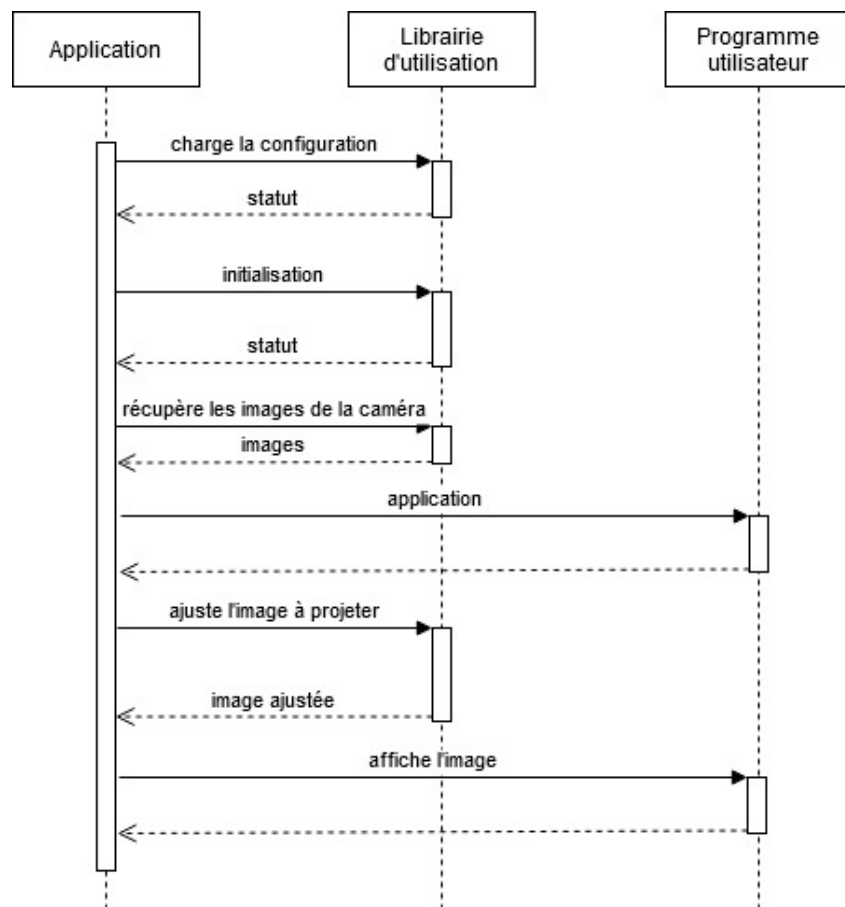


ILLUSTRATION 4.1 – Diagramme de séquence de la librairie d'utilisation

Une fois les étapes de configuration et initialisation effectuées avec succès, l'utilisateur de la librairie pourra récupérer les images de la caméra et effectuer un traitement dessus pour projeter son image après l'avoir ajustée.

Librairie de calibration

La librairie de calibration devra permettre à l'application d'exécuter correctement les routines et sauvegarder la configuration. Elle devra donc :

- charger certaines informations du fichier de configuration si nécessaire
- fournir les méthodes de calibration permettant de déterminer les données de configuration
- sauvegarder la configuration dans un fichier pour la rendre persistante

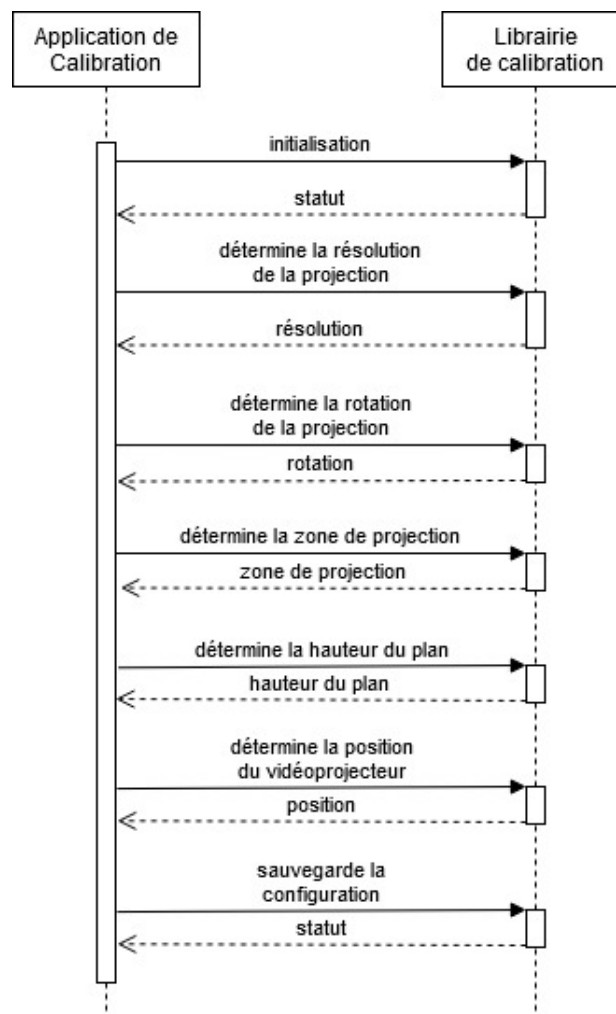
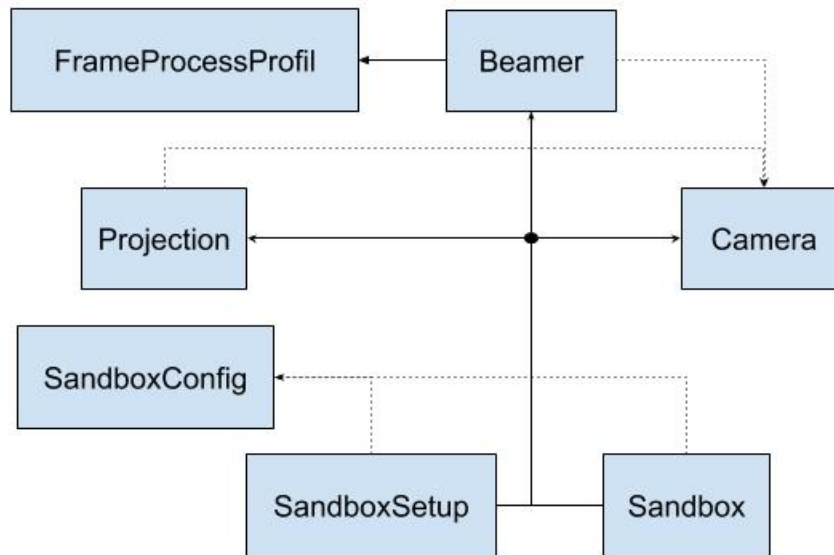


ILLUSTRATION 4.2 – Diagramme de séquence de la librairie de calibration

Dans ce scénario, nous allons récupérer les données de configuration en effectuant les routines de calibration, puis les sauvegarder dans un fichier. Ainsi nous avons notre configuration prête pour les applications utilisateurs.

ARCHITECTURE LOGICIELLE

5.1. LIBRAIRIES



$A \rightarrow B$: A instancie B

$A \cdots \rightarrow B$: A utilise B

ILLUSTRATION 5.1 – Relations entre les classes de l'API

Dans cette architecture, les classes sont assez représentatives de l'infrastructure réelle. Nous avons *Beamer* qui représente les informations liées au vidéoprojecteur utilisé et les méthodes spécifiques au beamer liées à l'étape de calibration. Elle contient aussi *FrameProcessProfil*, qui est un profil contenant des informations liées au traitement d'images et est utilisé lors de la calibration. *Camera* est l'interface utilisant la librairie realsense 2 (*Intel RealSense SDK 2.0 – Intel RealSense Depth and Tracking cameras*, 2020), permettant l'utilisation de la camera d'Intel. Elle est notamment utilisée par *Beamer* lors de la calibration et par *Projection* lors de l'utilisation.

Ensuite nous avons *Projection*, qui contient les traitements nécessaires à appliquer à une image projetée. Puis il y a *SandboxConfig*, qui est une classe permettant de lire et écrire les informations liées à la configuration de la librairie dans un fichier.

Enfin nous avons les interfaces de la librairie, respectivement *Sandbox* correspondant à la librairie d'utilisation permettant de développer les applications utilisateur et *SandboxSetup* corres-

pendant à la librairie de calibration et permettant de développer les applications de calibration qui généreront la configuration utilisée par *Sandbox*.

5.2. LIBRAIRIE D'UTILISATION

La librairie d'utilisation, aussi appelée *Sandbox* instancie *Camera*, *Beamer* et *Projection* et fait le lien entre le tout. Elle va s'occuper de fournir de quoi initialiser la caméra et charger la configuration. Elle fournira les méthodes permettant de récupérer les images de profondeur et de couleur de la caméra adaptées à la zone d'affichage du beamer, et de quoi ajuster une image à projeter sur le bac à sable, qui tiendra compte de la position du beamer, de sa rotation par rapport à la caméra, ainsi que de sa distance avec le sommet du bac à sable.

5.3. LIBRAIRIE DE CALIBRATION

La librairie de calibration *SandboxSetup* instancie aussi *Camera*, *Beamer* et *Projection* et permet d'accéder aux méthodes de configuration de la librairie. Elle permet de déterminer les variables liées à l'infrastructure qui serviront à avoir un rendu correcte sur le bac à sable. Hormis le fait que nous n'allons pas utiliser les méthodes de traitement d'images de *Projection*, *Camera* et *Beamer* seront sollicités pour déterminer la position du *Beamer* par rapport à la caméra et pour récupérer les images capturées sans traitement. Une fois la configuration déterminée, *Sandbox-Config* permettra de la sauvegarder.

5.4. APPLICATION DE CALIBRATION

Pour faciliter la génération du fichier de configuration, il a fallu faire une application plus agréable pour l'utilisateur. L'application de calibration utilise la librairie de calibration *SandboxSetup*.

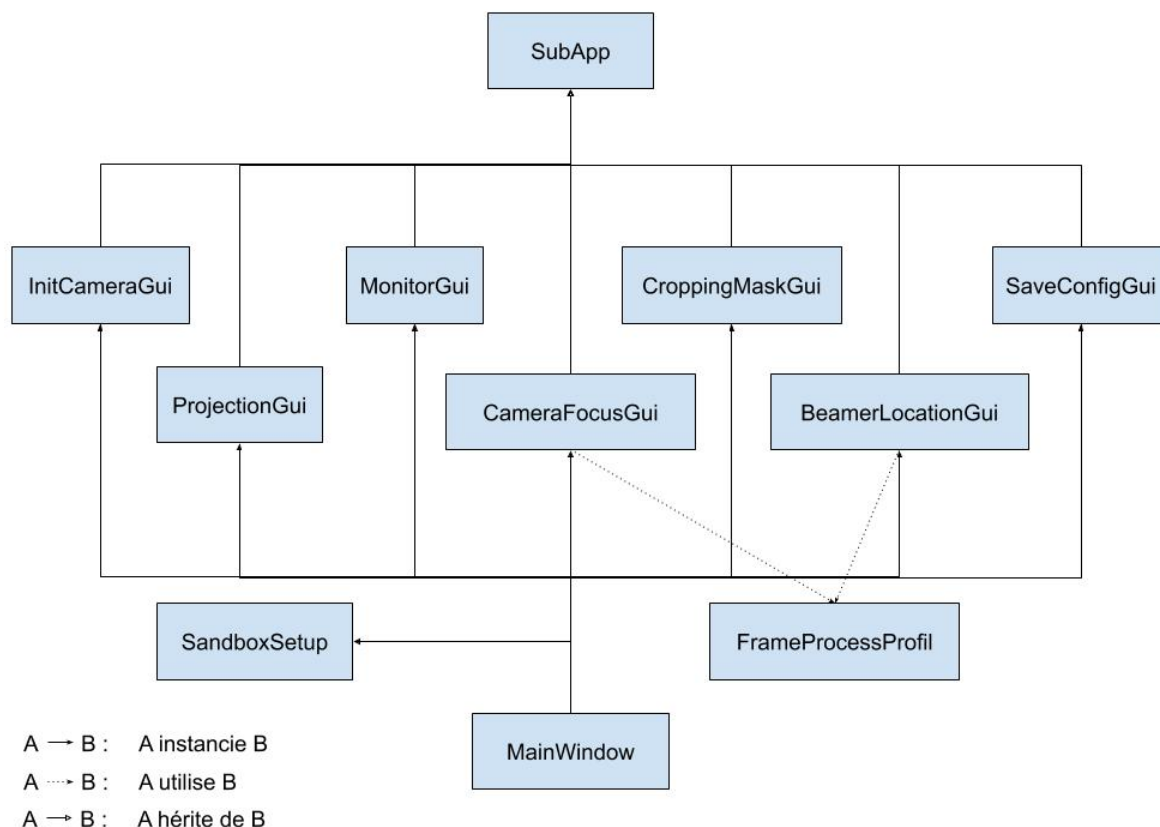


ILLUSTRATION 5.2 – Relations entre les éléments principaux de l'application de calibration

Cette application est basée sur *MainWindow*, qui sera l'interface parente tout au long de la procédure de calibration. Elle instancie *SandboxSetup*, afin qu'elle puisse utiliser l'infrastructure du bac à sable, puis instancie toutes les classes héritant de *SubApp*. *SubApp* représente une étape de la calibration, donc chaque classe en héritant correspondra à une phase de la configuration. En l'occurrence les classes *InitCameraGui*, *MonitorGui*, *ProjectionGui*, *CameraFocusGui*, *CroppingMaskGui*, *BeamerLocationGui* et *SaveConfigGui*.

InitCameraGui sert à initialiser la caméra et vérifier qu'elle soit disponible. *MonitorGui* sert à sélectionner la sortie correspondant au vidéoprojecteur et sa résolution. *ProjectionGui* permet de faciliter la mise en place du vidéoprojecteur et de la caméra. *CameraFocusGui* permet d'établir le profil *FrameProcessProfil* qui contient les valeurs permettant un meilleur rendu de l'image de la caméra lors de la calibration. *CroppingMaskGui* permet de déterminer la matrice de rotation liée à la rotation du beamer, ainsi que la zone d'intérêt des images de la caméra et la distance avec le plan se trouvant au dessus du bac à sable. Tous ces paramètres seront nécessaires à l'ajustement des images projetées. *BeamerLocationGui* permet d'exécuter la routine permet-

tant d'établir la position du vidéoprojecteur, qui sera elle aussi utilisée lors de l'ajustement des projections. Enfin *SaveConfigGui* permet de sauvegarder la configuration dans un fichier.

5.5. APPLICATIONS UTILISATEURS

Lorsqu'un utilisateur veut utiliser la librairie d'utilisation *Sandbox*, il devra toujours passer par ces étapes.

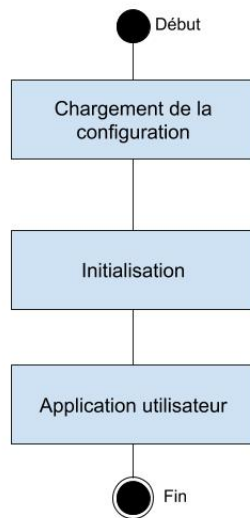


ILLUSTRATION 5.3 – Etapes d'initialisation de la librairie utilisateur

Il devra initialiser la librairie pour s'assurer que l'infrastructure est disponible. Puis charger la configuration en s'assurant qu'elle soit présente, pour ensuite exécuter son application avec la confirmation que la librairie fonctionnera correctement.

TECHNOLOGIES ET OUTILS

Le projet est basé le langage C++. Ainsi que d'autres librairies et framework.

6.1. OPENCV

Nous avons utilisé OpenCV (Open Source Computer Vision Library) qui est une librairie "spécialisée dans le traitement d'images en temps réel" (*OpenCV*, 2020).

6.2. REALSENSE 2

La librairie realsense 2 est une librairie multi-plateforme développée par Intel RealSense et permettant l'utilisation des ses caméras de profondeurs.

6.3. YAML-CPP

"yaml-cpp" est une librairie de lecture et d'écriture de fichier YAML écrite en C++ par Jesse Beder.

6.4. QT

"Qt" est un framework C++ permettant la création d'application graphique principalement, mais offre aussi d'autres éléments liés aux connexions réseaux, multi-threading ou d'accès aux données (*Qt*, 2020).

6.5. XRANDR

"XrandR" est une librairie C permettant de récupérer les informations des sorties vidéos et de les modifier (*libxrandr*, 2020).

6.6. PROFILING

Lors du projet, nous avons dû utiliser des outils de profiling afin de détecter les potentiels goulets d'étranglement. Les outils ci-dessous proposeront tous d'indiquer des informations après exécution du programme. Ces informations seront principalement basées sur l'arborescence d'appels de fonctions que créera le programme à son exécution, ainsi que les "coûts" de ces fonctions, qui représente une unité de temps, mais qui peut varier en fonction de l'outil.

Gprof

"Gprof" (*Gprof*, 2016) un outil de profiling qui permet de récupérer les informations liées au temps d'exécution d'un code dans un fichier et de les afficher dans le terminal. Il suffit d'ajouter *-pg* lors de la compilation des fichiers pour générer un fichier *gmon.out* à l'exécution du programme.

Valgrind

"Valgrind" (VALGRIND DEVELOPPERS, 2020) un autre outil de profiling, qui permet lui aussi de générer un fichier contenant les informations d'exécution. Il est notamment utilisé avec l'outil graphique Callgrind, qui permettra d'avoir une meilleure compréhension visuelle du cours de l'exécution du programme et de l'arborescence d'appels de fonctions.

Gperftools

"Gperftools" (SANJAY GHEMAWAT, 2008) l'outil de profiling utilisé par Google. Il permet aussi d'afficher une arborescence d'appels de fonctions, mais celui-ci est complet, contrairement à Callgrind qui n'affiche qu'une partie.

CONCEPTION ET IMPLÉMENTATION

7.1. LIBRAIRIE D'UTILISATION

Principe

La librairie d'utilisation *Sandbox* doit fournir une interface permettant l'affichage d'une image dans le bac à sable adapté à la topologie de celui-ci. Comme vu précédemment dans la description, plusieurs ajustements sont nécessaires. Nous allons donc voir plus précisément comment l'ajustement de l'image se fait.

En appelant la fonction *adjustProjection*, l'utilisateur passe son image en couleur à projeter :

- capture de la topologie du bac à sable
- initialisation du buffer de sortie
- appels à la fonction *adjustFrame* de la classe *Projection*
- retour du buffer de sortie

Nous avons donc un buffer pour l'image de sortie, ainsi qu'un autre contenant les profondeurs du bac à sable (représentées par des floats équivalent à des mètres). Pour rappel, notre but est de corriger la valeur des pixels à afficher à une position donnée.

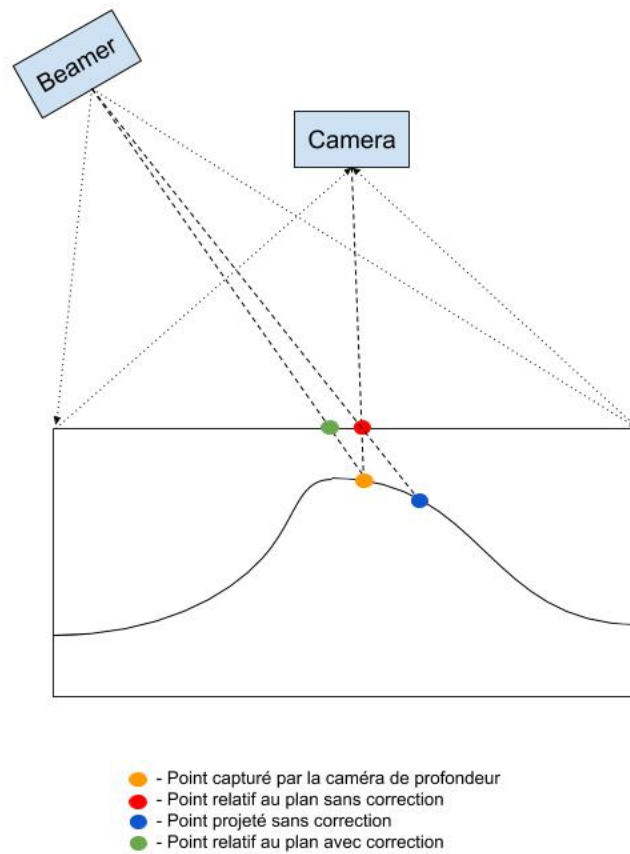


ILLUSTRATION 7.1 – Correction de la projection d'un pixel

Il faut donc se baser sur le point de vue de la caméra pour connaître la position du point depuis le point de vue du vidéoprojecteur. C'est à dire trouver la position verte, sinon la rouge sera par défaut et ne sera pas représentative de la réalité.

En regardant de plus près la fonction *adjustFrame*, on peut voir qu'elle suit cette démarche pour ajuster l'image :

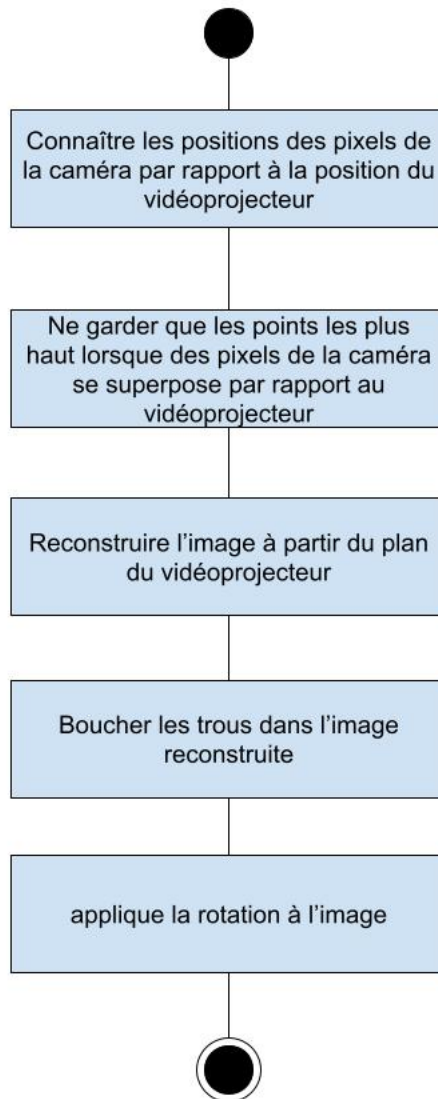


ILLUSTRATION 7.2 – Fil d'exécution de la correction d'une projection

Maintenant nous avons une compréhension plus précise de ce qu'il se passe lors du traitement d'une image. Nous verrons plus en détails la fonction *adjustFrame* dans *Projection*.

SandboxConfig

Bien que nous ayons vu le principe de l'ajustement d'une image, il est irréalisable tant que nous ne possédons pas les informations de configuration. C'est pourquoi dans la librairie, il y a une méthode permettant de lire le fichier de configuration écrit en YAML et de charger les informations dans leurs variables respectives. Si on regarde dans cette méthode, on s'aperçoit qu'elle utilise les méthodes statiques de la classe *SandboxConfig*, afin de charger les informations né-

cessaires lors du processus d'ajustement de l'image projetée. Grâce à la librairie `yaml-cpp`, il est très facile d'interagir avec les données. Il suffit de charger les données dans un nœud, pour ensuite interagir avec ce nœud comme avec un tableau associatif. De plus, il est aussi facile d'écrire les données dans un fichier, il suffit de traiter le nœud comme du texte lors d'un affichage dans la sortie standard en C++.

Camera

La librairie utilise la caméra D415 de Intel. Elle comprend deux récepteurs stéréo avec un projecteur qui servent à capturer l'image de profondeur et un récepteur RGB qui sert à capturer l'image de couleur.

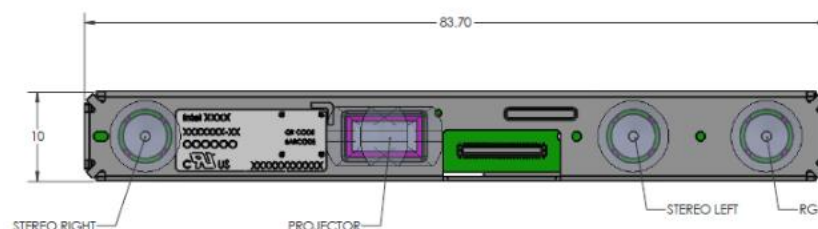


ILLUSTRATION 7.3 – Module de la RealSense D415. Source : Intel RealSense D400 Series Datasheet June 2020 (p.108)

La caméra peut fournir une résolution allant jusqu'à 1920x1080 pour la caméra de couleur et 1280x720 pour celle de profondeur. Comme les résolutions et potentiellement les champs de vision des objectifs diffèrent, Intel a mis en place une méthode permettant de s'assurer que l'image récupérée en couleur corresponde au champ de vision de celle de profondeur. Il s'agit de la classe *align* disponibles dans la librairie realsense 2.

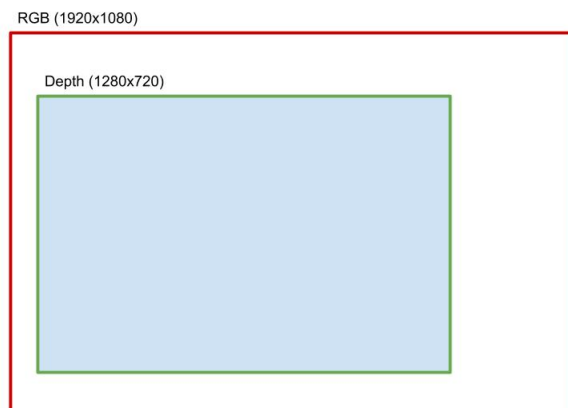


ILLUSTRATION 7.4 – Point de vue des objectifs de profondeur et de couleur de la caméra RealSense

Dans notre cas, *align* a été paramétrée pour s'aligner sur le flux de profondeurs, donc les images de profondeur et couleur seront basées sur l'objectif de profondeur (*rs-align*, 2020). Lorsque nous récupérons les images de profondeur et couleur, nous avons des matrices de 1280x720. Mais dans la librairie *sandbox*, nous ne renvoyons pas directement l'image capturée par la caméra, elle est rognée pour correspondre à la zone de projection du vidéoprojecteur.

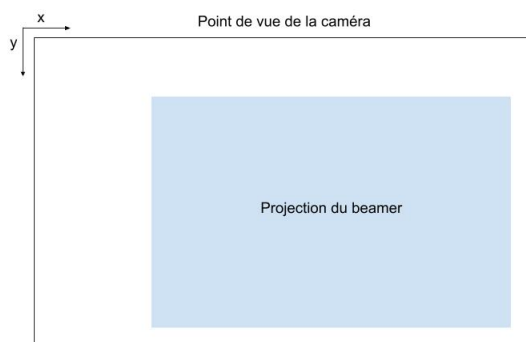


ILLUSTRATION 7.5 – Zone de l'image projetée par rapport au point de vue de la caméra

Autrement dit, on ne récupère que la zone bleue par rapport au point de vue de la caméra. Ce qui définit cette délimitation est un rectangle que nous avons appelé *CroppingMask* et qui est sauvegardé dans la classe *Camera*. Notons qu'il est important de permettre à la caméra de "chauffer" les capteurs de profondeurs, autrement ceux-ci ne capteront qu'une partie avec leur objectifs, voir rien du tout, ce qui retournera une image de profondeur vide.

Dans la classe *Camera*, nous utilisons les fonctions de projection et déprojection. Ces méthodes nous sont nécessaires pour passer de la matrice de profondeur à un repère 3D et inver-

sement. Les valeurs contenues dans la matrice de profondeur correspondent à la profondeur Z sur un repère 3D et non à la profondeur directe entre l'objet et l'objectif, ici appelée Range. Une méthode permettant de récupérer la Range existe dans la librairie realsense 2, mais nous ne l'utiliserons pas.

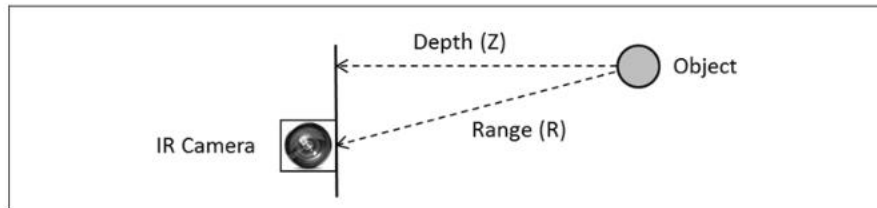


ILLUSTRATION 7.6 – Contenu de l'image de la caméra de profondeur. Source : Intel RealSense D400 Series Datasheet June 2020 (p.17)

La méthode de déprojection *rs2_deproject_pixel_to_point* permet de déprojeter un pixel en un point 3D où le point (0,0,0) est défini par la caméra (INTEL REALSENSE, 2019). La méthode est définie par des coordonnées correspondant à celles d'un pixel, de la profondeur qu'il y aurait à ce pixel et d'un profil intrinsèque défini par la caméra. La méthode de projection permet l'inverse, soit de passer d'un point 3D aux coordonnées d'un pixel. Dans les deux cas nous avons besoin de ce profil intrinsèque. C'est dans celui-ci qu'est décrit le profil de la caméra, le type de déformation lié à la lentille de l'objectif, la résolution de l'image et les informations liées à la position de la matrice produite par l'objectif.

Dans notre cas, le modèle de déformation est le Brown-Conrady, la librairie realsense fait donc une simple adaptation de repère. C'est à dire :

```
float x = point.x / point.z ;
float y = point.y / point.z ;
return cv : :Point2i( x*f.x+pp.x, y*f.y+pp.y) ;
```

Où *point* est notre point en 3D, puis les variables x et y multipliées aux coefficients de f qui permettent de passer du repère 3D à celui d'une matrice de pixels, tels que les coefficients de f correspondent à des multiples de pixels proportionnels à la taille de l'image produite par le capteur. De plus, il faut ajuster les coordonnées du pixel par rapport à son origine dans l'espace ou la matrice avec pp , car l'origine (0,0) dans l'image capturée se situe dans le coin haut-gauche, alors que l'origine (0,0,0) dans l'espace 3D se situe aux coordonnées indiquées par la variable (INTEL REALSENSE, 2019).

Nous nous sommes permis de réimplémenter ces deux fonctions, car nous avons besoins de modifier les valeurs f et pp , et nous ne voulions pas être dépendants de la librairie `realsense 2` dans le cas où une optimisation par parallélisation du traitement se ferait, car `realsense 2` ne propose pas de solution sur **Graphics Processing Unit (GPU)** pour le moment.

Comme les images capturées par la caméra et le profil intrinsèque sont liés, si l'on veut adapter les valeurs de f et pp tout en restant cohérent avec une image de dimensions différentes, il faut adapter l'image à la matrice capturée par la caméra pour qu'elles correspondent lorsqu'on les déprojettent. C'est pourquoi nous avons la méthode *getAdaptedIntrinsics* qui retourne ces deux paramètres ajustés à l'image à projeter.

Les images de profondeurs subissent aussi un filtre spatial et temporel. Le spatial permet de réduire le nombre d'informations manquantes dans l'image de profondeur et le temporel permet de stabiliser les valeurs, car les valeurs varient légèrement entre deux images même si rien n'a changé physiquement.

Projection

Projection est la classe qui nous permet d'ajuster l'image projetée sur le bac à sable en tenant compte des données de l'environnement. Son but est donc principalement tourné vers l'utilisation de la fonction *adjustFrame*. Dans cette fonction nous utilisons plusieurs buffers qui ont tous les mêmes dimensions : *resized_src*, *resized_depth*, *deprojectMap* et *frameMap*. Ainsi avec le pseudo code suivant nous pouvons reconstruire l'image à projeter :

- initialisation des buffers
- redimensionne les buffers (comprends l'image source et l'image de profondeur) à la taille de l'image de sortie
- dé-projette *resized_depth* par rapport à la position du vidéoprojecteur dans *deprojectMap* (indique où les pixels seront positionnés dans l'image de sortie)
- filtre les points dé-projetés aux mêmes coordonnées et ne garde que le point le plus haut en sauvegardant les coordonnées du pixel dans *deprojectMap* dans *frameMap* (indique les coordonnées du pixel source)
- construit l'image en lisant *deprojectMap* pour connaître le pixel de *resized_src* à sé-

lectionner et mettre dans le buffer de sortie

- rebouche les trous dans le buffer de sortie pour les pixels n’ayant pas de correspondance dans *frameMap*
- applique la rotation à l’image de sortie

Nous avons donc une initialisation des buffers à la taille de la projection de l’image. Suivi d’une redimension de la matrice de profondeur (*depth* devient *resized_depth*) et de l’image source (*src* devient *resized_src*) où la méthode *resize* d’OpenCV applique une interpolation bilinéaire par défaut (OPENCV DEV TEAM, 2019c).

Pour mieux comprendre à quoi servent les buffers dédiés à la fonction, voici de quoi imager la démarche et à quel moment ils interviennent :

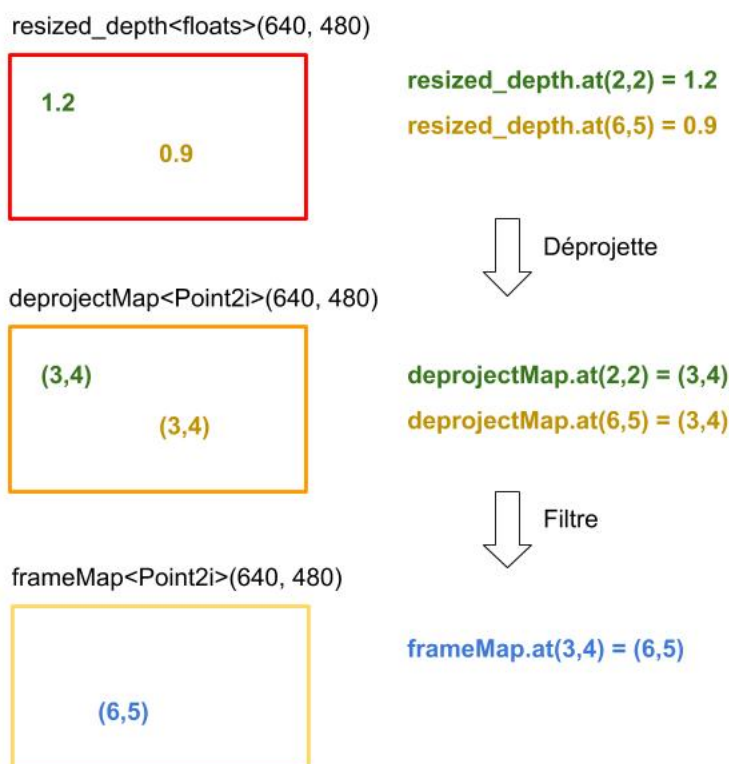


ILLUSTRATION 7.7 – Aperçu des buffers et leurs données lors de l’ajustement d’une image projetée

Donc si nous revenons à l’étape de déprojection, le but est de trouver à quel pixel de l’image projetée correspond le pixel de la matrice de profondeur, soit P’ dans le schéma ci-dessous.

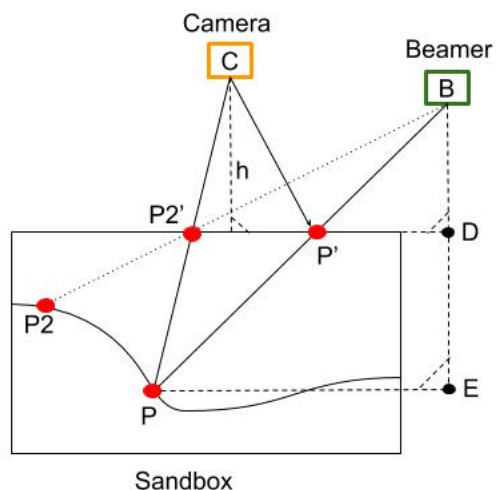


ILLUSTRATION 7.8 – Géométrie de l'adaptation d'un pixel lors de sa projection

Comme vu précédemment dans *Principe*, le but est de trouver P' , car sans adaptation, P sera projeté à $P2$. Mais pour trouver P' , il faut déjà connaître P . Grâce à la méthode *deproject-PixelToPoint* fournie par *Camera*, nous pouvons connaître les coordonnées de P en spécifiant ces coordonnées dans la matrice de profondeur et la profondeur (donc sa valeur).

Comme nous nous basons sur les valeurs de la matrice de profondeur pour établir un environnement 3D, cela implique que la base de ce repère est la caméra. De plus comme notre but est de calculer CP' qui se base sur le même plan que $P2'$, cela implique que la caméra soit perpendiculaire au plan, qui est le dessus du bac à sable. Enfin comme la position du vidéoprojecteur (Beamer) est relative à la caméra (Camera), nous pouvons assumer que BDP' et BEP sont des triangles rectangles.

Ce qui nous amène au calcul de P' en connaissant déjà au préalable : la hauteur du dessus du bac à sable h , la position du vidéoprojecteur B , la position du point P . Ainsi nous pouvons trouver P' grâce à la formule ci-dessous.

Objectif : $\overrightarrow{CP'}$

$$\begin{aligned}\overrightarrow{CP'} &= \overrightarrow{CB} + \overrightarrow{BP'} \\ \text{Où : } \overrightarrow{BP'} &= \alpha * \overrightarrow{BP}\end{aligned}$$

Thalès :

$$\frac{\|\overrightarrow{BD}\|}{\|\overrightarrow{BE}\|} = \frac{\|\overrightarrow{BP'}\|}{\|\overrightarrow{BP}\|}$$

Où :

$$\alpha = \frac{\|\overrightarrow{BD}\|}{\|\overrightarrow{BE}\|}$$

Donc :

$$\overrightarrow{BP'} = \frac{\|\overrightarrow{BD}\|}{\|\overrightarrow{BE}\|} * \overrightarrow{BP}$$

$$\overrightarrow{CP'} = \overrightarrow{CB} + \frac{\|\overrightarrow{BD}\|}{\|\overrightarrow{BE}\|} * \overrightarrow{BP}$$

ILLUSTRATION 7.9 – Logique de l’algorithme de deprojection

Une fois CP' trouvé, il suffit d'utiliser la méthode *projectPointToPixel* pour récupérer les coordonnées correspondantes dans la matrice de profondeur et les sauver dans notre matrice *deprojectMap*.

Une fois que toute notre matrice *deprojectMap* est remplie, nous allons essayer de commencer à reconstruire notre image à projeter. Pour ce faire, nous allons lire *deprojectMap* et assigner l'index de la cellule de *deprojectMap* dans le pixel cible. Comme si nous inversions les index avec les valeurs. Sauf qu'il se peut que plusieurs pixels de *deprojectMap* pointent vers la même cellule, nous devons donc vérifier que seul le pixel de *deprojectMap* ayant le sommet le plus haut (donc la valeur la plus petite, puisque les valeurs sont relatives à la caméra) soit assigné aux coordonnées cible dans *frameMap*.

Ensuite nous parcourons *frameMap* pour connaître à quel autre pixel correspond le pixel actuel et nous récupérons la valeur de l'image source à l'image de sortie. En faisant cette étape, il se peut que *frameMap* ait des pixels manquants, car les points de vues entre le vidéoprojecteur et la caméra diffèrent.

C'est pourquoi il s'ensuit l'étape de rebouchage de trous. Dans cette étape, nous parcourons à nouveau *frameMap*, mais cette fois à la recherche de valeurs manquantes. Lorsque nous en trouvons une, nous copions la valeur du premier voisin non vide se trouvant à une distance d'une case. Enfin, il ne reste plus qu'à appliquer la matrice de rotation à notre image de sortie.

Beamer

Dans le cas de la librairie d'utilisation, la classe *Beamer* sert à fournir la position 3D relative à la caméra de celui-ci et sa résolution. Ces informations sont toutes deux utilisées lors de l'adaptation de l'image projetée.

7.2. LIBRAIRIE DE CALIBRATION

La librairie de calibration *SandboxSetup* sert de deuxième interface, mais son but est de fournir les méthodes nécessaires à la génération d'un fichier de configuration complet. La classe *SandboxSetup* fournit les éléments comme la classe *Sandbox*, c'est à dire l'accès aux classes *Beamer*, *Camera* et *Projection*, mais aussi à *SandboxConfig*. Le principe est de paramétrer les classes comme si nous allions utiliser l'interface *Sandbox*, sauf que nous allons sauvegarder la configuration dans un fichier yaml grâce à la méthode fournie par *SandboxSetup*, elle permet de sauvegarder tout les éléments nécessaires en utilisant les fonctions de *SandboxConfig*.

La classe fournit aussi de quoi calculer la matrice de rotation et la zone d'intérêt de la capture de la caméra, car à ce stade, aucun traitement n'est fait sur les images. Nous utilisons simplement la caméra et renvoyons les informations telles quelles. Il est aussi possible de charger le profil *FrameProcessProfil* qui sera utile lors du traitement d'images durant la calibration. Nous verrons plus en détails dans la partie Beamer.

AdjustMatrix

La méthode permettant de déterminer la matrice de rotation dans la classe *SandboxSetup* prend simplement la liste des coins d'un rectangle et son centre, afin de déterminer son angle de rotation grâce aux points P0 et P3, où le coin de référence pour la rotation dépendra de P0. Une fois la matrice déterminée, nous la sauvegardons dans la classe *Projection*.

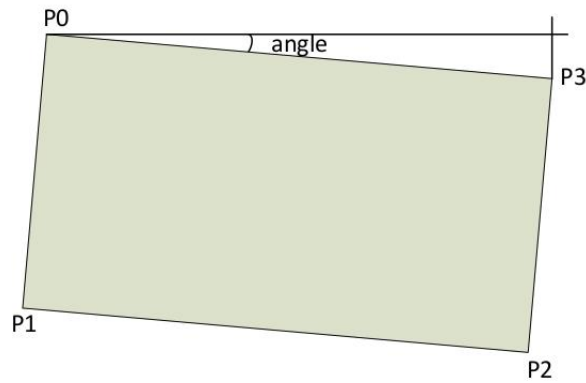


ILLUSTRATION 7.10 – Rotation nécessaire à la projection pour la redresser

Il est important de noter que comme la matrice redresse l'image, elle est dépendante de la rotation de la caméra si on applique la matrice à une image prise avec celle-ci.

CroppingMask

La méthode permettant de déterminer le masque de découpage utilise les mêmes points que pour la déterminer la matrice de rotation, sauf qu'on applique la rotation aux coins pour avoir un rectangle droit. Puis nous la sauvegardons dans la classe *Camera*.

Beamer

La classe *Beamer* propose elle aussi des méthodes liées à la calibration. En effet c'est même le cas pour la majorité d'entre elles.

Procédure

Ces méthodes sont basées sur une procédure de calibration précise. L'utilisateur devra se munir d'une cible comme ci-dessous afin de la réaliser.

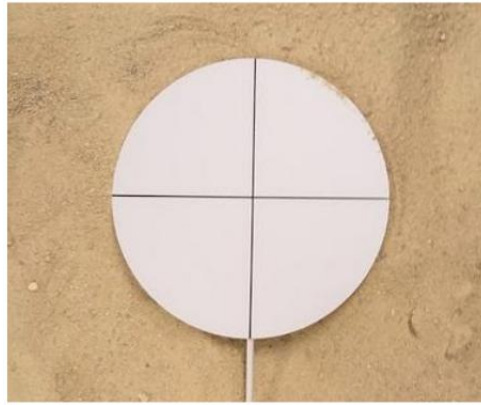


ILLUSTRATION 7.11 – Cible blanche nécessaire à la calibration

Une fois cette cible acquise, l'utilisateur pourra suivre la procédure suivante :

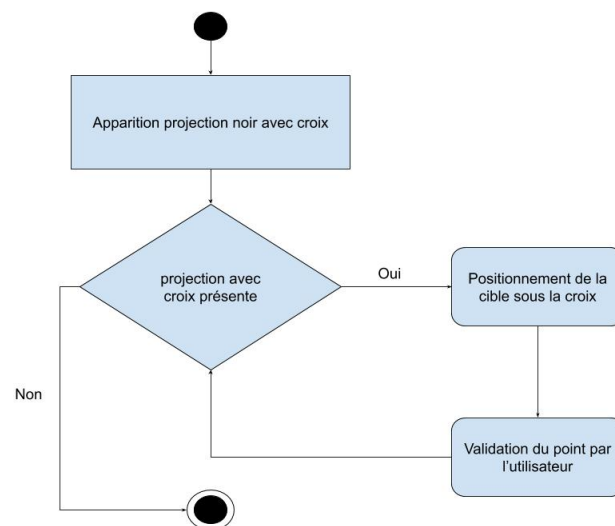


ILLUSTRATION 7.12 – Routine de calibration permettant d'approximer la position du beamer

Afin de trouver la position du vidéoprojecteur à la fin de cette procédure, les différentes méthodes devront suivre cette démarche :

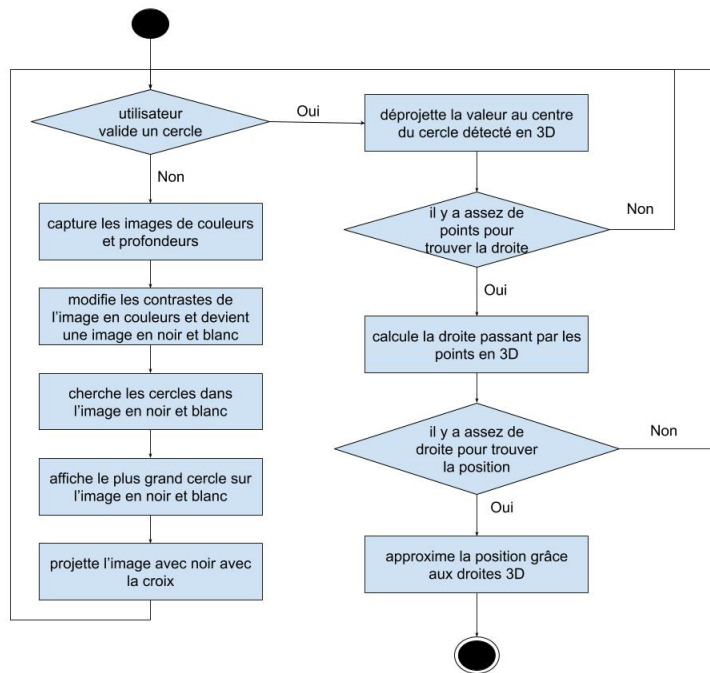


ILLUSTRATION 7.13 – Routine d’exécution détaillé de la calibration permettant d’approximer la position du beamer

L'utilisateur positionnera sa cible à l'endroit où les croix seront projetées. La méthode *getCrossList* nous permet de récupérer les positions des croix à afficher. Lorsqu'une croix est projetée, une capture est prise afin de localiser la cible, après l'avoir convertie en noir et blanc, nous pouvons modifier le contraste et la luminosité (avec *editContrast*), puis la passer à la méthode *findCircles* qui nous retournera les cercles détectés. Ensuite nous pouvons construire l'image à afficher avec la croix à l'aide de *buildCrossFrame*, qui affichera la croix à suivre et si la cible est détectée, en plus d'indiquer à quelle étape du processus nous en sommes.

Lors de la validation d'un point de l'utilisateur, il faut s'assurer d'avoir une cible détectée, car c'est à ces coordonnées que nous allons établir un point sur un repère en 3D, pour cela il suffira d'utiliser *deprojectPixel*. Une fois plusieurs points validés, nous pouvons calculer la droite passant au mieux par ces points grâce à la méthode *findLinearLineFrom*. Enfin, lorsque nous avons assez de droites, nous pouvons approximer la position du vidéoprojecteur avec *approximatePosition*.

getCrossList

Retourne une liste de coordonnées correspondant aux croix à viser lors de la calibration. Elles devront être affichées les unes après les autres.

editContrast

Cette méthode adapte le contraste et la luminosité d'une image en noir et blanc en suivant cette equation (OPENCV DEV TEAM, 2019b) :

$$g(i,j) = \alpha * f(i,j) + \beta \text{ où } \alpha > 0$$

findCircles

Cette méthode permet de récupérer la position des cercles détectés. Elle est basée sur la méthode *houghCircles* d'OpenCV défini par : *void HoughCircles(InputArray image, OutputArray circles, int method, double dp, double minDist, double param1=100, double param2=100, int minRadius=0, int maxRadius=0)*

HoughCircle est basé sur une matrice de compteurs, appelée accumulateur. Pour détecter un cercle, il faut tout d'abord appliquer un filtre Canny permettant de ne garder que les contours des formes dans l'image. Puis nous allons parcourir chaque pixel de notre image de contours et lorsque nous rencontrons un pixel de valeur non vide/noir, nous allons tracer un cercle autour de ces pixels, en traçant ce cercle, nous incrémentons de un aux mêmes coordonnées dans notre accumulateur. Une fois l'image parcourue, nous pouvons contrôler notre accumulateur pour trouver les cercles potentiels. Si les valeurs de l'accumulateur dépassent un certain seuil, alors nous avons un cercle centré à cette endroit de l'image de rayon égal à celui avec lequel nous avons parcouru notre image de contours précédemment.

- "image" est une image en niveaux de gris
- "circles" est une liste permettant à la fonction de sauvegarder les cercles détectés
- "method" indique à la fonction quelle méthode utiliser pour détecter les cercles. Dans notre cas, nous utilisons la méthode Hough-Gradient, c'est aussi la seule disponible pour le moment par OpenCV
- "dp" correspond au ratio inverse de la taille de l'accumulateur. Elle sera de même taille

- que l'image passée en paramètre si "dp" est égale à un
- "minDist" est la distance minimum entre deux cercles détectés
- "param1" correspond au seuil supérieur de la méthode Canny, donc le seuil indiquant un contour
- "param2" est le seuil indiquant si le centre d'un cercle existe dans notre accumulateur
- "minRadius" et "maxRadius" permettent de gérer le rayon minimum et maximum du cercle lors du parcours de l'image de contours incrémentant l'accumulateur

(TEAM, 2019)

buildCrossFrame

Construit une image où l'on indique la position de la croix à placer. Indique si un cercle a été trouvé, si c'est le cas, un carré vert sera visible, sinon il sera rouge. Il est aussi indiqué à quelle étape nous nous trouvons sur le total.

deprojectPixel

Permet de récupérer les coordonnées 3D à partir d'un cercle détecté et de la matrice de profondeur.

findLinearLineFrom

Cette méthode permet de trouver la droite passant le mieux entre les points donnés. Elle utilise la méthode *fitLine* d'OpenCV.

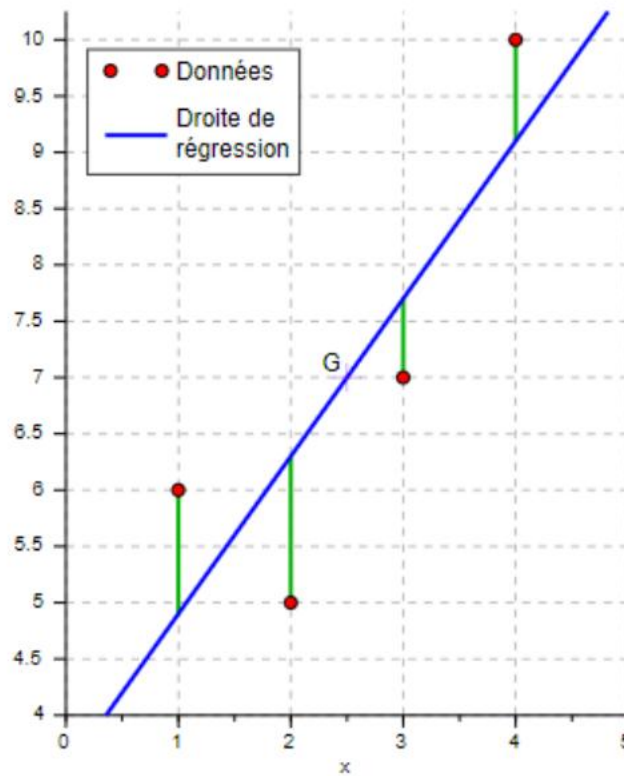


ILLUSTRATION 7.14 – Régression linéaire

approximatePosition

Cette méthode permet d'approximer la position 3D du vidéoprojecteur grâce à une liste de droites (qui sont définies par un point et une direction). Elle fait appel à la méthode *LineLineIntersect* qui permet de trouver les points les plus proches entre deux droites (PAUL BOURKE, 2020) :

$$Pa = P1 + mua * (P2 - P1)$$

$$Pb = P3 + mub * (P4 - P3)$$

Où Pa et Pb sont les points les plus proches.

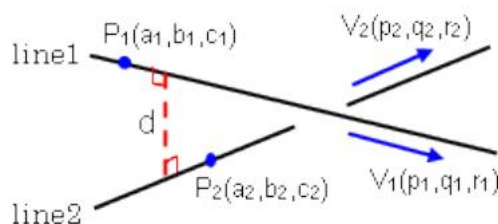


ILLUSTRATION 7.15 – Distance la plus courte entre deux droites

Une fois les points trouvés, nous faisons la moyenne de tous nos points pour approximer la position du vidéoprojecteur.

FrameProcessProfil

Cette classe sert à accompagner la classe *Beamer* lors de la phase de calibration, son but est de contenir les différents paramètres liés au traitement d'images que nous utiliserons plus tard dans les fonctions de *Beamer*. Elle contient notamment les variables de luminosité et contraste utilisable dans "editContrast", ainsi que les paramètres utilisés lors dans la fonction "findCircles" liés au traitement de "HoughCircles" d'OpenCV. Plus précisément : "minDist", "param1", "param2", "minRadius" et "maxRadius". Où "param1" fait référence au seuil supérieur lié à la méthode de contour Canny, et "param2" au seuil déterminant un cercle dans notre accumulateur dans la méthode de Hough. Les variables "minDist", "minRadius" et "maxRadius" sont tous trois en pourcentage par rapport à la largeur de l'image traitée. Donc nous avons :

- contrast : contraste de "editContrast"
- brightness : luminosité de "editContrast"
- minDistance : "minDist" de "HoughCircles"
- cannyEdgeThreshold : "param1" de "HoughCircles"
- houghAccThreshold : "param2" de "HoughCircles"
- minRadius : "minRadius" de "HoughCircles"
- maxRadius : "maxRadius" de "HoughCircles"

7.3. FICHER DE CONFIGURATION

Le fichier de configuration est le fichier permettant de sauvegarder les données nécessaires à l'utilisation de la classe *Sandbox* et déterminées lors de la calibration. Il est écrit au format YAML et est donc facilement lisible et modifiable.

AdjustingMatrix

Cette variable contient la matrice qui permet de corriger la rotation du vidéoprojecteur. Elle est sauvegardée dans le fichier sous forme de vecteur, mais il s'agit bien d'une matrice de 2x3 comme l'indiqueront la hauteur et la largeur. L'angle sur lequel est basé la matrice de rotation est aussi disponible. Si la matrice a de telles dimensions, c'est parce qu'elle est basée sur les transformations affines, on doit donc pouvoir appliquer une rotation, une translation ou un changement d'échelle avec celle-ci.

$$\begin{bmatrix} \alpha & \beta & (1 - \alpha) \cdot \text{center.x} - \beta \cdot \text{center.y} \\ -\beta & \alpha & \beta \cdot \text{center.x} + (1 - \alpha) \cdot \text{center.y} \end{bmatrix}$$

$$\begin{aligned} \alpha &= \text{scale} \cdot \cos \text{angle}, \\ \beta &= \text{scale} \cdot \sin \text{angle} \end{aligned}$$

ILLUSTRATION 7.16 – Formules déterminant la matrice de rotation. Source : docs.opencv.org, ref. URL08

De plus si l'on regarde la fonction d'OpenCV "warpAffine", qui applique notre matrice, on voit comment est appliquée la matrice :

$$\text{dst}(x, y) = \text{src}(M_{11}x + M_{12}y + M_{13}, M_{21}x + M_{22}y + M_{23})$$

ILLUSTRATION 7.17 – Formules appliquant la matrice de rotation. Source : docs.opencv.org, ref. URL08

Avec cette formule, on observe que seules les deux premières colonnes sont multipliées aux coordonnées, donc la troisième contient le décalage lié au centre de rotation en cas de translation de l'image et les deux premières sont celles liées à la rotation et à l'ajustement de l'échelle.

DistanceTopSandbox

Cette variable sert à déterminer la distance entre la caméra et le dessus du bac à sable, car c'est à cette distance que l'image virtuelle reconstruite pour le vidéoprojecteur se trouve. Comme la matrice de profondeur fournie par la caméra, est en float, cette distance l'est aussi.

CroppingMask

Le *CroppingMask* est le rectangle permettant de rogner l'image de la caméra afin d'avoir uniquement la zone où le vidéoprojecteur affiche dans le bac à sable. Comme c'est un rectangle dans le même sens que la caméra, il n'a pas de rotation et est dépendant de la résolution de la caméra. Nous pouvons donc manipuler sa position et ses dimensions si nous connaissons le matériel.

BeamerResolution

BeamerResolution permet de connaître la résolution de l'image à reconstruire et projeter. Elle est simplement constituée d'une largeur et d'une hauteur.

BeamerPosition

BeamerPosition est la position du vidéoprojecteur en 3D basée sur la caméra en mètres. Où dans le cas de la librairie realsense 2, le point (0,0,0) représente le centre, l'axe des x positifs va vers la droite, celui des y positifs vers le bas et celui des z positifs pointe tout droit. En sachant cela, nous pouvons, si besoin est, approximer directement la position du vidéoprojecteur par rapport à la caméra et la modifier. Les valeurs sont séparées par les trois axes et sont en float (INTEL REALSENSE, 2019).

FrameProcessProfil

Cette variable est la seule qui nous est utile uniquement pour l'application de calibration. Son but est de contenir les informations de la classe *FramProcessProfil*, car il est fastidieux de saisir ce profil à chaque calibration, si certains éléments de l'environnement où se trouve le bac à sable ne changent pas. Comme énoncé précédemment et avec des précisions :

- "contrast" doit être supérieur à 0.
- "brightness" n'a pas de restriction, sachant qu'un pixel doit contenir une valeur comprise entre 0 et 255.
- "minDistance" est en pourcentage par rapport à la largeur de l'image traitée supérieur à 0.
- "cannyEdgeThreshold" est compris entre 0 et 255

- "houghAccThreshold" est compris entre 0 et la largeur multiplié à la hauteur de l'image traitée
- "minRadius" est en pourcentage par rapport à la largeur de l'image traitée supérieur à 0 et est plus petit ou égal à maxRadius.
- "maxRadius" est en pourcentage par rapport à la largeur de l'image traitée supérieur à 0 et est plus grand ou égal à minRadius.

7.4. APPLICATION DE CALIBRATION

L'application de calibration est l'application permettant de créer le fichier de configuration complet et qui se base sur la classe *SandboxSetup*. Comme vu précédemment dans l'architecture *SubApp* correspond à une étape de calibration et *MainWindow* est la classe utilisant ces étapes.

SubApp

Cette classe représente une étape de la calibration, soit une forme dans l'application. Elle hérite de la classe *Widget* de Qt, afin de permettre une insertion de la classe qui en découle d'être traitée comme un *Widget*. Cette classe *SubApp* fournit quelques fonctions permettant de la gérer comme une étape d'un processus. Elle propose donc les méthodes clés suivantes :

- *checkRoutine()* : permet de vérifier si l'étape s'est bien déroulée en retournant un booléen
- *valideRoutine()* : permet de valider l'étape afin de passer à la suivante
- *cancelRoutine()* : permet d'annuler l'étape comme si celle-ci n'avait pas encore commencé

La classe permet aussi d'envoyer un signal avec "sendNotif" et contiendra un int comme état de l'étape en cours. Cela permet à celle-ci d'envoyer une notification d'état au parent, soit l'application de calibration qui gère les étapes.

MainWindow

MainWindow est la classe de notre application de calibration, c'est elle qui va regrouper et ordonner les différentes étapes afin de générer correctement le fichier de configuration final. Cette application va instancier les éléments nécessaire afin d'exécuter les étapes dans l'ordre ci-dessous. Les étapes sont directement représentées par leur classe respective héritant de *SubApp*. Nous avons donc une liste de *SubApp* exécutée dans cet ordre :

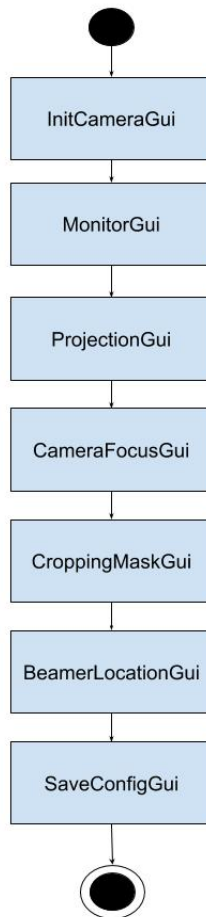


ILLUSTRATION 7.18 – Ordre d'exécution des classes des étapes de calibration

Afin de naviguer entre chaque étapes avec les boutons de l'interface "précédent" et "suivant", notre classe parente va exécuter "cancelRoutine" de la forme actuelle en cas de clique de l'utilisateur sur "précédent" ou va exécuter "verifyRoutine" si le bouton "suivant" est pressé, si la méthode ne retourne rien d'anormal, "valideRoutine" est à son tour exécutée avant de passer à la forme suivante.

Dans le cas où la forme actuelle a besoin d'interagir avec *MainWindow*, la forme peut envoyer un signal à travers "sendNotif" qui sera lié au slot "receiveNotif" de *MainWindow* et ainsi envoyer l'état.

InitCameraGui

Cette classe sert à vérifier la présence de la caméra et l'initialiser. Comme la routine d'initialisation est bloquante, elle est exécutée dans un thread à part.

MonitorGui

MonitorGui permet de choisir la sortie correspondant au vidéoprojecteur. Elle utilise la librairie *xrandr* afin de récupérer les informations liées aux écrans connectés et leurs résolutions. De plus un aperçu de l'écran est visible lors de la sélection d'une sortie. Le but était d'utiliser Qt afin de récupérer les écrans et résolutions de ceux-ci. Malheureusement, Qt ne permet de récupérer que les résolutions actives des écrans actifs. En faisant des recherches, nous avons trouvé que le peu de personnes voulant accéder à ces informations sous les systèmes linux, utilisaient *xrandr*, ou essayaient. La librairie offre une interface plus haut niveau, mais compatible uniquement avec Windows. Au vu du manque de documentation et du peu utilisation de la librairie sur les système linux, les utilisateurs ont tendances à exécuter la commande dans une console et traiter sa sortie. Comme il était fastidieux d'utiliser la librairie et que ça n'était pas une priorité, nous avons opté pour le traitement de la sortie de la console comme solution temporaire. Puis plus tard dans le développement, nous sommes revenus dessus et avons modifié le code pour qu'il utilise la librairie C. Afin de récupérer ces informations, il faut tout d'abord ouvrir une connexion avec le serveur gérant les différent écrans. Puis récupérer l'écran virtuel "screen", qui contient les informations des différentes fenêtres observant cet écran virtuel. Grâce à "screen" et ses informations, nous pouvons déterminer les sorties vidéo présentes et leurs résolution. De plus lorsqu'une sortie vidéo n'est pas connectée, elle possède une liste vide de résolution.

ProjectionGui

Cette étape projette un écran bleu dans la sortie sélectionnée à l'étape *MonitorGui*. Grâce à l'aperçu de la caméra en couleur, l'utilisateur peut ajuster la position de la caméra, sa rotation, l'affichage du vidéoprojecteur et sa netteté. Il est important que la caméra soit perpendiculaire au plan au-dessus du bac à sable.

CameraFocusGui

Cette étape permet d'ajuster les paramètres de traitement d'image, afin d'améliorer la détection de notre cible. Cela nous sera utile lors de la procédure de localisation du vidéoprojecteur. Notre interface ressemble à ceci :

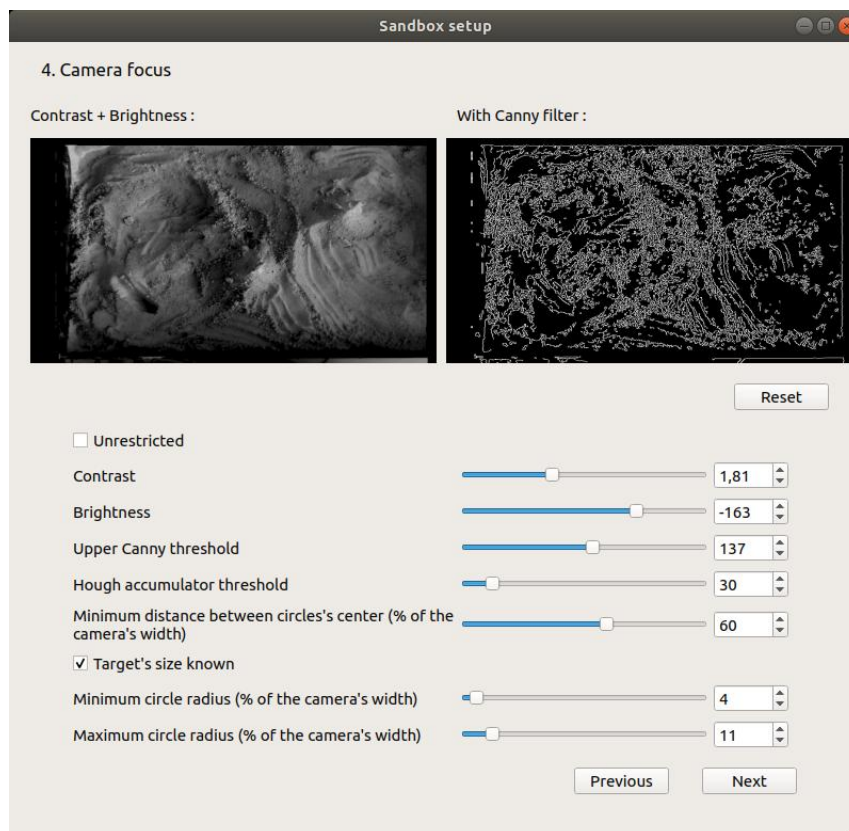


ILLUSTRATION 7.19 – Interface graphique de l'application de calibration, étape d'ajustement du traitement d'images

Tout d'abord, nous pouvons voir que nous avons deux retours vidéos. Le premier applique la méthode "editContrast" vu précédemment dans la classe *Beamer*. Le deuxième retour montre le premier retour, mais avec l'application de la fonction "Canny" d'OpenCV. Dans les deux cas, l'image traitée est une image en niveau de gris.

Ensuite observons la partie interactive de notre interface. Elle est composée de barres et de saisies numériques. Tout d'abord, nous avons les paramètres de contraste "Contrast" et luminosité "Brightness" affectant notre premier rendu. Puis nous avons les paramètres liés à la fonction "HoughCircles" d'OpenCV :

- Upper Canny threshold : Seuil affectant la détection de contours de la méthode Canny
- Hough accumulator threshold : Seuil déterminant un cercle dans notre accumulateur
- Minimum distance... : Distance minimum entre deux cercle détectés. La valeur est un pourcentage de la largeur
- Target's size known : Permet de préciser une fourchette de taille pour la cible à détecter
- Minimum circle radius... : Taille minimum du rayon du cercle lors de la détection

— Maximum circle radois... : Taille maximum du rayon du cercle lors de la détection

Finalement nous avons la checkbox "Unrestricted", actuellement, les valeurs sont bloquées à un intervalle de valeur raisonnable pour la majorité des cas. Mais si une des échelles ne convient pas à l'utilisateur, il peut cocher "Unrestricted" afin d'avoir le choix complet sur ses valeur dans la limite de la théorie.

CroppingMaskGui

Cette étape permet à l'utilisateur de définir le masque de découpage de la caméra. Cette classe utilise la classe *MaskEdit*, elle nous permet de dessiner le masque de découpage de l'utilisateur sur le flux vidéo de la caméra. Elle est nécessaire, car Qt ne fournit pas d'événement de dessin, le seul disponible étant celui exécuté à l'apparition de l'élément et dessine donc tout l'élément. Nous avons donc créé un composant affichant uniquement l'image de la caméra avec le masque tracé par dessus.

Durant cette étape, l'utilisateur peut modifier la position des coins du masque en les attrapant et déplaçant.

Lors de la validation, la rotation du masque afin de le redresser est appliquée autour du coin supérieur gauche, et la matrice de rotation sauvegardée est basée sur la rotation centrée au pixel haut gauche de l'image.

BeamerLocationGui

Cette étape est celle où nous déterminons la position du vidéoprojecteur. Pour ce faire, l'application est basée sur la routine énoncée dans *Beamer* de *SandboxSetup*. L'application va laisser tourner dans un autre thread l'affichage et le traitement d'image de la photo capturée par la caméra, et ainsi laisser le thread principal s'occuper de valider un point lorsque l'utilisateur clique sur le bouton "Lock".

Dans le thread secondaire, nous suivons cette démarche :

- récupération des images de couleur et profondeur de la caméra
- cherche les cercles dans l'image de couleur avec application du traitement sur l'image (qui utilise notre profil établi dans *CameraFocusGui*)
- traitement de l'image de couleur pour l'image d'aperçu (uniquement le contraste et la luminosité)

- dessine les cercles sur l'image d'aperçu
- construit l'image à projeter avec la croix
- affiche l'image d'aperçu et projette la croix

Dans le thread principal, nous avons la méthode de validation d'un point :

- déprojection du point en 3D
- trouve la meilleur droite passant par les points enregistrés, s'il y en a assez
- fini la routine s'il y a assez de droite
- approxime la position du vidéoprojecteur à partir des droites estimées

SaveConfigGui

Cette dernière étape sauvegarde la configuration actuelle dans un fichier de configuration à l'endroit où l'application est exécutée. Elle affiche un message indiquant si la configuration a bien été sauvée.

TESTS ET RÉSULTATS

8.1. OPTIMISATION

La librairie fonctionne, mais elle souffre de mauvaises performances, c'est pourquoi le but est d'entamer une amélioration de celles-ci. Pour ce faire, nous avons utilisé des outils de profiling, afin de localiser les zones à améliorer. Tout d'abord nous avons commencé à utiliser Valgrind, car son interface permettait de lire les données graphiquement, contrairement à Gprof qui retourne les informations dans la console. Ensuite en analysant les données retournées par Valgrind, nous voulions voir si les conclusions tirées d'un outil de profiling s'appliquaient pour tout outils. Nous avons donc testé avec Gperftools, qui est un autre outil graphique de profiling. Nous avons remarqué que les deux nous donnaient différents retours, mais que certaines informations étaient quand même redondantes. Enfin en testant avec Gprof, nous avons aussi remarqué qu'une fonction se démarquait des autres. Cette fonction permettait de reconstruire l'image lors de que la fonction d'ajustement de l'image projetée ressortait, plus précisément à cause du parcours de chaque pixel qu'elle effectue pour ajuster les valeurs de la sortie.

L'idée de paralléliser le traitement de l'image était alors concevable, mais pas raisonnable pour le moment, car il fallait s'assurer que le code était bien optimisé sur [Central Processing Unit \(CPU\)](#), avant de l'adapter sur [GPU](#). Donc nous avons décidé d'améliorer le code existant.

Pour ce faire, nous nous sommes attardés sur la gestion de la mémoire des buffers utilisés lors d'une utilisation standard de la librairie. C'est à dire, récupération d'une image de couleur ou profondeur, puis ajustement de l'image projetée. En ce qui concerne la caméra, nous avons limité les buffers à la capture des images couleur et profondeur (avec chacune leur buffer) et le retour d'une copie de ces buffers lors de la récupération de ces images.

Puis lors de l'ajustement de l'image, nous gardons en mémoire les buffers utilisés dans la fonction. C'est à dire celui contenant l'image de sortie et les quatres autres utilisés dans la fonction d'ajustement. Ce qui nous fait cinq zones mémoires de la taille de l'image de sortie. Enfin une copie est faite de l'image de sortie, ce qui nous rajoute un buffer.

Lors de l'analyse avec Gperftools, nous avons obtenu un graphique (disponible en annexe). Nous sommes mitigés face à ce graphique, car nous avons des informations qui paraissent cohérentes, mais nous avons aussi l'inverse, comme le fait que la fonction "HoughCircle" soit

appelé lors d'une redimension d'une matrice.

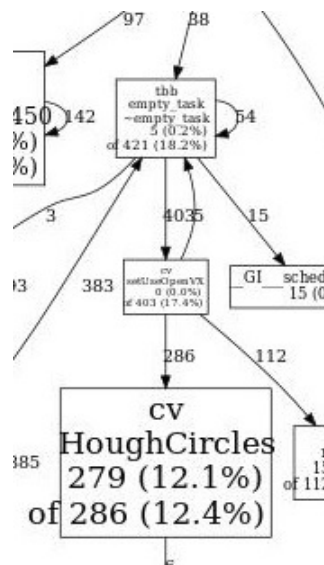


ILLUSTRATION 8.1 – Temps passé dans HoughCircle lors de l'exécution du programme utilisateur

Alors que cette fonction ne devrait pas être appelée, le programme a passé quand même 279 "tick" sur les 1875 totaux dans cette fonction. Toutefois, le fait que le filtre spatial de la librairie realsense prenne une place aussi importante lors de l'exécution est tout à fait envisageable.

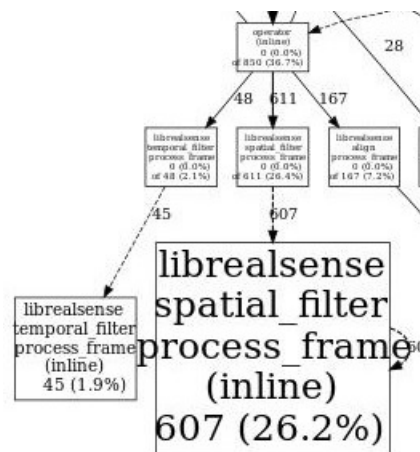


ILLUSTRATION 8.2 – Temps passée dans spatial filter lors de l'exécution du programme utilisateur

On remarque qu'elle a un compteur à 607, soit 26% du temps d'exécution totale. De plus nous voyons aussi que la méthode "copyPixelsInto" se démarque de la majorité.

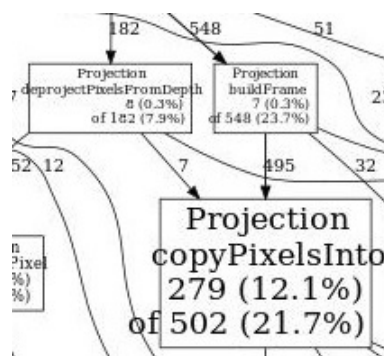


ILLUSTRATION 8.3 – Temps passée dans copyPixelsInto lors de l'exécution du programme utilisateur

Enfin, nous voyons que 21% du temps d'exécution est passé dans la fonction et ses sous-appels de fonction, dont un peu plus de la moitié directement dans "copyPixelsInto". Cependant, comme nous avons décidé de simplifier le code, cette méthode n'existe plus et ce graphique est donc moyennement représentatif de l'exécution standard de la librairie. Comme le temps manquait, nous n'avons pu aller plus loin dans l'optimisation.

8.2. TESTS ET RÉSULTATS

Nous avons testé les points clé de la librairie. Tout d'abord nous avons testé la précision de la librairie en utilisant l'application de coloration de niveau sur un terrain plat et en creusant des trous comme s'il s'agissait d'une grille. Nous avons vérifié que seule la marge d'erreur notable était liée à l'approximation de la position du vidéoprojecteur dans la configuration.

Ensuite, nous avons vérifié la résolution des images projetées, en s'assurant qu'elle ne dépendait pas de la résolution de la caméra. Pour tester nous avons simplement affiché une image de haute résolution et le résultat était positif.

Puis, nous voulions tester le gain de performance lié à l'optimisation de la mémoire. Pour ce faire, nous avons calculé le temps d'initialisation de nos variables. Malheureusement, l'allocation atteignait que rarement 1 milliseconde sans l'optimisation et jamais avec.

Enfin, nous avons fait une application C++ de tests de performance, calculant le temps passé sur chaque fonctions principales utilisées dans une exécution standard et répétée d'une application. En répétant ces opérations un nombre de fois prédéfini, les temps sont enregistrés dans un fichier et peuvent être lus par une application python qui indiquera le temps moyen passé dans chacune des fonctions et le nombre d'images par seconde moyen.

Temps moyen en millisecondes :

Version	Capture	recupère l'image de profondeur	recupère l'image de couleur	Ajuste l'image à projeter	moyenne d'IPS
basse qua- lité et imprécis	18.37	0.06	0.05	84.61	9.7
moyenne qualité et précis	15.06	0.12	0.01	57.6	13.74
bonne qua- lité et précis	14.76	0.31	0.11	210.25	4.44

TABLEAU 8.1 – Résultats des tests d'exécution de l'application de tests

Dans ce tableau, nous pouvons voir que la première version avait 9 images par secondes en moyenne, car la correction de la précision et d'autres facteurs ont permis d'accélérer le processus global, notamment celui de l'ajustement de l'image. Mais ce qui est impressionnant, c'est l'augmentation du temps de traitement entre la deuxième version et la troisième. Ce changement est dû à l'image de profondeur utilisée afin d'adapter l'image, dans la deuxième version, la résolution de l'image de profondeur est propre à la caméra de profondeur, mais dans la dernière version elle est ajustée à la résolution de l'image à projeter, soit 1400x1050 dans cette série de test. Les moyennes présentes dans ce tableau sont tirés d'un échantillonnage de 200 images projetées.

8.3. UTILISATION

La librairie générée par le Makefile retourne un fichier "libsandbox.so", qu'il va falloir inclure dans les projets C++. Pour ce faire, il faudra indiquer le chemin des en-têtes de fichiers lors de la compilation et indiquer le chemin vers le fichier .so lors du lien entre les librairies. De plus, pour pouvoir exécuter l'application de calibration, il faudra indiquer à la console où trouver le fichier .so nécessaire à l'exécution de l'application. Pour ce faire, il suffit de sauvegarder le chemin vers le fichier .so dans la variable d'environnement LD_LIBRARY_PATH. Une fois la calibration faite, l'application utilisateur devra respecter cette routine, afin de s'assurer du bon fonctionnement de la librairie.

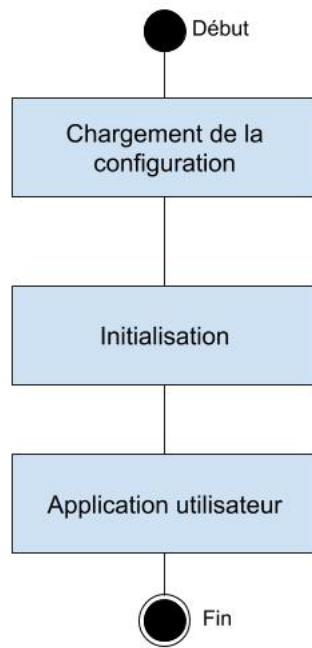


ILLUSTRATION 8.4 – Routine d’exécution d’un programme utilisateur

Nous avons l’initialisation qui permet de vérifier que les composants physiques soient disponibles. Puis le chargement de la configuration. Une fois tout validé, l’application utilisateur pourra s’exécuter.

CONCLUSION

Les architectes paysagistes peinent à modifier leurs maquettes de terrain une fois qu'elles sont créées. C'est pourquoi en se basant sur le projet de réalité augmentée de l'UC Davis, nous avons mis en place un bac à sable de réalité augmentée qui leur servira de substitut à leurs maquettes. De plus, il est désormais facile de déterminer la configuration d'autres infrastructures et dans différents milieux, grâce à l'application de calibration réalisée en Qt qui générera un fichier YAML contenant la configuration. L'application produite est basée sur notre API de calibration en C++, afin de permettre la création d'autres applications de calibration si la nôtre ne convenait pas à leur besoin. En complément de la librairie de calibration, une autre librairie d'utilisation en C++ a été fournie afin de permettre la création d'applications utilisateurs. Cette dernière permet de lire la configuration déterminée grâce à l'application de calibration et permet d'utiliser la caméra et de projeter des images adaptées à l'infrastructure. Finalement une application de démonstration affichant les niveaux du bac en couleurs a été développée grâce à l'API d'utilisation.

9.1. AMÉLIORATIONS

La manière de déterminer la matrice de rotation pourrait être revue, afin de prendre en compte le centre du rectangle dessiné par l'utilisateur lors de la phase de calibration, cela permettrait la mise en place du vidéoprojecteur plus permissive. De plus, il serait judicieux d'essayer d'améliorer la vitesse de l'application des filtres sur les images prises par la caméra. Ainsi que l'amélioration de la vitesse de traitement d'une image à projeter grâce à de la parallélisation, ce qui permettrait aux applications utilisateurs d'être plus réactives. Finalement, nous pourrions faciliter l'installation de la librairie, afin de permettre aux développeurs C++ d'inclure la librairie directement à la racine du système d'exploitation. Ainsi que, la présence d'un exécutable pour l'application de calibration et la possibilité de choisir le lieu de sauvegarde du fichier de configuration en fin de calibration.

9.2. PERSPECTIVES

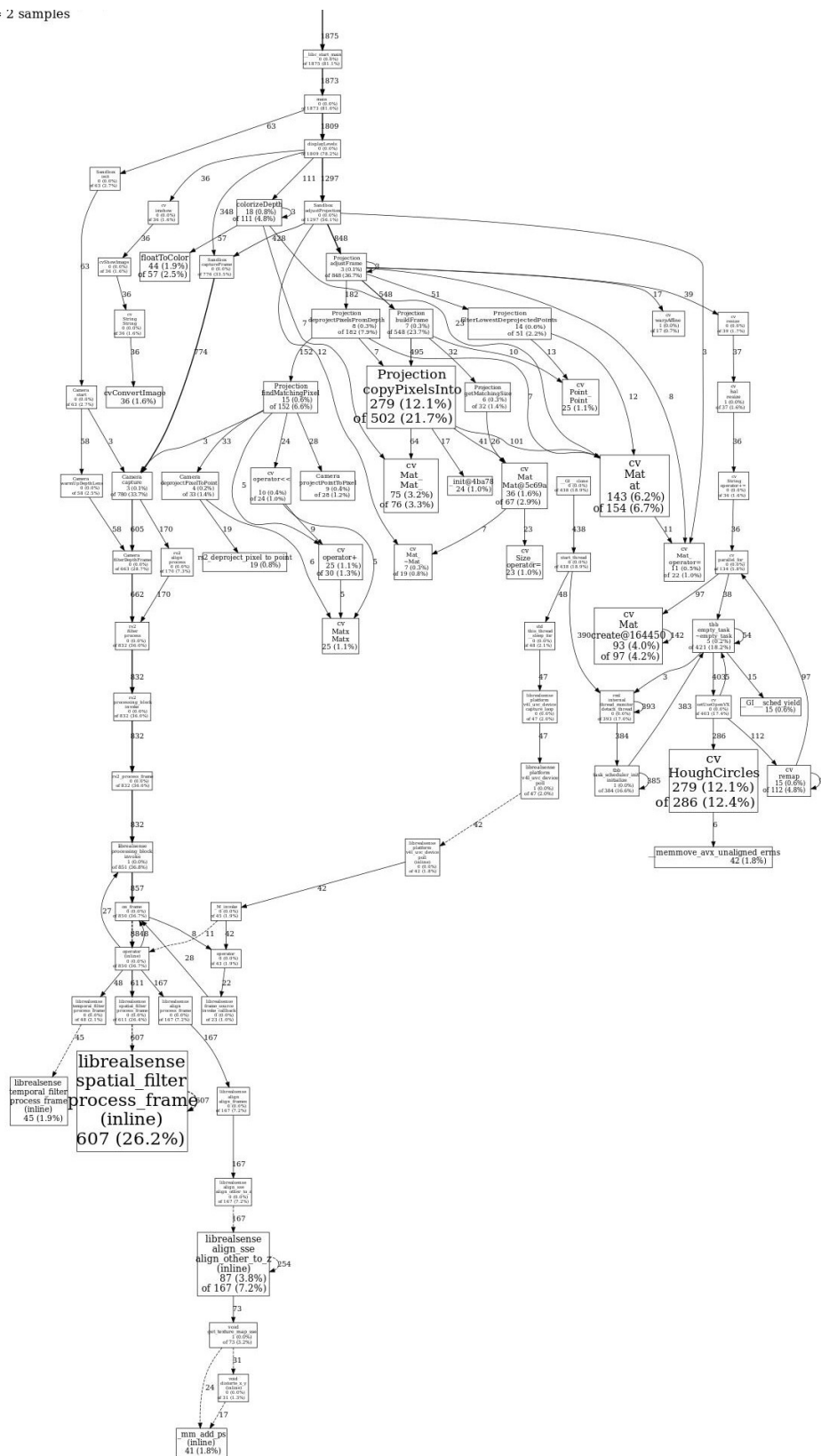
La flexibilité apportée lors de ce projet permet la création de différentes applications utilisateurs. Comme une application de reproduction de modèles de topologie, qui permettrait aux

architectes paysagistes de faire une simple sauvegarde de leur topologie et de la reproduction dans le bac une fois qu'ils en auraient besoin. Ou encore une application de bac à distance, qui connecterait deux bacs distants afin que les utilisateurs voient en temps réel à travers le monde et concrètement la topologie envisagée par leur partenaire.

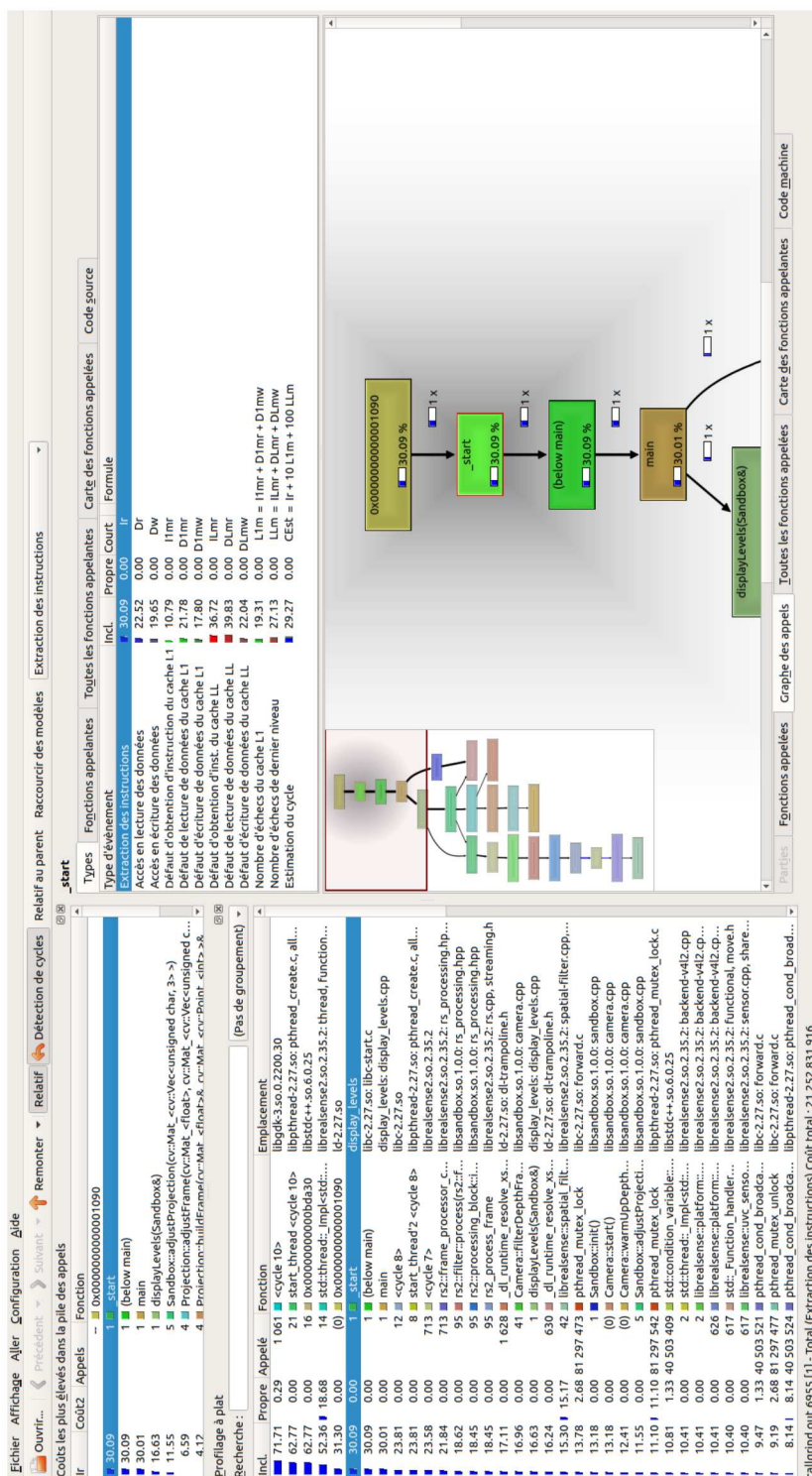
ANNEXES

ANNEXE 1 : ARBORESCENCE GPERFTOOLS

ipped edges with ≤ 2 samples



ANNEXE 2 : INTERFACE VALGRIND



ANNEXE 3 : EXEMPLE D'UTILISATION DE LA LIBRAIRIE

```
int main(){
    Sandbox sandbox;
    char *sandbox_conf_file = (char*)"./sandbox_conf.yaml";

    if(sandbox.loadConfig(sandbox_conf_file)){
        std::cout << "Failed to load the configuration" << std::endl;
        return 1;
    }

    if(sandbox.init()){
        std::cout << "Failed to initilize sandbox" << std::endl;
        return 1;
    }

    displayLevels( sandbox, sandbox.getProjection()->getDistanceTopSandbox(), 0.17f );
}
```

```
void displayLevels(Sandbox &sandbox, float top, float height){
    // Needed because the values retrieve from depth frames can change a bit although nothing has changed physically
    const float DEPTH_MARGIN_ERROR = 0.01f;

    char windowName[] = "Sandbox";
    cv::namedWindow(windowName, CV_WINDOW_NORMAL);
    cv::setWindowProperty(windowName, CV_WND_PROP_FULLSCREEN, CV_WINDOW_FULLSCREEN);

    sandbox.captureFrame();
    cv::Mat1f depth_cache = sandbox.getDepthFrame();
    cv::Mat1f new_depth = cv::Mat(depth_cache.size(), CV_32F);
    cv::Mat3b colorized = cv::Mat(depth_cache.size(), CV_8UC3);
    cv::Mat3b res = cv::Mat(depth_cache.size(), CV_8UC3);

    do{
        sandbox.captureFrame();
        new_depth = sandbox.getDepthFrame();
        // doesn't show what is above the top of sandox
        depth_cache.copyTo( new_depth, (new_depth < top) );
        // doesn't update little differences due to the camera depth captors
        new_depth.copyTo( depth_cache, (cv::abs(depth_cache-new_depth) > DEPTH_MARGIN_ERROR) );

        colorized = colorizeDepth(depth_cache, top, height);
        res = sandbox.adjustProjection(res, depth_cache);

        cv::cvtColor(res, res, CV_RGB2BGR);
        cv::imshow(windowName, res);
    } while (cv::waitKey(10) != ESCAPE_CHAR);

    cv::destroyAllWindows();
}
```

ANNEXE 4 : EXEMPLE DE FICHIER DE CONFIGURATION YAML

```
AdjustingMatrix:
  angle: -0.63377702050241091
  width: 3
  height: 2
  matrix: [0.999938846, -0.0110612698, 0, 0.0110612698, 0.999938846, 0]
DistanceTopSandbox:
  distance: 1.00800002
CroppingMask:
  x: 156
  y: 95
  width: 452
  height: 338
BeamerResolution:
  width: 1400
  height: 1050
BeamerPosition:
  x: 0.05
  y: 0.2
  z: -0.3
FrameProcessProfil:
  contrast: 1.8100000000000001
  brightness: -163
  minDistance: 60
  cannyEdgeThreshold: 137
  houghAccThreshold: 30
  minRadius: 4
  maxRadius: 11
```

ANNEXE 5 : API SANDBOX

AR Sandbox

[Main Page](#)
[Classes ▾](#)
[Files ▾](#)

Sandbox Class Reference

```
#include <sandbox.h>
```

Public Member Functions

Camera *	getCamera ()
Beamer *	getBeamer ()
Projection *	getProjection ()
int	init ()
void	captureFrame ()
cv::Mat_< cv::Vec3b >	getColorFrame ()
cv::Mat_< float >	getDepthFrame ()
cv::Mat_< cv::Vec3b >	adjustProjection (cv::Mat_< cv::Vec3b > &frame)
cv::Mat_< cv::Vec3b >	adjustProjection (cv::Mat_< cv::Vec3b > &frame, cv::Mat_< float > &depth)
int	loadConfig ()
int	loadConfig (char *path)

Detailed Description

API to build augmented reality applications which use the configuration file

Member Function Documentation

◆ adjustProjection() [1/2]

cv::Mat_<cv::Vec3b> Sandbox::adjustProjection (cv::Mat_< cv::Vec3b > & frame)

Adjust the frame to the topology by taking a picture with the depth camera

Parameters

frame Image to be adapted on the sandbox

Returns

The image adjusted to the topology in RGB

◆ adjustProjection() [2/2]

```
cv::Mat_<cv::Vec3b> Sandbox::adjustProjection ( cv::Mat_< cv::Vec3b > & frame,  
                                              cv::Mat_< float > & depth  
                                              )
```

Adjust the frame to the topology by adapting to the topology passed with the frame

Parameters

frame Image to be adapted on the sandbox

depth Depth frame representing the topology of the sand

Returns

The image adjusted to the topology in RGB

◆ captureFrame()

```
void Sandbox::captureFrame ( )
```

Capture a frame from the camera, this update the depth buffer and the color buffer of the camera

◆ getColorFrame()

```
cv::Mat_<cv::Vec3b> Sandbox::getColorFrame ( )
```

Get a copy of the colored RGB frame from the buffer

Returns

Colored frame represented by a matrix filled of 3 bytes values

◆ getDepthFrame()

```
cv::Mat_<float> Sandbox::getDepthFrame ( )
```

Get a copy of the depth frame from the buffer with values in meter

Returns

Depth frame represented by a matrix filled of float values

◆ init()

```
int Sandbox::init ( )
```

Initialize the components needed for the applications

Returns

int indicating if there was an error, where 0 indicates no error

◆ loadConfig() [1/2]

```
int Sandbox::loadConfig ( )
```

Load the configuration from the file at the default location

Returns

int indicating if there was an error, where 0 indicates no error

◆ loadConfig() [2/2]

```
int Sandbox::loadConfig ( char * path )
```

Load the configuration from the file at the specified location

Parameters

path to the configuration file

Returns

int indicating if there was an error, where 0 indicates no error

ANNEXE 6 : API SANDBOXSETUP

AR Sandbox

Main Page	Classes ▾	Files ▾
SandboxSetup Class Reference		
#include <sandboxSetup.h>		
Public Member Functions		
Projection *	getProjection ()	
Camera *	getCamera ()	
Beamer *	getBeamer ()	
int	saveConfig (char *path)	
int	saveConfig ()	
int	loadFrameProcessProfil (char *path)	
int	loadFrameProcessProfil ()	
int	loadCroppingMaskAndAdjustingMatrix (char *path)	
int	loadCroppingMaskAndAdjustingMatrix ()	
void	setupAdjustMatrix (std::vector< cv::Point2i > rectPoints, cv::Point2i center)	
void	setupCroppingMask (std::vector< cv::Point2i > rectPoints, cv::Point2i center)	
Detailed Description		
API to build setup applications which will generate the configuration file		
Member Function Documentation		

◆ loadCroppingMaskAndAdjustingMatrix() [1/2]

```
int SandboxSetup::loadCroppingMaskAndAdjustingMatrix ( char * path )
```

Load the **CroppingMask** and the **AdjustingMatrix** from a specific file

Parameters

path Location of the file to read from

Returns

int indicating if there was an error, where 0 indicates no error

◆ loadCroppingMaskAndAdjustingMatrix() [2/2]

```
int SandboxSetup::loadCroppingMaskAndAdjustingMatrix ( )
```

Load the **FrameProcessProfil** from the default location

Returns

int indicating if there was an error, where 0 indicates no error

◆ loadFrameProcessProfil() [1/2]

```
int SandboxSetup::loadFrameProcessProfil ( char * path )
```

Load the **FrameProcessProfil** from a specific file

Parameters

path Location of the file to read from

Returns

int indicating if there was an error, where 0 indicates no error

◆ loadFrameProcessProfil() [2/2]

int SandboxSetup::loadFrameProcessProfil ()

Load the **FrameProcessProfil** from the default location

Returns

int indicating if there was an error, where 0 indicates no error

◆ saveConfig() [1/2]

int SandboxSetup::saveConfig (char * path)

Save the configuration in the file at the path indicated

Parameters

path Location of the file will be save

Returns

int indicating if there was an error, where 0 indicates no error

◆ saveConfig() [2/2]

int SandboxSetup::saveConfig ()

Save the configuration in the file at default path

Returns

int indicating if there was an error, where 0 indicates no error

◆ setupAdjustMatrix()

```
void SandboxSetup::setupAdjustMatrix ( std::vector< cv::Point2i > rectPoints,  
                                       cv::Point2i               center  
                                       )
```

Find the adjusting matrix from the coordinates of the corner of the rectangle and the center of rotation

Parameters

rectPoints List of the points forming the rectangle
center Coordinates of the center of rotation

◆ setupCroppingMask()

```
void SandboxSetup::setupCroppingMask ( std::vector< cv::Point2i > rectPoints,  
                                       cv::Point2i               center  
                                       )
```

Find the cropping mask from the coordinates of the corner of the rectangle and the center of rotation by applying the rotation on the corners

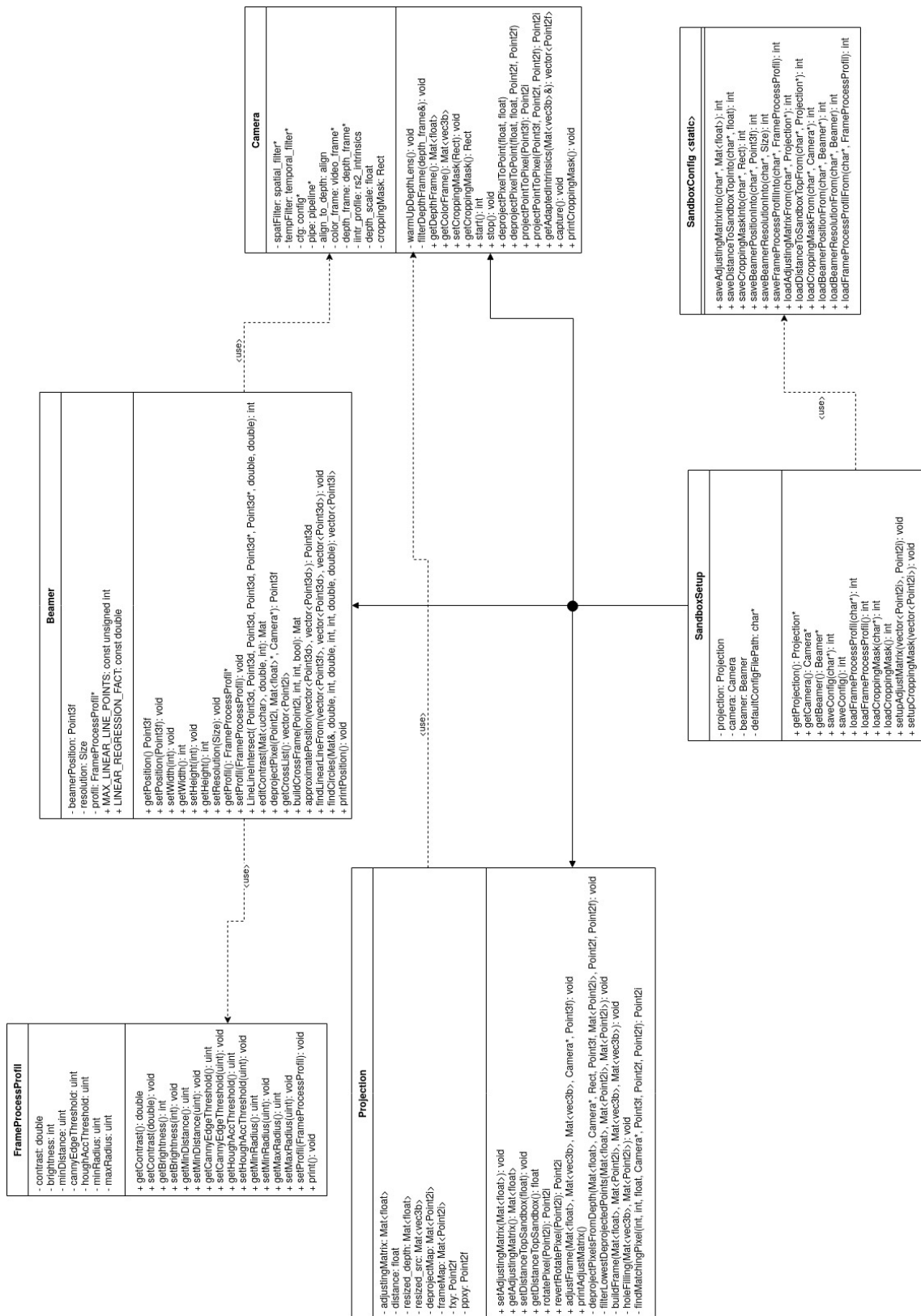
Parameters

rectPoints List of the points forming the rectangle
center Coordinates of the center of rotation

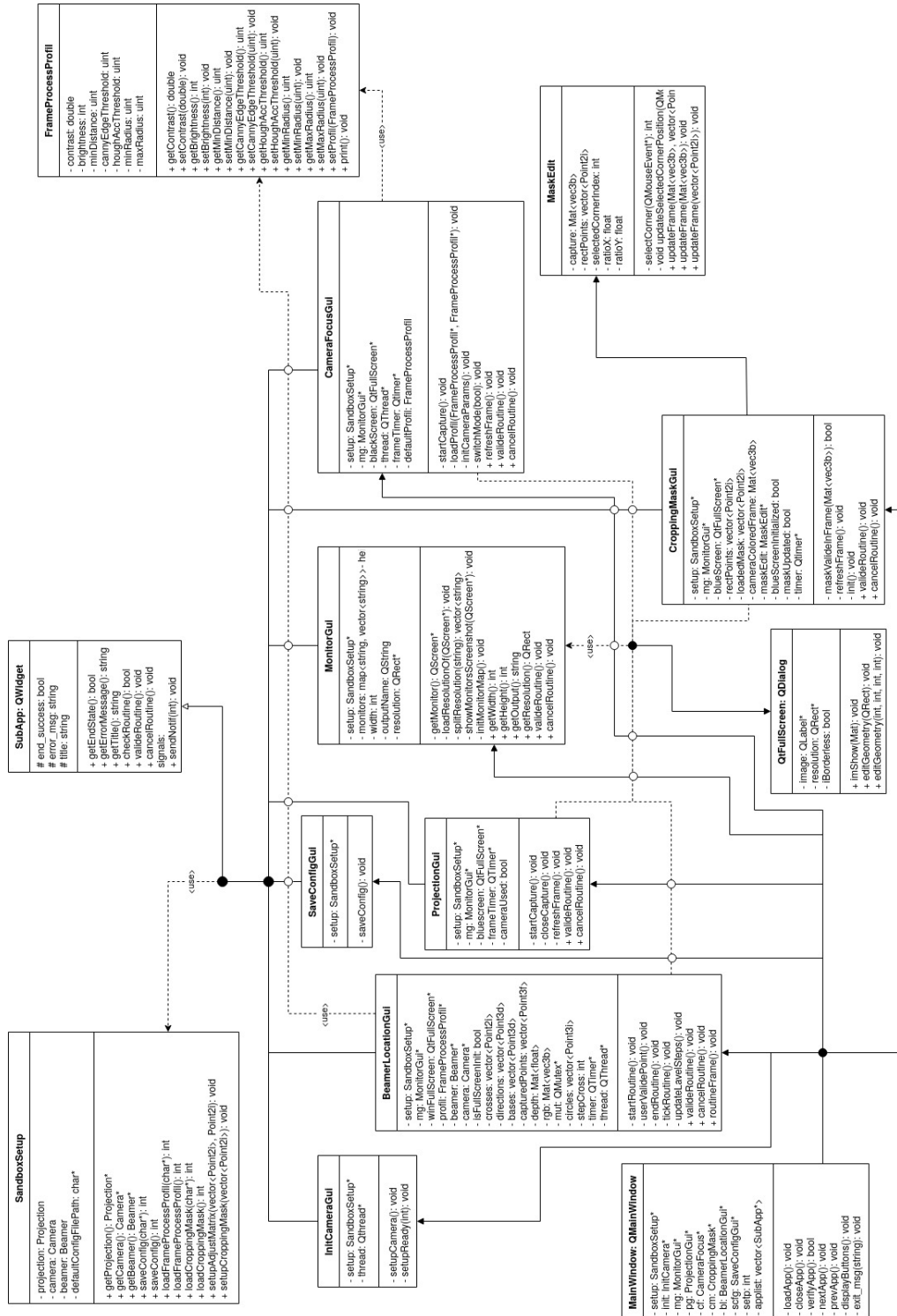
ANNEXE 7 : DIAGRAMME UML API SANDBOX



ANNEXE 8 : DIAGRAMME UML API SANDBOXSETUP



ANNEXE 9 : DIAGRAMME UML APPLICATION DE CALIBRATION



RÉFÉRENCES DOCUMENTAIRES

- Augmented reality sandbox* | Award-winning, certified | Ships internationally [ar-sandbox.com] [online] [visité le 2020-08-16]. Disponible à l'adresse : <http://ar-sandbox.com/>.
- BEDER, Jesse, 2020. *jbeder/yaml-cpp* [online] [visité le 2020-08-13]. Disponible à l'adresse : <https://github.com/jbeder/yaml-cpp>. original-date : 2015-03-30T02:52:32Z.
- CHRIS DEMETRIOU, 2007. *Google CPU Profiler Binary Data File Format* [gperftools.github.io] [online] [visité le 2020-08-15]. Disponible à l'adresse : <https://gperftools.github.io/gperftools/cpuprofile-fileformat.html>.
- DAVIS, UC, 2016. *Hardware – Augmented Reality Sandbox* [UC Davis Augmented Reality Sandbox] [online] [visité le 2020-08-13]. Disponible à l'adresse : <https://arsandbox.ucdavis.edu/instructions/hardware-2/>.
- Gprof*, 2016. In : *Wikipédia* [online] [visité le 2020-08-16]. Disponible à l'adresse : <https://fr.wikipedia.org/w/index.php?title=Gprof&oldid=127793681>. Page Version ID : 127793681.
- INTEL REALSENSE, a. *Depth Camera D415* [Intel® RealSense™ Depth and Tracking Cameras] [online] [visité le 2020-08-15]. Disponible à l'adresse : <https://www.intelrealsense.com/depth-camera-d415/>.
- INTEL REALSENSE, b. *Depth Camera D435* [Intel® RealSense™ Depth and Tracking Cameras] [online] [visité le 2020-08-15]. Disponible à l'adresse : <https://www.intelrealsense.com/depth-camera-d435/>.
- INTEL REALSENSE, c. *Intel® RealSense™ Technology* [Intel] [online] [visité le 2020-08-16]. Disponible à l'adresse : <https://www.intel.com/content/www/us/en/architecture-and-technology/realsense-overview.html>.
- INTEL REALSENSE, d. *IntelRealSense/librealsense* [GitHub] [online] [visité le 2020-08-15]. Disponible à l'adresse : <https://github.com/IntelRealSense/librealsense>.
- INTEL REALSENSE, e. *Stereo Depth* [Intel® RealSense™ Depth and Tracking Cameras] [online] [visité le 2020-08-16]. Disponible à l'adresse : <https://www.intelrealsense.com/stereo-depth/>.

INTEL REALSENSE, 2019. *Projection in Intel RealSense SDK 2.0* [Intel® RealSense™ Developer Documentation] [online] [visité le 2020-08-13]. Disponible à l'adresse : <https://dev.intelrealsense.com/docs/projection-in-intel-realsense-sdk-2#distortion-models>.

INTEL REALSENSE, 2020f. *Get started* [Intel® RealSense™ Developer Documentation] [online] [visité le 2020-08-13]. Disponible à l'adresse : <https://dev.intelrealsense.com/docs>.

Intel RealSense SDK 2.0 – Intel RealSense Depth and Tracking cameras [Intel® RealSense™ Depth and Tracking Cameras] [online] [visité le 2020-08-16]. Disponible à l'adresse : <https://www.intelrealsense.com/sdk-2/>.

Kinect, 2020. In : *Wikipédia* [online] [visité le 2020-08-13]. Disponible à l'adresse : <https://fr.wikipedia.org/w/index.php?title=Kinect&oldid=173049699>. Page Version ID : 173049699.

libxrandr [online] [visité le 2020-08-16]. Disponible à l'adresse : <https://www.x.org/wiki/libraries/libxrandr/>.

MANTIDPROJECT. *Profiling with Valgrind* [mantidproject] [online] [visité le 2020-08-13]. Disponible à l'adresse : <https://developer.mantidproject.org/ProfilingWithValgrind.html>.

MICHELE BRUSASCA, 2012. *Jean Nouvel Design* [Atelier Figura/Sfondo] [online] [visité le 2020-08-13]. Disponible à l'adresse : <https://www.figurasfondo.fr/maquette-darchitecture-en-carton-gris/> Section : Maquettes d'architecture.

MICROSOFT, 2020. *Buy the Azure Kinect developer kit – Microsoft* [Microsoft Store] [online] [visité le 2020-08-13]. Disponible à l'adresse : <https://www.microsoft.com/en-us/p/azure-kinect-dk/8pp5vxmd9nhq>.

OpenCV, 2020. In : *Wikipédia* [online] [visité le 2020-08-15]. Disponible à l'adresse : <https://en.wikipedia.org/w/index.php?title=OpenCV&oldid=972100697>. Page Version ID : 972100697.

OpenCV, 2020. In : *Wikipédia* [online] [visité le 2020-08-15]. Disponible à l'adresse : <https://fr.wikipedia.org/w/index.php?title=OpenCV&oldid=166957581>. Page Version ID : 166957581.

OPENCV DEV TEAM. *OpenCV : cv : :Mat Class Reference* [OpenCV] [online] [visité le 2020-08-13]. Disponible à l'adresse : https://docs.opencv.org/3.4/d3/d63/classcv_1_1Mat.html.

OPENCV DEV TEAM, 2019a. *Affine Transformations — OpenCV 2.4.13.7 documentation* [OpenCV] [online] [visité le 2020-08-13]. Disponible à l'adresse : https://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/warp_affine/warp_affine.html.

OPENCV DEV TEAM, 2019b. *Changing the contrast and brightness of an image! — OpenCV 2.4.13.7 documentation* [OpenCV] [online] [visité le 2020-08-13]. Disponible à l'adresse : https://docs.opencv.org/2.4/doc/tutorials/core/basic_linear_transform/basic_linear_transform.html.

OPENCV DEV TEAM, 2019c. *Geometric Image Transformations — OpenCV 2.4.13.7 documentation* [OpenCV] [online] [visité le 2020-08-13]. Disponible à l'adresse : https://docs.opencv.org/2.4/modules/imgproc/doc/geometric_transformations.html#resize.

OPENCV DEV TEAM, 2019d. *Geometric Image Transformations — OpenCV 2.4.13.7 documentation* [OpenCV] [online] [visité le 2020-08-13]. Disponible à l'adresse : https://docs.opencv.org/2.4/modules/imgproc/doc/geometric_transformations.html?highlight=warpaffine#warpaffine.

OPENCV DEV TEAM, 2019e. *Mat - The Basic Image Container — OpenCV 2.4.13.7 documentation* [OpenCV] [online] [visité le 2020-08-13]. Disponible à l'adresse : https://docs.opencv.org/2.4/doc/tutorials/core/mat_the_basic_image_container/mat_the_basic_image_container.html.

PAUL BOURKE. *Point, Line, Plane* [paulbourke.net] [online] [visité le 2020-08-13]. Disponible à l'adresse : <http://paulbourke.net/geometry/pointlineplane/>.

Qt, 2020. In : *Wikipédia* [online] [visité le 2020-08-15]. Disponible à l'adresse : <https://fr.wikipedia.org/w/index.php?title=Qt&oldid=170935077>. Page Version ID : 170935077.

QT COMPANY LTD, 2019. *Qt Documentation* [Qt] [online] [visité le 2020-08-13]. Disponible à l'adresse : <https://doc.qt.io/>.

rs-align [Intel® RealSense™ Developer Documentation] [online] [visité le 2020-08-16]. Disponible à l'adresse : <https://dev.intelrealsense.com/docs/rs-align>.

SANDBOX, The home of augmented reality. *Software* [The home of augmented reality sandbox] [online] [visité le 2020-08-14]. Disponible à l'adresse : <https://isandbox.co.uk/software/>.

SANJAY GHEMAWAT, 2008. *Gperftools CPU Profiler* [gperftools.github.io] [online] [visité le 2020-08-13]. Disponible à l'adresse : <https://gperftools.github.io/gperftools/cpuprofile.html>.

SINHA, Utkarsh. *Circle Hough Transform - AI Shack* [AI Shack] [online] [visité le 2020-08-16]. Disponible à l'adresse : <https://aishack.in/tutorials/circle-hough-transform/>.

SUGIH JAMIN. *EECS 380 : gprof Quick-Start Guide* [Umich.edu] [online] [visité le 2020-08-13]. Disponible à l'adresse : http://web.eecs.umich.edu/~sugih/pointers/gprof_quick.html.

Table 1 . Comparative specifications of Microsoft Kinect v1 and v2. [ResearchGate] [online] [visité le 2020-08-14]. Disponible à l'adresse : https://www.researchgate.net/figure/Comparative-specifications-of-Microsoft-Kinect-v1-and-v2_tbl1_313333776.

TEAM, opencv dev, 2019. *Feature Detection — OpenCV 2.4.13.7 documentation* [OpenCV] [online] [visité le 2020-08-13]. Disponible à l'adresse : https://docs.opencv.org/2.4/modules/imgproc/doc/feature_detection.html?highlight=houghcircles#houghcircles.

UC DAVIS, 2016a. *Augmented Reality Sandbox* [UC Davis Augmented Reality Sandbox] [online] [visité le 2020-08-13]. Disponible à l'adresse : <https://arsandbox.ucdavis.edu/>.

UC DAVIS, 2016b. *Augmented Reality Sandbox* [UC Davis Augmented Reality Sandbox] [online] [visité le 2020-08-13]. Disponible à l'adresse : <https://arsandbox.ucdavis.edu/>.

UNIVERSAL TERMINAL SYSTEMS (LLC), a. *Augmented reality sandbox | Award-winning, certified | Ships internationally* [ar-sandbox.com] [online] [visité le 2020-08-13]. Disponible à l'adresse : <http://ar-sandbox.com/>.

UNIVERSAL TERMINAL SYSTEMS (LLC), b. *Augmented reality sandbox* | *Award-winning, certified* | *Ships internationally* [ar-sandbox.com] [online] [visité le 2020-08-13]. Disponible à l'adresse : <http://ar-sandbox.com/>.

VALGRIND DEVELOPPERS. *Valgrind Home* [Valgrind] [online] [visité le 2020-08-15]. Disponible à l'adresse : <https://valgrind.org/>.

YAML, 2020. In : *Wikipédia* [online] [visité le 2020-08-16]. Disponible à l'adresse : <https://fr.wikipedia.org/w/index.php?title=YAML&oldid=173456652>. Page Version ID : 173456652.