Rapport d'ingénieur
Projet de 2ème année
Filière : Informatique des systèmes embarqués

SaHaRA V2

Présenté par : **Barthelat Lou et Monfroy Wendy**

Responsable ISIMA : Mesnard Emmanuel **20 mars 2019**

Responsable entreprise : Guillaume Bacques **projet de 60h**

Campus des Cezeaux. 1 rue de la Chébarde. TSA 60125. 63170 Aubière CEDEX.

Remerciements

Nous tenons tout d'abord à remercier Emmanuel Mesnard ainsi que Guillaume Bacques pour nous avoir proposé ce projet, ainsi que toute l'équipe du projet GeolVir3D dont le porteur de projet Philippe Labazuy et Karim Kelfoun. Nous adressons également nos remerciements à notre référent Romuald Aufrère.

Nous remercions le projet CAP20-25 pour avoir financé l'ordinateur performant utilisé pour ce projet.

Nous souhaitons également remercier Rémi Alègre et Yann Duband pour avoir proposé et réalisé le projet initial, et plus particulièrement M. Alègre pour nous avoir soutenu dans ce projet et nous avoir aidé à comprendre la première version du projet SaHaRA.

Nous adressons également nos remerciements à M. Hennequin et à M. Mesnard pour les cours de Unity et les travaux pratiques mis en place dans le cadre de la filière F1. Nous tenons enfin à adresser nos remerciements à Mme Mouzat pour les cours de communication qu'elle nous a dispensés ainsi que pour les instructions qu'elle a fournis concernant le présent rapport.

Table des figures et illustrations

Résumé

Le projet **SaHaRA V2** nous a été proposé par M. Emmanuel Mesnard dans le but de poursuivre et d'améliorer le projet originel SaHaRA.

L'objectif ici est d'exploiter les performances améliorées de la **Kinect V2** en transposant le travail réalisé lors du projet SaHaRA puis d'ajouter de nouvelles fonctionnalités pour permettre l'exploitation de la **réalité augmentée*** dans une formation de géologie.

Le développement de ce projet a été réalisé à l'aide du logiciel **Unity3D**. Nous avons d'abord étudié le fonctionnement de cet outil ainsi que l'exploitation qu'il nous permet de la **Kinect V2**. Nous avons ensuite essayé plusieurs méthodes d'affichage d'un gradient de couleur ou toute autre texture à partir de la **heightmap*** obtenue avant de sélectionner celle qui nous semblait la plus efficace à mettre en œuvre. Pour finir, nous nous sommes concentrées sur l'aspect calibration et interface du système afin de le rendre facilement utilisable et adaptable dans un environnement de travail qui peut être amené à varier.

A ce jour, nous avons un produit qui répond à la requête initiale, à savoir le passage du projet SaHaRA en version 2 en exploitant la **Kinect V2**. Cependant, nous n'avons pas pu complètement répondre à la demande d'ajoute de nouvelles fonctionnalités par manque de temps.

Mots-clés : Unity3D, SaHaRA V2, Réalité Augmentée, Kinect V2, Heightmap

Abstract

The **SaHaRA V2** project was proposed to us by Mr. Emmanuel Mesnard, aiming to continue and improve the original SaHaRA project.

The goal here is to use the improved performances of the **Kinect V2** by transposing the work done during the SaHaRA project and then adding new features to allow the exploitation of **augmented reality** in a geology formation.

The development of this project was realized using the **Unity3D** software. We first studied the operation of this tool as well as the exploitation it allows us of the **Kinect V2**. We then tried several methods to display a color gradient or other texture from the **heightmap** before selecting the one that seemed the most efficient to implement. Finally, we focused on the calibration and interface aspect of the system to make it easily usable and adaptable in a work environment that may vary.

To date, we have a product that responds to the initial request, namely the passage of SaHaRA version 2 by exploiting the **Kinect V2**. However, we were not able to completely respond to the demand for new features due to lack of time.

Key-Words: Unity3D, SaHaRA V2, Augmented Reality, Kinect V2, Heightmap

Table des matières

Introduction

Le projet SaHaRA v2 nous a été confié par notre professeur M. Emmanuel Mesnard dans le cadre du projet de 2ème année de l'ISIMA. Ce projet à été proposé en partenariat avec l'OPGC (Observatoire de Physique du Globe de Clermont-Ferrand) situé sur le campus des Cézeaux et financé par le projet CAP 20-25.

L'OPGC est un laboratoire de recherche et d'observation en géologie qui offre à ses étudiants une formation complète et approfondie notamment en météorologie et sur l'études des volcans. Dans le but d'améliorer l'accessibilité et la compréhension de leur formation, l'équipe de chercheurs de l'OPGC, composée de M. Guillaume Bacques, M. Philippe Labaeui et M. Karim Kelfoun, s'est tournée vers M Emmanuel Mesnard afin d'étudier la possibilité d'utiliser et améliorer le bac à sable habillé en réalité augmentée (SaHaRA) mis au point par M. Rémi Alègre et M. Yann Duband quelques années plus tôt dans le cadre de leur projet de deuxième année de l'ISIMA.

C'est donc dans cette optique que s'inscrit le présent projet SaHaRA V2. Il s'agit pour nous de récupérer le travail effectué au préalable par le précédent groupe, de le basculer en version 2 en utilisant la Kinect V2 plus performante que sa version 1, puis d'y ajouter des fonctionnalités plus pertinentes dans le cadre d'une étude géologique d'un sol. Ce projet à pour but de faciliter l'appréhension des notions de courbes de niveau ou de couches géologiques, entre autres choses, en permettant aux étudiants de visualiser en trois dimensions ce qu'ils ont l'habitude d'étudier en deux dimensions.

A l'heure actuelle, il existe de nombreux bacs à sables en réalité augmentée de démonstration s'appuyant sur la Kinect V1 comme c'était le cas pour le projet SaHaRA que nous avons repris. Cependant, il existe très peu de ces dispositifs utilisant la Kinect V2, pourtant plus performante que sa petite sœur. Cela s'explique, à priori, par l'arrêt rapide de sa commercialisation, ce qui la rend moins accessible au public de développeur et rend donc sa documentation bien moins fournie que pour sa première version.

Pour réaliser ce projet, il nous a fallu étudier en détails ce qui avait été réalisé au préalable par le précédent groupe avant de l'adapter aux spécificités de la nouvelle Kinect afin de produire un rendu* similaire à la précédente version. Puis il nous a fallu l'adapter aux demandes spécifiques de M. Bacques pour l'OPGC.

Pour présenter ce projet, nous verrons dans un premier temps le contexte dans lequel il s'inscrit. Puis, dans un deuxième temps, nous détaillerons notre démarche scientifique, nos choix ainsi que nos difficultés. Enfin, dans un troisième temps, nous décrirons précisément le fonctionnement de notre produit fini avant de discuter des résultats obtenus.

I. Contexte du projet

1. L'origine du projet

Depuis juin 2018, à l'OPGC (Observatoire de Physique du Globe de Clermont-Ferrand) et avec le CNRS (Centre National de Recherche Scientifique), Philippe Labazuy est porteur du projet GeolVir3D : Géologie Virtuelle et Réalité Augmentée en 3D. Leur objectif sur deux ans est de pouvoir déployer des outils permettant d'observer des objets naturels et de virtualiser des environnements géologiques.

C'est dans le cadre de GeolVir3D que naît l'idée de SaHaRA V2. Il s'agit de créer à partir d'un projet déjà existant une structure utilisable par des géologues, notamment dans le cadre de formations. Le projet est donc suivi par Philippe Labazuy, Karim Kelfoun et particulièrement Guillaume Bacques, qui a suivi la progression du projet de très près et a aiguillé l'orientation des recherches.

2. Détail de l’installation

Le bac à sable en réalité augmentée, abrégé SaHaRA (Sable Habillé en Réalité Augmentée), est un dispositif installé à ISIMA depuis quelques années et notamment mis en avant lors d'événements comme les portes ouvertes de l’école. Il s’agit d’un bac à sable classique surmonté d’une installation comportant 3 composants principaux : la Kinect V2, un vidéoprojecteur et une machine pour faire tourner le programme.

Figure I.2.1 — Le bac à sable SaHaRA

La Kinect V2 est utilisée pour l’acquisition des données de profondeur. En effet, sa caméra de profondeur permet de détecter à quelle distance un point se situe par rapport à l’appareil, et permet de manipuler les données nécessaires à la détermination de la hauteur du sable. Le nom anglais Depthmap* est souvent employé pour désigner cette carte de profondeur. Visuellement, il s’agit d’une image en niveaux de gris qui diffèrent en fonction de la distance par rapport à l’objectif. Avec les données de base, un objet apparaît de plus en plus clair en se rapprochant de la Kinect, et le fond apparaît plus sombre. Si un objet est trop près de la caméra de profondeur, il apparaît noir (pas de données).

Un vidéoprojecteur orienté en direction du sable permet de renvoyer l’image calculée par le programme en temps réel sur le sable. Il englobe une marge autour du bac à sable afin de pouvoir faire des ajustements du côté logiciel.

L'armature a été construite par Emmanuel Mesnard en 2017 pour la première version du projet. Elle permet de fixer le vidéoprojecteur et la Kinect de façon à ce que la structure reste stable. De plus, il est possible d'accéder au programme afin de le modifier tout en observant les changements.

Unity 3D est un logiciel imposé pour la programmation du bac. Ce logiciel est muni d'un moteur de jeu extrêmement puissant et capable de répondre aux attentes du projet en termes de fluidité et de rapidité d'exécution. En effet, chaque image doit s'afficher en temps réel à la vitesse de 30 images par seconde afin que l'œil humain ne détecte qu'un mouvement uniforme.

Unity met beaucoup d'objets et de modèles préfabriqués à la disposition des utilisateurs, avec une interface très visuelle qui peut être utilisée en minimisant le temps passé en programmation classique. Néanmoins, dans le cadre de ce projet particulier, il est indispensable de créer des objets propres à l'utilisation du bac à sable, donc de partir de zéro et de coder traditionnellement les entités. Le langage utilisé pour créer des scripts au sein de ce logiciel est le C#, qui permet d'optimiser un programme précis et d'effectuer exactement la tâche nécessaire.

La version déjà existante :

La version 1 du Sahara a été créée dans l'optique de proposer une expérience de réalité augmentée ludique et attractive au public. Elle est implémentée avec des détails esthétiques comme l'apparition de fleurs ou des couleurs réalistes, mais aussi avec un jeu vidéo. Elle permet de faire des démonstrations de façon autonome à un groupe de personnes. Cette version, équipée de la première Kinect (V1), existe déjà dans plusieurs universités dans le monde, notamment UCLA (University of California, Los Angeles) où un concept très similaire a été mis en place en 2015. Le projet, financé par la NSF (National Science Foundation), a été instauré pour l'éducation scientifique informelle d'enfants, et centré sur la formation des lacs et des plans d'eau. Le produit final est idéal pour une utilisation dans des musées de sciences en exposition en tant qu'activité ludique.

La première Kinect a été utilisée, comme dans le SaHaRA V1. Le vidéoprojecteur est disposé dans un angle par rapport à la configuration mise en place à ISIMA, où il est quasiment centré. Leur programme tourne sous Linux avec le Vrui VR toolkit et la latence (qui n'est pas indiquée mais visible sur les vidéos) est bien supérieure à celle du SaHaRA développé sous Unity. En effet, elle utilise un objet nommé Vrui codé en C++ pour gérer la réalité augmentée, ce qui n'est pas optimisé pour cette utilisation.

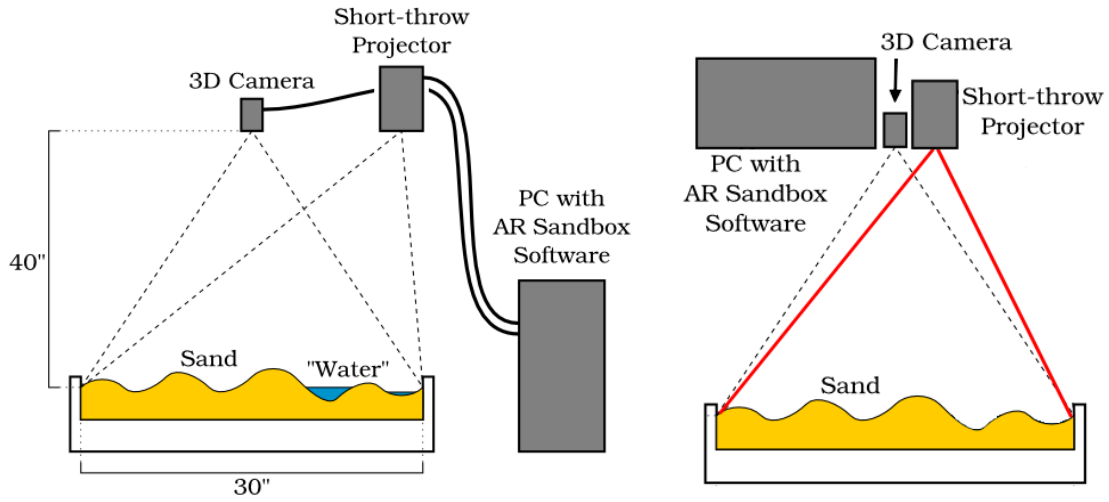


Figure I.2.2 — Comparaison de la disposition des bacs en réalité augmentée. UCLA (à gauche) et ISIMA (à droite)

3. Objectifs du projet

Le SaHaRA V2 doit prendre en compte plusieurs améliorations. La première est l'amélioration physique du matériel. La caméra utilisée pour détecter la profondeur du terrain est une caractéristique de la Kinect. Il n'existe au monde aucun bac en réalité augmenté équipé de la Kinect V2. Ce dispositif offre pourtant une qualité d'image largement supérieure à la version précédente, notamment dans l'acquisition des données de profondeur. En effet, la première version de la Kinect ne renvoyait pas une image aussi détaillée et les contours étaient moins distincts, comme on peut le constater sur la figure I.3.1.

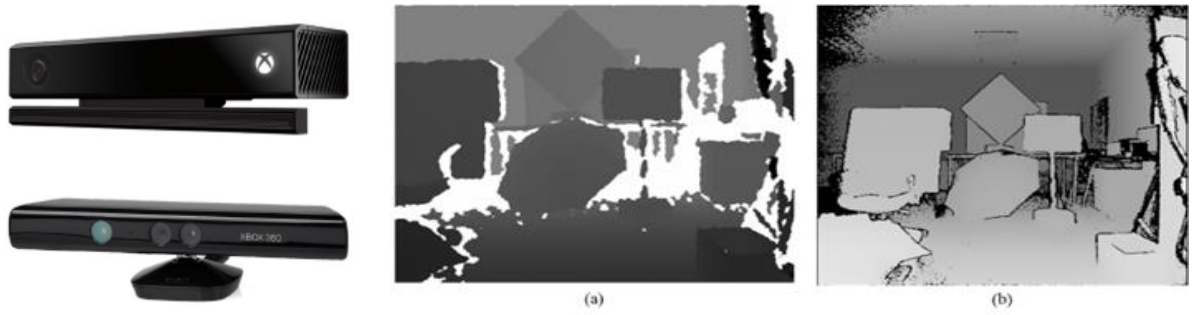


Figure I.3.1 — Comparaison visuelle des données de profondeur : Kinect V1 (à gauche) et V2 (à droite)
(image de gauche) Kinect V2 (en haut) et Kinect V1 (en bas)

La mise à jour matérielle peut donc permettre une précision accrue et nécessaire afin d'obtenir des données utilisables par des géologues. De plus, une machine plus puissante nous a été fournie afin de conserver une qualité d'image optimale. La résolution de l'image en profondeur est de 512 par 424 pixels et la fréquence de rafraîchissement est de 30 Hz, ce qui est assez rapide pour apparaître très fluide à l'œil nu. En comparaison, la caméra de profondeur de la Kinect V1 n'offrait une résolution que de 320 par 240 pixels, comme visible dans le tableau comparatif des performances ci-dessous.

	Kinect V1		Kinect V2	
	Résolution (pixel * pixel)	Fréquence de rafraîchissement de l'image (Hz)	Résolution (pixel * pixel)	Fréquence de rafraîchissement de l'image (Hz)
Couleur	640*480	30	1920*1080	30
Profondeur	320*240	30	512*424	30
Infrarouge	320*240	30	512*424	30

Figure I.3.2 — Tableau comparatif des performances des Kinect V1 et V2 en qualité d'image

En plus de mettre en place du matériel à la pointe de la technologie, l'objectif est d'afficher sur le sable un gradient de couleur optimisé ainsi que des courbes de niveau. En comparaison avec la V1, le projet est moins axé sur l'esthétique et plus dans l'optique de pouvoir développer un outil scientifique avancé permettant de faire des démonstrations dans le cadre de la géologie.

Dans le cadre de l'évaluation à ISIMA, l'objectif est d'atteindre un volume horaire de 60 heures de travail pour chaque membre du binôme, ainsi que de rédiger un rapport de projet et de préparer une soutenance finale. Il est important d'apprendre à gérer le temps imparti sur environ 5 mois ainsi que la communication avec les tuteurs et les membres de l'entreprise. Des réunions doivent être mises en place régulièrement pour faire le point sur les difficultés rencontrées et l'avancement du projet.

D'un point de vue plus personnel, ce projet permet au binôme de se familiariser avec le colossal moteur apporté par Unity. En effet, une des difficultés avec la prise en main de ce logiciel est de ne pas se perdre dans la multitude d'options, d'objets et de fonctions à la disposition de l'utilisateur. L'utiliser en projet permet donc d'appuyer les connaissances acquises en fin de deuxième année et offre un bagage de taille pour les compétences professionnelles.

4. Stratégie de résolution

Nous avons l'intention de diviser ce projet en différentes étapes, la première étant la prise en main de Unity. Le logiciel, extrêmement complet et complexe, doit être au moins partiellement maîtrisé afin de pouvoir appréhender le fonctionnement de la première version. Il s'est avéré que la version 1 n'était pas réutilisable telle qu'elle a été codée. De plus, les fonctions utilisées pour la gestion de la Kinect V1 diffèrent de celles qui doivent être utilisées dans ce nouveau projet. Une fois les outils de base comme les terrains, les scènes et les caméras maîtrisés, il est nécessaire de faire le lien entre Unity et la Kinect V2 afin de pouvoir exploiter le flux de données du capteur de profondeur dans le logiciel. L'utilisation du SDK est indispensable pour pouvoir utiliser l'appareil sur l'ordinateur, et ainsi pouvoir comparer les technologies et les différences entre les deux versions.

Même s'il s'est très vite avéré que la version 1 du projet SaHaRA n'était pas utilisable sans la Kinect V1 et sans le Wrapper* mis en place par l'équipe précédente, il est important de comprendre l'essentiel du code qui a été rédigé pour cette première version. Nous avons donc organisé un rendez-vous est avec l'un des étudiants qui ont mis au point la version 1 du SaHaRA. Il nous explique notamment comment générer une image à partir des données de profondeur et suggère de décrire tous les objets dans des scripts, expliquant que la version 1 utilise trop d'objets préfabriqués et est donc peu optimisée de ce point de vue.

La deuxième étape consiste en une optimisation du bac à sable ainsi développé pour une utilisation dans un cadre géologique. Une fois la mise à niveau du bac à sable pour être compatible avec une Kinect V2, il était demandé de réaliser quelques modifications pour correspondre au formalisme utilisé en géologie. Parmi ces modifications figurait notamment le passage de données 2D en données 3D.

Le tuteur de projet, M. Mesnard est régulièrement consulté pour parler des objectifs et adapter en fonction de l'avancement du projet. De l'équipe de GeolVir3D, M. Bacques est régulièrement invité à assister à des démonstrations des nouvelles fonctions mises en place.

Les deux membres du binôme travaillent sur des solutions aux problèmes individuellement, puis mettent en commun une à trois fois par semaine afin de choisir la solution la plus efficace et adaptée à chaque situation. Ainsi, chacune développe ses compétences personnelles et le travail dans la salle de projet est plus efficient.

Les fonctions principales sont dans l'ordre chronologique : l'affichage de la carte des profondeurs sous forme d'image en nuances de gris, l'affichage d'un gradient en fonction de la profondeur de chaque pixel, le rognement (crop) de la zone de vue afin de ne travailler que sur le bac à sable et pas sur le cadre plus large, la déformation de l'image pour compenser le décalage dans le positionnement du vidéoprojecteur, l'affichage du résultat en plein écran sur cet appareil, ainsi que l'affichage de courbes de niveau.

II. Réalisation

1. Les outils à disposition

1.1. La Kinect 2

Il s'agit de l'outil central du présent projet, il nous est donc nécessaire d'en comprendre le fonctionnement ainsi que ses limites.

Figure 1.2.1 — Kinect2

Caractéristiques :

Contrairement au premier projet SaHaRA, nous utilisons la deuxième version de la Kinect, plus récente et donc plus performante que son ancienne version. On note que cette version permet de bien meilleures performances, notamment au niveau du flux de profondeur qui est au format 512x424 pixels contre 320x240 pixels pour la précédente version. Cela permet une meilleure définition, surtout pour de plus grandes distances.

Par ailleurs, cette Kinect présente une profondeur de champ d'environ 8m pour une précision de l'ordre de quelques micromètres (encore une fois plus précis que son ancienne version). Une telle précision n'est pas nécessaire pour ce projet puisque nous ne faisons pas de mesures précises (cela serait tout de même possible et exploitable), cependant, la fluidité du traitement des données n'en est pas perturbée. En revanche, on remarque la présence d'une zone d'environ 30cm à proximité du capteur pour

laquelle il est impossible d'obtenir une mesure de profondeur. Cette zone sera donc à prendre en compte si la structure vient à être modifiée (dans le cas d'un modèle miniaturisé pour entrer dans une salle de classe par exemple).

D'autre parts, la nouvelle Kinect dispose d'un champ de vision plus large que sa précédente version : 70° horizontaux et 60° verticaux contre 57° et 43,5°. Que ce soit pour l'ancienne version ou la Kinect V2, le champ de vision du capteur de profondeur est parabolique, ce qui influe évidemment sur les profondeurs mesurées. Il existe en effet un écart entre la hauteur cartésienne du sable par rapport à celle de la Kinect et cette hauteur en coordonnées sphérique effectivement mesurée. Nous devrions donc théoriquement observer une variation de couleur lors de la projection du gradient correspondant à la profondeur sur une surface plane. Cependant, cet effet s'est avéré négligeable dans la pratique : la dérive en profondeur n'était pas perceptible compte tenu du faible angle qu'il existe entre le capteur et le bord du bac à sable.

Bibliothèques Unity3D disponibles :

Pour le précédent projet, le choix avait été fait d'utiliser un Kinect Wrapper plutôt que le SDK disponible dans l'Asset Store d'Unity pour des raisons de compatibilités de version. En effet, le SDK est régulièrement mis à jour pour correspondre aux mises à niveaux de la Kinect alors que le wrapper choisi était adapté à une version plus ancienne de la Kinect ; leur choix s'était donc naturellement porté sur cette deuxième solution.

Dans le cas présent, le problème est inverse : la Kinect V2 est compatible avec le SDK qui est à jour pour cette version précise, mais elle n'est plus compatible avec le Kinect Wrapper majoritairement utilisé pour la première version du SaHaRA. Nous verrons un peu plus loin que ceci pose quelques problèmes pour la bonne marche de notre projet.

Il est important de noter que la commercialisation du kit de développement de la Kinect V2 a rapidement été arrêtée peu après ça sortie. La Kinect V2 est bien sûr toujours disponible sur le marché puisqu'il s'agit d'une pièce fragile de la célèbre console de Windows. Cependant, l'adaptateur permettant l'exploitation de la Kinect V2 par un ordinateur n'est plus aisément accessible.

1.2. Unity3D

Tout comme le projet SaHaRA, Le projet SaHaRA V2 exploite également le logiciel Unity puisque cet outil avait été choisi avec soin et qu'il aurait été malvenu de recommencer sur une autre plateforme ce qui avait déjà été réalisé. Pour des raisons de stabilité, la version utilisée était la dernière version d'Unity3D disponible au 1er septembre 2018, soit la version 2018.2.6f1. Pour toute la durée, aucune mise à jour du logiciel n'aura été effectuée car Unity à la mauvaise réputation de faire apparaitre des bugs dans un projet en cours.

Il s'agit d'un logiciel comportant un moteur graphique et physique, permettant de nombreux traitements de données et laissant une grande liberté dans l'architecture d'un projet. Il existe de nombreuses façons d'aborder un même projet sous Unity puisqu'il existe de nombreux objets utilisables présentant quelques variations dans leur traitement. Il est donc possible d'adapter notre utilisation du logiciel en fonction du format de données que l'on utilise, voire même aux habitudes de programmation du développeur.

En effet, Unity rend la programmation majoritairement par objet possible tout comme la programmation majoritairement par script. Le projet SaHaRA V2 s'appuie sur cette deuxième méthode de développement tandis que son ancienne version faisait plus appel à un développement par objets.

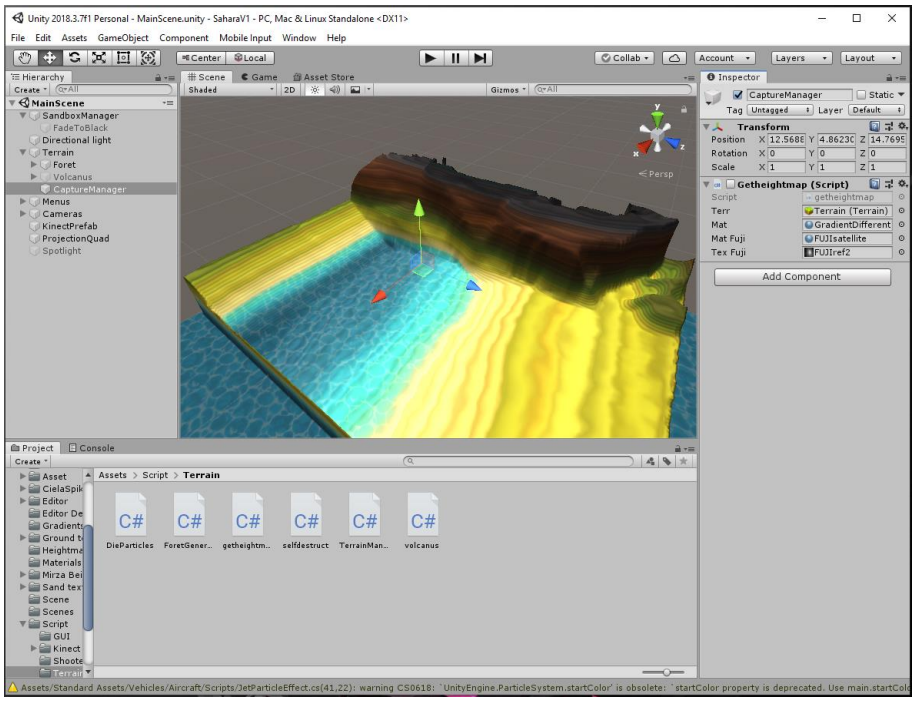


Figure II.1.2.1 — Interface du logiciel Unity

Ce logiciel s'appuie sur le scripting* en C# et permet de créer des projets compatibles avec de nombreuses plateformes (PC, Mac, consoles) ce qui lui procure une notoriété importante. Unity étant un logiciel assez accessible, il est apprécié du grand public et de fait, beaucoup de documentation et d'exemples d'utilisations sont disponibles sur Internet. Il existe d'ailleurs quelques tutoriels concernant l'utilisation de la Kinect 1 ou un bac à sable similaire à celui du projet SaHaRA. Cependant, nous n'avons que très peu trouvé d'équivalents pour un bac à sable exploitant une Kinect V2, cela est probablement dû à l'arrêt soudain de sa commercialisation. Ce manque de documentation accessible aura pour conséquence de ralentir l'avancement du projet, notamment à son commencement.

1.3. Le matériel physique

Le vidéoprojecteur :

Il s'agit d'un élément important du système puis qu'il nous permet de mettre effectivement en place l'aspect réalité augmentée du projet. Il n'est pas nécessaire de le modifier lors de la mise à niveau du dispositif car il a déjà été réglé pour projeter parfaitement en plein écran sur la zone du sable.

La carte graphique :

La carte graphique est en effet un élément indispensable pour un rendu final fluide. En effet, Unity s'appuie énormément sur ce composant lorsqu'il s'agit de calculer et d'afficher un rendu visuel. Le processeur n'a alors pas besoin de traiter les calculs liés à l'affichage et peut donc être exploité en amont pour exécuter plus de consignes et/ou pour améliorer la durée de rafraichissement d'une frame*game.

Ici, il s'agit du modèle GTX1080 du fabricant NVIDIA et fait partie de l'ordinateur financé par le projet CAP 20-25.

Cependant, l'accès à une carte graphique ne doit pas dissuader le développeur d'optimiser son code. En effet, le projet SaHaRA peut à terme être exploiter par différentes machines potentiellement moins performantes que celle dont nous disposons pour le développement.

2. Mise en place du projet

2.1. Répartition du temps de travail

Après avoir pris connaissance du projet et après un premier entretien avec notre tuteur ainsi que notre contact de l'OPGC, nous avons pu établir un premier plan général présentant la marche à suivre pour réaliser notre projet. Voici le diagramme de Gantt prévisionnel qui en découle :

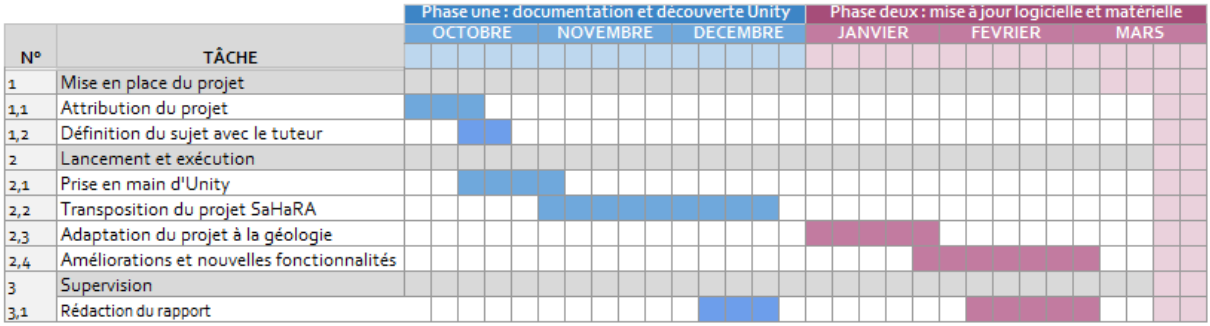
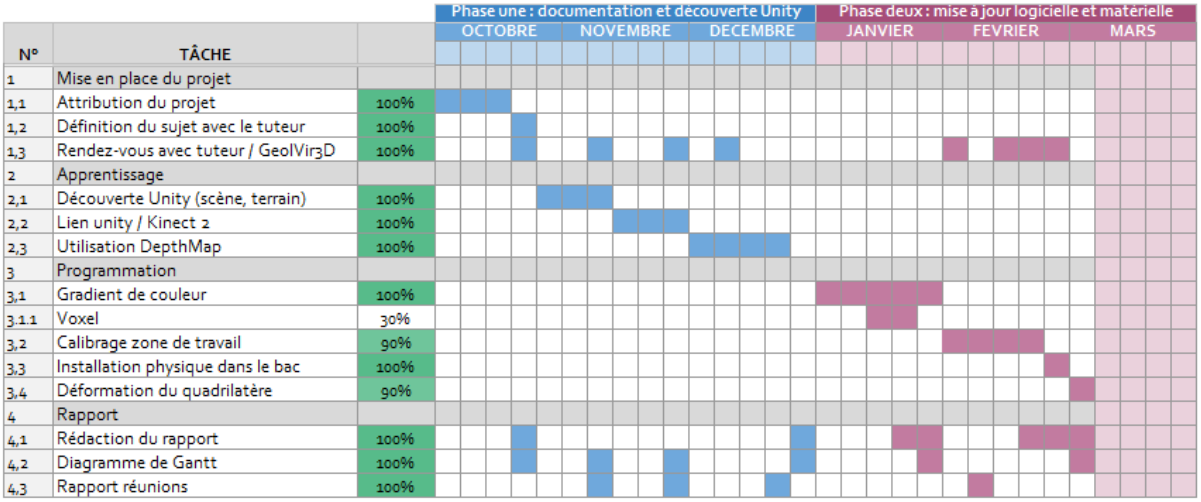


Figure II.2.1.1 — Diagramme de Gantt prévisionnel

Au commencement du projet, il a été envisagé de diviser le planning en deux parties principales. D'abord, la prise en main logicielle et matérielle de Unity, de la Kinect et de tous les outils à disposition. Dans cette partie, il est nécessaire de maîtriser Unity suffisamment pour pouvoir comprendre le code du SaHaRA V1 et voir si certaines portions sont réutilisables avec la nouvelle version. La seconde partie consiste en l'écriture des nouvelles fonctions. En effet, le produit doit être adapté pour une utilisation dans le cadre géologique.

Finalement, il s'est avéré que le temps consacré à la mise à niveau du projet SaHaRA prenait bien plus de temps que prévu. Cela explique la divergence observée entre le diagramme de Gantt attendu et celui obtenu.



N°	TÂCHE	
1	Mise en place du projet	
1,1	Attribution du projet	100%
1,2	Définition du sujet avec le tuteur	100%
1,3	Rendez-vous avec tuteur / GeolVir3D	100%
2	Apprentissage	
2,1	Découverte Unity (scène, terrain)	100%
2,2	Lien unity / Kinect 2	100%
2,3	Utilisation DepthMap	100%
3	Programmation	
3,1	Gradient de couleur	100%
3.1.1	Voxel	30%
3,2	Calibrage zone de travail	90%
3,3	Installation physique dans le bac	100%
3,4	Déformation du quadrilatère	90%
4	Rapport	
4,1	Rédaction du rapport	100%
4,2	Diagramme de Gantt	100%
4,3	Rapport réunions	100%

Figure II.2.1.2 — Diagramme de Gantt réel

2.2. Découverte d'Unity

La découverte du logiciel Unity fait entièrement partie des objectifs du projet. En effet, l'utilisation de ce logiciel est au programme de l'école seulement plus tard au cours du cursus, et l'outil n'a donc jamais été employé auparavant. Un temps considérable a donc été alloué à la maîtrise des outils essentiels de l'environnement de développement avant même de pouvoir établir les besoins précis du projet. Sans comprendre le logiciel un minimum, il est impossible de réfléchir à des solutions à mettre en place, ni d'évaluer le temps de travail pour l'écriture de nouvelles fonctionnalités. Cette phase d'apprentissage s'est prolongée sur les quelques premières semaines de travail.

En se familiarisant avec le logiciel, il a été possible de faire le choix des outils à utiliser au sein de Unity. Le choix se fait sur plusieurs critères, notamment l'optimisation et la simplicité d'utilisation. En effet, le temps étant limité, il est impossible de se permettre d'apprendre des centaines de nouveaux concepts pour créer des objets très complexes. L'avantage de la plupart des objets basiques est leur optimisation : ils sont plus légers et l'écriture de script est également moins lourdes qu'avec toutes les fonctionnalités intégrées dans les objets préfabriqués du logiciel.

Objet Plan :

Il s'agit d'un objet carré en seulement 2 dimensions. Le plan est texturé de façon à ce que l'image entière apparaisse exactement une fois sur sa surface : l'image est donc étirée en fonction de la taille du carré pour s'adapter au support. Cet objet est souvent utilisé pour créer un sol ou un mur dans une scène. Il est également possible de l'utiliser pour projeter une image sur sa surface, ce qui est intéressant pour pouvoir gérer facilement les dimensions du visuel projeté.

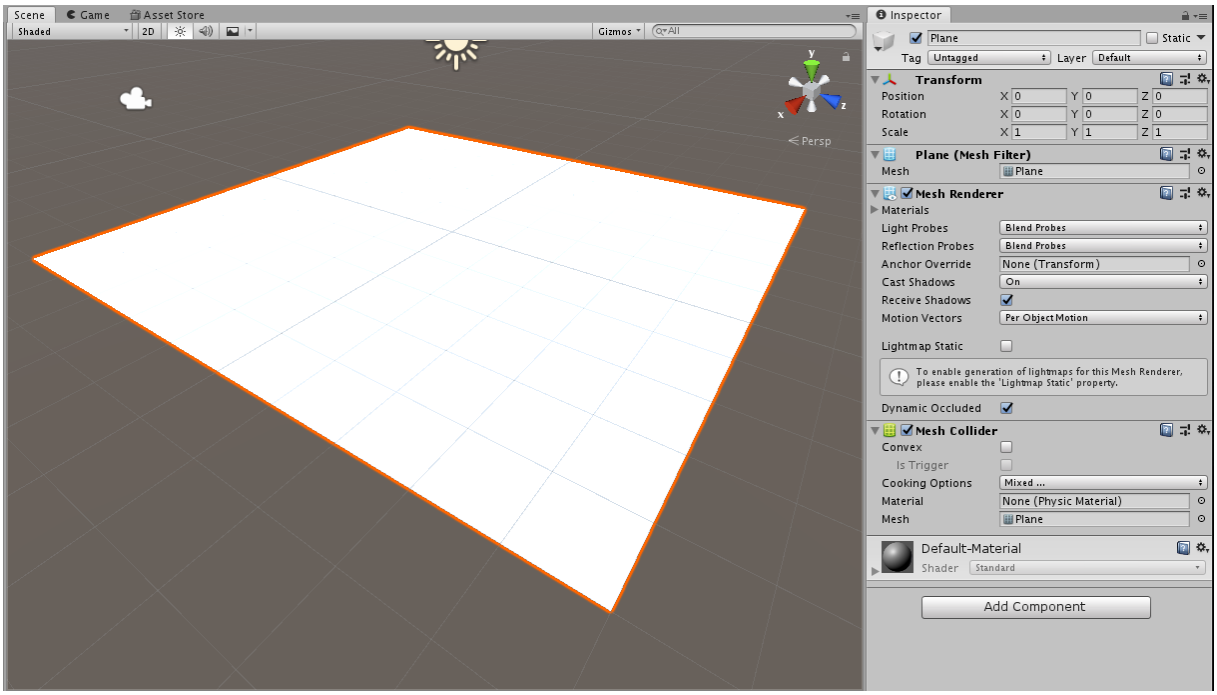


Figure II.2.2.1 — L'objet Plan dans Unity

Objet Terrain :

Il s'agit d'un plan pour lequel il est possible d'altérer l'altitude localement, par script ou par les outils qui y sont intégrés. Il est toujours possible de lui appliquer une texture. Il s'agit d'un objet bien plus gourmand en ressources qu'un simple plan. En effet, il possède une fenêtre d'inspection particulière avec de nombreuses fonctions intégrées telles que modifier l'altitude manuellement, peindre la texture, donner une valeur de hauteur à un point ou encore adoucir le dénivelé. Cet outil est donc très complet et optimisé pour la création d'un paysage dans une scène, mais encore trop lourd pour l'utilisation que l'on veut en faire.

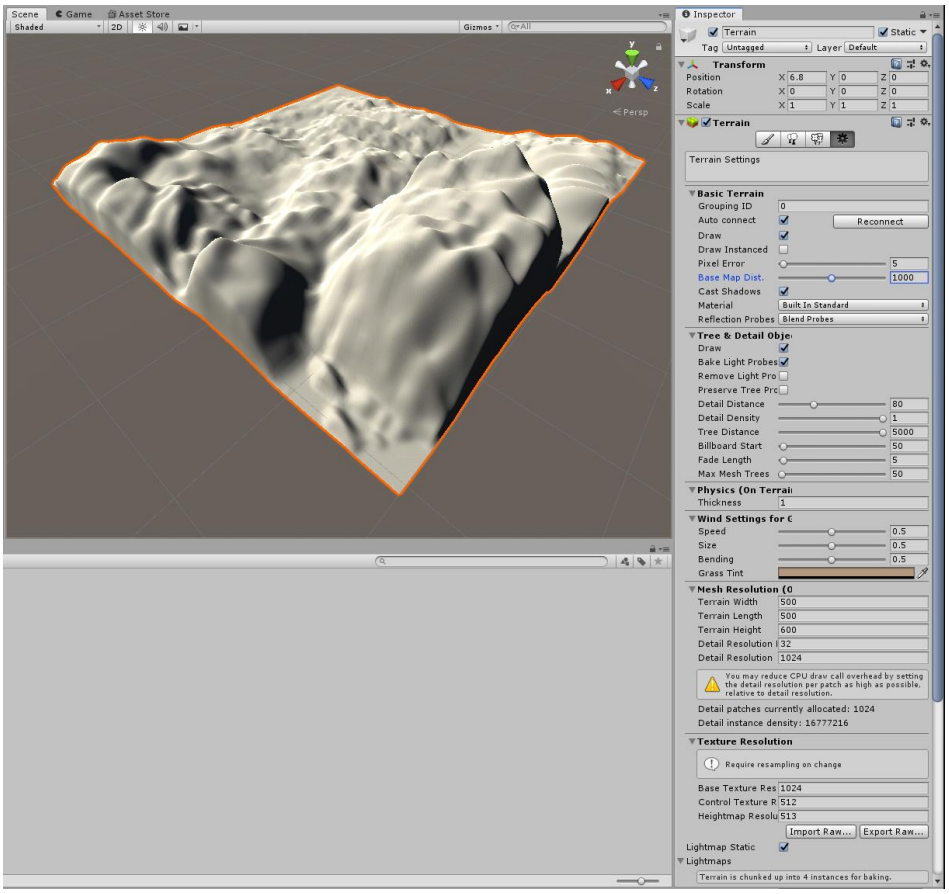


Figure II.2.2.2 — L'objet Terrain dans Unity

Création d'une surface :

Un processus de rendu peut être décomposé en plusieurs éléments. D'abord, on a le modèle de base contenant les sommets, coordonnées de textures, normales et autres caractéristiques comme les couleurs. Il y a également un Game Object* auquel sont assignés un Mesh Renderer* qui lui-même utilise un Material*. Enfin, le Material contient les textures que l'on veut appliquer à l'objet ainsi qu'un Shader*.

Le Mesh Renderer associé au Game Object utilise la géométrie générée par le Mesh Filter* (un attribut qui permet d'utiliser un Mesh* stocké dans les Assets*) pour pouvoir renvoyer le résultat visuel. Il utilise un Material qui leur est assigné, dont les propriétés peuvent être intégralement modifiées. On peut notamment l'éditer en modifiant la texture qui lui est assignée, la couleur, mais également modifier directement le Shader associé.

Le Shader est une caractéristique liée au Material qui contient le code et les instructions à exécuter par la carte graphique. Il s'agit en fait d'un script qui contient des calculs mathématiques et des algorithmes qui définissent l'apparence que doit prendre chaque pixel projeté sur la surface d'un objet.

Il existe des Shaders pré fabriqués qui sont déjà implémentés dans Unity, mais il est intéressant de les écrire soi-même afin d'appliquer des effets après le traitement. En effet, le Shader standard par exemple exécute des calculs d'éclairage complexes et réalistes, ce qui est idéal pour une scène immersive, mais pas du tout adapté à l'affichage d'objets sur le bac à sable.

Les Shaders peuvent être par exemple des Shaders de surface (Surface Shader), facilitant l'écriture pour les interactions avec la lumière. Un autre modèle très utilisé est le Unlit Shader, qui en opposition n'interagit pas avec les lumières, mais est principalement utilisé pour afficher des effets spéciaux réalistes.

2.3. Analyse du projet SaHaRA V1

La compréhension du code de la version 1 a été possible une fois les bases de Unity maîtrisées. Le groupe de projet précédent avait averti dans le rapport que la reprise du projet serait compliquée à mettre en place, mais il était important de prendre note de ce qui avait été fait et avec quels outils. En effet, plusieurs barrières ont empêché la reprise du code.

Tout d'abord la barrière technique a été un obstacle. Dans la version 1, beaucoup d'objets différents et complexes en structure ont été modifiés. Un des membres du binômes explique lors d'une rencontre qu'ils se sont effectivement laissés emportés par le nombre de ressources immense qu'offre la plateforme Unity. Il a recommandé de ne pas se laisser avoir par la multitude d'objets puissants et lourds qui est disponible sur Internet, mais plutôt de partir d'objets basiques et de scripts simples.

Le second problème que nous avons rencontré avec leur code est qu'il n'est pas utilisable dans l'état actuel. En effet, le groupe précédent utilisait leur Wrapper personnel pour pouvoir le compiler et le lancer. Ils ont aussi rencontré des problèmes de compilations qui sont évoqués dans la partie destinée aux résultats. De plus, les fonctions de Kinect V1 comme la Kinect V2 se sont avérées parfois instable sous Unity.

La décision a donc été prise de ne pas réutiliser l'ancien code. Il semblait plus simple et plus cohérent d'écrire des scripts propres à cette version en choisissant des objets et des méthodes connues et maîtrisées.

3. Transposition du projet SaHaRA

Après la phase d'étude préalable, nous avons commencé à effectivement développer la nouvelle version du projet SaHaRA. Etant donné notre choix d'exploiter un plan plutôt qu'un terrain pour des raisons d'efficacité, nous avons dû reprendre le projet de zéro. Nous avons cependant essayé de garder une architecture similaire, bien que simplifiée. En effet, le projet SaHaRA initial mettait en œuvre de nombreux scripts et objets qui étaient de l'ordre de la démonstration ou du jeu puisqu'il s'agissait de leur objectif ; nous avons donc ignoré ces parties-là.

3.1. La heightmap

Le projet SaHaRA ne peut exister sans exploiter des données de profondeur, c'est pourquoi nous commençons naturellement par récupérer ces données.

Nous appelons heightmap le tableau d'entiers de dimension 512x424 stockant pour chaque pixel la valeur de sa distance au capteur. Ce tableau est récupéré assez simplement en interrogeant le capteur de profondeur de la Kinect V2 grâce à un script Unity. A chaque rafraichissement de frame, un nouveau jeu de données de profondeur est sollicité de telle sorte que 30 fois par seconde, la distance à chaque pixel est mise à jour et ce, jusqu'à l'arrêt du programme.

```
if (_Reader != null)
{
    var frame = _Reader.AcquireLatestFrame();
    if (frame != null)
    {
        frame.CopyFrameDataToArray(_Heightmap);
        frame.Dispose();
        frame = null;
    }
}
```

Figure II.3.1.1 — Code de récupération des données de profondeur

Une fois que nous disposons de cette cartographie d'altitudes, il nous est possible de réaliser différents traitements de données afin de les rendre visibles et donc compréhensibles et exploitables pour un opérateur. Pour ce faire, nous avons réalisé un script qui, pour chaque pixel du tableau, associais un niveau de gris en fonction de sa valeur (plus la zone correspondante est haute, plus le pixel sera foncé).

Nous avons ensuite assemblé les couleurs de pixels dans une même texture avant de l'envoyé dans la carte graphique pour l'affichage.

Tout ce traitement, comme les traitements que nous aborderons plus tard, se font sur une texture appliquée à un plan. Cela a pour effet de considérablement diminuer la latence du dispositif par rapport au projet SaHaRA initial. En effet, le plan étant plus léger en termes de données que l'objet terrain, la carte graphique a besoin de fournir moins d'efforts pour calculer ce qui doit être affiché sur chacun des pixels du vidéoprojecteur : ce calcul se fait normalement toujours en fonction de la hauteur, de l'inclinaison et de l'éclairage de la surface virtuelle à projeter, or pour un plan parallèle à l'écran de projection, ces trois variables sont réduites à leur plus simple expression et sont même constantes.

3.2. Le gradient

Dès lors que nous avons réalisé un premier traitement des données de profondeur, nous pouvons passer à la mise en place d'un gradient de couleur. C'est une technique rendu visuel très répandue lorsqu'il s'agit de rendre compte de variations d'altitudes.

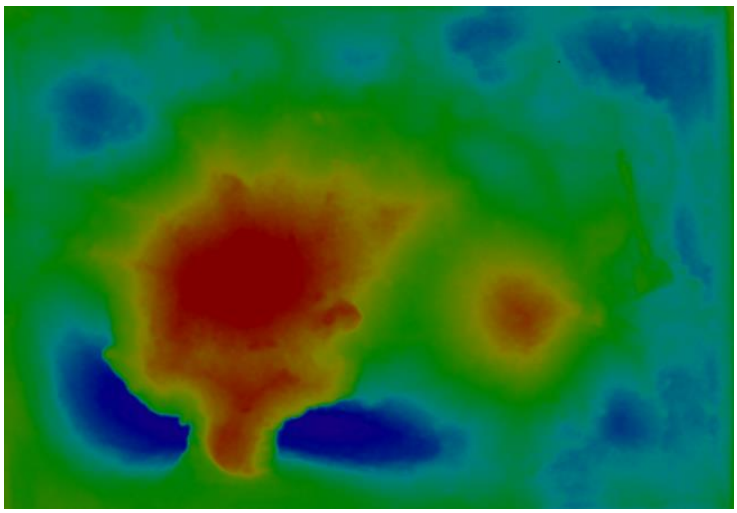


Figure II.3.2.1 — Rendu graphique du gradient de couleur

Deux méthodes ont été envisagées pour aborder cette partie : appliquer un gradient à partir d'une texture déjà existante ou bien calculer ce gradient. Dans les deux cas, il s'agit pour chaque pixel de récupérer la valeur de son altitude par rapport

au capteur avant d'y appliquer une texture. Les deux membres du binôme ont travaillé parallèlement sur ces deux méthodes mais par manque de temps, c'est celle du calcul direct qui a été retenue.

Pour réaliser effectivement le rendu sous forme de gradient de couleur, la valeur de distance du pixel était transformée en une valeur de couleur. Cette couleur était ensuite passée au format HSV qui présente l'avantage de coder sa teinte sur une unique variable contrairement au format RGB qui lui utilise trois variables. Une fois fait, il ne restait plus qu'à appliquer une variation linéaire de la teinte en fonction de la hauteur du pixel puis de sauvegarder la teinte ainsi obtenue dans la texture finale de la même façon que pour l'affichage de la heightmap.

```
// Update is called once per frame
void Update () {
    ushort[] depthmap = myDSM.GetData();
    for (int i = 0; i < 512; i++)
    {
        for(int j = 0; j < 424; j++)
        {
            float _depth = depthmap[j * 512 + i];
            float _color = _depth / 10000f;
            float _hue;
            float _sat;
            float _val;
            Color.RGBToHSV(new Color(_color, _color, _color, 1.0f), out _hue, out _sat, out _val);

            if ((_color < HeigthFromFloor) && (_color > HeigthFromFloor-SandHeight))
            {
                _sat = 1;  // maximum saturation for vivid colors
                _val = 0.5f;    // 0 = black 1 = white
                _hue = (7/SandHeight)*_color - 0.7f * (HeigthFromFloor/SandHeight-1);
                // hue is linear from lower position (blue = 0.7) to upper position (red = 0)
            }
            depthTexture.SetPixel(i, j, Color.HSVToRGB(_hue, _sat, _val));
        }
    }
    depthTexture.Apply(); // copy the texture in the GPU
    material.mainTexture = depthTexture;
}
```

Figure II.3.2.2 — Code d'application du gradient de couleur

3.3. La calibration

Maintenant que le gradient de couleur est fonctionnel, il est nécessaire de délimiter la zone dans laquelle les données de profondeur seront effectivement traitées, autant pour optimiser l'utilisation du processeur que pour ajuster l'affichage sur le sable.

Tout d'abord, le fait de permettre une calibration de la profondeur du bac à sable permet d'ajuster l'affichage du gradient par rapport aux hauteurs minimales et maximales du sable, le gradient s'affichant de façon linéaire entre une position basse et une position haute.

Par ailleurs, le capteur de profondeur de la Kinect V2 capte des « images » plus larges que le bac à sable sur lequel nous travaillons. Il existe donc une zone de quelques centimètres autours du bac que nous pouvons ignorer lors du calcul et de l'application de la texture

En revanche, le projecteur est réglé pour projeter une image exactement de la dimension du bac : il faut donc que la zone de travail délimité précédemment soit projetée en plein écran sur le vidéoprojecteur. Pour ce faire, il est nécessaire de pouvoir associer les quatre coins du bac à sable vu par la Kinect aux quatre coins de l'écran de projection du projecteur.

Afin de pouvoir déformer la zone de projection (objet plan) et l'adapter au bac à sable physique, la solution qui a été sélectionnée est la suivante. L'utilisateur clique successivement sur les 4 coins du bac qu'il voit à l'écran, et la fonction de déformation calcule un quadrilatère à partir de ces 4 points. L'image projetée est donc déformée avec le plan, c'est une des propriétés de cet objet. De plus, lorsque l'on sélectionne ces 4 points, on peut utiliser 2 points opposés pour rogner le bord de l'image pour ne pas projeter hors du bac. Ainsi, sans duplication de code, quand l'utilisateur clique sur les 4 angles du bac dans l'ordre, l'image est rognée, redimensionnée et déformée pour apparaître parfaitement sur la surface du sable réel.

Pour pouvoir déformer le plan de projection, dès que l'acquisition des 4 points a été effectuée, ils doivent être attribués aux 4 angles du plan. Pour cela, il a été choisi de créer un Mesh de façon procédurale. Il est défini par des points dans le repère en 3 dimensions, des coordonnées de texture (dites normales) et des triangles. Les points définissent bien sûr les angles du polygone, les coordonnées de texture désignent elles de quelle façon la texture va s'appliquer sur l'objet (dans quel sens par exemple, ou selon quel angle) et les triangles (définis dans le sens horaire) définissent la surface à afficher. De cette manière, il est facile de modifier chaque point dès que le dernier des 4 clics a été effectué.

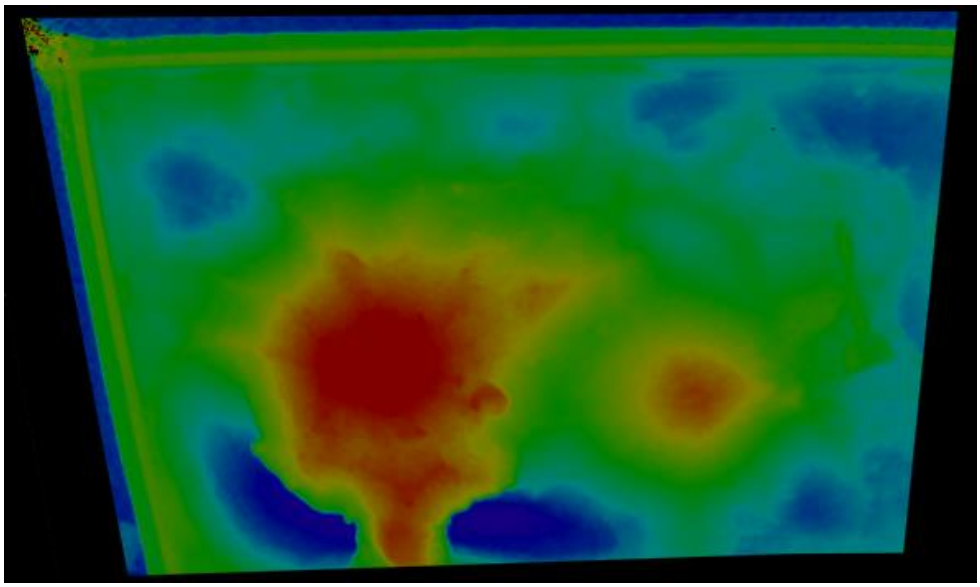


Figure II.3.3.1 — Rendu graphique après calibration et déformation

3.4. La mise en place dans la structure physique

Une fois que le rendu graphique était satisfaisant, le matériel, Ordinateur et Kinect V2, a été déplacé dans le dispositif SaHaRA.

Après avoir effectué quelques tests pour vérifier le bon fonctionnement du projet et l'alignement des différentes parties, il était nécessaire de paramétrer correctement l'affichage sur les deux écrans distincts. Pour ce faire, il a fallu ajouter une caméra dans Unity dédiée au deuxième écran et y ajouter un script pour forcer son affichage sur le vidéoprojecteur. A ce moment-là, le projet était prêt à être exporté sous forme d'application pour PC. Cependant, après compilation, l'application démarrait effectivement mais le shader restait inactif. A ce jour, ce problème n'a malheureusement pas été résolu par manque de temps.

III. Résultats et bilan

1. Fonctionnement du bac à sable

Ce projet s'axe autours de deux principaux objets dans Unity : l'objet Sandbox Setup qui contient entre autres le plan de projection, les scripts de récupération des données et d'affichage, et l'objet Caméras qui contient la Main Camera pour l'affichage sur l'écran principal et la caméra secondaire pour la projection sur le sable. Ces objets sont vides et servent simplement à organiser l'architecture de façon logique en exploitant le principe d'héritage*.

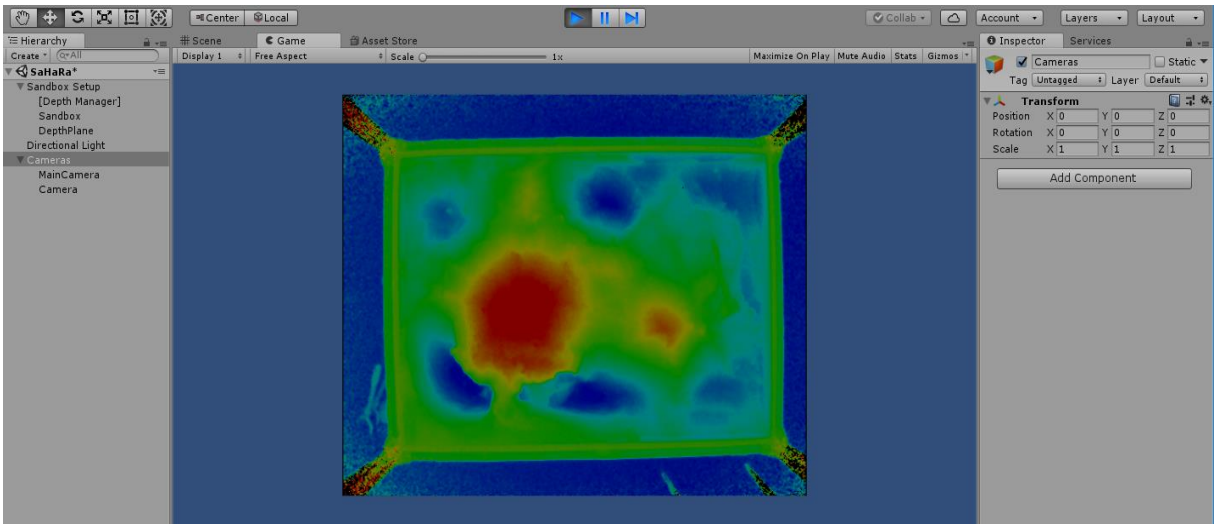


Figure III.1 — Architecture du projet SaHaRA V2

[Depth Manager] :

Il s'agit d'un objet simplement composé du script permettant d'initialiser et de récupérer les données du capteur de profondeur de la Kinect V2.

Depth Plane :

Il s'agit du plan qui sert de support de projection pour nos textures. C'est dans cet objet que se trouve le Mesh qui est utilisé pour l'application des textures.

Sandbox :

Il s'agit également d'un objet simplement composé d'un script. Ce script récupère les données des objets Depth Manager et Depth Plane pour ensuite réaliser tous les traitements nécessaires avant d'afficher la texture finale sur le plan.

2. Problèmes rencontrés

2.1. Les difficultés

L'ensemble de ce projet a représenté un vrai défi dans le cadre de la seconde année dans l'école. De la reprise du code à l'implémentation de nouvelles fonctions, de nombreux problèmes se sont posés.

Le premier obstacle rencontré a été l'utilisation de Unity. Avec le code de la précédente version, il pourrait sembler que ce soit aisé de se familiariser avec le langage. Loin de là, il a été nécessaire de se former individuellement sur les bases de l'outil et d'approfondir certaines fonctions afin de seulement pouvoir comprendre une partie du code. Découvrir Unity avec la frustration de ne pas pouvoir commencer vraiment à écrire les fonctions principales s'est avéré être un vrai défi, plus difficile que l'on aurait pu s'imaginer. Le nombre d'objet et de fonctions intégrées est tel que faire le tri dans cette source d'informations demande de se familiariser avec beaucoup de concepts à la fois. Les deux premiers mois de travail ont donc été peu motivants, dû au fait qu'il n'existait pas de preuve physique de la progression du projet et du travail fourni.

Une fois le logiciel maîtrisé, il s'est avéré quasiment impossible de reprendre le code de la version 1 du SaHaRA. Cela est dû à plusieurs raisons comme les mises à jour logicielles, les incompatibilités entre les Kinects et l'utilisation de nombreux objets différents dans la version initiale. Comme précisé dans le rapport du groupe précédent, *"(...) [L'outil utilisé] n'a jamais fonctionné sur l'ordinateur mis à disposition dans la salle d'immersion. C'est pourquoi seules les versions du projet entièrement compilées s'avèrent utilisables. La reprise du projet par d'autres étudiants est donc pour l'instant difficile, nous présentons toutefois une solution à ce problème dans les perspectives. Quoiqu'il arrive, la version de Kinect que nous avons utilisé est assez instable et supporte mal le changement de PC ou même de port USB."*

Plusieurs heures peu fructueuses pour essayer de gagner du temps en récupérant quelques fonctions ont été passées en vain. Il a donc été décidé de partir de zéro pour écrire un code clair et lisible, avec des fonctions maîtrisées, et entièrement commenté et séparé en catégories. Le scripting C# est un langage nouveau, mais les concepts objets sont maîtrisés et il n'est pas trop compliqué de s'habituer à cette nouvelle rédaction. Le fait de progresser sans le support de l'ancien projet s'est avéré motivant et efficace, et tout le code du projet a été rédigé de zéro.

Il n'a pas été facile de tout refaire avec les connaissances acquises du logiciel. Le sujet du projet étant particulièrement pointu, très peu de documentation existe sur ce type de réalisation. Contrairement à la plupart des cas en informatique, il n'était pas possible de débloquer une situation avec une recherche sur Internet ou en demandant

de l'aide à un collègue. Il a fallu utiliser les connaissances tout juste acquises pour contourner les problèmes et s'adapter aux imprévus rencontrés.

Le manque d'expérience a engendré de nombreux ratés dans les tentatives d'écriture de fonction. Pour l'écriture du code qui génère le gradient, plusieurs idées ont été implémentées, notamment celle de générer un mesh avec des voxels* (des pixels en trois dimensions). Cette méthode s'est avérée très lourde, même avec le matériel très performant fourni. Comme cela prenait beaucoup de temps à développer, cette fonction a été abandonnée. Un gradient de couleur a été généré manuellement afin de répondre parfaitement au problème posé. La solution est plus légère, plus simple à la compréhension et facile à adapter pour d'éventuelles modifications.

2.2. Défauts subsistants

Même si beaucoup de problèmes ont été résolus ou évités, le produit fini ne correspond pas exactement au résultat attendu. Le fait de ne pas pouvoir utiliser la version 1 a fait perdre un temps considérable par rapport à ce qui était prévu. Les fonctions liées au domaine de la géologie n'ont donc pas pu être implantées, malgré une tentative d'affichage de courbes de niveau. De plus, des problèmes pour générer une version compilée du projet ont été rencontrées. Il semblerait que certaines fonctions liées à la Kinect V2 fonctionnent mal avec les fichiers à disposition.

Le groupe qui a travaillé sur la version précédente a également évoqué des problèmes de compatibilités et des incohérences à la compilation. Ces aléas sont probablement dus à la version de Kinect utilisée qui est assez instable avec les Assets employés dans Unity.

3. Validation du produit final

Bien que ce projet ne réponde pas entièrement à toutes ses ambitions et malgré les difficultés rencontrées, le projet SaHaRA V2 répond en grande partie a l'objectif initial, à savoir la mise à niveau du projet initial pour être compatible avec la Kinect V2. En l'occurrence, il ne s'agit pas à proprement parler d'une mise à niveau car le projet à été recommencé de zéro. On note également l'absence de menu de paramétrage dans la nouvelle version contrairement à la précédente, qui n'a pas pu être implémenté par manque de temps. Cependant, le rendu final est assez similaire à la première version du projet SaHaRA.

Conclusion

L'objectif initial du projet SaHaRA V2 se divisait en deux axes principaux : la mise à jour logicielle et matérielle du bac à sable, puis l'adaptation de ce projet pour qu'il s'inscrive dans le cadre du projet GeolVir3D. Il était donc nécessaire d'implémenter des fonctions compatibles avec le nouveau matériel, notamment la Kinect V2. Les objectifs de ce projet ont été partiellement remplis. En effet, toute la partie de mise à jour a bien été effectuée, mais le temps a manqué lorsqu'il a fallu implémenter des fonctions propres au domaine de la géologie. Le temps passé à apprendre à utiliser le géant qu'est Unity3D explique ce décalage par rapport aux attentes initiales.

Les difficultés rencontrées en essayant de reprendre la version 1 ont également pris un temps non négligeable dans ce projet. Le mode de travail du binôme précédent était effectivement très différent, et le grand nombre d'objets peut être déconcertant pour des débutants dans l'utilisation du logiciel.

Ce projet a néanmoins été très bénéfique en termes d'apprentissage personnel. En effet, développer des compétences avec un outil aussi complet que Unity3D aura certes pris un temps considérable dans le cadre du projet, mais ce sont aussi des connaissances extrêmement utiles pour des ingénieurs en informatique. Son utilisation insolite avec la Kinect V2 est unique et quasiment aucune documentation n'existe à ce sujet, ce qui rend ces compétences précieuses et intéressantes pour la culture générale en milieu professionnel.

Malgré les difficultés rencontrées, tout le code a été rédigé de façon optimisée et commenté précisément de façon à pouvoir être repris facilement par une autre équipe. Pour pouvoir implémenter des fonctions plus poussées comme la création d'un terrain à partir d'un modèle, il faudra toujours travailler en restant très proche des membres du projet GeolVir3D afin de respecter précisément la demande des géologues. En effet, ce domaine étant loin des sujets traités dans l'école, il est indispensable de prendre le temps de comprendre ce qui est attendu et de l'adapter dans la mesure du possible. Les membres du projet avaient par exemple évoqué des structures en couches pour simuler des strates géologiques, de la gestion des fluides ou encore des courbes de niveau. Nous espérons donc que le projet tel qu'il est actuellement sera repris avec plus d'aisance par un prochain groupe afin de mettre en place ces solutions évoquées.

Références bibliographiques

Communiqué scientifique : “Comparison of Kinect v1 and v2 Depth Imagesin Terms of Accuracy and Precision” de Oliver Wasenmüller et Didier Stricker avec le German Research Center for Artificial Intelligence (DFKI)

Rapport fonctionnement Kinect de V. Le Mellay - A. Paillard - K. Marburger

Bac à Sable en Réalité Augmentée : rapport de projet d’ingénieur de Rémi Alègre et Yann Duband

Références webographiques

Manuel d’utilisation Unity 3D
https://docs.unity3d.com/Manual/index.html (consulté le 23 octobre 2018)

Chaîne YouTube de Brackeys : Destinée aux tutoriaux Unity
https://www.youtube.com/user/Brackeys (consulté le 23 octobre 2018)

University of California, Los Angeles, Augmented Reality Sandbox
https://arsandbox.ucdavis.edu/ (consulté le 9 novembre 2018)

Lexique

Assets : dossier dans lequel tous les fichiers utiles pour le programme sont utilisés. C'est l'équivalent de la boîte à outil pour Unity.

Depthmap ou Heightmap : en français "carte des profondeurs". Il s'agit entre autres d'une image en nuances de gris dont la luminosité varie en fonction de la distance de l'objet à la caméra. Plus un objet est près, plus il apparaît clair. S'il est trop près, la distance ne peut pas être calculée et il apparaît noir.

Frame : nom donné aux images consécutives qui apparaissent à une fréquence (Frame Rate) assez rapide pour apparaître comme un mouvement à l'œil humain. À 30 FPS (frames per second / images par seconde), l'image apparaît animée.

Game Object : classe de base pour toutes les entités créées dans Unity. Chaque script ou autre attribut doit être associé à un Game Object.

Héritage : système de classement des dépendances entre objets. Un objet hérite des caractéristiques de son objet père. Pour un père objet vide, seule sa position relative par rapport à celle d'un objet fil est héritée.

Material : appliqué sur un Mesh Renderer. Il permet de définir quel Shader utiliser pour afficher ce matériau et les valeurs spécifiques pour paramétrer ce Shader.

Mesh : un objet qui s'apparente à un maillage. Il contient des vecteurs et des tableaux de triangles.

Mesh Renderer : attribut d'un Game Object pour projeter un Material sur l'objet en question.

Mesh Filter : un attribut qui permet d'utiliser un Mesh stocké dans les Assets.

Réalité Augmentée (Augmented Reality) : abrégé RA ou AR

Rendu : un processus informatique qui calcule l'image 2D vue par une caméra dans un logiciel de modélisation en 3D. Il calcule par exemple les sources de lumière, la perspective, et n'affiche pas les objets cachés derrière d'autres.

Scripting C# : permet d'écrire des fonctions en langage objet afin d'obtenir un effet qui répond à un problème particulier, qui ne peut pas être traité efficacement par de la simple gestion d'objets.

Shader : appliqué sur un Material. Il permet de définir une méthode d'affichage de rendu sur l'objet sous forme de formules mathématiques et d'algorithmes.

Voxel : pixel en trois dimensions. Le mot "pixel" étant la contraction de Picture Element (élément d'image), le "voxel" signifie de la même façon Volume Element (élément de volume).

Wrapper : un programme dont la fonction principale est d'appeler une autre fonction. Comme le nom anglais le suggère, il "emballe" le produit fini.