Daéren RIGAUD
Parcours 2023
Année universitaire 2022-2023

Master Imagerie, Robotique et Ingénierie pour le vivant (IRIV)
Parcours Topo
Mémoire de stage de master 2

Restitution 4D du château de la Wasenbourg dans le cadre du projet INTERREG VI

Rapport de Projet de Fin d'Études (PFE)

Laboratoire ICube
24 Boulevard de la Victoire,
67000 Strasbourg

Encadrant/Directeur de PFE : Koehl Mathieu
(mathieu.koehl@insa-strasbourg.fr)
Correcteur de PFE : Moisan Emmanuel
(emmanuel.moisan@insa-strasbourg.fr)

Résumé du projet (en français)

Restitution 4D du château de Wasenbourg dans le cadre du projet INTERREG VI

Dans un objectif de revalorisation des châteaux rhénans (français et allemands) encadré par le projet INTERREG VI, nous avons effectué la restitution 4D du château de Wasenbourg, situé dans le Bas-Rhin. Nous avons exploité des données lasergrammétriques et photogrammétriques acquises par nos soins sur le terrain pour obtenir un nuage de points dense, qui sera utilisé pour effectuer un maillage du modèle 3D de l'état actuel (en ruines). Nous avons complété ce nuage de points avec des données Lidar HD issues de l'IGN pour construire le MNT afin d'avoir une représentation du sol autour du château. En explorant les différentes sources historiques non métriques (croquis, essais de restitution, discussions avec un archéologue, descriptions de l'histoire du château), nous avons restitué le château à l'état historique de sa construction (XIIIe siècle), à partir de primitives géométriques en utilisant le logiciel Blender. Afin d'apporter une transparence dans la restitution, nous avons défini des niveaux de certitude pour chaque objet 3D, représentés par une échelle de couleurs formant la première représentation de l'état historique. La seconde exploite la méthode de cartographie UV pour appliquer au modèle des textures réalistes. Enfin, le modèle 4D a été exporté et travaillé sur le logiciel Unity pour proposer une visite virtuelle immersive, interactive et collaborative dans le Cube VR, nouvelle technologie remplaçant les casques VR traditionnels.

Résumé du projet (en anglais)

4D rendering of Wasenbourg castle as part of INTERREG VI project

As part of the INTERREG VI project to upgrade French and German castles on the Rhine, we carried out a 4D restitution of Wasenbourg castle in the Bas-Rhin region of France. We used lasergrammetric and photogrammetric data acquired in the field to obtain a dense point cloud, which will be used to mesh the 3D model of the current (ruined) state. We supplemented this point cloud with HD Lidar data from IGN to build the DTM, in order to obtain a representation of the ground around the castle. By exploring the non-metric historical sources (sketches, restitution tests, discussions with an archaeologist, descriptions of the castle's history), we restored the castle to the historical state of its construction (13th century), from geometric primitives using Blender software. To bring transparency to the restitution, we defined levels of certainty for each 3D object, represented by a color scale forming the first representation of the historical state. The second uses UV mapping to apply realistic textures to the model. Finally, the 4D model was exported and worked on using Unity software to offer an immersive, interactive and collaborative virtual tour in the Cube VR, a new technology replacing traditional VR headsets.

Remerciements

Je tenais à remercier mon encadrant et correcteur de PFE Mathieu Koehl pour m'avoir accueilli dans le laboratoire de recherche et m'avoir fait confiance pour réaliser ce projet. Je remercie également mon autre correcteur de PFE Emmanuel Moisan, qui m'a beaucoup aidé pour mener à bien ce projet et qui a répondu à mes questions, notamment concernant le fonctionnement de certains logiciels (CloudCompare ou Blender par exemple). Je tenais également à remercier mes confrères et consœurs du laboratoire ICube, en particulier à Jade-Emmanuelle Heitz qui participe au projet INTERREG VI avec moi, pour m'avoir apporté des idées et des solutions sur lesquelles j'ai pu me baser pour répondre à des problèmes que j'ai eus au cours de ce projet, le tout dans une ambiance agréable et chaleureuse. Je tenais également à remercier Jacky Koch, archéologue de l'entreprise Archéologie Alsace, pour le temps consacré à nous partager ses connaissances sur les châteaux forts alsaciens ainsi que pour les corrections du modèle au cours du projet. Je remercie également Solène Meignen, ingénieure pédagogique, ainsi que Philippe Seitier, enseignant à l'INSA Toulouse, et Bastien Sommeria Klein, technicien de la DSIN à l'INSA Strasbourg, pour les explications concernant l'utilisation du Cube VR.

Je remercie également ma consœur Zoé Papirer pour son soutien incommensurable au cours du projet et pour ses précieux conseils pour la rédaction de ce rapport. Enfin, je remercie ma famille pour m'avoir permis de suivre la voie de la topographie à l'INSA Strasbourg.

Table des matières

Introduction : définition des objectifs et contexte historique du château

La modélisation 3D est une notion très répandue dans le milieu de la topographie et du génie civil de nos jours. De nombreux logiciels permettent de traiter des données 3D terrestres de différentes manières, avec des outils polyvalents selon la situation.

Une de ses applications que nous allons étudier dans ce Projet de Fin d'Etudes est la restitution de monuments, partiellement ou complètement en état de ruines. En effet, reconstruire et représenter virtuellement des éléments toujours présents aujourd'hui ainsi que des éléments disparus suscite un engouement dans le milieu archéologique, touristique et aussi scientifique. Nous parlons alors de modélisation 4D lorsque nous mettons en valeur plusieurs phases historiques d'une même construction.

L'objectif de ce projet est donc d'effectuer une restitution 4D du château de Wasenbourg, surplombant le village de Niederbronn-les-Bains dans le Bas-Rhin. Plus exactement, dans la mesure où nous sommes 2 étudiants à initier le projet de revalorisation des châteaux rhénans dont la durée est fixée à 3 ans, nous cherchons à définir plusieurs méthodes pour optimiser et instruire la restitution d'un château en ruines sous plusieurs phases historiques (au travers de guides, vidéos tutoriels, etc.) avec application concrète sur le château de Wasenbourg, pour permettre à d'autres acteurs (entre autres des étudiants en PFE pour les années suivantes) de découvrir ce type de projet dans de bonnes conditions à l'avenir. De plus, nous avons effectué des recherches pour mettre en valeur le modèle final au travers d'une visite interactive en réalité virtuelle (VR) dans le Cube VR, technologie récente développée par *Virtual Concept* proposant une expérience différente par rapport à d'autres méthodes conventionnelles comme le casque VR. Nous cherchons alors à répondre à plusieurs problématiques, à la fois pour le laboratoire ICUBE-TRIO et pour l'avancement du projet INTERREG VI – Châteaux Rhénans:

- Comment modéliser le château de Wasenbourg et le restituer à plusieurs étapes historiques ?
- Comment instruire les méthodes utilisées à des personnes néophytes dans le domaine de la modélisation 3D ?
- Comment mettre en valeur le livrable à travers le Cube VR ?

Après avoir défini un état de l'art présentant plusieurs sources bibliographiques dans la thématique de la modélisation et de la restitution archéologique, une description de la chaîne de traitement appliquée dans ce projet sera effectuée, en commençant par les acquisitions terrain et le traitement des données, puis la modélisation 3D de l'état existant. L'aspect 4D avec la modélisation de l'état historique sera ensuite développé, en concluant sur la mise en valeur du modèle dans le Cube VR.

Contexte historique du château de Wasenbourg

Dans la mesure où aujourd'hui le château de Wasenbourg est en ruines, nous souhaitons effectuer une modélisation 4D en restituant les éléments d'origine. C'est pour cela que comprendre le contexte historique de ce château est capital pour créer le plus fidèlement possible les différentes parties du château. Entre autres, nous recherchons en particulier des

éléments d'archives tels que des plans 2D, des croquis ou du moins des cotations pour éviter de modéliser uniquement à partir d'hypothèses, et de définir les différentes époques dans lesquelles le château sera modélisé.

Salesse (2018) propose une monographie qui retrace, entre autres, une chronologie historique du château de Wasenbourg, à partir des nombreuses références dédiées à ce sujet. Toutefois, comme l'auteur le décrit, beaucoup d'évènements historiques ne sont pas garantis et restent hypothétiques, par manque d'informations précises. Mais elles restent tout de même les théories les plus plausibles, s'appuyant généralement sur des trouvailles faites autour du site (roches en ruines appartenant à un ancien édifice démoli, objet datant d'une ère ancienne, etc.), par hasard ou à travers des fouilles organisées par un archéologue.

Selon les différentes théories recensées par Salesse (2018), nous pouvons distinguer 4 époques concernant le château de Wasenbourg :
L'époque préromaine (avant le Ier siècle après J.C) : des objets archéologiques datant du néolithique jusqu'aux celtes (haches de bronze, pointes de flèches, grattoirs en silex, etc.) ont été retrouvés dans la région, et notamment autour de Niederbronn-les-Bains, ce qui attesterait d'une présence humaine avant l'occupation romaine d'après l'Encyclopédie de l'Alsace (1986). De plus nous pouvons encore aujourd'hui retrouver les fondations des murs d'une enceinte à environ 200m en aval du château qui aurait pu servir de lieu rituel, nommé le jardin des fées par Charles Matthis (1851-1925), archéologue autodidacte et spécialiste de l'histoire des Vosges ayant beaucoup apporté à la transmission de connaissances historiques ainsi qu'à la préservation du site. Mais les informations sont trop hypothétiques pour pouvoir effectuer une restitution fiable de cette période.

L'époque romaine (I-Ve siècle après J.C) : des vestiges d'un temple romain dédié au dieu Mercure ont été trouvés, sur le site non loin du château. Salesse (2018) évoque également une stèle mentionnant la VIIIe légion romaine qui a été également retrouvée au XVIIIe siècle, attestant d'une présence militaire. Cela justifierait alors la présence du Wachtfels, le rocher de grès placé au nord-est du plateau. Il servait de poste d'observation qui surplombe la vallée, et pas d'assise pour le temple dont des vestiges sont placés en dessous (voir Figure 1) comme nous pourrions le penser aux premiers abords. En réalité, ces pierres ont été placées par Matthis pour éviter leur destruction. Mais le temple à l'origine n'était pas placé ici. Dans tous les cas, le château ne semble pas être construit à cette époque, nous ne représenterons donc pas le site dans cette période.

Figure 1. Rocher du Wachtfels avec les vestiges du temple romain.

L'époque post-romaine et médiévale (VI-XVIe siècle après J.C) : Cette période est de loin la plus complexe puisque beaucoup de sources se contredisent sur l'historique du site de Wasenbourg, et notamment la construction du château. Les différents auteurs se coordonnent pour estimer sa création au XIIIe siècle, mais la date précise n'est pas établie (peut-être 1272 ou 1275). C'est en 1592 que le château est mentionné comme étant en ruines d'après Salesse

(2018). Nous ne connaissons pas les raisons exactes, mais c'est la piste la plus probable pour dater la situation de ruines que nous pouvons observer encore aujourd'hui.

L'époque moderne (XVII-XXIe siècle après J.C) : Au cours de cette période, de nombreux archéologues, dessinateurs et historiens amateurs ou professionnels se sont intéressés au château de Wasenbourg. Plusieurs associations se sont relayées pour préserver ce patrimoine, notamment le Club Vosgien de Niederbronn-Reichshoffen (dont Matthis est l'un des membres fondateurs). Des travaux de dévégétalisation ont été effectués (en 2005 et 2017), ainsi que des projets de restauration mineures comme la réparation de la fenêtre gothique sur la façade Est avec mise en place d'escaliers en bois (qui n'existent plus aujourd'hui) pour observer en hauteur en 1909, la réparation de la cheminée en 2009 ou encore la mise en place des vestiges du temple romain sur le Wachtfels en 1912. Toutefois, nous ne représenterons pas ces légères modifications par manque de temps au cours de cette étude.

Nous pouvons déduire de cette définition historique du château 2 étapes historiques que nous allons représenter pour ce projet : l'état actuel en ruines ainsi que l'état initial (c'est-à-dire la période de sa construction, au XIIIe siècle). La figure 2 met en évidence ces 2 états avec une photo prise sur le site et un essai de restitution d'un dessinateur et historien.

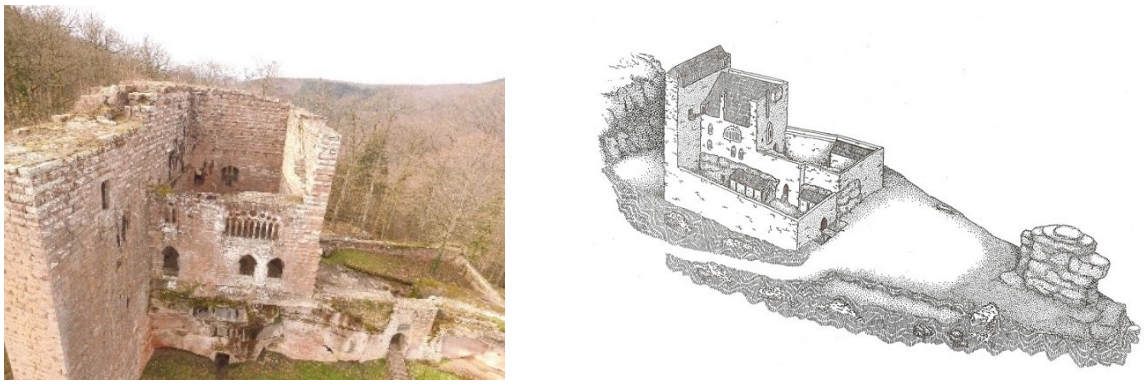

Figure 2. Photographie aérienne du château de Wasenbourg (état actuel) / Essai de restitution du château de Wasenbourg à l'époque médiévale par Mengus (2004).

1/ Etat de l'art sur la restitution 4D appliquée au château de Wasenbourg

1.3/ Communication et mise en valeur du modèle

Après avoir réussi à effectuer une modélisation 4D du modèle, en tenant compte de la visualisation des textures et de la représentation des incertitudes, le choix de la méthode de communication des données est primordial. C'est notamment le cas dans le cadre de notre projet où la restitution archéologique du château de Wasenbourg a pour but d'être exploitée pour la revalorisation touristique de ce dernier. Bien que, d'après le cahier des charges il est demandé d'effectuer une vidéo de visite virtuelle, il est intéressant de parcourir dans cette partie plusieurs méthodes pour mettre en valeur le modèle et le diffuser à un public pas nécessairement spécialiste de l'histoire du château.

a) La vidéo de visite virtuelle

De nos jours, l'une des méthodes les plus répandues pour visualiser un modèle 3D sans utiliser une représentation physique (comme une maquette en bronze par exemple) est la visite virtuelle sous forme de vidéo. Comme son nom l'indique, le principe est de présenter le modèle 3D en se déplaçant dans l'environnement 3D, comme une visite. Facile à concevoir avec des logiciels de modélisation 3D disposant de modules d'animation comme Blender (Blender@2023) ou des logiciels de capture vidéo, cette méthode a été exploitée sur plusieurs PFE précédents, notamment ceux de Rocha (2022) et Benazzi (2018) qui ont communiqué leurs résultats sous forme vidéo.

Afin de proposer une visite virtuelle agréable à regarder, il semble nécessaire d'avoir un défilement de la caméra fluide pendant l'intégralité de la vidéo. Pour cela, la plupart des logiciels de modélisation 3D utilisent des points de jonction, définis par l'utilisateur, sur lesquels vont passer la caméra virtuelle (qui est un objet 3D) qui enregistre la vidéo. Ces points forment alors un chemin de caméra. D'autres paramètres tels que la vitesse de déplacement ou l'orientation de la caméra peuvent également être définis par l'utilisateur. Rocha (2022) a proposé une planification des chemins de caméra pour sa visite virtuelle, en exploitant le croquis 2D du château de Lichtenberg. Pour avoir plusieurs séquences vidéo, 4 chemins de caméra ont été effectués (voir figure 13).

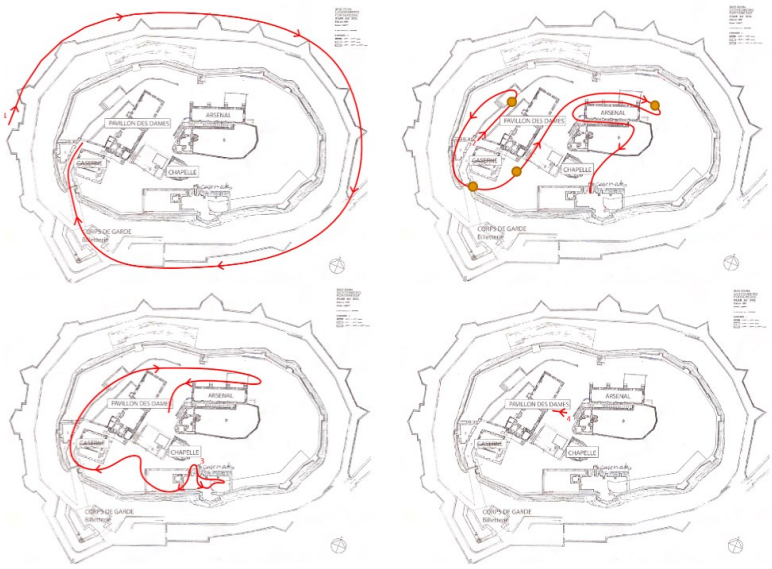

Figure 3. Planification des chemins de caméra pour la vidéo de visite virtuelle du château de Lichtenberg par Rocha (2022).

Cette méthode a pour avantage de permettre l'utilisation de logiciels de montage pour agrémenter la vidéo. Nous pouvons penser par exemple à une voix-off servant de guide pendant la visite virtuelle, ou l'utilisation de textes pour légender des éléments architecturaux du château. De plus, la vidéo permet de définir un rythme de visite idéal, tout en permettant de visualiser le modèle à différents angles. Cela peut s'avérer intéressant pour observer en détail une partie du château non accessible au public, ou pour représenter plusieurs phases historiques. Toutefois, cette méthode ne permet aucune interaction directe avec l'utilisateur.

b) Les méthodes plus interactives : représentations VR, AR et applications mobiles

Une autre approche pour communiquer et mettre en valeur un modèle 3D est l'interaction entre ce dernier et l'utilisateur. En effet, intégrer un public dans une scène virtuelle permet un accès libre à des zones interdites, visualiser de plus près une zone en hauteur comme si nous y étions vraiment, ou encore proposer des interactions ludiques voire didactiques pour le plus jeune âge. De plus en plus de sites touristiques s'y intéressent. Nous pouvons citer le château de Versailles, le Vatican ou encore le British Museum qui proposent des visites virtuelles en VR (*Virtual Reality)*, AR (*Augmented Reality*) ou sur applications mobiles interactives (BeauxArts@2023). Entre autres, Benazzi (2018) s'est intéressé à intégrer les données du site de Kagenfels pour une immersion en réalité virtuelle.

5/ Mise en valeur du modèle : représentation dans le Cube VR

Après avoir effectué toutes les démarches de la restitution 4D, la dernière étape de la chaîne de traitement (voir figure 3) consiste à mettre en valeur le modèle afin de le communiquer au public. Après discussion avec les différents acteurs du projet, il a été décidé que les châteaux français seront présentés dans une même vidéo à travers une visite virtuelle. Plus exactement, il est prévu que nous travaillions en collaboration avec un monteur vidéo en lui fournissant les données nécessaires. Nous avons vu en partie 1.3.a) que cette méthode est répandue dans le cadre de la présentation d'une restitution archéologique, et présente plusieurs avantages. Toutefois, afin d'obtenir un rendu visuel soigné, nous avons jugé que les différents châteaux modélisés doivent avoir une cohérence entre eux, que ce soit dans le choix de lumière, des textures, de vitesse et d'orientation de la caméra, etc. Par manque de temps et dans la mesure où d'autres châteaux seront modélisés dans le futur proche, nous avons décidé d'attendre que tous les châteaux soient modélisés pour harmoniser le rendu visuel (à travers un logiciel commun), plutôt que de proposer tout de suite une vidéo qui sera surement remodifiée par la suite.

Pour cette raison, nous nous sommes tournés sur une autre méthode de communication, bien plus interactive que la vidéo de visite virtuelle : la représentation du modèle 4D dans le Cube VR. Comme expliqué dans l'introduction, cette nouvelle technologie acquise en décembre 2022 par l'INSA Strasbourg propose une exploration VR différente des méthodes conventionnelles telles que le casque. Dans la mesure où aucun projet n'a encore été proposé sur le Cube dans l'établissement, nous avons pour objectif de comprendre le fonctionnement de ce dernier et de découvrir un maximum d'outils pour proposer une visite virtuelle interactive du château de Wasenbourg, en exploitant les différentes phases historiques représentées.

Dans cette ultime partie, nous allons présenter plus en détails le fonctionnement du Cube, ainsi que le logiciel *Unity* sur lequel nous allons utiliser des outils intégrés pour proposer la visualisation VR.

5.1/ Présentation du matériel utilisé

a) Le Cube VR

Pour commencer, il est important de comprendre le matériel qui sera utilisé pour cette représentation. Le Cube VR, développé par la société VirtualConcept (VirtualConcept@2023), correspond à un système de réalité virtuelle utilisant 5 projecteurs sur 5 faces blanches, formant une partie de l'intérieur d'un cube de 3x3x3 mètres (voir figure 34). Une homographie est appliquée aux différentes projections pour proposer une visualisation en temps réel cohérente entre les différentes faces, sans bordures visuelles. De plus, le système propose une visualisation 3D pour améliorer les reliefs.

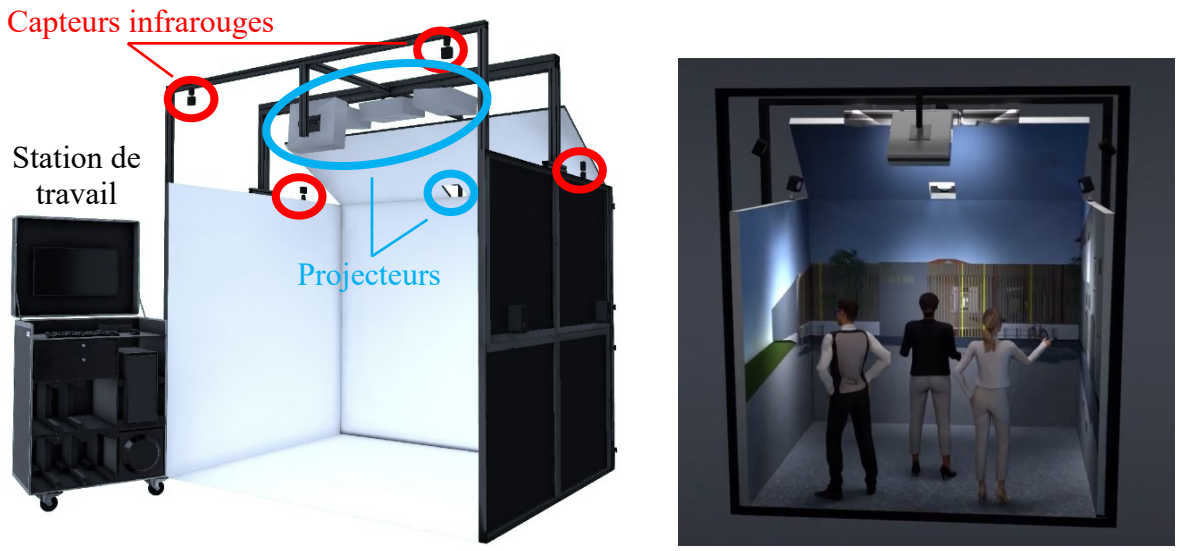

Figure 4. Description des composants du Cube VR (DEC@2023) / Exemple d'application du Cube VR (VirtualConcept@2023).

L'utilisateur est placé à l'intérieur. Comme nous pouvons le voir dans la figure 35, il dispose de lunettes 3D ainsi que des manettes jumelles connectées en Bluetooth sur un ordinateur qui calcule le rendu graphique. Bien qu'en théorie, plusieurs types de manettes sont utilisables, nous disposons d'une paire de Joycon, issue de la console de jeu Nintendo Switch, développée par la firme japonaise Nintendo. Ces 3 composants (lunettes 3D + 2 manettes) disposent de cibles sphériques (6 pour les lunettes, 3-4 pour chaque manette), permettant de capter les mouvements grâce à un système de *tracking* infrarouge 3D, conçu par Optitrack. Au total, 4 capteurs sont placés dans le Cube. De plus, jusqu'à 5 personnes peuvent être immergées dans le monde virtuel en même temps. Les autres joueurs utilisent également des lunettes 3D, mais ne disposant pas de capteurs. Par conséquent, le système de *tracking* ne s'effectue que sur le joueur principal.

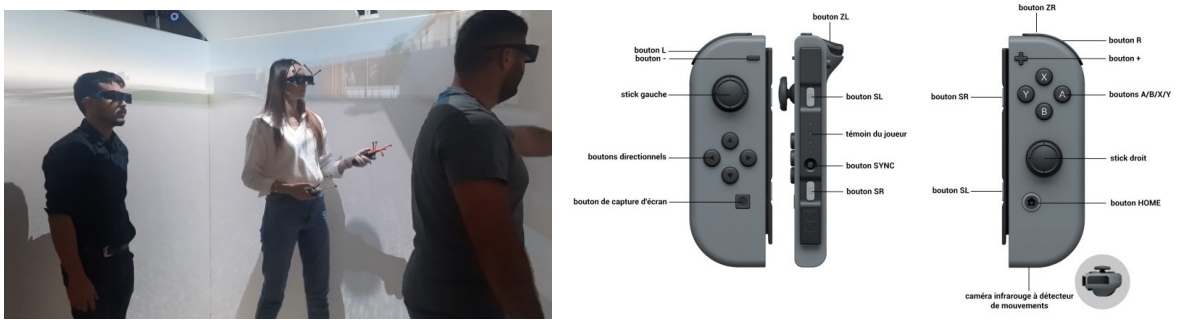

Figure 5. Autre exemple d'application du Cube VR (VirtualConcept@2023) / Description des composants des Joycon (JeuxActu@2017).

Comme expliqué précédemment, un ordinateur permet d'effectuer les calculs et la représentation VR du Cube. Plusieurs logiciels sont installés, avec des utilisations différentes :

- Le logiciel Motive, développé par Optitrack, permet la calibration et la détection des capteurs infrarouges.
- Le logiciel DEC, qui est un espace permettant de télécharger, partager des modèles VR à exploiter sur le Cube. Il est également possible de monitorer les projecteurs ainsi que des paramètres lors d'une représentation.
- Le logiciel Unity, qui sert à concevoir la représentation VR.

b) Le logiciel *Unity* avec le SDK DEC

Bien que nous ayons principalement effectué la modélisation 4D sur Blender, nous devons travailler désormais sur le logiciel gratuit Unity, développé par Unity Technologies, afin de proposer une représentation dans le Cube VR. Tout comme Blender, Unity propose beaucoup d'outils propres à un logiciel de modélisation 3D. Mais il est plus exact de le considérer comme un moteur de jeu. En effet, il est plus orienté dans la conception de jeux-vidéo, notamment grâce à sa possibilité de construire une application indépendante sur de nombreux supports (IOS/Andoid, PC, consoles de jeu, VR, etc.) ainsi que des outils d'interaction avec l'utilisateur.

Pour être plus précis, le logiciel Unity se décompose en 2 éléments complémentaires :

- Le logiciel Unity Hub, qui correspond à un gestionnaire de projets Unity. Il permet de retrouver des projets Unity déjà ouverts, de créer des projets en suivant un *template* (gabarit qui modifie l'interface, les éléments préfabriqués par défaut selon l'objectif du projet), de savoir quelle(s) version(s) du logiciel sont installée(s), ou encore d'accéder aux diverses ressources en ligne du site et de la communauté.
- Le logiciel Unity Editor, qui permet de modifier un projet Unity à travers les différents outils. Dans la mesure où cela correspond au logiciel principal sur lequel nous allons travailler, nous l'appellerons pour la suite du rapport Unity.

Une fonctionnalité d'Unity qui va nous intéresser dans ce projet est la possibilité d'intégrer des programmes, des scripts dans le projet. Par défaut, Unity utilise le langage C# en exploitant le logiciel *VisualStudio*, intégré dans le logiciel. Mais il est également possible d'intégrer des ensembles de scripts préfabriqués pour faciliter la programmation. Nous appelons cela des kits de développement logiciel, ou SDK (*Software Développement Kit*). La société DEC a alors conçu un SDK regroupant de nombreux scripts utiles pour la représentation VR, et notamment pour le Cube VR. C'est ce que nous allons utiliser ici. De plus, nous disposons d'un *template* de projet personnalisé comprenant le SDK DEC déjà importé ainsi que des éléments 3D dans la scène que nous pourrons exploiter. Bien qu'il soit possible d'intégrer ce *template* sur Unity Hub pour y accéder facilement, cela n'a pas été mis en place pour notre Cube. Nous avons alors procédé en copiant un projet vierge qui comporte tous les éléments dont nous avons besoin.

Afin d'éviter des problèmes de compatibilité, nous utilisons une version ancienne de Unity dans laquelle a été conçu le SDK, c'est-à-dire la version 2019.4.40f1.

5.2/ Premières approches de *Unity* et import du modèle

a) Description de l'interface, des *GameObjects* et des *Components*

Maintenant que nous avons découvert le matériel à notre disposition, nous pouvons nous intéresser au traitement du modèle 4D sur Unity. Pour commencer, il faut comprendre l'interface du logiciel avec ses différentes fenêtres. Il est possible de modifier la position et le contenu de ces dernières, mais nous avons souhaité conserver l'interface par défaut, qui se veut à notre sens complète et bien répartie. Comme nous pouvons le voir dans la figure 36, nous avons 5 fenêtres principales :

- Au milieu haut se trouve la scène 3D. Des outils de navigation, de sélection, de modification ou de création d'objets sont présents, assez similaires à un logiciel comme Blender (mis à part la différence des raccourcis clavier). Dans cette fenêtre se trouve un

autre onglet, *Game,* qui propose une prévisualisation du rendu final, avec possibilité d'essayer les interactions que nous avons programmées.

- En bas à gauche se trouve la fenêtre des *assets.* Cela correspond à un gestionnaire des différents fichiers et éléments à importer dans le logiciel (interne au projet). Ils ne sont cependant pas encore intégrés dans la scène du moteur de jeu. Il est possible de regrouper les différents fichiers en dossiers, comme dans un explorateur de fichiers classique. Dans le *template* à notre disposition, plusieurs dossiers vides sont déjà présents pour séparer selon le type (modèle, son, script, etc.). Mais il faut savoir que dans un projet Unity vierge aucun dossier n'est présent, il faut les concevoir soi-même si besoin. Nous y trouvons également l'onglet *Console* qui, comme tout logiciel de programmation, recense tous les messages de résultats, de commentaires ou d'erreurs des différents scripts et fonctions du logiciel quand ils sont utilisés.
- En haut à gauche correspond à la fenêtre de la hiérarchie, qui comprend tous les éléments (*GameObjects*) calculés et représentés dans la scène du moteur de jeu. Le principe est assez similaire au système de collections de Blender.
- A droite se trouve la fenêtre d'*inspector*, présentant les propriétés de l'élément sélectionné. Dans le cas d'un *GameObject*, tous les *Components* sont affichés, avec la possibilité de modifier les paramètres.

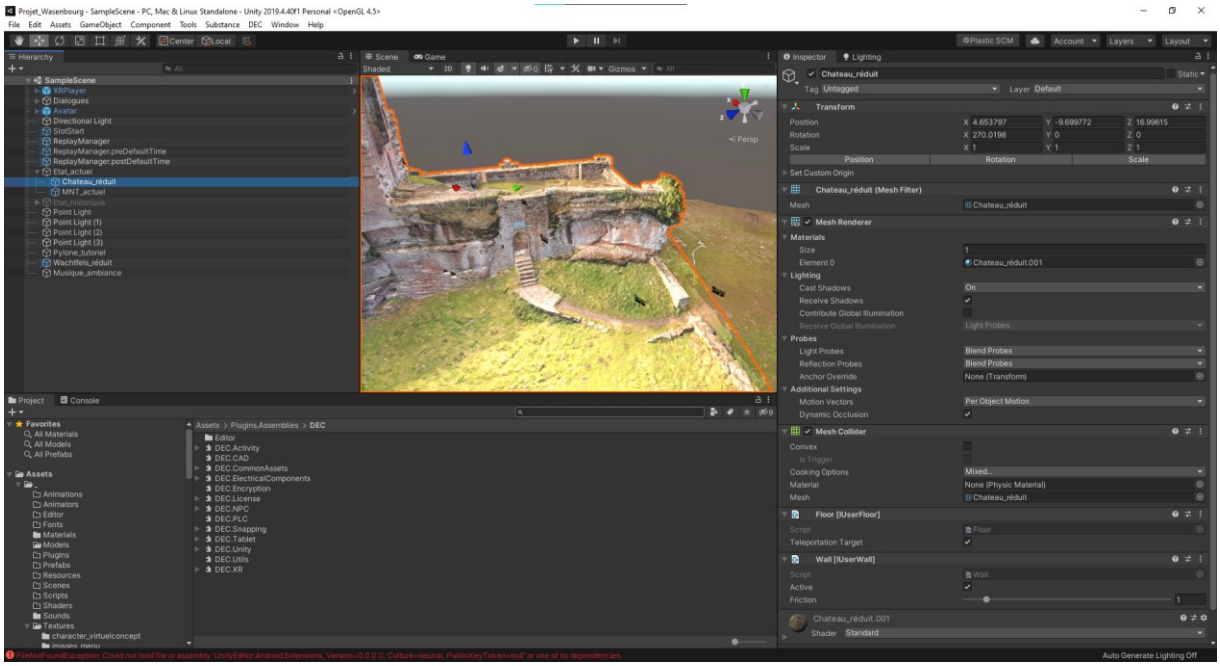

Figure 6. Présentation de l'interface par défaut d'Unity.

Il faut savoir que dans Unity, tout élément intégré dans la scène (et donc présent dans la fenêtre de la hiérarchie) est considéré comme un *GameObject* (objet de jeu). Cela peut correspondre par exemple à un objet 3D, un son audio, une image 2D, voire rien. Ce sont alors les *Components* (composants) qui déterminent toutes les spécificités de l'objet : position, type de modèle, son audio émis, effet de collision, texturage, interaction selon conditions, etc. Tous ces *Components* correspondent à des scripts dont il est possible de définir des paramètres de plusieurs types (case à cocher, valeur numérique, choix parmi un menu déroulant, nom du *GameObject*, etc.). Pour un *GameObject* vide, seul le *Component Transform* est présent, contenant les informations sur la position de l'origine, la rotation et l'échelle de l'objet. Mais il est possible de créer un *GameObject* d'un type spécifique dans le menu du même nom, situé en haut à gauche dans l'interface (voir figure 37). Dans ce cas, le *GameObject* disposera par défaut

des *Components* associés selon le type choisi. Si nous prenons l'exemple d'un Cube, les *Components* seront :

- *Transform*, commun à tout *GameObject.*
- *Mesh filter*, script permettant de définir la référence du maillage (dans notre exemple, le modèle *Cube*, présent par défaut dans tout projet Unity)
- *Mesh Renderer,* script permettant d'effectuer le rendu du maillage référencé dans le *Component Mesh filter*, avec quelques paramètres concernant les effets d'ombres et de lumière.
- *Mesh Collider*, script permettant d'appliquer un *Collider* (collisionneur) sur l'objet, qui est une surface invisible (correspondant approximativement à la surface de l'objet) qui prend en charge les collisions de l'objet avec les autres éléments de la scène. Des variantes de ce script sont utilisées pour certains modèles 3D, utilisant une primitive géométrique plutôt qu'une estimation d'un maillage (*Box Collider* pour un cube, *Capsule Collider* pour un cylindre par exemple).
- *Material*, définissant le matériau choisi pour le texturage. Par défaut, une texture grise unie est appliquée.

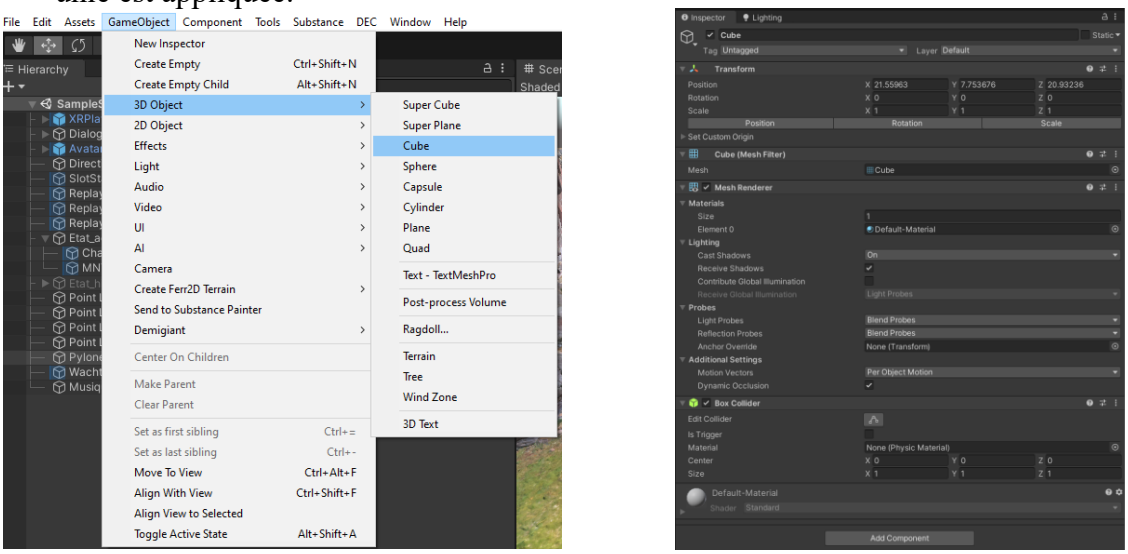

Figure 7. Exemple de création d'un cube sur Unity. / Fenêtre inspector du cube créé.

Bien qu'en théorie, nous pouvons créer n'importe quel élément à partir d'un *GameObject* vierge tant que nous ajoutons les *Components* correspondants, il est plus pratique d'utiliser ces objets préconstruits. Mais dans la mesure où nous avons déjà conçu le modèle 4D sur Blender, nous n'irons pas en détail sur tous les composants et objets associés disponibles sur Unity, nous nous limiterons à ceux qui nous seront utiles dans le rendu final.

b) Import du modèle 4D

La première étape est donc d'importer le modèle 4D sur Unity. D'après la documentation du logiciel (Unity@2022), les formats de modèles 3D pris en charge sont FBX, OBJ, DAE (Collada) et 3DS. Parmi ces propositions Blender peut exporter au format FBX, OBJ et DAE. Il est toutefois recommandé d'utiliser les formats FBX et OBJ, bien plus répandus pour la plupart des logiciels de modélisation 3D. La principale différence entre ces 2 formats est que le format OBJ utilise un 2ème fichier, au format MTL, qui contient toutes les informations liées au matériau du modèle (donc du texturage). Nous avons essayé d'importer le modèle sous ce format mais le matériau ne s'applique pas automatiquement. D'après certains forums de discussion dédiés, des solutions existent mais nécessitent un script ou ne fonctionnent que sur

une version d'Unity précise. C'est pour cela que nous nous sommes orientés sur le format FBX, n'utilisant qu'un seul fichier et présentant moins de problèmes pour importer la texture.

2 solutions sont alors possibles : importer sur Unity le fichier FBX, ou importer le fichier Blender (format blend). En effet, Unity peut lire certains formats de fichiers de projet de logiciel de modélisation 3D, dont celui-ci. Mais pour être plus exact, dans ce cas Unity effectue un pré-export en format FBX du modèle. Quelques légères différences subsistent entre ces 2 solutions (par exemple le fichier blend s'actualise automatiquement si ce dernier a été modifié), mais ce ne sera pas très important dans le cadre de notre projet.

Quelle que soit la méthode, pour exporter correctement le modèle issu de Blender sur le projet Unity, il faut extraire en parallèle les textures du projet sur Blender. En effet, par défaut sur ce logiciel les fichiers de textures sont importés depuis leur répertoire d'origine sur le disque. Il est possible de les associer au projet de sorte que les données externes soient intégrées directement dans le fichier FBX ou blend. Mais nous avons découvert que lors de l'import sur Unity, les textures ne sont pas détectées, du moins uniquement celles qui correspondent à une couleur unie. Les objets 3D qui utilisent la cartographie UV comme matériau ont une texture par défaut (gris). La solution trouvée est donc la suivante :

- (a) Sur Blender, associer tous les fichiers de textures au projet (*File > External Data > Pack Ressources)* pour ensuite les désassocier (*File > External Data > Unpack Ressources > Use files in current directory).* Dans le même répertoire que le fichier de projet, un dossier *textures* est créé, contenant toutes les données externes utilisées (dans notre cas, les différentes images correspondant aux différentes cartes UV).
- (b) Sur Unity*,* importer dans la fenêtre des *assets* le fichier de projet blend ainsi que le dossier contenant les textures en effectuant un glisser-déposer (utiliser l'export du fichier au format FBX fonctionne également).
- (c) En sélectionnant le fichier de projet, la fenêtre *inspector* présente les propriétés de l'import ainsi qu'une prévisualisation du modèle sur la scène. Dans l'onglet *Materials*, le logiciel recense tous les matériaux utilisés et propose de les redéfinir sur Unity. Il est possible de sélectionner un par un les matériaux pour chaque objet, mais ici nous allons extraire le dossier *textures* créé précédemment pour remplir automatiquement tous les champs (*Extract materials* > sélectionner dans le répertoire des *assets* lié au projet le dossier *textures* importé à l'étape précédente). Pour chaque objet, un fichier de matériau, au format MAT, est créé dans le même répertoire que le dossier de textures.
- (d) Glisser-déposer le fichier blend de la fenêtre des *assets* dans la fenêtre de la hiérarchie.

La figure 38 présente des captures d'écrans de ces étapes pour cette solution. Dans cet exemple, nous avons importé le fichier de projet *Test.blend* qui est une copie d'une des versions du modèle 4D du château. Il est important de préciser que ces données ont été placées dans le répertoire *Editor* dans les *assets*, mais il est possible de les séparer dans des répertoires plus appropriés (notamment le répertoire *Textures* pour le dossier de textures du modèle, et le répertoire *Models* pour le fichier de projet).

Nous avons remarqué à travers la prévisualisation du modèle que même sans extraire les matériaux (étape c), les textures sont bien affichées. Cela est dû au fait que Unity récupère les informations liées aux matériaux utilisés dans le projet Blender. Mais ces derniers ont pour désavantage de ne pas être modifiables (les paramètres sont grisés). Le fait de créer des nouveaux fichiers de matériaux dans l'environnement de Unity (en utilisant l'outil d'extraction

des matériaux) corrige ce problème. Cela est important car, même si en théorie toutes les informations du matériau peuvent être importées, en pratique seule une partie des informations sont récupérées. Cela se remarque notamment sur la cartographie UV : lorsqu'un matériau est sélectionné, l'*inspector* propose de définir plusieurs paramètres, et notamment plusieurs cartes : *albedo, metallic, normal, height, occlusion map*, etc. (voir détails en partie 1.1.c). Dans notre cas, bien que plusieurs cartes aient été appliquées sur Blender (*Diffuse, normal, displacement, Occlusion map*), par défaut seules l'*albedo map* et/ou la *normal map* ont été identifiées. Cela est dû au fait que Unity ne comprend pas le système de *Geometry Nodes* de Blender (voir annexe A.1), et ne réussit à déduire que les cartes UV les plus génériques. Il faut donc pour chaque matériau vérifier que les différentes textures sont bien appliquées, et rajouter celles qui sont manquantes (par le biais d'un glisser-déposer). Il faut aussi veiller à ce que les différentes cartes appliquées sur Blender puissent être adaptées à celles utilisées par Unity. Par exemple, nous avons vu dans l'état de l'art (partie 1.1.c) que, malgré de grandes similarités, la *diffuse map* utilisée sur Blender est différente de l'*albedo map* même si elles ont la même fonction. Certains paramètres peuvent être modifiés. Après plusieurs essais, nous avons conclu que l'utilisation des *albedo map, normal map* et *occlusion map* sont suffisants pour obtenir un rendu graphique équivalent à celui obtenu sur Blender, sans avoir à effectuer des modifications spécifiques (pour l'*albedo map*, Unity propose un paramétrage de la lumière cohérent avec la *Diffuse map* importée). Il faut également vérifier la résolution des textures en les sélectionnant dans les *assets*. L'*inspector* présente alors les paramètres d'import de l'image et notamment la résolution, qui est modifiable : 1k (1024x1024 pixels), 2k (2048x2048 pixels), 4k (4096x4096 pixels), etc.

En ce qui concerne la fenêtre de hiérarchie, il faut savoir que Unity ne prend pas en compte le système de collections de Blender. En effet, comme nous pouvons le voir dans la figure 38, lorsqu'un modèle est placé dans la scène (étape d), un élément parent est créé (qui correspond à un *GameObject* vierge ayant le nom du fichier importé), qui regroupe tous les éléments du projet Blender comme enfants. Pour être plus précis, ce groupe est importé en tant que *prefab* (élément préfabriqué), utilisé en principe pour des groupes de *GameObjects* configurés. Cela se distingue par une icône de cube bleu dans la hiérarchie. Afin de déplacer les éléments enfants indépendamment de l'élément parent, il faut désassembler le *prefab* (Clic droit dans la hiérarchie > *Unpack prefab*). Pour pouvoir répartir les objets dans plusieurs groupes, il faut créer un *GameObject* vide et mettre en enfant les *GameObjects* que nous souhaitons rajouter (glisser-déposer).

Tout comme pour Blender, nous nous sommes interrogés pour l'import sur Unity du modèle à la phase historique, disposant de 2 représentations (textures réalistes et échelle de couleur représentant l'incertitude) : est-il préférable d'importer un seul modèle avec 2 groupes de texture, ou plutôt 2 modèles différents avec leurs propres textures ? Malgré le fait que dédoubler le modèle augmente le poids du projet, il est plus simple sur Unity d'afficher/masquer un groupe de *GameObjects* dans la scène que de modifier un par un le matériau des différents éléments présents sur la scène. Puisque nous allons chercher par la suite à proposer une interaction permettant de permuter rapidement parmi les 3 représentations, nous allons faire le choix de prendre plusieurs modèles avec leurs propres textures.

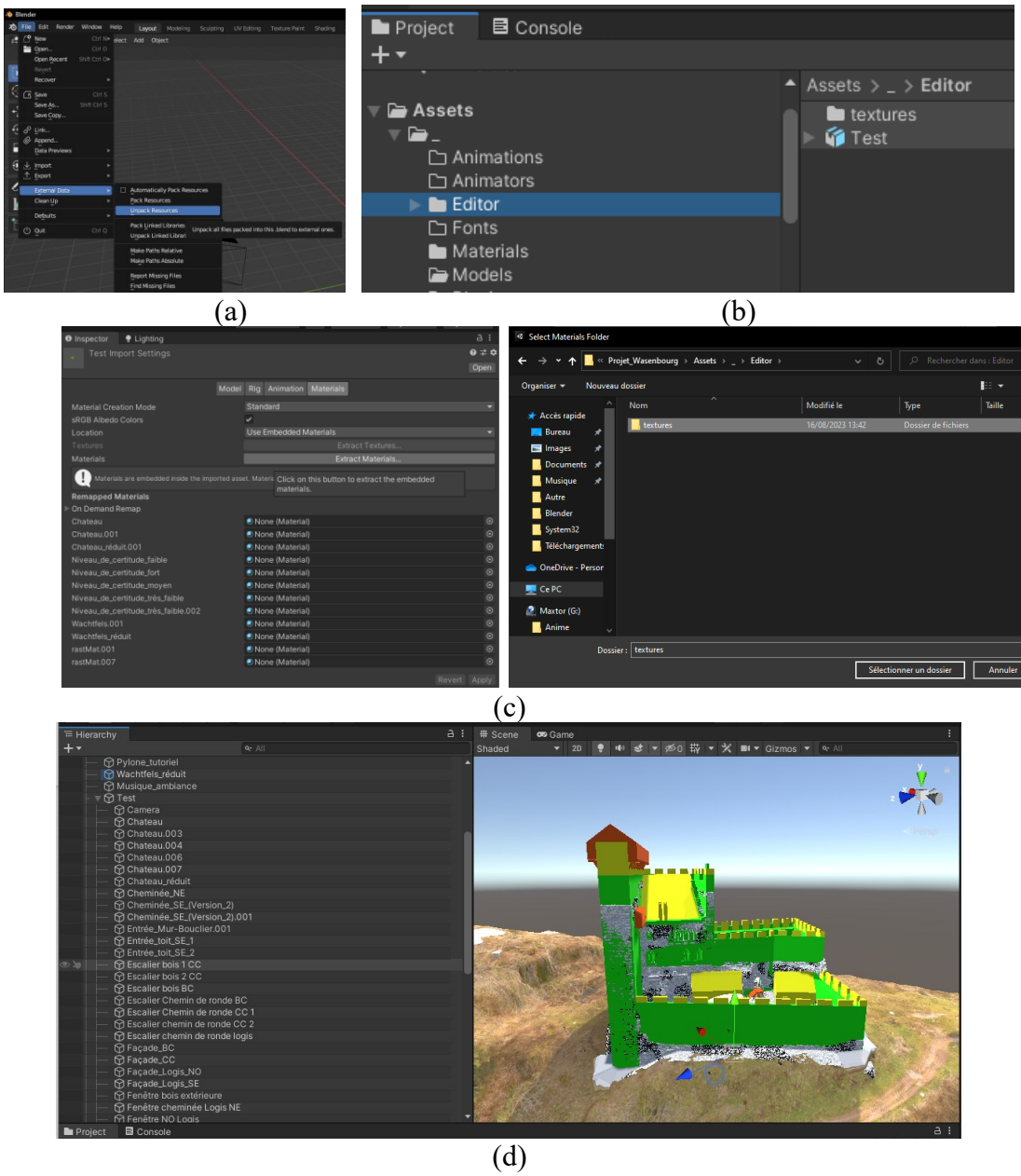

Figure 8. Captures présentant les étapes pour importer le modèle 4D de Blender vers Unity.

5.3/ Exploitation d'un menu interactif et description du mouvement dans le monde virtuel

a) Description du XRPlayer et de la représentation dans le Cube

Nous avons pu découvrir en partie 7.2 une première approche des outils et interfaces génériques du logiciel Unity. Nous allons désormais nous intéresser plus en détails aux éléments propres à la représentation dans le Cube VR. Pour commencer, comme expliqué en partie 7.1.b, nous avons utilisé un *template* contenant, en plus du SDK DEC, des éléments 3D. Il s'agit plus précisément de *prefab* déjà intégrés à la scène. Celui qui va nous intéresser est nommé *XRPlayer*. La principale difficulté ici est que, à la rédaction de ce rapport, aucune

documentation écrite n'a été encore été effectuée ni sur l'utilisation des scripts du SDK DEC, ni sur l'utilisation de ce *prefab*. Nous avons uniquement à notre disposition des tutoriels vidéo réalisés par VirtualConcept, expliquant les bases de Unity ainsi que l'utilisation de quelques scripts du SDK. Par conséquent, concernant le *prefab* nous avons progressé à l'aveugle.

À travers plusieurs essais et recherches détaillés, nous avons tout de même réussi à comprendre le principe du *prefab XRPlayer*. Ce dernier regroupe beaucoup d'éléments, il serait fastidieux de tous les décrire un par un. Nous allons décrire uniquement les différents sous-ensembles du *prefab* tout en allant plus en détail sur ceux que nous allons exploiter. Tout d'abord, le *GameObject XRPlayer* n'est pas vide : il contient plusieurs composants complémentaires, notamment le composant *Player* qui permet en un clic (sur le bouton vert représentant une icône du Cube) de construire l'application au format .exe permettant de visualiser la scène du projet en 3D dans le Cube. Il est également possible de créer l'application en utilisant les outils conventionnels de Unity (*File* > *Build Settings* > Sélectionner le format PC > Cave > *Build*), mais sur les différents essais effectués nous avons constaté plusieurs erreurs à l'exécution du programme (l'effet 3D ne se lance pas, l'application rencontre une erreur fatale), là où en utilisant le raccourci lié au composant nous ne rencontrons aucun problème spécifique. Nous procéderons désormais ainsi pour générer le programme. Ce dernier est placé automatiquement dans le dossier *Build* du répertoire du projet Unity, accompagné de fichiers complémentaires de données (voir figure 39).

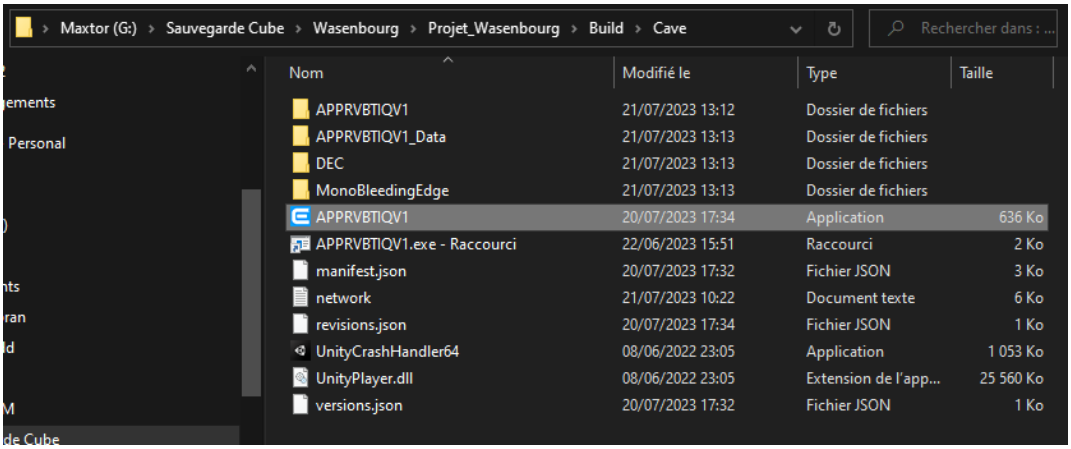

Figure 9. Contenu du dossier après avoir construit le programme (application à exécuter en surbrillance).

Afin de mieux comprendre les différents éléments du *prefab*, il faut savoir que lors de l'utilisation du Cube VR, les différentes faces projetées représentent une vue immersive 3D à la 1ère personne. Mais la station de travail représente en même temps une vue alternative de la scène, sans effets 3D. Nous pouvons y voir la vue à la 1ère ou la 3ème personne d'un avatar virtuel animé, représentant un homme en tenue d'ouvrier, personnifiant le joueur principal du Cube en copiant ses mouvements. La position et l'orientation de la tête de l'avatar correspondent alors au champ de vision de l'utilisateur sur les faces du Cube. De plus, dans cette vue une interface permet de quitter l'application, modifier des paramètres, prendre des captures photo et vidéo, etc.

Le *prefab XRPlayer* comporte 2 *GameObjects* enfants, eux-mêmes parents d'autres groupes d'éléments :

- Le *GameObject Root*, contenant tous les éléments liés au joueur de manière générale
- Le *GameObject UIRoot*, contenant tous les éléments liés à l'interface de la vue alternative.

Nous ne serons pas amenés à travailler sur l'*UIRoot*, nous allons donc nous attarder sur le *Root*, contenant tout élément pouvant contribuer à l'immersion VR de l'utilisateur. Entre autres, nous pouvons citer le groupe *CaveUser* qui contient dans les groupes *Right* et *Left* (symbolisant les mains gauches et droites) du *GameObject* parent *MultiFace* les modèles 3D des manettes affichés dans la scène (qui représentent en réalité des manettes PS Move, conçues par Sony). Ces derniers sont accompagnés d'un faisceau lumineux indiquant où pointe la manette, ce qui peut être pratique pour viser un élément (un bouton par exemple). Des scripts sont également appliqués pour permettre la détection de mouvements et de déplacer les manettes dans la scène virtuelle en corrélation avec le déplacement des manettes de l'utilisateur. Dans ce même groupe se trouvent les objets servant à la détection du mouvement de la tête (dans le groupe *Head*) et plus globalement du corps (dans le groupe *Body*), permettant de définir en temps réel le déplacement de la caméra dans le Cube et du modèle 3D de l'ouvrier, contenu dans un autre *prefab* présent par défaut dans le *template* nommé *Avatar* comme nous pouvons le voir dans la figure 40.

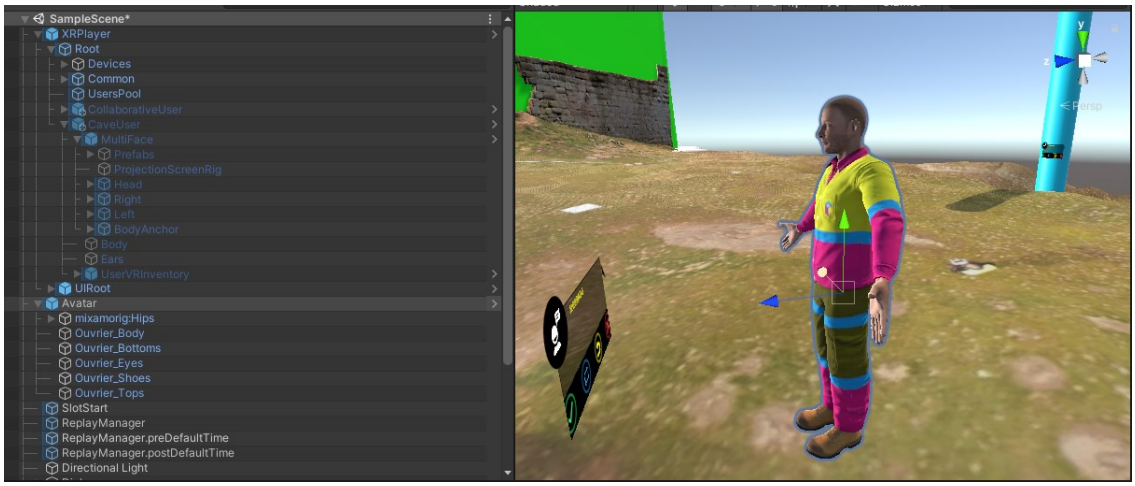

Figure 10. Détail des prefab par défaut dans le template (à gauche) avec le modèle 3D de l'avatar dans la scène (à droite).

b) Navigation dans le monde virtuel

Lorsque le programme s'exécute, les différentes faces du Cube affichent le point de vue de l'avatar, qui est situé à l'origine du repère cartésien (car c'est aux coordonnées (0,0,0) que les *prefab* associés sont placés par défaut, nous n'avons pas modifié dans le projet Unity la position de ces éléments). L'orientation de la tête est prise en compte (grâce aux cibles placées sur les lunettes, voir partie 7.1.a), ce qui modifie l'orientation du champ de vision. De plus, lorsque l'utilisateur se déplace physiquement dans le Cube, les projections 3D sont modifiées de sorte que la vision de la caméra virtuelle soit cohérente avec la vision du joueur. Par exemple, si nous nous approchons d'une face en particulier, un effet de distorsion est appliqué sur les autres faces pour rajouter de la profondeur. Cela donne l'impression que nous nous sommes éloignés des autres faces comme si nous nous sommes légèrement déplacés dans le monde virtuel. Pour pouvoir se déplacer, il faut utiliser les 2 manettes à notre disposition (voir figure 35). Dans les paramètres présents dans l'interface de la vue alternative, il est possible d'alterner entre 3 modes de déplacement :

- **Par translation** : mode de déplacement immersif où l'avatar se déplace en effectuant une translation, en prenant compte de la gravité (sol) et des collisions avec les objets de la scène (mur). Pour effectuer une translation, il faut déplacer le *joystick* analogique de la manette en avant ou en arrière. L'avatar se déplacera dans la direction que vise la

manette (par exemple, si la manette vise à gauche par rapport à l'orientation de l'avatar, ce dernier se déplacera vers la gauche). Orienter les *joysticks* à gauche ou à droite permet d'effectuer une rotation dans la scène, sans déplacer l'avatar. La vitesse de translation et de rotation peut être modifiée dans les paramètres. Les commandes attribuées aux Joycon étant identiques, orienter les *joysticks* des 2 manettes dans la même direction simultanément permet de doubler la vitesse. Ce mode de déplacement propose une navigation dans le monde virtuel réaliste.

- **En volant** : mode de déplacement similaire à la translation mais en ne prenant pas compte de la gravité et des collisions. L'orientation verticale des manettes permet de prendre de la hauteur. Cela est intéressant pour voir plus en détail des éléments à forte altitude ou pour traverser facilement des objets 3D, permettant de passer d'une zone à une autre plus rapidement.
- **Par téléportation** : mode de déplacement plus rigide où le déplacement s'effectue par téléportation de l'avatar d'une position à une autre. Il faut utiliser les *joysticks* pour faire apparaître une zone au sol devant la manette, qui indique la position dans laquelle l'avatar va se téléporter lorsque le *joystick* sera relâché. Cela est intéressant pour éviter un déplacement de la scène constant, ce qui permet d'atténuer l'effet de *Motion sickness* (mal des transports) qui rend l'expérience VR désagréable pour les personnes sensibles.

Il est important de préciser que, pour que les modes de déplacement par translation et par téléportation fonctionnent correctement, il faut indiquer au programme les modèles 3D qui doivent être considérés comme du sol. Cela est possible en utilisant des scripts du SDK DEC.

c) Exploitation des scripts *Floor* et *Wall* sur les modèles 3D

Afin de proposer une navigation immersive, il faut appliquer à la scène les notions de collisions et de gravité à travers les modèles 3D de la scène. Pour cela, nous pouvons utiliser les scripts *Floor* et *Wall* du SDK DEC.

L'ensemble des scripts sont au format DLL (*Dynamic Link Library*, Bibliothèque de liens dynamiques), qui contiennent un ensemble de fonctions et de classes compilées et utilisées par d'autres programmes. Tous ces scripts sont regroupés dans un *Assembly*, qui correspond au fichier *DEC.XR.dll*, placé dans le répertoire *Assets>Plugin_Assemblies>DEC*. Ce fichier est également au format DLL. Puisque les lignes de code contenues dans ces fichiers sont déjà compilées, il est difficile de les lire avec des logiciels de programmation. Plusieurs solutions existent, mais nous avons choisi d'utiliser le logiciel JetBrains dotPeek, qui permet de décompiler un *Assembly* en code au format C# (JetBrains@2023), en isolant les différents scripts associés. Grâce à cela nous pouvons analyser le code des différents composants que propose le SDK. Par manque de temps, nous n'avons pas réussi à définir avec certitude le fonctionnement de l'ensemble des scripts, mais nous avons pu mieux comprendre le comportement des scripts principaux que nous allons utiliser dans ce projet.

Le script ***Floor*** permet, lorsqu'il est ajouté en tant que composant d'un modèle 3D (*Add Component*>Utiliser la barre de recherche pour trouver le script), de donner à ce dernier le comportement d'un sol. Pour fonctionner, le modèle 3D doit obligatoirement disposer d'un composant créant un *Collider* (*Mesh Collider, Box collider,* etc. Voir partie 7.2.a). Si ce n'est pas le cas, le composant *Floor* affiche un message d'alerte à ce sujet et propose de créer en un clic le composant correspondant. Entre autres, ce script définit la fonction *AllowTeleportation*. Ce dernier vérifie si, pour les polygones du *Collider* situé dans la zone sur laquelle l'avatar va

se déplacer ou se téléporter, l'angle formé entre la normale du polygone et l'axe vertical (axe Z dans le repère cartésien) est inférieur à une variable nommée *maximumSlopeAngle*, représentant l'angle de pente maximale. Si la condition est réussie, alors le polygone est considéré comme un sol, l'utilisateur peut s'y déplacer (lorsque le mode de déplacement est défini par translation, le modèle 3D fait opposition à la force de gravité appliquée sur l'avatar) ou se téléporter. Pour faire simple, le script applique l'effet de sol uniquement sur les parties qui ne sont pas trop pentues, en fonction d'un seuil d'angle maximal. Par défaut, la variable *maximumSlopeAngle* est fixée à 45°. Mais il est possible de définir cette valeur en tant que paramètre dans le composant en utilisant la variante du script, nommée *Floor (custom slope)*, ayant la même fonction mais en laissant au développeur le choix de définir pour la variable *maximumSlopeAngle* une valeur comprise entre 0 et 90°. La figure 41 présente les composants liés à ces 2 scripts dans l'*inspector.* Dans les 2 cas, le paramètre *Teleportation Target*, définissant si le modèle 3D peut être ciblé pour une téléportation ou non, est également présent.

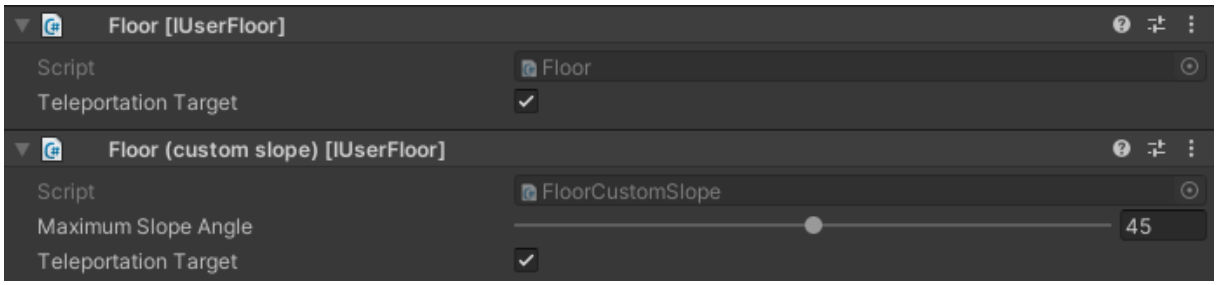

Figure 11. Définition des composants Floor et Floor (custom slope) dans l'inspector.

Il est important de préciser que, lorsque nous sommes dans le mode de déplacement par translation, si nous naviguons entre 2 zones de sol ayant une hauteur différente (par exemple lorsque nous montons un escalier), jusqu'à un certain seuil de différence de hauteur l'avatar va monter sur la zone la plus haute. Nous n'avons pas réussi à trouver la valeur exacte de ce seuil (aucune variable ou fonction n'y fait allusion dans le script *Floor*), mais après plusieurs essais nous pouvons l'estimer à environ 60-80cm. Cette valeur est cohérente avec la hauteur maximale moyenne qu'un être humain adulte puisse grimper en utilisant uniquement ses jambes.

Le script ***Wall*** propose d'appliquer à un modèle 3D un comportement de mur, c'est-à-dire de bloquer l'avatar lorsqu'il rentre en collision avec celui-ci. Tout comme pour le script *Floor,* pour que le composant fonctionne un *collider* doit également être appliqué. Pour être plus précis, si la translation effectuée par l'avatar n'est pas perpendiculaire à la surface du *collider* dans la zone de contact, le mouvement n'est pas forcément bloqué. Il est modifié en fonction de la variable *Friction*, définie en tant que paramètre du composant, sur une plage de 0 à 10. Lorsque la valeur est fixée à 0, l'avatar glisse le long de la paroi du mur selon la direction du mouvement (si l'avatar avance sur un mur en se tournant sur sa droite, il va se déplacer le long du mur vers sa droite), il n'y a donc pas de friction. Mais plus la valeur augmente, plus la différence d'angle entre la normale du *collider* et du vecteur de translation de l'avatar devra être élevée pour pouvoir glisser le long du mur, sinon le mouvement sera complètement bloqué. Après plusieurs essais, nous avons estimé que la valeur par défaut (1) représente un bon équilibre d'effet de friction. Enfin, nous précisons que le composant dispose du paramètre *Active* permettant au développeur d'activer ou de désactiver l'effet de mur, sans avoir à supprimer le composant. La figure 42 présente le composant dans l'*inspector*.

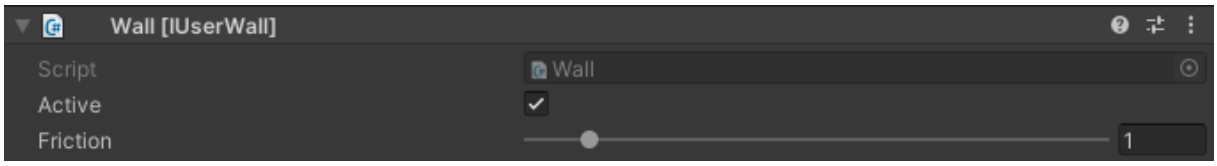

Figure 12. Définition du composant Wall dans l'inspector.

Nous pouvons donc utiliser ces scripts sur les différents modèles 3D. Toutefois, nous avons constaté que dans plusieurs cas l'utilisation d'un seul script peut ne pas être suffisante. Prenons l'exemple d'un mur avec chemin de ronde. Nous souhaitons à la fois que le mur bloque le mouvement et aussi avoir la possibilité de marcher sur la partie supérieure comme le faisaient les gardes à l'époque médiévale pour surveiller le château. Utiliser uniquement le script *Floor* permet bien de pouvoir se déplacer sur le dessus du mur, mais sur la façade l'avatar n'est pas bloqué, il traverse le mur. Cela est dû au fait que l'angle de pente de la façade (qui vaut environ 90°) est supérieur à l'angle maximal de pente (45°), cette partie ne peut pas être considérée comme un sol et ne possède donc pas les propriétés associées. De même, si nous appliquons uniquement le script *Wall*, la façade du mur va bloquer correctement le joueur, mais ce dernier ne pourra pas marcher sur la partie supérieure, il va traverser la surface et tomber à l'intérieur du mur. Cela s'explique par le fait que lorsque l'avatar se situe sur le dessus d'un objet 3D, il n'est pas considéré comme étant en collision avec ce dernier. Par conséquent, la surface supérieure de l'objet n'a pas le comportement d'un mur. Ces défauts nuisant à l'immersion dans le monde virtuel, nous avons trouvé comme solution d'appliquer les 2 scripts sur le même *GameObject*. Dans ce cas, les effets se complémentent, proposant un comportement à la fois sur la partie supérieure et sur les parties latérales de l'objet. En utilisant les paramètres par défaut, nous n'avons pas rencontré au cours de nos essais des conflits dans le comportement des scripts (nous n'avons pas trouvé des situations où le mouvement est bloqué sur une surface prévue pour se comporter comme un sol, ou de pouvoir marcher et grimper sur une surface prévue pour se comporter comme un mur).

Nous avons décidé d'appliquer cette solution pour l'ensemble des objets 3D de la scène. En effet, excepté certains objets 3D comme le MNT où le script *Wall* n'apporte pas de comportement supplémentaire dans le moteur de jeu (puisque par définition le MNT représente le sol), nous recommandons d'appliquer les 2 scripts sur tous les objets pour éviter tout problème de collision ou de gravité lors de la navigation dans le monde virtuel, sans avoir besoin au préalable d'analyser chaque objet 3D pour déterminer si nous souhaitons le considérer comme un sol ou un mur.

d) Utilisation du *Canvas UserVRInventory* pour l'affichage des différentes représentations du château

Nous avons décrit en partie 7.3.c une méthode pour proposer une navigation dans le monde virtuel réaliste, qui prend en compte les autres objets 3D. Nous cherchons désormais à répondre à la problématique de la visualisation des différents modèles du château. En effet, nous disposons dans la scène de 3 groupes de *GameObjects* différents, chacun correspondant à l'une des représentations du château (état actuel, état historique avec textures réalistes, état historique avec échelle de couleurs définissant le niveau de certitude). Placés dans la même zone dans le repère, il n'est pas possible de les afficher en même temps. L'objectif est donc de proposer au joueur la possibilité d'alterner entre ces différentes représentations, sans avoir à modifier le projet.

Dans les tutoriels vidéo à notre disposition, il est proposé d'intégrer une tablette DEC, qui est un *prefab* représentant une tablette tactile qu'il est possible d'attraper dans la scène VR via les manettes. Des outils sont intégrés à la tablette, comme un appareil photo ou une lampe torche. Nous pourrions alors utiliser cette tablette pour y intégrer des outils personnalisés, notamment pour choisir la représentation du château à effectuer. Toutefois, la tablette étant un objet virtuel physique, il peut s'avérer encombrant à transporter lors de la visite virtuelle. C'est pour cela que nous avons décidé d'utiliser à la place l'interface d'inventaire intégré par défaut dans le *template*, correspondant au *prefab UserVRInventory* intégré dans le *XRPlayer* (voir figure 40). En effet, en appuyant sur le bouton directionnel haut ou sur le bouton X, un menu apparaît au-dessus du modèle 3D de la manette qui l'a activé, et suivant son mouvement. Compte-tenu de l'interface, nous pouvons supposer que ce menu sert à faire apparaître des outils ou des équipements d'ouvrier, dans le cadre de simulations de chantier ou d'assemblage de pièces mécaniques. Le menu dispose de 2 onglets, l'un représentant les outils et l'autre les équipements. 10 boutons sont disponibles pour chaque onglet, le joueur peut les activer en appuyant sur la gâchette (ZL ou ZR) de l'autre manette en visant avec le faisceau laser. Par défaut le menu est vide, aucun objet n'est présent. Par rapport à la tablette DEC, ce menu a pour avantage de rester fixé à la manette sans être un objet 3D physique qui doit être transporté (le menu peut être désaffiché en rappuyant sur le même bouton qui a permis de l'afficher). Nous allons alors concevoir nos propres boutons qui afficheront la représentation choisie.

Le *prefab UserVRInventory* est en réalité un *canvas* (canevas) propre à Unity, qui permet de représenter des interfaces utilisateurs, ou *User Interface* (UI). D'après la documentation de Unity (Unity@2023), le canevas est un *GameObject* disposant du composant *Canvas*. La figure 43 présente les différents éléments du *prefab.* Il serait fastidieux de tous les présenter, nous allons uniquement nous intéresser au *GameObject InventoryContent* qui regroupe tout le contenu associé à l'onglet d'inventaire. Nous pouvons y retrouver les éléments liés aux 10 boutons (*UserVRInventorySlot01, UserVRInventorySlot02,* etc.), regroupés par rangée (*Row01* et *Row02*). Pour tous les *UserVRInventorySlot*, le composant *Button* est intégré (voir figure 43). Il est possible de définir l'image représentée par le bouton (*Target Graphic*), ainsi que les couleurs appliquées à l'image en fonction de la situation du bouton (1 couleur différente lorsque le bouton est visé par le pointeur d'une manette, lorsqu'il est pressé, lorsqu'il est sélectionné). Nous avons modifié les couleurs par défaut pour qu'elles soient plus distinguées entre elles.

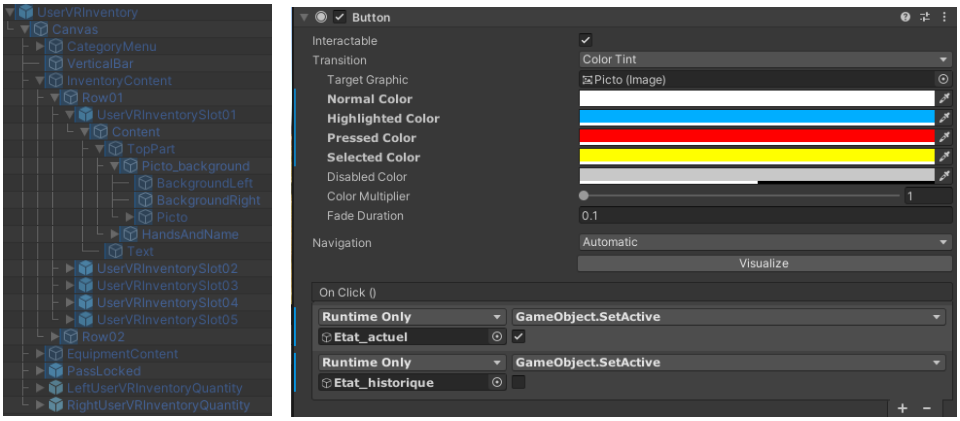

Figure 13. Description du prefab UserVRInventory dans la hiérarchie / Description du composant Button dans l'inspector.

Ce qui est le plus intéressant dans ce composant est la fonction *On Click,* qui permet d'appliquer des fonctions lorsque l'utilisateur appuie sur le bouton. C'est ainsi que nous allons pouvoir afficher la représentation du château souhaitée, en utilisant la fonction *GameObject.SetActive*. Cette dernière propose en paramètre d'activer ou de désactiver le *GameObject* en entrée dans

la scène. La solution que nous proposons utilise simultanément 3 fonctions *SetActive* pour un même bouton. Lorsque l'utilisateur appuie sur le bouton, l'une des fonctions active le *GameObject* parent contenant tous les objets 3D propres à la représentation souhaitée (en effet, l'action d'activer/désactiver un *GameObject* s'applique aussi à tous les éléments enfants), les 2 autres fonctions désactivent les 2 autres représentations. Nous modifions alors ainsi les 3 premiers boutons de la 1re rangée, 1 pour chaque représentation. Pour chaque *UserVRInventorySlot*, l'image peut être modifiée dans le *GameObject* enfant *Picto*, à travers le composant *Image* (paramètre *Source Image*). Nous avons donc intégré dans les *Assets* des pictogrammes en noir et blanc au format JPEG téléchargés sur le site Shutterstock (Shutterstock@2023), représentant un château en ruines et un château en bon état (voir figure 44), représentant respectivement l'état actuel et l'état historique (avec textures réalistes). En ce qui concerne l'état historique avec représentation du niveau de certitude, nous avons utilisé le même pictogramme mais en vert et blanc.

Figure 14. Pictogrammes utilisés dans le menu interactif dans la représentation VR.

Nous avons également souhaité ajouter du texte sous les boutons pour indiquer la représentation affichée. Les *UserVRInventorySlot* disposent d'un *GameObject* enfant nommé *Text,* prévu à cet effet. Nous avons alors rentré pour chaque bouton le texte approprié, mais dans la scène VR ces textes ne s'affichent pas dans les menus. Notre hypothèse est qu'il est prévu dans le canevas que ce texte s'affiche que s'il considère qu'un objet d'inventaire est présent à l'emplacement du bouton. Si ce n'est pas le cas, le bouton est fonctionnel mais est considéré comme vide, les éléments secondaires tels que le texte décrivant le nom de l'objet ne s'affichent pas. Ici, nous avons donc modifié le comportement par défaut des boutons, mais n'avons pas rajouté d'objets dans l'inventaire. Par manque de temps nous n'avons pas réussi à résoudre ce problème, mais ce serait une perspective intéressante à développer à l'avenir.

5.4/ Implémentation d'outils pour l'amélioration de l'expérience VR

Dans les parties 7.2 et 7.3, nous avons proposé des solutions pour effectuer une représentation 4D du château dans le Cube VR. Dans cette section, nous allons nous intéresser à des outils plus spécifiques visant à améliorer l'expérience du joueur sur l'aspect interactif, visuel et instructif de l'exploration virtuelle.

a) Création de boîtes de dialogue interactives

Pour compléter cette visite VR du château, nous avons jugé intéressant d'apporter des textes descriptifs, décrivant l'histoire du château dans les différentes parties comme lors d'une visite d'un monument, ainsi que de proposer un tutoriel pour expliquer les différentes commandes et interactions possibles. Nous pouvons pour cela intégrer dans la scène Unity le *prefab UIDialog*, disponible dans les *Assets* par défaut dans le *template* (taper dans la barre de recherche *UIDialog*). Ce *prefab* correspond à une boîte de dialogue flottante apparaissant dans la scène lorsque le joueur s'en approche, défilant un texte. Cela s'effectue à travers le script *Dialog Controller* (propre au SDK DEC) permettant de fournir tous les paramètres nécessaires. La figure 45 présente le script dans l'*inspector*, où les paramètres sont divisés en 4 parties : *Base settings, Dialog Settings, UI Component settings et Position/Look at Settings.* Des paramètres tels que la vitesse de défilement du texte (*Text Speed*), la portée d'activation de la boîte de dialogue (*Activation Range*, où l'élément apparait et disparait lorsque l'avatar entre ou sort de la zone d'activation) ou la possibilité que la boîte de dialogue s'oriente en direction de l'avatar (*Look at User*) sont présents.

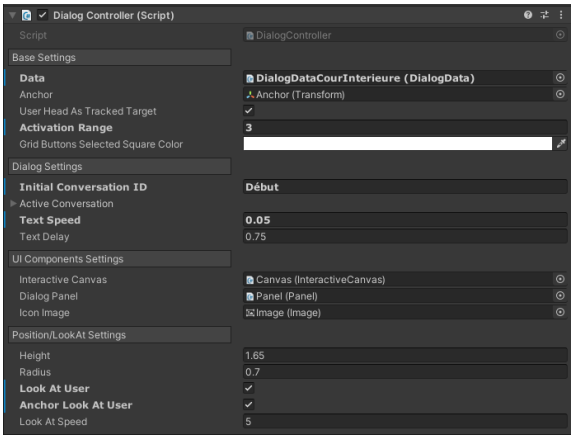

Figure 15. Description du script Dialog Controller intégré dans l'UIDialog.

Le texte à afficher n'est pas contenu dans l'*UIDialog*. En effet, il faut créer un *GameObject* vide contenant le composant *DialogData*. Ce dernier permet de créer plusieurs conversations, chacun disposant d'un identifiant (ID). Pour chaque conversation il est possible de définir le contenu textuel (*Dialog Text*), ainsi que des boutons (*Footers Buttons*). 4 types de boutons sont disponibles avec leur propre logo (boutons valider, annuler, répéter et retour en arrière). Le développeur peut décider lesquels seront utilisés pour la boîte de dialogue, ainsi que la fonction exécutée lorsque l'utilisateur appuie sur le bouton (à travers la fonction *On click* que nous avons déjà évoqué en partie 7.3.d). Dans notre projet, nous utilisons uniquement le bouton de type valider (*Validate*), permettant lorsqu'il est pressé :

- d'afficher la conversation suivante dans la boîte de dialogue, en utilisant la fonction *DialogController.DisplayConversation* avec en entrée l'*UIDialog* et l'ID de la conversation à afficher.
- de masquer la boîte de dialogue si la conversation actuelle est la dernière, en utilisant la fonction *DialogController.UI_HideUIDialog*.

Afin d'associer l'*UIDialog* et le *DialogData*, il faut indiquer dans le paramètre *Data* du composant *Dialog Controller* (voir figure 45) le *GameObject* disposant du composant *DialogData*, ainsi que l'ID de la conversation de départ (paramètre *Initial Conversation ID*). La figure 46 présente l'exemple de la paramétrisation du *DialogData* pour la boîte de dialogue décrivant la cour intérieure du château, avec sa représentation dans la scène VR. Nous retrouvons en-dessous de la boite flottante le bouton valider, qui a pour effet lorsqu'il est pressé de masquer l'élément de la scène. Dans cet exemple, une seule conversation est utilisée, dans la mesure où le texte est court. Mais dans d'autres cas, utiliser plusieurs conversations permet de limiter la taille des boîtes de dialogues (qui s'adaptent selon la longueur du texte), pour éviter que le contenu dépasse du champ de vision.

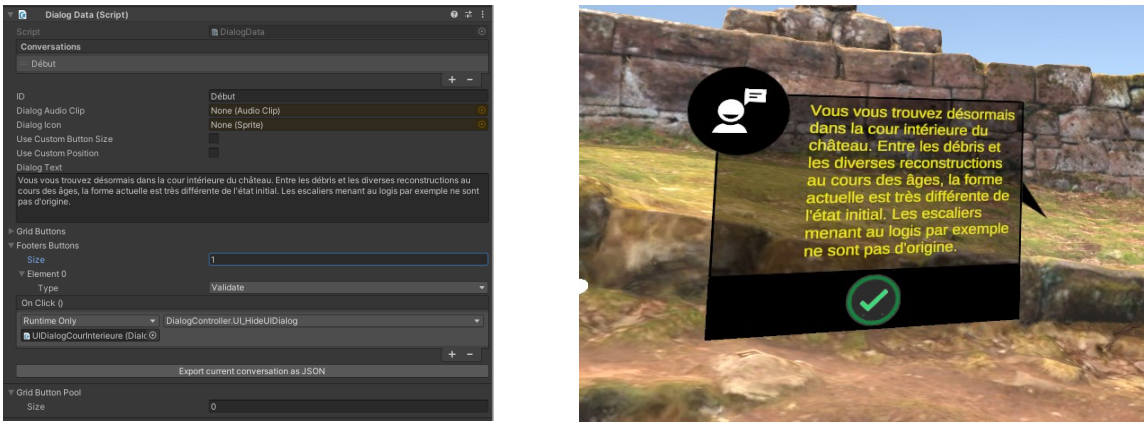

Figure 16. Description du script Dialog Data / Représentation de l'UIDialog dans la scène virtuelle.

b) Ajout de sons audios dans la scène

Il est possible sur Unity d'ajouter des sons audio (format MP3), considérés comme des *AudioClip* dans l'interface du logiciel. La méthode la plus simple d'intégrer un son dans la scène est de créer un *GameObject* disposant du composant *Audio Source*, permettant d'importer en paramètres un *AudioClip* (voir figure 47). Afin que l'utilisateur puisse entendre des sons, il faut un *GameObject* disposant du composant *AudioListener*. Dans notre cas, le *prefab CaveUser* du *XRPlayer* dispose par défaut de ce composant, dans le *GameObject Ears*, simulant le fait que l'avatar entende les sons depuis ses oreilles. La position des *GameObject* disposant de ce composant peut être important dans la mesure où il est possible de modifier la zone dans laquelle le son peut être entendu. Cela s'effectue à travers le paramètre *Spatial Blend*, sur une plage de 0 à 1. A 0 le son est entendu quelle que soit la position de l'*AudioListener*, à 1 le son ne peut être entendu que dans la zone d'écoute 3D.

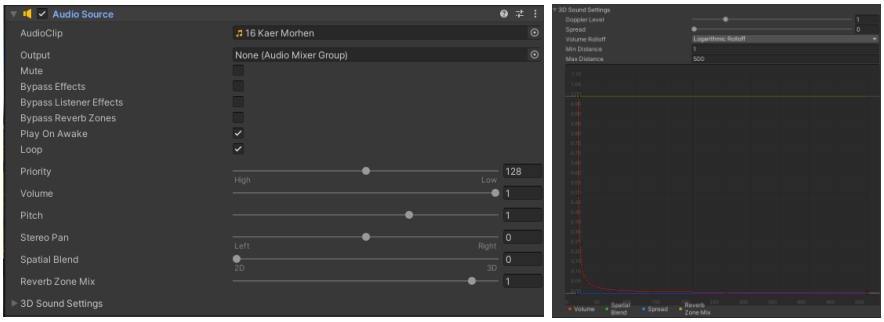

Figure 17. Description du composant Audio Source dans l'inspector.

Dans notre projet, nous avons uniquement ajouté une musique d'ambiance, rappelant l'ère médiévale, pour améliorer l'immersion de la visite. Mais d'autres utilisations peuvent être réfléchies en perspective. Par exemple, il faut savoir que dans les *UIDialog* (voir partie 5.4.a), il est possible d'ajouter un *AudioClip* qui s'exécute lorsque le texte défile. Nous pourrions alors penser à intégrer une voix qui récite le texte affiché dans les boîtes de dialogue, dans le principe d'un audioguide. Ceci améliorerait la communication des informations historiques pour l'utilisateur.

c) Perspectives d'outils à implémenter dans le projet

Pour conclure cette partie, nous allons présenter plusieurs outils décrits dans les tutoriels vidéo à notre disposition qui n'ont pas été ajoutés dans le projet dans le temps imparti, mais que nous avons estimé intéressants à évoquer en perspective d'amélioration :

- *Occlusion culling* : Unity a la possibilité d'appliquer cet effet (présenté en partie 1.1.c) dans la scène (*Window* > *Rendering* > *Occlusion culling*). Une fenêtre apparaît, permettant de régler certains paramètres. Pour calculer l'*Occlusion culling* (*Bake*), il faut définir tous les objets 3D de la scène qui sont statiques (c'est-à-dire qui ne sont pas amenés à se déplacer, au travers d'une animation par exemple). Cela aurait pour effet d'améliorer les performances graphiques. Pour guise d'information, la représentation dans le Cube VR oscille entre 20 et 40 FPS (*Frame Per Second*, soit le nombre d'images par seconde). En comparant la représentation dans le Cube du modèle 3D du château à l'état actuel de base avec celui réduit à travers l'utilisation de la *Normal map* (voir partie 3.2.b), nous obtenons une amélioration d'environ 10 FPS. Ceci implique que les pistes d'optimisation du rendu entraînent des conséquences non négligeables sur les performances dans la représentation VR.
- *Sprite* : ce type de *GameObject* permet d'ajouter une image 2D dans la scène 3D. Nous aurions pu l'utiliser pour afficher le tableau 7 présentant l'échelle de couleurs utilisée pour représenter le niveau de certitude de l'état historique, permettant au joueur d'accéder à ces informations directement dans la scène virtuelle.
- *Skybox* : ceci correspond à un matériau qui s'applique sur le fond de la scène virtuelle, tel un décor qui s'étend à l'horizon. Nous avons conservé la *Skybox* par défaut, qui donne une sensation de vide autour du château. Il serait possible d'utiliser une photographie 360° prise sur le site pour renforcer l'immersion.
- *DoTweenPath* : ce script propose de définir des points de passage dans la scène, formant un chemin sur lesquels les *GameObjects* disposant du composant peuvent se déplacer. Nous pouvons alors suggérer un mode de visite alternatif où la caméra virtuelle se déplace automatiquement autour du château dans la scène, dans le même principe qu'une vidéo de visite virtuelle (voir partie 1.3.a).

- *Rotating Door* : pour renforcer l'immersion dans la visite VR, il est possible de définir des interactions avec les objets 3D. Par exemple, le script *Rotating door* propose d'appliquer une rotation sur un modèle 3D, simulant l'ouverture d'une porte lorsque le joueur interagit avec l'objet.

Conclusion

Pour conclure, nous avons pu constater au travers d'une synthèse bibliographique la diversité des méthodes et réflexions pouvant s'appliquer sur un type de projet tel que la restitution 4D archéologique. Plusieurs enjeux et problématiques sont à prendre en compte dans la démarche globale selon le temps imparti, le cahier des charges à respecter, ou tout simplement la volonté de l'acteur du projet.

Nous avons réussi à suivre la chaîne de traitement de la figure 2 en respectant les différentes étapes, en commençant par les acquisitions lasergrammétriques et photogrammétriques relevées sur le terrain. Les traitements effectués sur les différents logiciels adaptés permettent d'obtenir un nuage de points dense, complet, et géoréférencé.

Ce nuage a été par la suite utilisé pour la modélisation 3D du MNT et de l'état actuel du site, comprenant le château en ruines ainsi que le rocher du Wachtfels. Pour cette étape, nous avons cherché à trouver les méthodes et logiciels les plus adaptés pour classifier les points au sol et modéliser au mieux, selon des critères de précision, de qualité visuelle ou encore selon le niveau de connaissance de l'utilisateur. Ceci est important pour permettre aux prochains acteurs du projet INTERREG VI d'optimiser leur temps pour manipuler le large panel d'outils, d'appareils et de méthodes pouvant être employés pour une restitution 4D. Nous avons conclu que le logiciel Metashape propose de nombreux avantages, notamment la possibilité d'utiliser des acquisitions photogrammétriques pour effectuer une cartographie UV sur le modèle 3D, tout en ouvrant des pistes d'optimisation du rendu au travers de la méthode de réduction du nombre de polygones en utilisant la *normal map*.

Nous avons également pu nous essayer aux rôles d'historien et d'infographiste au travers de la modélisation 4D sur le logiciel Blender. Nous avons alors réuni 3 représentations du modèle du château : l'état actuel, l'état historique avec des textures réalistes, et avec une échelle de couleurs représentant le niveau de certitude avec lequel a été restitué chaque élément.

Nous avons tout de même pu explorer une nouvelle technologie pour communiquer et mettre en valeur le modèle 4D, qui est le Cube VR. Nous avons pu exploiter plusieurs scripts du SDK ainsi que des outils propres au moteur de jeu Unity pour proposer une visite interactive et didactique du château de Wasenbourg à travers les âges.

Plusieurs perspectives sont à prendre en considération vis-à-vis de ce projet si nous disposions de plus de temps. Nous pouvons penser par exemple à améliorer le texturage en utilisant d'autres cartes UV présentées dans l'état de l'art, ou encore proposer d'autres méthodes de communication et de mise en valeur du modèle (réalité augmentée, visite virtuelle interactive sur un site internet). Nous pouvons également évoquer la modélisation paramétrique de certains éléments complexes tels que les fenêtres et portes, intégrés dans une bibliothèque d'objets. Ceci permettrait de gagner du temps sur la modélisation d'éléments complexes.

Liste des tableaux

Table des illustrations

Bibliographie

a23d@2023 (2023). *Different maps in PBR Textures*. URL : https://www.a23d.co/blog/different-maps-in-pbr-textures/

Agisoft@2023 (2023). *Texture map types*. URL : https://agisoft.freshdesk.com/support/solutions/articles/31000154587-texture-map-types

Angles, B. (2019). *Modélisation géométrique par primitives*. Thèse de doctorat

Autodesk@2023 (2018). *Familles Revit : présentation détaillée | Autodesk University*. URL : https://www.autodesk.com/autodesk-university/fr/article/Revit-Families-Step-Step-Introduction-2018

Barratt, R.P. (2016). Interpreting and Presenting Archaeological Sites Using 3D Reconstruction: Virtual Exploration of the Xaghra Brochtorff Circle in Gozo. Disponible sur : https://www.academia.edu/35713061/Interpreting_and_Presenting_Archaeological_Sites_Using_3D_Reconstruction_Virtual_Exploration_of_the_Xaghra_Brochtorff_Circle_in_Gozo Consulté le : 20 avril 2023

BeauxArts@2023 (2023). *Tour du monde des visites virtuelles les plus bluffantes*. URL : https://www.beauxarts.com/vu/tour-du-monde-virtuel-des-musees-comme-si-vous-y-etiez/

Benazzi, T. (2018). *Restitution 4D du Château du Kagenfels par combinaison de l'existant et d'hypothèses archéologiques pour une visite virtuelle du site*. Mémoire de Projet de Fin d'Etudes. INSA DE STRASBOURG. Disponible sur : http://eprints2.insa-strasbourg.fr/3513/

Bern, M. et Eppstein, D. (2000). QUADRILATERAL MESHING BY CIRCLE PACKING, *International Journal of Computational Geometry & Applications*, 1004, p. 347-360. Disponible sur : https://doi.org/10.1142/S0218195900000206

Blender@2023 (2023). *UV Tools — Blender Manual*. URL : https://docs.blender.org/manual/fr/dev/modeling/meshes/editing/uv.html

Bruna, R. (2014). *Modélisation 3D de la chapelle Saint-Laurent et de la place du Château (secteur 3) pour extraction de données archéologiques et visite virtuelle.* masters. INSA de Strasbourg. Disponible sur : http://eprints2.insa-strasbourg.fr/1777/ Consulté le : 10 février 2023

Cartier, L. (2019). *Modélisation 3D du château disparu des Wurtemberg à Horbourg-Wihr et*

exploitation de la réalité augmentée pour une mise en valeur dans la trame urbaine contemporaine. Mémoire de Projet de Fin d'Etudes. INSA DE STRASBOURG. Disponible sur : http://eprints2.insa-strasbourg.fr/3829/

Cohen-Or, D., Chrysanthou, Y. et Silva, C. (2001). A Survey of Visibility for Walkthrough Applications, *Proceedings of SIGGRAPH* [Preprint]

Coorg, S. et Teller, S. (1997). Real-time occlusion culling for models with large occluders, in *Proceedings of the 1997 symposium on Interactive 3D graphics - SI3D '97. the 1997 symposium*, Providence, Rhode Island, United States: ACM Press, p. 83-ff. Disponible sur : https://doi.org/10.1145/253284.253312

Czarnowsky, C. (1937). Die Wasenbourg bei Niederbronn-les-bains, in *D'Elsässer Kalender Hüsfrind*, p. 117-122.

DEC@2023 (2023). *Salle immersive 3D - DEC*. URL : https://www.dec-industrie.com/Salle-immersive-3D

Dell'Unto, N., Ferdani, D., Leander Touati, A.-M., Dellepiane, M. et Callieri, M. (2013). Digital reconstruction and visualization in archaeology Case-study drawn from the work of the Swedish Pompeii Project, in, p. 621-628. Disponible sur : https://doi.org/10.1109/DigitalHeritage.2013.6743804

Desjardin, E., Nocent, O. et Runz, C. de (2012). Prise en compte de l'imperfection des connaissances depuis la saisie des données jusqu'à la restitution 3D, *Archeologia e Calcolatori*, Sup. 3, p. 385.

DiFrancesco, P.-M., Bonneau, D. et Hutchinson, D.J. (2020). The Implications of M3C2 Projection Diameter on 3D Semi-Automated Rockfall Extraction from Sequential Terrestrial Laser Scanning Point Clouds, *Remote Sensing*, 1211, p. 1885. Disponible sur : https://doi.org/10.3390/rs12111885

Dufaÿ, B. (2014). La modélisation 3D de grands ensembles monumentaux de la restitution au public à la recherche scientifique, p. 149-163.

Encyclopédie de l'Alsace (1986). Editions Publitotal (Encyclopédie de l'Alsace, vol. 12). Disponible sur : https://books.google.fr/books?id=i1MvAAAAMAAJ

Favre-Brun, A. (2013). Architecture virtuelle et représentation de l'incertitude: analyse des solutions de visualisation de la représentation 3D. *Colloque Virtual Retrospect 2013*

Hassan, M. (2016). *Proposed workflow for UV mapping and texture painting*. Disponible sur : https://urn.kb.se/resolve?urn=urn:nbn:se:bth-12799 Consulté le : 2 août 2023

Kazhdan, M., Chuang, M., Rusinkiewicz, S. et Hoppe, H. (2020). Poisson Surface Reconstruction with Envelope Constraints, *Computer Graphics Forum*, 395, p. 173-182. Disponible sur : https://doi.org/10.1111/cgf.14077

JetBrains@2023 (2023). *dotPeek : Décompilateur .NET et navigateur assembleur gratuit par JetBrains*. URL : https://www.jetbrains.com/fr-fr/decompiler/

JeuxActu@2023 (2017). URL : https://www.jeuxactu.com/nintendo-switch-la-manette-joy-con-en-detail-107553.htm

Lague, D., Brodu, N. et Leroux, J. (2013). Accurate 3D comparison of complex topography with terrestrial laser scanner: Application to the Rangitikei canyon (N-Z), *ISPRS Journal of Photogrammetry and Remote Sensing*, 82, p. 10-26. Disponible sur : https://doi.org/10.1016/j.isprsjprs.2013.04.009

Landes, T., Grussenmeyer, P. et Boulaassal, H. (2011). Les principes fondamentaux de la lasergrammétrie terrestre: acquisition, traitement des données et applications, *XYZ,* 129, p. 25-38.

Landes, T., Heissler, M., Koehl, M., Benazzi, T. et Nivola, T. (2019). Uncertainty Visualization Approaches for 3d Models of Castles Restituted from Archeological Knowledge, *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 42W9, p. 409-416. Disponible sur : https://doi.org/10.5194/isprs-archives-XLII-2-W9-409-2019

Levy, B., Petitjean, S., Ray, N. et Maillot, J. (2002). Least Squares Conformal Maps for Automatic Texture Atlas Generation, *ACM Trans. Graph.*, 21, p. 362-371. Disponible sur : https://doi.org/10.1145/566654.566590

Liu, L., Gotsman, C., Zhang, L., Xu, Y. et Gortler, S. (2008). A Local/Global Approach to Mesh Parameterization, *Computer Graphics Forum*, 27. Disponible sur : https://doi.org/10.1111/j.1467-8659.2008.01290.x

Luximon, A. et Luximon, Y. (2012). Shoe-Last Design and Development, in, p. 193-212. Disponible sur : https://doi.org/10.1201/b13021-13

Matthis, C. (1902). Communications du Club Vosgien, 36. Disponible sur : https://gallica.bnf.fr/ark:/12148/bpt6k9625641c/f1.item

Matthis, C. (1906). *Die Wasenburg: Eine elsässische Ritterburg im 14. Jahrhundert und ein römischer Merkurtempel*. Heitz. Disponible sur : https://books.google.fr/books?id=tVy8mgEACAAJ

Mengus, N. et Rudrauf, J.-M. (2013). *Châteaux forts et fortifications médiévales d'Alsace*. La Nuée Bleue Strasbourg

Nivola, T. (2018). *Modélisation 3D du castellum de Horbourg-Wihr et exploitation de la réalité augmentée pour une mise en valeur dans la trame urbaine contemporaine*. Mémoire de Projet de Fin d'Etudes. INSA DE STRASBOURG. Disponible sur : http://eprints2.insa-strasbourg.fr/3441/

Richalet-Chaudeur, T. (2022). *Relevé et modélisation 3D des galeries des contremines de la citadelle de Doullens (Somme)*. Mémoire de Projet de Fin d'Etudes. INSA DE STRASBOURG. Disponible sur : http://eprints2.insa-strasbourg.fr/4859/

Rocha, M. (2022). *Levé et numérisation du château de Lichtenberg en vue d'une proposition de visite virtuelle du site à des périodes remarquables*. Mémoire de Projet de Fin d'Etudes.

INSA DE STRASBOURG. Disponible sur : http://eprints2.insa-strasbourg.fr/4882/

Rocheleau, M. (2011). La modélisation 3D comme méthode de recherche en sciences historiques

Rodríguez-Gonzálvez, P., Muñoz-Nieto, A.L., del Pozo, S., Sanchez-Aparicio, L.J., Gonzalez-Aguilera, D., Micoli, L., Gonizzi Barsanti, S., Guidi, G., Mills, J., Fieber, K., Haynes, I. et Hejmanowska, B. (2017). 4D RECONSTRUCTION AND VISUALIZATION OF CULTURAL HERITAGE:ANALYZING OUR LEGACY THROUGH TIME, *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XLII-2/W3, p. 609-616. Disponible sur : https://doi.org/10.5194/isprs-archives-XLII-2-W3-609-2017

Runz, C. de (2008). *Imperfection, temps et espace : modélisation, analyse et visualisation dans un SIG archéologique*. phdthesis. Université de Reims - Champagne Ardenne. Disponible sur : https://theses.hal.science/tel-00560668 Consulté le : 20 avril 2023

Salesse, J. (2018). *Wasenbourg si tu m'étais contée...* Les amis de Wasenbourg

Sabaliauskas, M. et Marcinkevičius, V. (2015). An Investigation of ABF++, LSCM, and ARAP Methods for Parametrization of Shoetrees, *Mokslas - Lietuvos ateitis*, 7, p. 295-299. Disponible sur : https://doi.org/10.3846/mla.2015.786

Sanseverino, A., Limongiello, M. et Fiorillo, F. (2022). UAV photogrammetric survey and Image-Based elaborations for an Industrial Plant - DISEGNARECON, 15, p. 15.1-15.10. Disponible sur : https://doi.org/10.20365/disegnarecon.29.2022.15

Sheffer, A., Levy, B., Mogilnitsky, M. et Bogomyakov, A. (2005). ABF++ : Fast and Robust Angle Based Flattening, *ACM Trans. Graph.*, 24, p. 311-330. Disponible sur : https://doi.org/10.1145/1061347.1061354

Sheffer, A. et de Sturler, E. (2001). Parameterization of Faceted Surfaces for Meshing using Angle-Based Flattening, *Engineering with Computers*, 17, p. 326-337. Disponible sur : https://doi.org/10.1007/PL00013391

Shutterstock@2023 (2023). *Castle Ruins Stock Vector (Royalty Free)*. URL : https://www.shutterstock.com/image-vector/castle-ruins-148332056

Sorkine, O. et Alexa, M. (2007). As-rigid-as-possible surface modeling, in *Proceedings of the fifth Eurographics symposium on Geometry processing*. Goslar, DEU: Eurographics Association (SGP '07), p. 109-116.

Sorkine, O. et Cohen-Or, D. (2001). Warped Textures for UV Mapping Encoding
Unity@2023 (2023). *Unity - Manual: Albedo Color and Transparency*. URL : https://docs.unity3d.com/Manual/StandardShaderMaterialParameterAlbedoColor.html

Spiria@2023 (2016). *Comprendre le "UV-mapping" et les textures*. URL : https://www.spiria.com/fr/blogue/applications-desktop/comprendre-le-uv-mapping-et-les-textures/

Verhoeven, G.J. (2017). COMPUTER GRAPHICS MEETS IMAGE FUSION: THE POWER OF TEXTURE BAKING TO SIMULTANEOUSLY VISUALISE 3D SURFACE FEATURES AND COLOUR, *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, IV-2-W2, p. 295-302. Disponible sur : https://doi.org/10.5194/isprs-annals-IV-2-W2-295-2017

VirtuelConcept@2023 (2023). *VirtualConcept - Accueil*. URL : https://virtuelconcept32.com/

Wang, Y., Bu, J., Li, N., Song, M. et Tan, P. (2012). Detecting discontinuities for surface reconstruction, in *Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012)*. *Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012)*, p. 2108-2111.

Wang, Z., Luo, Z., Zhang, J. et Saucan, E. (2016). ARAP++: an extension of the local/global approach to mesh parameterization, *Frontiers of Information Technology & Electronic Engineering*, 176, p. 501-515. Disponible sur : https://doi.org/10.1631/FITEE.1500184

Weatherill, N.P. (1992). Delaunay triangulation in computational fluid dynamics, *Computers & Mathematics with Applications*, 245, p. 129-150. Disponible sur : https://doi.org/10.1016/0898-1221(92)90045-J

Webster, N. (2017). High poly to low poly workflows for real-time rendering, *Journal of Visual Communication in Medicine*, 40, p. 40-47. Disponible sur : https://doi.org/10.1080/17453054.2017.1313682

Yuksel, C., Lefebvre, S. et Tarini, M. (2019). Rethinking Texture Mapping, *Computer Graphics Forum*, 382, p. 535-551. Disponible sur : https://doi.org/10.1111/cgf.13656

Zhang, H. et Hoff, K.E. (1997). Fast backface culling using normal masks, in *Proceedings of the 1997 symposium on Interactive 3D graphics - SI3D '97*. *the 1997 symposium*, Providence, Rhode Island, United States: ACM Press, p. 103-ff. Disponible sur : https://doi.org/10.1145/253284.253314

Zhang, W., Qi, J., Peng, W., Wang, H., Xie, D., Wang, X. et Yan, G. (2016). An Easy-to-Use Airborne LiDAR Data Filtering Method Based on Cloth Simulation, *Remote Sensing*, 8, p. 501. Disponible sur : https://doi.org/10.3390/rs8060501

Zhang, J., Yao, Y. et Deng, B. (2021). Fast and Robust Iterative Closest Point, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, p. 1-1. Disponible sur : https://doi.org/10.1109/TPAMI.2021.3054619