



Ecole Nationale des Sciences Géographiques



Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

Rapport de stage

Cycle des ingénieurs diplômés de l'ENSG 2^{ème} année

RÉALISATION D'UNE SALLE IMMERSIVE



Co-funded by the
Erasmus+ Programme
of the European Union

Mathéo MARECHAL

6 Septembre 2023

Non confidentiel Confidentiel IGN Confidential Industriel Jusqu'au ...

Organisme de stage

Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud
Route de Cheseaux 1, CH-1401 Yverdon-les-Bains, Suisse

Maître(s) de stage

Adrien Gressin, Professeur en photogrammétrie et télédétection

Bertrand Cannelle, Professeur en BIM et géovisualisation et responsable de filière géomatique

Professeur(s) référents(s)

Victor Coindet, Professeur et Responsable du Pôle d'Enseignement en Technologies des Systèmes d'Information

Stage du 22 Mai au 18 août 2023

Nombre de pages : 30 dont 11 d'annexes

Version : 1

Remerciements

Je souhaite ici exprimer mes sincères remerciements à Adrien Gressin et Bertrand Cannelle pour ces trois mois vraiment enrichissants. Leur soutien a été d'une grande aide lorsque nécessaire, et bien que supervisant le projet d'un peu plus loin que moi, je trouve que leurs conseils étaient toujours précis, m'aidant à garder le recul nécessaire.

Je tiens aussi à les remercier pour le sujet passionnant qui m'ont donné et leur attitude positive et accueillante a largement contribué à faire de ce stage une expérience vraiment agréable. Un travail ambitieux et motivant dans une atmosphère sympathique, d'aide et d'échanges, voilà le résumé de ce stage. En plus, ayant cassé mon téléphone lors de ce stage, ils n'ont pas hésité à m'en prêter un jusqu'à mon retour chez moi, renforçant ainsi ma gratitude envers eux.

J'en profite par ailleurs pour remercier plus globalement tous ceux qui m'ont aidé dans le groupe Géomatique dès que j'avais besoin de quelque chose ou un problème informatique.

Je tiens également à exprimer ma reconnaissance envers le service des relations internationales pour leur aide précieuse dans la facilitation de mon accès à l'école et mon installation.

J'adresse finalement ma gratitude à l'ensemble de la HEIG-VD et à l'ENSG pour avoir favorisé une telle opportunité et aussi un grand merci pour l'appui financier de la Fondation ENSG et d'Erasmus+ qui a été d'une importance capitale, allégeant vraiment les contraintes financières.

Résumé

Mon but a été de réaliser une salle immersive à l'aide de vidéoprojecteurs, comme si nous transformions les murs en écrans, en projetant un environnement virtuel grâce à Unreal Engine, un moteur de jeu vidéo qui nous permet notamment de se mouvoir dans le monde que nous allons visualiser.

Ainsi, je me suis dans un premier temps imprégner de ce moteur et de ses plug-ins utiles à la projection, ici : nDisplay et Switchboard. Ensuite, l'immersion étant l'unique critère de la réussite, il a fallu étudier comment corriger la projection sur n'importe quelle surface et bien veiller à ce que les projections correspondent bien entre elles.

Finalement, parmi toutes les possibilités de configuration possible, je montrerai le résultat avec celle choisie.

Mots clés : salle immersive, vidéoprojecteurs, Unreal Engine, corriger la projection, recouvrement

Abstract

My goal here was to create an immersive room using projectors, as if we were transforming the walls into screens, projecting the desired virtual environment using Unreal Engine, a video game engine that allows us to navigate within the visualized world.

To begin with, I familiarized myself with this engine and its useful plugins for projection, namely nDisplay and Switchboard. Then, since immersion was the key criterion for the success of the room, it was necessary to study how to correct the projection on any surface and ensure that the projections aligned seamlessly.

Finally, among all the possible configuration options, I will showcase the results with the chosen setup.

Table des matières

Introduction

I - Les fondamentaux de la projection

1 Les comprendre

- a. Ses caractéristiques
- b. L'exemple classique
- c. Bougeons légèrement le projecteur

2. Les corriger

- a. Dans la théorie
- b. Dans la pratique - partie terrain

II - La projection dans Unreal Engine

1 Présentation rapide

- a. Pourquoi un moteur de jeu vidéo ?
- b. Les outils proposés pour la projection

2 Intégration de la configuration des projections

- a. Construction des écrans virtuels à partir des formes projetées
- b. Dans la pratique de la correction - partie Unreal Engine
- c. Recouvrement

III - Mise en place finale

1 Placement des projecteurs

2 Perspectives d'améliorations

Conclusion

Introduction

Dans le cadre de ma formation à l'ENSG Géomatique, j'ai eu la chance de réaliser un stage à la HEIG-VD car cette école et plus spécifiquement son groupe Géomatique, aimerait avoir à sa disposition une salle immersive comme outil de visualisation 3D et m'a alors chargé de la réalisation de ce fabuleux projet.

Une salle immersive, c'est une pièce où les participants sont immergés dans des environnements virtuels grâce à des écrans ou des projections. Utilisant la réalité virtuelle ou augmentée, ces salles offrent une expérience potentiellement interactive, qui ouvrent des possibilités diverses dans le divertissement, la formation et la simulation. Dans le cas de cette école, cette salle pourrait être montrée au grand public à l'occasion de portes-ouvertes et servirait plus généralement pour visualiser des données 3D appartenant aux projets d'enseignants, d'associés ou d'étudiants.

Concernant ce stage, son but premier est la création d'un prototype fonctionnel d'une salle immersive en utilisant plusieurs vidéoprojecteurs (ensuite appelés projecteurs) et un moteur de jeu vidéo, dans une salle borgne dédiée pour l'occasion. Ce but donne alors lieu à différents critères, comme par exemple que les projections soient d'abord correctes, c'est-à-dire qui ne présentent pas de déformations ou au mieux, qui ne bloquent pas l'immersion. Un autre critère intuitif est que ces projections doivent se joindre et enfin, le dernier est que les utilisateurs pourront se déplacer en temps réel à l'intérieur de l'environnement virtuel projeté.

Cependant, avec cette salle, nous allons utiliser nos projecteurs dans des configurations originales qui créent des déformations nuisant justement à l'immersion. Ainsi, un de nos grands objectifs va être de trouver un protocole qui pourrait corriger n'importe quelle projection. Cela nous aidera à nous affranchir du contexte spatial et augmentera nos possibilités dans les dispositions. De cette manière, la compréhension des fondamentaux de la projection devient obligatoire et représentera alors notre première grande partie.

Dans notre sillage, il a aussi été de mise d'étudier les différents outils disponibles parmi les moteurs de jeu vidéo existants pour déterminer celui qui sied le plus à notre besoin. Ce qui nous donne la deuxième partie de ce rapport qui présente ainsi notre choix : Unreal Engine et en même temps, qui nous fait voir ce que permet un moteur de jeu vidéo sur les projections, notamment concernant les deux objectifs de correction ou de recouvrement des projections.

Finalement, avec toutes ces connaissances, nous pourrons alors passer à la troisième et dernière partie où nous essayerons de créer une configuration qui remplit ces objectifs et nous verrons si nous avons réussi à accomplir notre but.

Lors de ce rapport, nous allons suivre ce genre de configuration qui servira d'exemple de configuration originale, un exemple qui s'inscrit comme cela dans un repère terrain que nous allons garder tout au long de ce rapport :

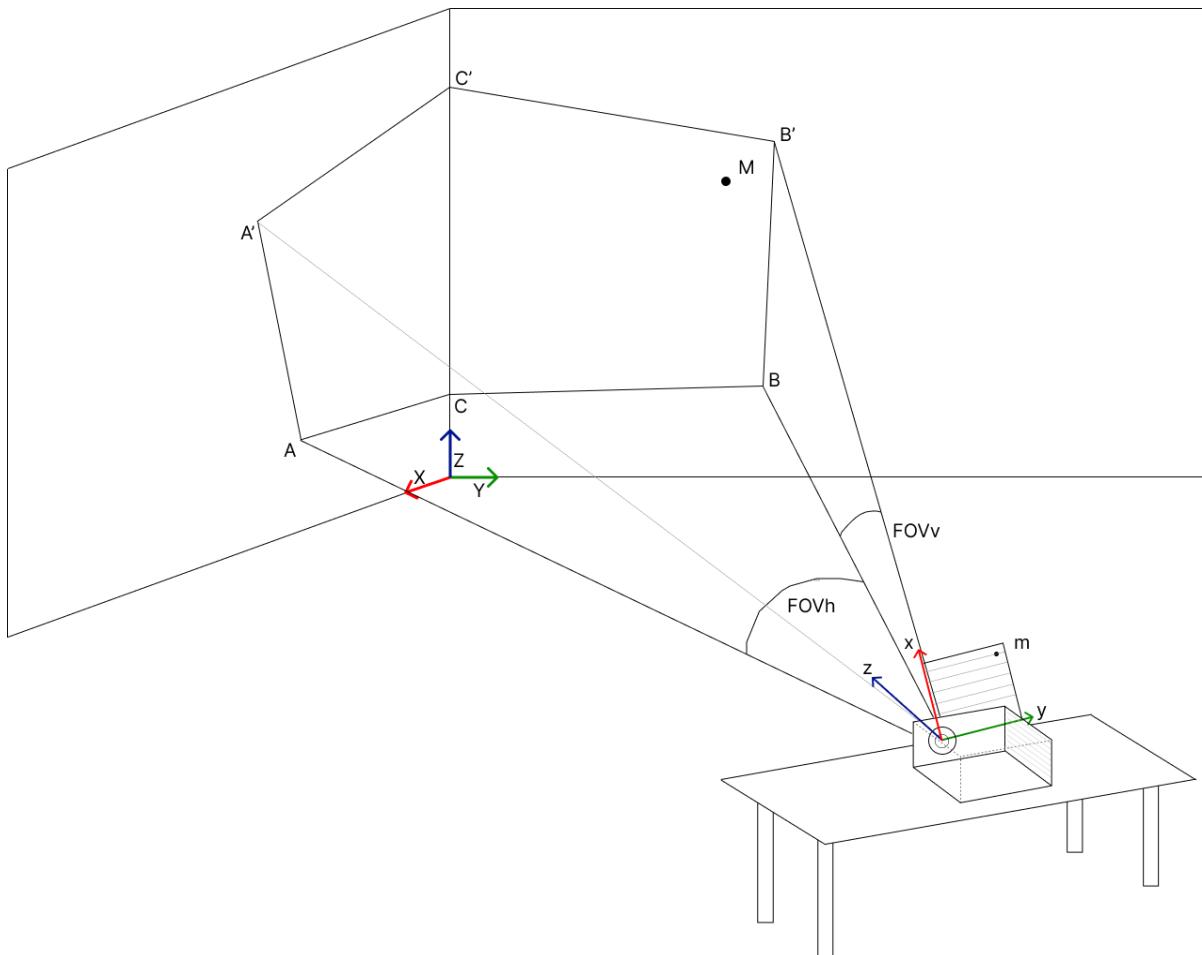


Figure 1 : Schéma présentant une projection originale réalisée ‘dans un angle’

Notons que nous utiliserons par la suite “projecteurs” pour remplacer “vidéoprojecteurs”. Le terme forme projetée sera le terme pour désigner le polygone ACBB'C'A'. La configuration pour l'ensemble des dispositions des projecteurs, et donc aussi des projections. La surface de projection sera le plan qui possède la forme projetée. Unreal pour Unreal Engine.

1 Les fondamentaux de la projection

Dans cette partie, nous allons voir ce qu'est une projection et comprendre pourquoi nous avons des déformations puis nous allons voir comment les corriger.

1. Les comprendre

a. Ses caractéristiques

Pour commencer, étudier une projection, c'est d'une part l'étude des caractéristiques intrinsèques du projecteur et ensuite, l'étude sa disposition par rapport à la surface de projection, c'est-à-dire ses caractéristiques extrinsèques. L'idée est ici de savoir ce qui définit la projection et trouver les responsables de ce que nous observons, ce qui semble inévitable si nous voulons comprendre puis manipuler chacun de ces paramètres. Nous saurons aussi ce qui tend à changer à l'avenir.

Comme dit précédemment, un projecteur possède d'abord des caractéristiques qui lui sont propres, ses fameuses caractéristiques intrinsèques. Par exemple :

- La résolution du projecteur (x, y),
étant par ailleurs la seule caractéristique qui n'a pas d'impact direct sur la forme observée,
ce premier et deuxième terme font respectivement référence à la taille de l'image en pixel
horizontalement et verticalement. Notons alors qu'une résolution deux fois plus grande
permet une image avec quatre fois plus de pixel

- Le FOV (horizontal et vertical) ou champ de vision du projecteur fait référence à l'angle de
vue couvert par l'image projetée. Souvent, un projecteur émet naturellement vers le haut, de 0
à 40 degrés par exemple, ce qui propose ce genre de projection :

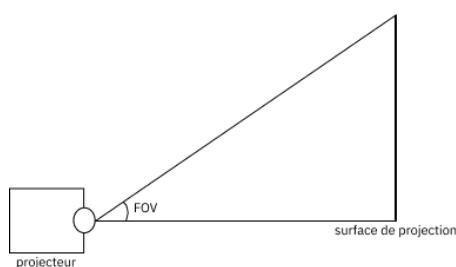


Figure 2 : Schéma de côté présentant la projection naturelle du FOV vertical

Effectivement, ce n'est pas parmi ces caractéristiques que nous trouverons les responsables des déformations, mais notons que la qualité de la projection et comment nous allons placer les projecteurs dépendent de celles-ci. Ce sont bien ces caractéristiques qui vont permettre certaines configurations et nous en refuser d'autres.

Ensuite, nous avons les caractéristiques extrinsèques, c'est-à-dire les caractéristiques du projecteur qui sont uniquement liées à la surface de projection. Par exemple :

- la distance de projection,

comme nous venons de le voir, ce paramètre est lié au FOV et à eux deux, ils influencent la taille de la projection, nous verrons plus tard concrètement comment cela se répercute

- l'orientation et l'inclinaison du projecteur,

prenons cet exemple de projection pour d'abord poser ce qu'est l'orientation :

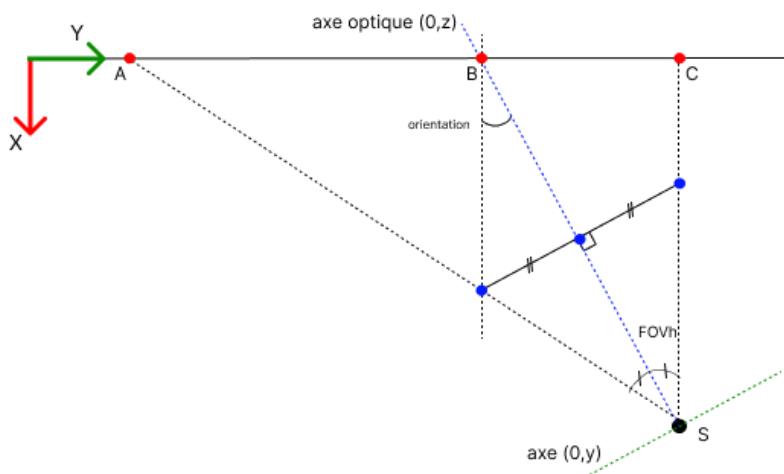


Figure 3 : Schéma de dessus présentant l'orientation du projecteur

Sur ce schéma, l'orientation est l'angle formé par l'axe optique ($0, z$) du projecteur dans le plan (O, X, Y). À noter que si nous donnons une inclinaison au projecteur, la même logique s'applique : l'axe optique ne sera plus coplanaire avec ce plan, et appelons alors inclinaison l'angle formé par l'axe optique dans le plan (O, X, Z). Ainsi, nous pourrons à l'avenir introduire une matrice de rotation pour passer du repère image (O, x, y, z) au repère terrain (O, X, Y, Z) qui contiendra alors ces angles.

Ici, l'image a à sa disposition le même nombre de pixels entre A et B qu'entre B et C, ce qui donne alors des lignes plus ou moins étirées selon l'endroit de l'image et de cette manière, nous observons que ce sont ces angles qui sont bien les auteurs des déformations.

b. L'exemple classique

Continuons cette hypothèse et intéressons-nous au cas classique de la projection pour comprendre comment ces caractéristiques se manifestent lors d'une projection que nous connaissons. Cela nous permettra par la suite de comprendre plus en détails les projections plus originales.

Dans cette partie, nous allons étudier la projection faite en salle de classe ou en conférence par exemple. En somme, la projection qui a pour but une forme rectangulaire sans déformation. Voici cette configuration que nous pouvons théoriquement seulement approcher :

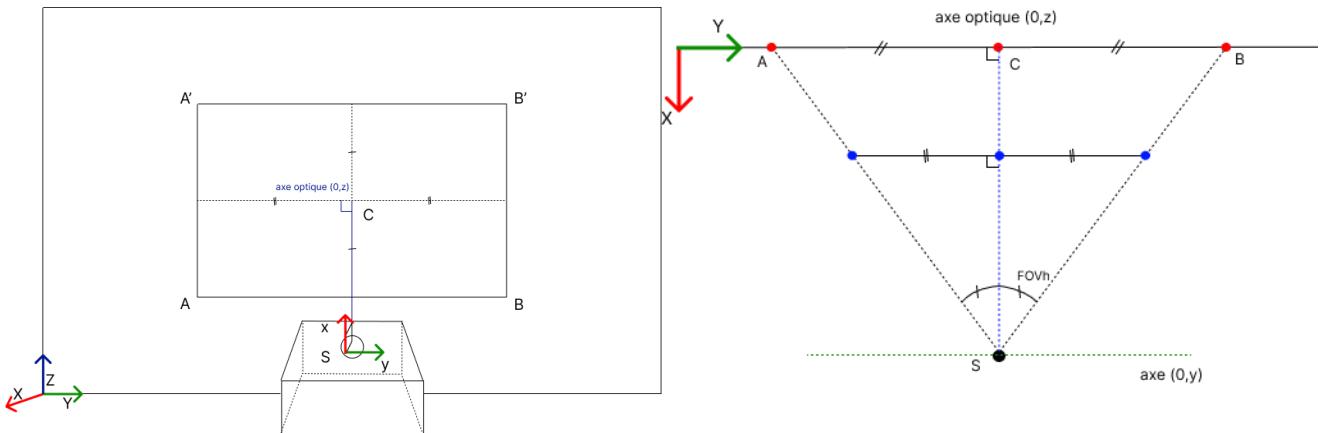


Figure 4.a : Schéma de la projection classique

Figure 4.b : Schéma de dessus de cette projection classique

Si nous comparons avec la configuration en figure 3, la première observation serait de voir que les milieux sont cette fois-ci conservés. En effet, ici, les quatres coins sont à égale distances du centre de projection mais aussi du projecteur. Pourquoi cela ?

Instinctivement, nous avons placé le projecteur le plus droit possible, horizontalement et verticalement, autrement dit l'axe optique est perpendiculaire à la surface de projection. De cette manière, on réalise une projection avec une orientation et une inclinaison très particulières puisque le plan de l'image ($0, x, y$) est parallèle avec le plan de la surface de projection. Bien que ces deux propositions soient équivalentes, cette condition est alors nécessaire pour réaliser une projection sans déformation.

Si nous continuons la comparaison d'un point de vue résolution cette fois-ci, nous avons bien à disposition autant de pixels entre A et C qu'entre C et B. Pour plus de détails, imaginons une projection réalisée avec un projecteur Haute Définition (1280x720) placé à 2m du centre de la projection avec un champ de vision horizontale de 60 degrés.

alors, avec le deuxième schéma, nous observons que : $\tan\left(\frac{FOVh}{2}\right) = \frac{AB}{2 \cdot SC}$

donc, si nous fixons le FOVh et augmentons la distance de projection, la projection s'élargit ; même résultat si nous fixons la distance de projection mais augmentons le FOVh.

Nous obtenons par ailleurs une longueur de projection environ égale à 2,30 mètres pour 1280 pixels, ce qui nous donne la moyenne de la longueur d'un pixel à environ 1,8 mm. Là où dans une configuration bien étirée, i.e. avec une orientation importante, la longueur du pixel le plus éloigné du centre de projection peut-être deux fois plus grande .

La question est désormais de savoir ce qu'il se passe si nous déplaçons l'axe optique ? Quelle est vraiment l'influence de l'orientation sur la forme de la projection ?

Voyons voir cela dans une configuration un peu plus originale.

c. Bougeons légèrement le projecteur

Dans cette partie, nous allons observer puis expliquer ce qu'il se passe si nous décalons horizontalement l'axe de projection.

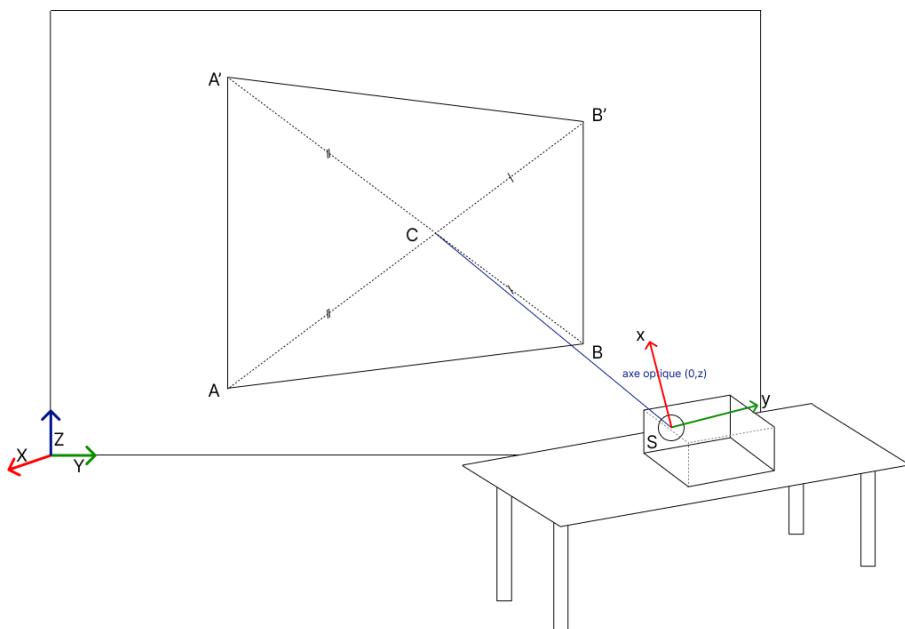


Figure 5 : Schéma de la projection avec une légère orientation du projecteur par rapport au mur

Comme dans les précédentes projections, la forme observée est le résultat de l'intersection du cône de projection avec la surface, le mur sur lequel on projette.

Cependant, les deux coins gauches de la projection sont ici plus éloignés du projecteur et comme nous avions précisé qu'en augmentant la distance, la projection s'agrandit, nous retrouvons alors cette forme qui semble maintenant logique puisque cela ne concerne pas que les coins mais aussi tous les

pixels de l'image. Effectivement, même à plus petite échelle, ce constat peut être fait pour chaque pixel projeté. Petite parenthèse : AA' et BB' restent verticales car l'inclinaison n'a pas changée.

Nous pouvons ainsi imaginer que la correction de cette projection est une projection qui présente elle-même des déformations mais qui, combinée à la déformation déjà présente, simule à nouveau les milieux conservés que nous pouvons observer en salle de classe. Une projection qui viendrait en quelques sortes inverser ce phénomène.

2. Les corriger

Grâce à la partie précédente, nous commençons à apercevoir une idée de la correction à appliquer. Il est alors temps de la concrétiser dans celle-ci maintenant que nous avons toutes les connaissances nécessaires.

Nous allons ici pousser notre raisonnement un peu plus loin et peu à peu intégrer Unreal Engine et ses spécificités pour trouver un protocole qui corrigera n'importe quelle projection. Pour cela, mettons nous dans le cas de la projection originale et compliquée visible sur le schéma en figure 1 où nous réalisons une projection dans un angle, c'est-à-dire sur deux murs.

a. Dans la théorie

Rentrons dans la partie théorique du protocole. Même lors de cette projection, rien ne nous empêche de séparer l'image projetée en deux si plus tard nous arrivons à les corriger individuellement et que nous arrivons ensuite à les joindre ; ce qui sera le cas grâce à Unreal Engine. Comme cela, même en les séparant en deux formes bien distinctes, la projection pourra tout de même être considérée comme complètement corrigée.

Maintenant, prenons à part une de ces deux formes et imaginons que nous avons accès à la disposition du projecteur par rapport à celle-ci, nous avions vu que nous pouvons mathématiquement la corriger en donnant à notre projecteur une image compensée où la combinaison de la déformation de celle-ci plus celle due à la configuration spatiale se compense. Autrement dit, l'idée concrète est de trouver un moyen qui donne au projecteur l'image apparaissant dans le plan qui crée cette déformation inverse.

Pour créer cela, ce plan imaginaire devrait être le plan symétrique au plan du mur par rapport au plan (O, x, y) du repère image translaté sur le point $(0, 0, 0)$ qui appartient aux deux murs. Voici un schéma de l'idée :

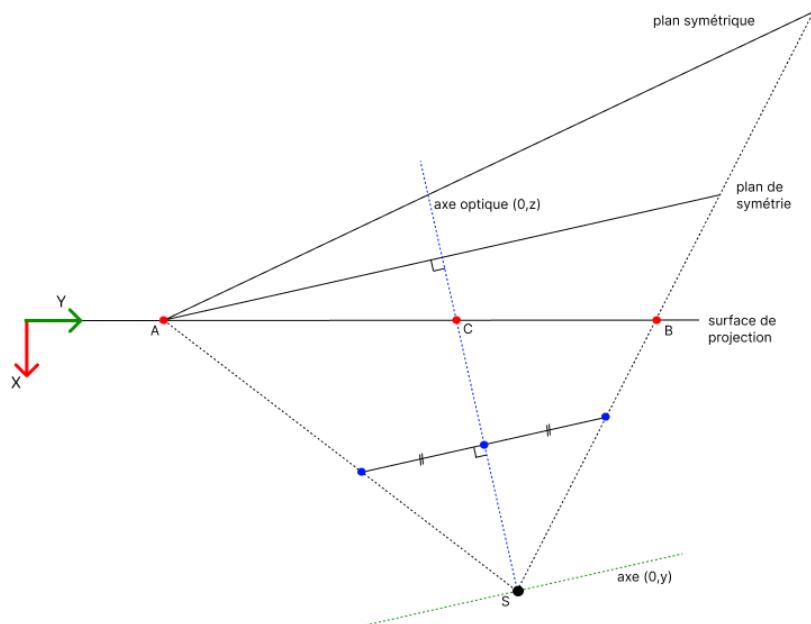


Figure 6 : Schéma de dessus de l'idée de la correction présentant le plan de symétrie et ce plan symétrique

De cette manière, nous corrigéons en effet les déformations en rectifiant la distance au projecteur sur l'ensemble de la projection et l'image projetée sera celle qui apparaît si nous réalisons la projection directement sur le plan de symétrie, donc sans déformation.

Dans notre configuration originale, ce plan du repère image poussé jusqu'au coin de l'angle du mur ressemblerait à cela :

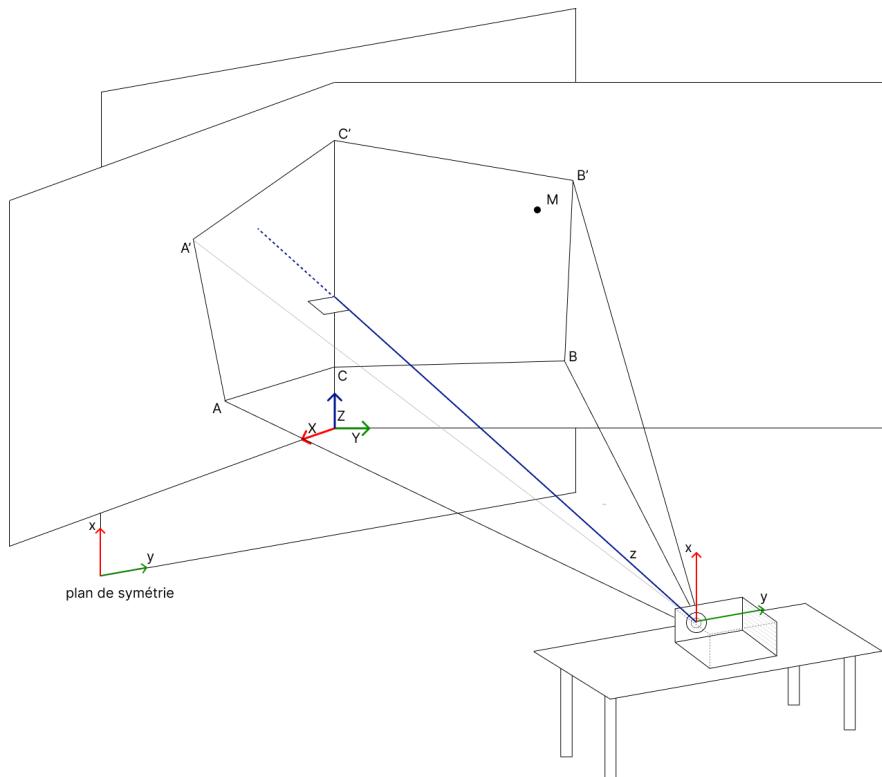


Figure 7 : Schéma de la projection en figure 1 avec le plan de plan de symétrie poussé vers l'angle

Maintenant, l'hypothèse de départ était d'avoir la disposition du projecteur par rapport à sa projection donc pour obtenir l'image qui corrigera la forme en question, il va nous falloir au minimum les coordonnées de ses coins mais aussi la matrice de rotation du projecteur et ses coordonnées dans le repère terrain que nous avons fixé. Avec cela, nous aurons d'ailleurs la possibilité d'obtenir ce plan de symétrie et ensuite, nous pourrons s'il le faut calculer les coordonnées de la forme corrective qui appartient à ce dernier. "S'il le faut", car les options laissées par Unreal permettront de ne finalement pas y avoir recours. Pour obtenir les coordonnées des coins, nous pouvons simplement les mesurer à l'aide d'un double-mètre, ce qui garantie un ordre de précision de l'ordre du millimètre mais pour le reste, il est obligatoire de réaliser une Direct Linear Transformation (DLT), un algorithme qui établit le lien entre les points de la scène et leurs projections sur l'image.

Pour faire simple, nous réalisons cela en trouvant l'équation de colinéarité qui donne le point de l'image en fonction de paramètres physiques dans le repère terrain. Comme cela, nous linéarisons grâce à 11 paramètres chaque point de l'image. Ensuite, en prenant quelques valeurs qui serviront d'observations, nous résolvons par moindres carrés ces paramètres et à partir de là, nous avons finalement accès à la position du projecteur dans le repère terrain et la matrice de rotation qui lie ce repère terrain au repère image.

b. Dans la pratique - partie terrain

Dans cette dernière partie, nous obtenons le protocole de la DLT que nous allons décrire ci-dessous :

- La première chose à faire est de régler physiquement le projecteur pour que la projection soit au moins verticalement correcte au niveau de l'angle. Cela n'est pas difficile à réaliser mais aidera au résultat pour la suite.

Toujours en guise de préparation, on utilise le repère (O, X, Y, Z) :

pour que nos deux plans soient visibles et atteignables facilement. Comme cela, nous pouvons supposer qu'ils sont bien orthogonaux, ce qui implique par exemple que chaque position sur le plan (O, X, Z) a alors forcément un Y égale à 0.

- Il faut ensuite faire correspondre les coordonnées en pixel (x, y) de la projection avec leurs positions (X, Y, Z) dans l'espace. Pour ce faire, projetons en plein écran un damier d'une taille connue. De cette manière, les coordonnées en pixel de chaque coin de chaque case du

damier étaient connues et il nous reste alors qu'à mesurer les coordonnées spatiales correspondantes. Il ne reste qu'à consigner ces données dans un tableau, puis dans une bibliothèque python.

- Ensuite, la deuxième étape est d'implémenter la résolution du problème (ici en python) et de cela, on obtient les paramètres recherchés.
- Nous obtenons en sortie les paramètres recherchés. Voici un exemple d'un plot python :

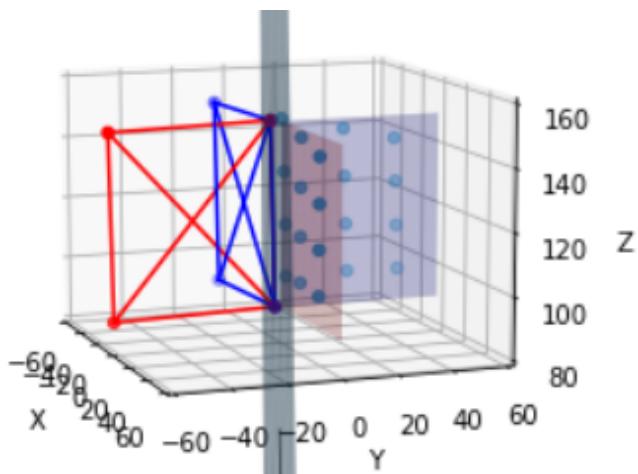


Figure 8 : Schéma où nous retrouvons les observations (points bleus), les plans des murs en gris clair, le plan de symétrie en gris foncé et les formes symétriques

Pour conclure cette partie, nous avons maintenant notre idée de comment faire pour projeter en ayant aucune déformation et dans n'importe quelle situation ; et finalement, nous utiliserons ce protocole pour régler chaque projecteur que nous utiliserons.

La partie théorique du protocole pour corriger les projections s'arrête ici, avec l'idée de donner au projecteur l'image qui apparaît sur le fameux plan symétrique et avec les moyens pour obtenir toutes les informations spatiales nécessaires pour ensuite pouvoir les faire transcrire dans Unreal Engine.

Ainsi, il nous reste la question de comment allons nous utiliser ces informations pour “gérer la projection”. Il nous reste aussi la question de comment joindre les deux corrections pour projeter dans un angle. Toutes ces questions trouveront leurs réponses dans la partie suivante sur la projection dans Unreal, le moteur de jeu vidéo choisi pour l'occasion.

2 La projection dans Unreal Engine

Voyons maintenant comment concrétiser cette idée dans Unreal.

Notre objectif est ici l'étude de ce que propose Unreal en termes de projection. Comme notre premier objectif, en connaître davantage nous apportera d'autant plus d'options, des options qui nous facilitent ensuite l'immersion.

1. Présentation rapide

a. Pourquoi un moteur de jeu vidéo ?

Avant de s'attaquer à cela, il est important de mettre au clair pourquoi nous avons besoin d'un moteur de jeu vidéo. Par exemple, posons nous ces questions :

- Comment projeter seulement l'environnement virtuel choisi ?
- sur l'entièreté de l'écran ?
- avec plusieurs projecteurs ?
- en réussissant à faire bouger nos projections en se baladant ?

Effectivement, c'est bien grâce à un moteur comme Unreal que l'on va pouvoir faire cela. C'est aussi le moyen le plus logique car notre jeu de données qui représente notre environnement comprendra des objets 3D et non de simples images, comme nous pouvons le voir lors d'une balade en street view.

De plus, comme nous l'avons vu précédemment, nous avons besoin d'un outil qui gère la projection et ses déformations. On pourrait aussi ajouter l'argument de l'ergonomie, de la fluidité et de l'optimisation des performances qui sont des critères déjà caractéristiques d'un moteur de jeu vidéo.

Maintenant, la question que l'on pourrait se poser est pourquoi Unreal Engine et non Unity par exemple, ou simplement pourquoi pas un autre ? Pour commencer, le débat s'est tout de suite tourné vers Unreal ou Unity car ces deux moteurs restent les plus connus et les plus documentés ; ce qui a été à pris en compte en sachant que j'allais utiliser un outil que je ne connaissais pas. Ensuite, le choix a été réalisé selon ces critères :

	Unreal engine	Unity
Installation	facile mais longue	facile et rapide
Utilisateur	exige un PC performant user-friendly	aucune exigence mais moins friendly
Programmeur	en C++ ou Blueprint communauté documentation	en C# communauté documentation
Projection	prise en charge native, explicit	possible, moins explicit

Figure 9 :Tableau comparatif des avantages et inconvénients de Unreal Engine et Unity

Ainsi, en faisant le bilan, comme le stage reste centré sur les projections alors, avec un ordinateur assez performant, nous n'aurions aucun problème à utiliser les outils disponibles et prévus dans Unreal. De plus, n'ayant pour l'instant reçu aucune formation en C# ou en C++, le système de Blueprint permet de rapidement s'acclimater aux objets et aux fonctions utilisables.

b. Les outils proposés pour la projection

Une dernière fois avant de rentrer dans le vif du sujet, faisons d'abord le tour des outils mis à disposition pour la projection dans Unreal. Commençons d'abord par des exemples de terrains, l'endroit où nous disposons nos objets, i.e. ce qui apparaît lors du lancement du jeu:

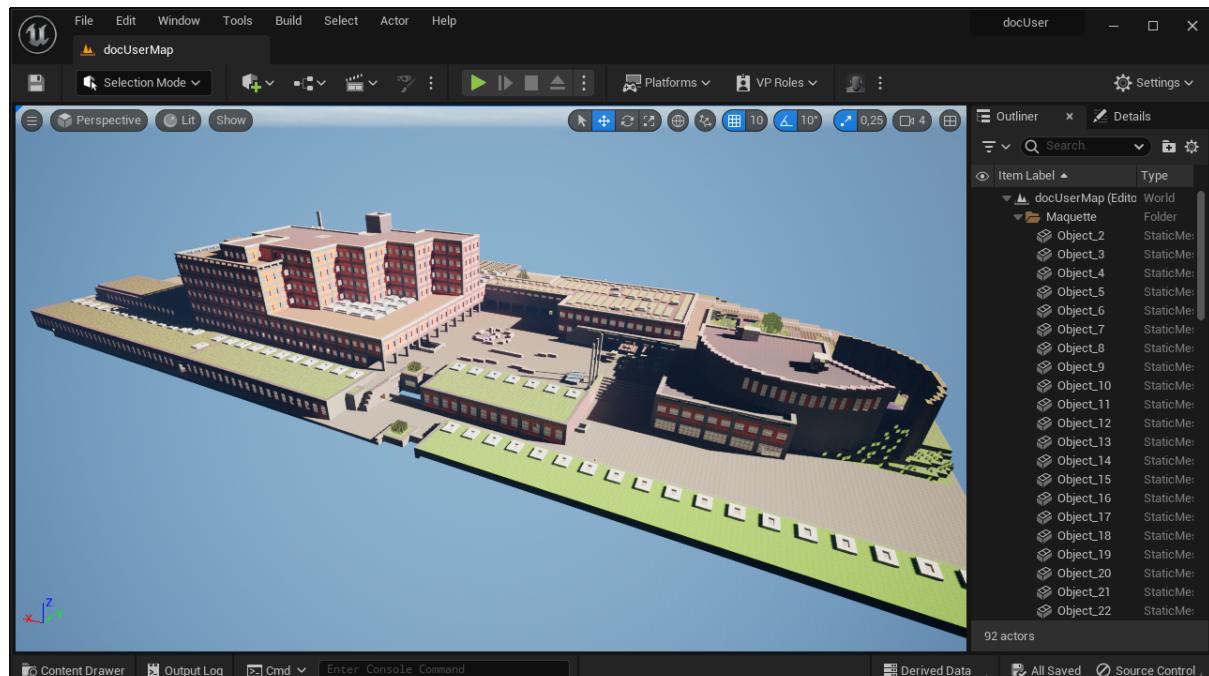


Figure 10 : Capture d'écran d'un ‘level’ d’Unreal Engine, précisément une maquette de la HEIG VD

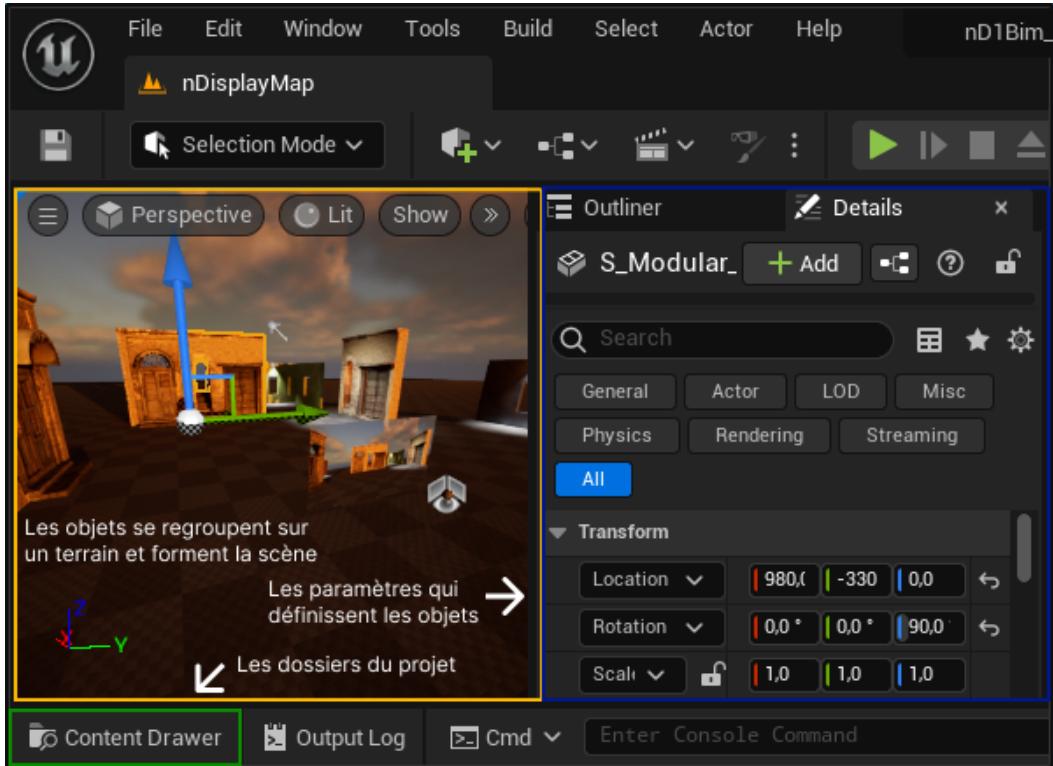


Figure 11 : Capture d'écran d'un ‘level’ présentant les différentes possibilités d’Unreal

Maintenant, pour comprendre comment les projections se traduisent dans Unreal, il est d'abord nécessaire de savoir que la conception d'un environnement virtuel est facilitée grâce à la réutilisation d'objets déjà créés. Ces objets préexistants, tels que les acteurs ou les personnages, sont classés en fonction de leurs caractéristiques spécifiques. Les acteurs peuvent être des objets statiques, des caméras ou même des sources lumineuses, tandis que les personnages sont des avatars jouables dotés de comportements et d'animations prédéfinis.

Pour chaque sorte d'objet, il y a ce genre de hiérarchie :

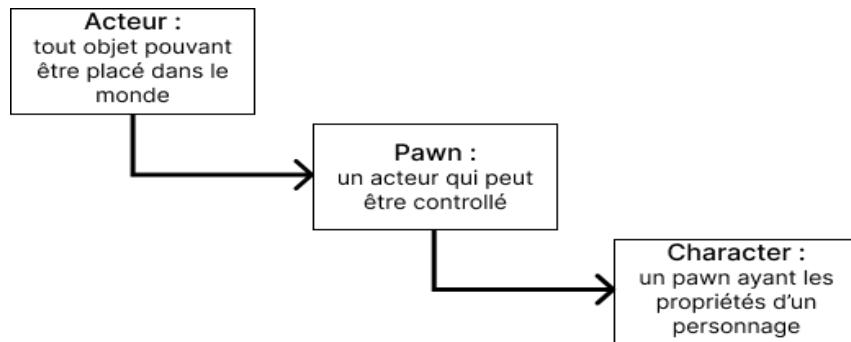


Figure 12 : Schéma présentant la hiérarchie des classes d’objets

C'est cette classification qui permet aux développeurs de tirer parti de bibliothèques existantes et de gagner du temps tout en créant leurs environnements. Dans notre cas, nous utiliserons les configurations nDisplay qui simulent la projection directement dans le moteur. Voici à quoi ressemble la configuration classique :

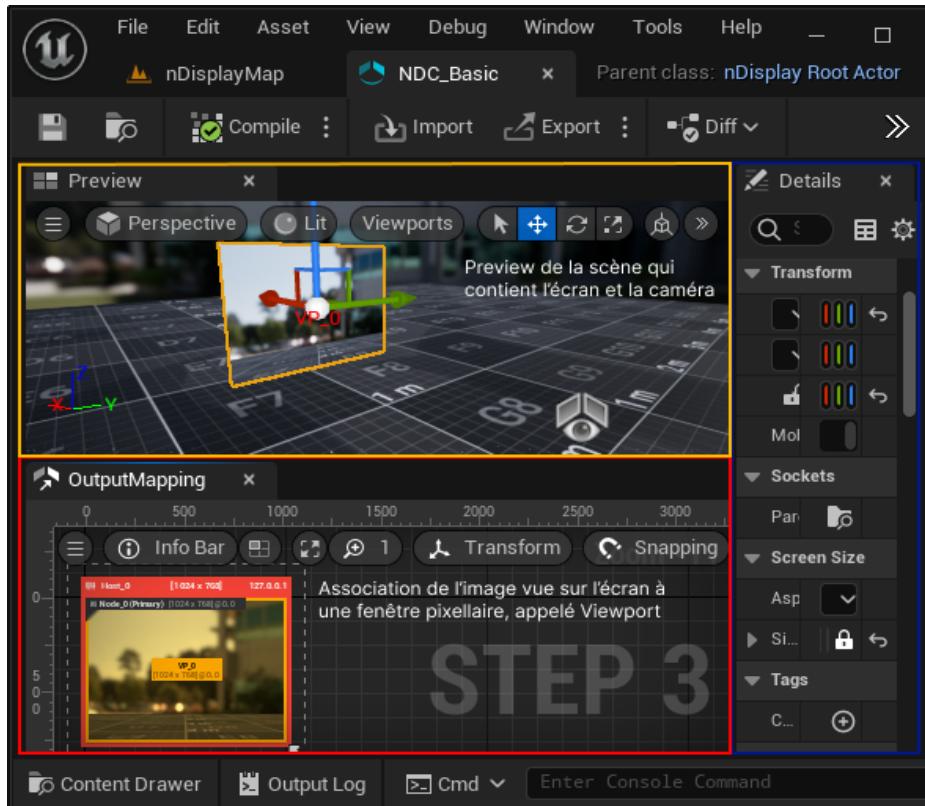


Figure 13 : Capture d'écran de la configuration de base proposée par nDisplay

Dans cette scène, nous retrouvons classiquement ce qui va être le projecteur et la surface de projection qui sont bien des objets que l'on peut transformer. Enfin, dans le cadran rouge, c'est ici qu'il est possible d'attribuer l'image vue sur l'écran depuis le point de vue de la caméra à une place précise en pixels lorsque nous “lancerons” la configuration. On appelle d'ailleurs “Viewport” la fenêtre pixellaire créée pour recevoir cette image.

Nous retrouvons ainsi ce qui va nous servir d'une part à simuler la projection sur la forme corrective et ensuite, attribuer cette image au bon projecteur.

Une fois placée sur le terrain, il ne nous reste plus qu'à “lancer” la configuration nDisplay et notre première projection pourra alors être considérée complète. En effet, nous venons d'effectuer l'attribution mais il faut maintenant un moyen pour afficher cette image. Ce moyen prévu par Unreal est un plug-in nommé Switchboard. Le voici :

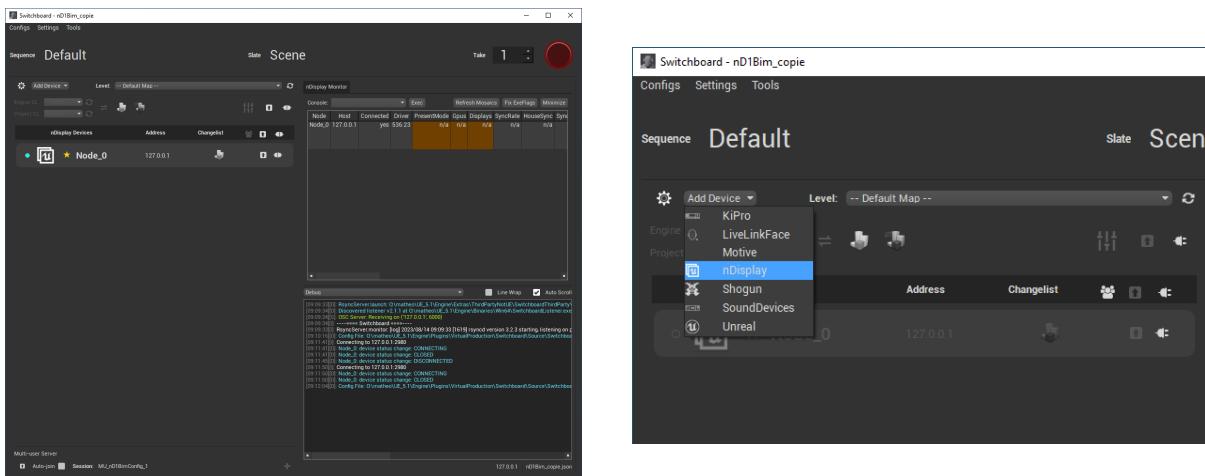


Figure 14 : Captures d'écran du plugin Switchboard

Dans les “devices” possibles, nous retrouvons notamment la possibilité de lancer notre configuration nDisplay mais aussi notre projet Unreal puisqu’effectivement, Switchboard permet aussi la création d’une session qui permet de lier la configuration au projet. Ainsi, une fois connectées à cette même session, les changements effectués dans le projet sont directement visibles en temps réel sur la projection.

C’est donc ce point clé qui est intéressant pour nous qui voulons nous déplacer dans l’environnement virtuel. Concrètement, il nous suffit d’attacher la caméra de navigation dans le projet à notre configuration nDisplay pour nous déplacer dans le jeu.

Sinon, à travers ces différentes captures d’écrans, nous pouvons d’ailleurs voir que cette session peut être rejoint par un autre ordinateur en spécifiant les adresses IP. En effet, cela ne sera pas utile dans notre cas puisque les projecteurs utilisés sont directement branchés sur le même PC et représentent alors une suite d’écrans, mais notons pour l’instant que cette option existe. Nous y reviendrons.

Gardons alors en tête que ce que nous venons de voir est l’utilisation la plus simple possible avec un seul écran, situé sur le même axe que la caméra et un seul viewport. Il faut cependant savoir que toutes les configurations sont réalisables vu que l’on peut ajouter et transformer tous ces objets.

Nous aurons à l’avenir différents viewports qui reçoivent différentes images issus de différents écrans. Ainsi, la question de la première partie sur comment joindre les projections se résout ici.

2. Intégration de la configuration des projections

a. Construction des écrans virtuels à partir des formes projetées

Maintenant que nous avons une vision d’ensemble sur comment projeter dans Unreal, nous pouvons alors passer à l’intégration de la réalité spatiale de nos projecteurs et des formes sur lesquelles nous voulons projeter.

Pour rappel, l'idée fondamentale de la correction est de donner au projecteur l'image de notre environnement qui compense la déformation de la projection. Nous avons aussi accès aux coordonnées du projecteur et de chaque coin de la forme à corriger.

Dans Unreal, l'image que l'on donne au projecteur, autrement dit, celle qui va être donnée au Viewport, provient de ce que voit une caméra coupée par un écran. Or, ces deux objets peuvent être positionnés où nous le voulons. De plus, n'importe quel objet Mesh peut servir d'écran dans cette configuration, alors nous pouvons créer, par exemple avec Blender, des plans où les coordonnées des coins sont manipulables. Ainsi, nous pouvons recréer les formes mesurées en amont, dans Blender et les importer ensuite dans Unreal.

Un exemple de plan réalisé dans Blender :

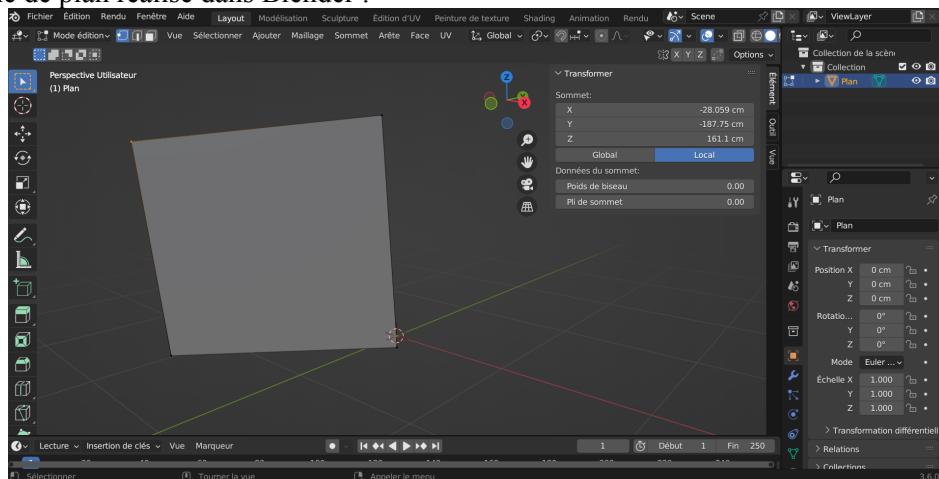


Figure 15 : Capture d'écran de la création d'un plan dans Blender

De cette manière, nous pouvons complètement matérialiser n'importe quelle configuration dans Unreal.

b. Dans la pratique de la correction - partie Unreal Engine

Finalement, il va falloir s'intéresser à comment faire pour obtenir cette image compensée.

La première idée a été de directement créer les formes correctives dans Blender et les placer dans la configuration. Avec la caméra placée à la position du projecteur, nous obtenons alors des résultats assez satisfaisants mais qui ne l'étaient plus lorsque le projecteur a été placé à l'envers pour la configuration finale.

Nous parlerons plus tard de pourquoi ce choix mais ce simple résultat a permis de mieux comprendre comment était créée l'image accueillie par l'écran. Pour vous montrer cela, voici un écran importé dans Unreal :

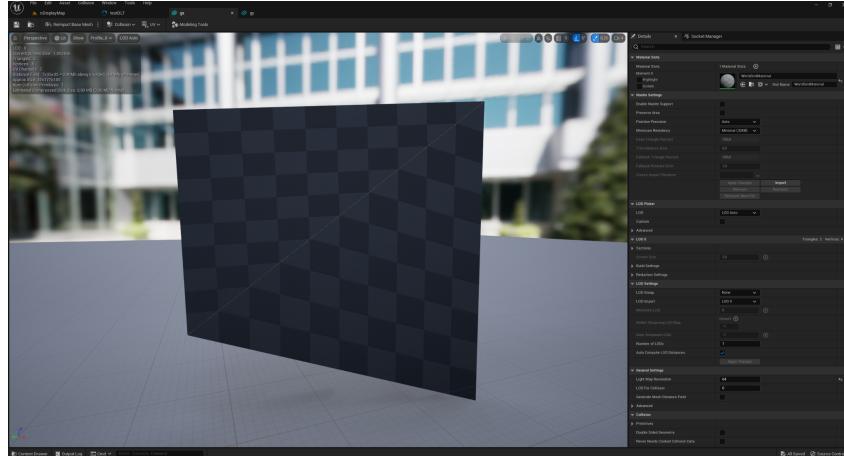


Figure 16 : Capture d'écran d'un écran importé depuis Blender

Où l'image sur l'écran une fois projetée se retrouve alors comme cela :

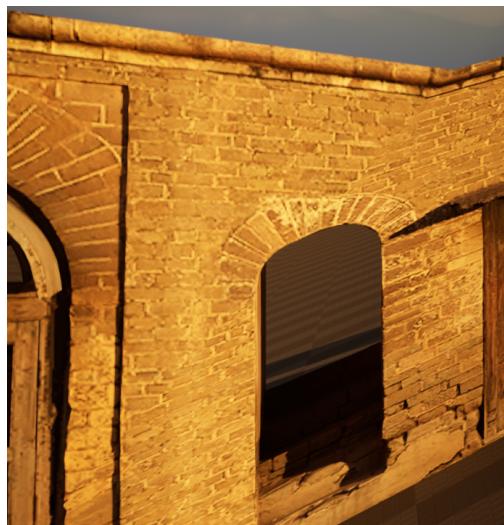


Figure 17 : Capture d'écran des conséquences de cet écran

Ce qui nous fait arriver à un concept clé de la correction : la UV Map. Comme vous pouvez le voir, cette dernière est celle qui dicte à quel endroit le pixel de l'image d'origine va se retrouver et il se trouve que naturellement, les plans sont découpés en deux triangles inégaux et les lignes droites d'un mur suivent alors ces lignes, ce qui devient la cause d'une déformation.

Pour corriger une projection, la solution alors proposée est de réunir les deux formes dans un projet Blender et de créer la UV Map unique de chaque forme selon le point de vue d'une caméra aussi placée dans le projet aux coordonnées du projecteur. De cette manière, nous retrouvons bien notre

idée de correction décrite dans la grande première partie, mais sans avoir besoin de calculer le plan de symétrie ou ce fameux plan imaginaire.

Ce qui nous donne rapidement la projection corrigée :

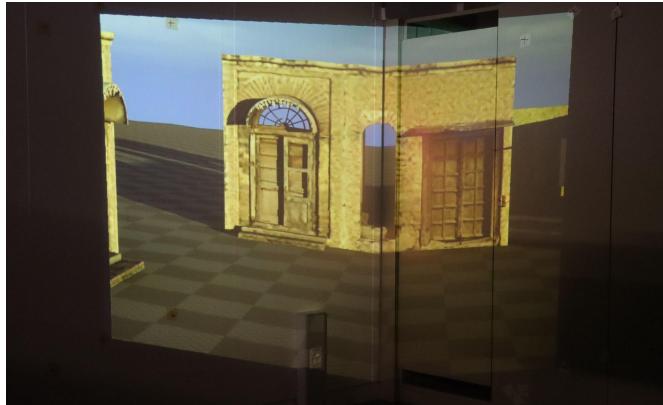


Figure 18 : Photo de la projection une fois corrigée, ne présentant plus de déformations bloquant l'immersion

Pour plus de détails sur comment réaliser ce genre d'UV Map, une documentation utilisateur est disponible en annexe qui décrit depuis le début tout le processus de correction

c. Recouvrement

Parlons maintenant d'un autre grand point qui appuie le choix de cette UV Map : le recouvrement entre les projections. En effet, il faut bien que les projections coïncident pour rendre la projection immersive.

Or, nous avons vu que l'environnement projeté par les viewports proviennent de ce que voient les caméras sur des écrans, cela implique alors que les formes des projections qui se joignent doivent alors appartenir au même plan pour contenir les mêmes objets.

Cela veut donc dire que prendre les formes physiques était le seul choix possible pour que les projections coïncident.

Voici par exemple deux projections avec un large recouvrement :

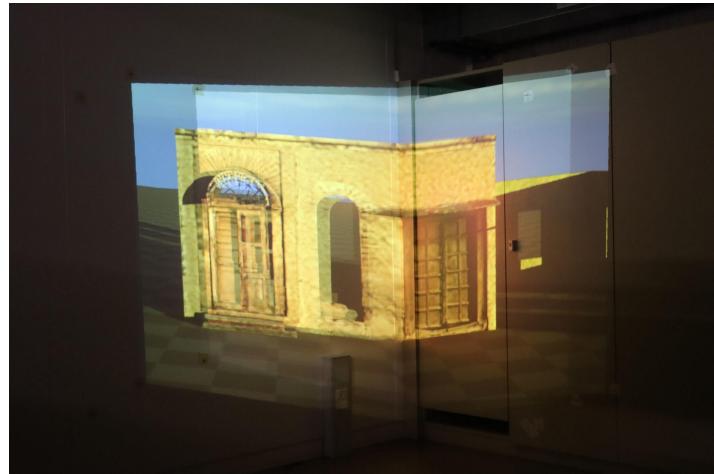


Figure 19 : Photo de deux projections avec un large recouvrement

Même si ce test a été réalisé avec une seule DLT, ce qui nuit forcément à la précision, il semble cependant qu'il y a un léger problème de parallaxe car les caméras ne sont pas situées au même endroit. Par ailleurs, les différences d'intensités lumineuses seront aussi à penser.

Pour conclure sur le recouvrement lors de la réalisation d'une configuration à plusieurs projecteurs, il est effectivement de mise d'imposer l'appartenance au même plan. Concernant le reste, dans notre cas avec le temps à disposition, il a été plus judicieux de faciliter la configuration et de penser à différents moyens pour contourner ces problèmes.

Par exemple, comme nous choisissons très précisément ce qui sera projeté, en plaçant nos projecteurs de manière intelligente, les problèmes d'intensités lumineuses peuvent être minimisés, voire invisibles.

Nous pouvons désormais conclure sur la correction que nous apportons aux projections puisqu'il nous reste alors qu'à réaliser les trois protocoles pour les trois projections. En effet, nous avons couvert d'une part sa partie théorique vue dans la première partie et sa partie réalisation grâce à Blender et Unreal dans celle-ci.

3 Mise en place finale

1. Placement des projecteurs

Maintenant que nous avons appris à projeter sans déformation et cela même sur deux murs, nous pouvons enfin commencer à imaginer la configuration finale de nos projecteurs puisque l'idée de base ressemblait à ce genre de positionnement :



Figure 20 : Photo d'une configuration test, servant d'exemple typique

À projeter à gauche avec un projecteur à droite, un au centre et un à gauche pour projeter à droite, les trois faces sont ainsi majoritairement couvertes. De plus, les projecteurs sont placés à l'envers et en hauteur car ces derniers projettent naturellement vers le haut et de cette manière, cela permet d'avoir une ligne droite horizontale qui fait la liaison entre les projections.

Il faut aussi penser à la configuration d'un point de vue réalisation, point de vue qui est aussi décrit dans la documentation utilisateur et d'un point de vue pratique, il a par exemple ici été fait en sorte que les boutons pour allumer les projecteurs soient accessibles sans les bouger. Étant en permanence en phase de test, une disposition peut durer plusieurs jours et a besoin de ce genre de détails.

Prenons maintenant en exemple la dernière configuration réalisée. Une configuration où une seule DLT a été faite, où le recouvrement n'est pas le plus intelligent mais une configuration qui a été réalisée en guise de présentation rapide de la marche à suivre. La voici :



Figure 21 : Photo de la dernière configuration réalisée

Après les coordonnées des différents coins prises, il nous reste qu'à produire nos plans à partir de ces formes, de les importer et nous avons alors la configuration nDisplay suivante, une fois placée sur le terrain :

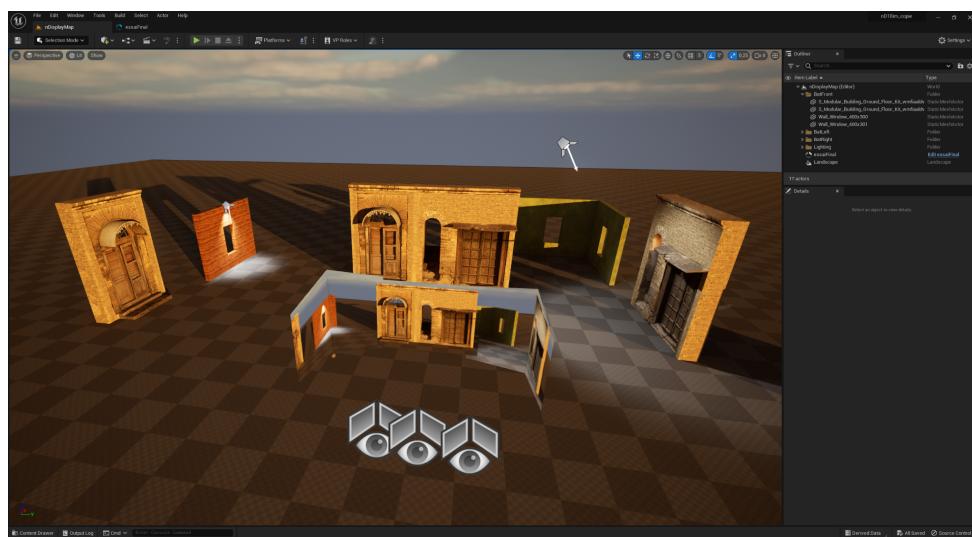


Figure 22 : Capture d'écran du level d'Unreal comprenant la configuration nDisplay des projections
Et enfin, le résultat de la projection :



Figure 23 : Photo du résultat de la projection dans cette configuration

2. Perspectives d'améliorations

Maintenant, nous allons ici discuter des résultats et ensuite de ce qui peut être amélioré.

Commençons par souligner que manquant de temps pour réaliser une configuration avec le moins de défaut possible, celle-ci a le mérite d'avoir été réalisée en environ 1 heure et semble tout de même assez satisfaisante. Il est effectivement possible de retrouver les défauts de recouvrement mentionnés plus tôt mais cela devient normal à ce niveau de précision.

Ce qui nous intéresse ici, c'est d'une part que c'est seulement le premier essai réalisé avec les UV Maps sur le groupe de projection et de l'autre que ce sont des corrections qui sont individuellement correctes et qui tendent à se joindre correctement.

Pour de meilleurs résultats, il aurait fallu commencer par fixer au plafond les projecteurs pour faire en sorte que les projections gauche et droite soient symétriques par rapport à la droite qui passe perpendiculairement par le centre du mur du fond. Ils partageraient de cette manière énormément de caractéristiques communes et il serait possible de fixer le projecteur central de telle manière à parfaitement joindre ces deux projections. Cela facilitera l'immersion et nous verrons les projections comme une seule mais le résultat en lui même sera aussi meilleur. Il va aussi de mise que toutes les DLT doivent être réalisées pour la précision des plans et projecteurs.

Ceci dit, la correction d'une projection apporte déjà de bonnes compensations mais concernant le cœur de son amélioration, les UV Maps semblent remplis de potentiels pour parfaitement afficher les projections en tant qu'ensemble puisqu'elles peuvent même être retouchées à la main dans le pire des cas. Ainsi, même si le résultat n'est pas encore parfait, les pistes sont en tout cas bien réelles.

Parlons aussi de la salle en elle-même. Comme nous avons pu le constater, les projections ont été réalisées avec une rangée d'armoires en tant que mur du fond. Cela ne nous dérange pas pour les tests réalisés ici mais des solutions comme tendre une toile blanche ou la pose de panneaux nous enlèverait ces traits visibles et les différences de relief observables.

Faisons une dernière parenthèse sur les projections que nous voulons actualiser en bougeant dans le jeu. La fluidité et la rapidité des changements entre les fenêtres d'Unreal et de la projection ne dépendent que du nombre d'images à afficher et de ce que permettent les performances de l'ordinateur. Par exemple, si nous utilisons un seul projecteur dans un angle, deux formes sont créées, ce qui implique la création de deux écrans et de deux images à calculer. Ici, les performances assurent encore la fluidité mais ce n'est plus le cas si nous demandons le calcul de 5 images comme dans la

configuration finale. La solution serait alors d'utiliser plusieurs ordinateurs et de relier leurs adresses IP avec Switchboard.

Ce que nous pourrions aussi imaginer pour la suite est de créer la salle nous même avec des écrans déplaçables. Dans cette idée, nous fixons aussi ici les trois projecteurs au plafond mais pour projeter le plus droit possible directement. Chacun regardant avec son axe optique pointé sur le face d'en face. De cette manière, les corrections à apporter seront déjà minimisées, et les jonctions entre les projections devraient se faire alors aussi plus proprement.

Une autre perspective pour l'avenir de cette salle est l'utilisation de projecteurs à courte focale, ce qui pourra simplifier les idées de réalisation de différentes configurations.

Conclusion

Passons maintenant à la conclusion. Le but premier de notre stage était, pour rappel, un prototype fonctionnel d'une salle immersive. Nous avions alors énuméré trois critères : si les corrections sont individuellement correctes, si les projections se joignent et si nous pouvons nous déplacer à l'intérieur de l'environnement virtuel.

Comme le contexte spatial de la salle l'obligeait, il a fallu trouver un protocole généralisant pour toutes les configurations, la marche à suivre pour se rapprocher de ces objectifs.

Ce que nous avons d'ailleurs fait en trouvant ce protocole semble réunir ces trois conditions ou en tout cas peut au moins être perçu comme une preuve de concept solide que cela est réalisable de cette manière. Effectivement, quelques améliorations rendraient la salle plus immersive mais nous sommes dans l'ensemble satisfaits des résultats.

Bibliographie

[1] nDisplay Quick Start for Unreal Engine.. Unreal Engine Documentation. Visité en Juin - Juillet.

<https://docs.unrealengine.com/5.1/en-US/ndisplay-quick-start-for-unreal-engine/>

[2] Switchboard Quick Start for Unreal Engine. Unreal Engine Documentation. Juin - Juillet.

<https://docs.unrealengine.com/5.1/en-US/switchboard-quick-start-for-unreal-engine/>

[3] UV Tools. Blender. Août.

<https://docs.blender.org/manual/en/latest/modeling/meshes/editing/uv.html>

[4] Plus généralement la documentation Unreal Engine. Juin - Août

<https://docs.unrealengine.com>

[4] Blender Stack Exchange. Août

<https://blender.stackexchange.com/>

[4] Introduction to UV Editor. Blender. Août.

<https://docs.blender.org/manual/en/latest/editors/uv/introduction.html>

[5] How to Sync nDisplay Nodes with Switchboard + Helix Core. Jase Lindgren, Perforce.. Juillet.

<https://www.perforce.com/blog/vcs/unreal-ndisplay-switchboard>

[6] How to make a live playable anamorphic billboard using nDisplay and Unreal Engine. Jasper Golding. Juin - Juillet. <https://www.youtube.com/watch?v=lS2wbGq1RH0>

[7] Apprendre Unreal Engine 5 de A à Z (Formation Débutants). Evans Bohl.

<https://www.youtube.com/watch?v=hn98tbztoBg>

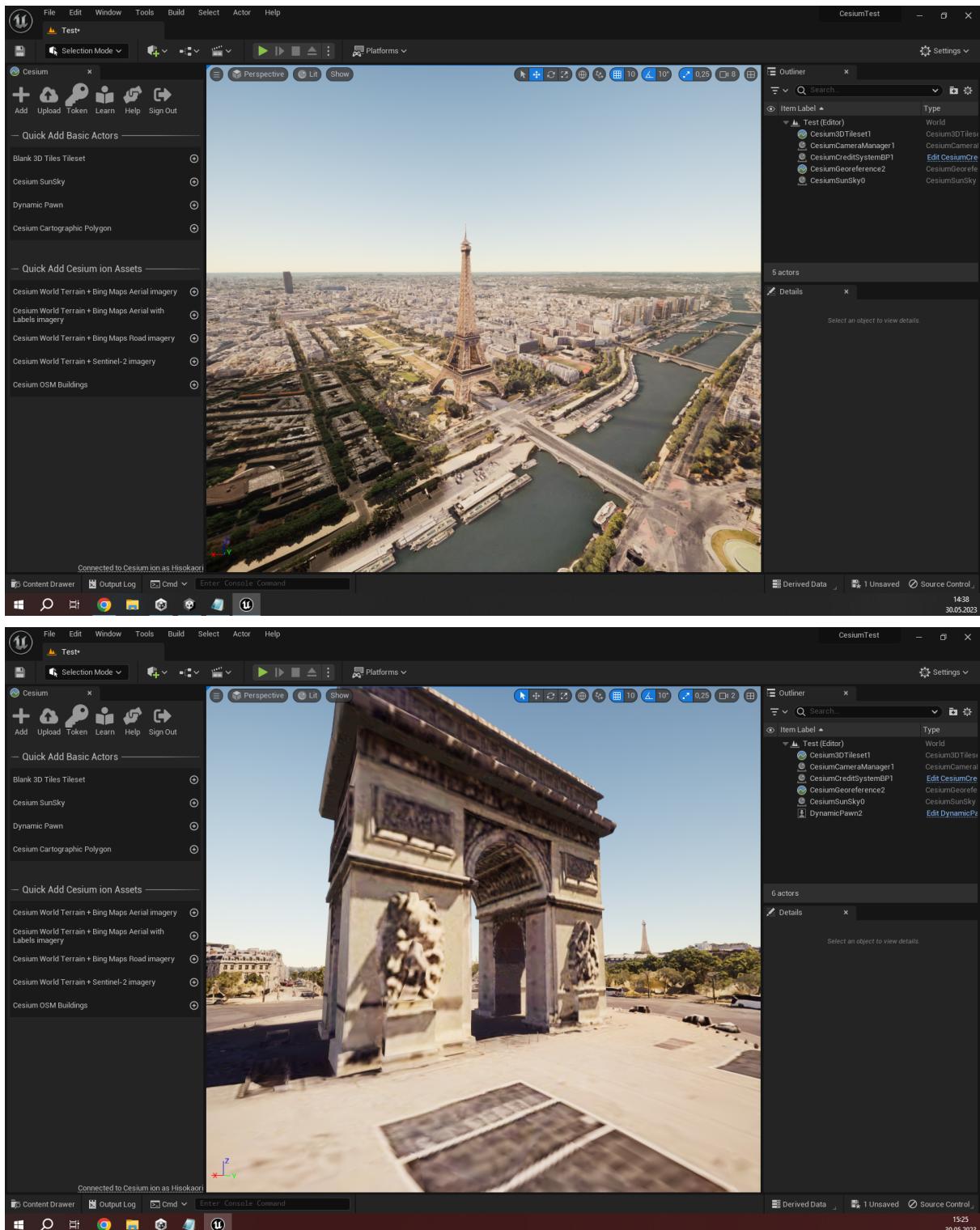
[8] E1: nDisplay Config/Switchboard Setup - In-Camera VFX Virtual Production - Vp Toolkit UE 4.27. Ian Fursa. Juillet. <https://www.youtube.com/watch?v=u5bgaze7aLw>

[9] Blender "Project from View" UV unwrapping tutorial. Août.

<https://www.youtube.com/watch?v=oHlyg6j-5Gg>

Annexe

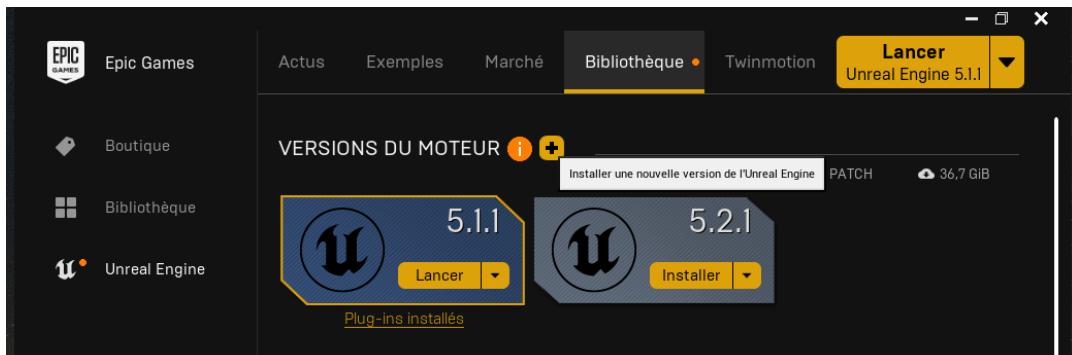
A. Le plugin Cesium for Unreal, un exemple de données 3D à visualiser



B. Documentation Utilisateur

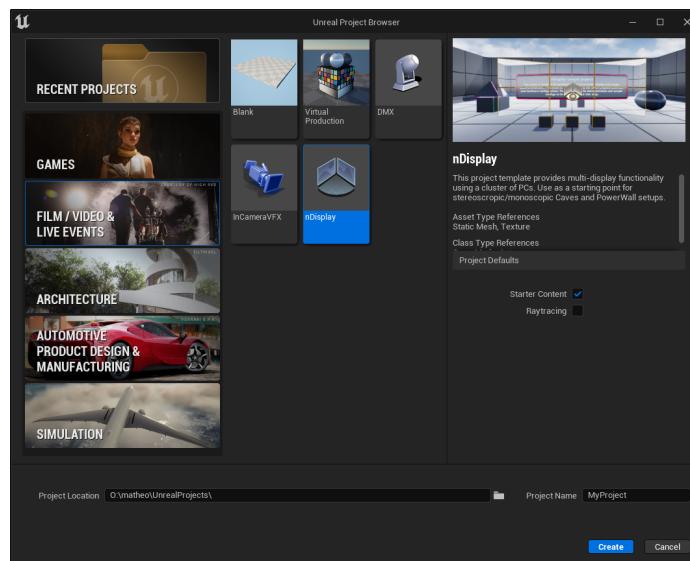
I. Installation

- Télécharger Epic Games et installer la version que vous souhaitez, ce projet a été réalisé en 5.1 mais les mêmes outils sont disponibles en 4.27 ou en 5.2.



conseil : dans les options, décocher les plateformes cibles qui ne vous intéressent pas (Android, IOS, Linux), il faut tout de même prévoir un espace de stockage conséquent pour Unreal et vos projets

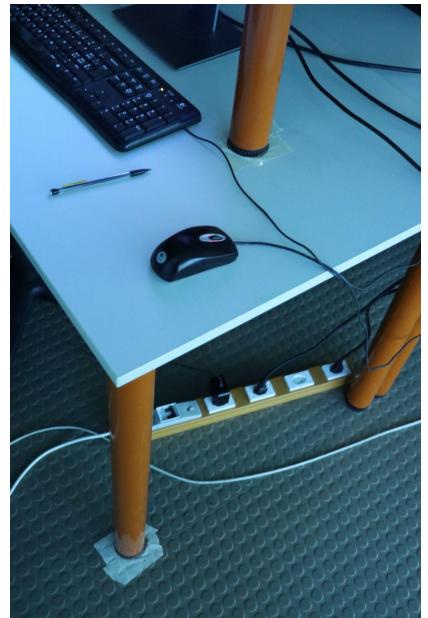
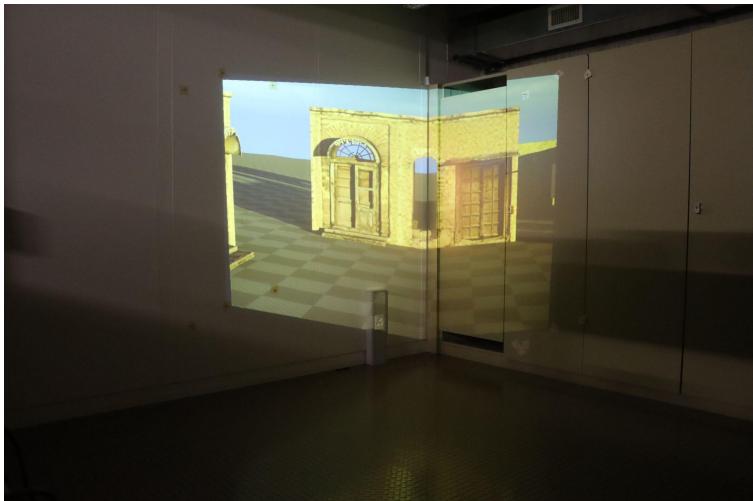
- Après le lancement, créez votre projet à l'aide du template nDisplay



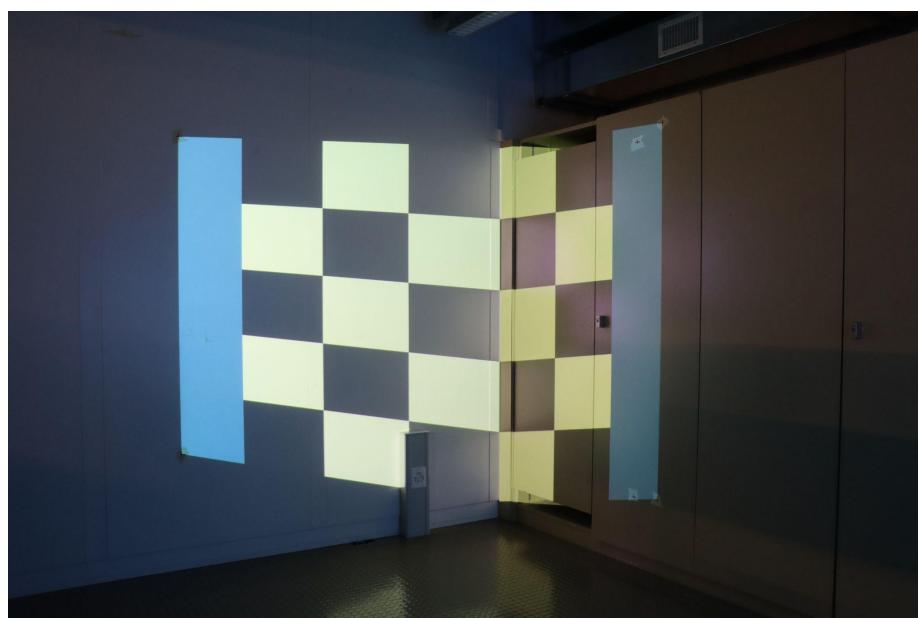
vous aurez ainsi accès aux configurations basiques, sinon il est possible d'aller dans Edit > Plugins pour activer nDisplay et Switchboard dans un projet quelconque

II. Dans la salle

- Dans le cas où vous créez une configuration qui est faite pour durer, il est primordial que vos vidéoprojecteurs ne bougent pas ou qu'ils puissent facilement être remis dans leur configuration. Sans fixation réelle, il est possible de se fabriquer des vérificateurs sur les projecteurs et sur les projections.



De plus, il est conseillé de veiller à la verticalité de chaque projection réalisée dans un angle en affichant le damier comme cela, car cette forme projetée sera plus tard séparée en deux écrans qui auront seulement en commun l'arête de l'angle du mur qui est considérée verticale :



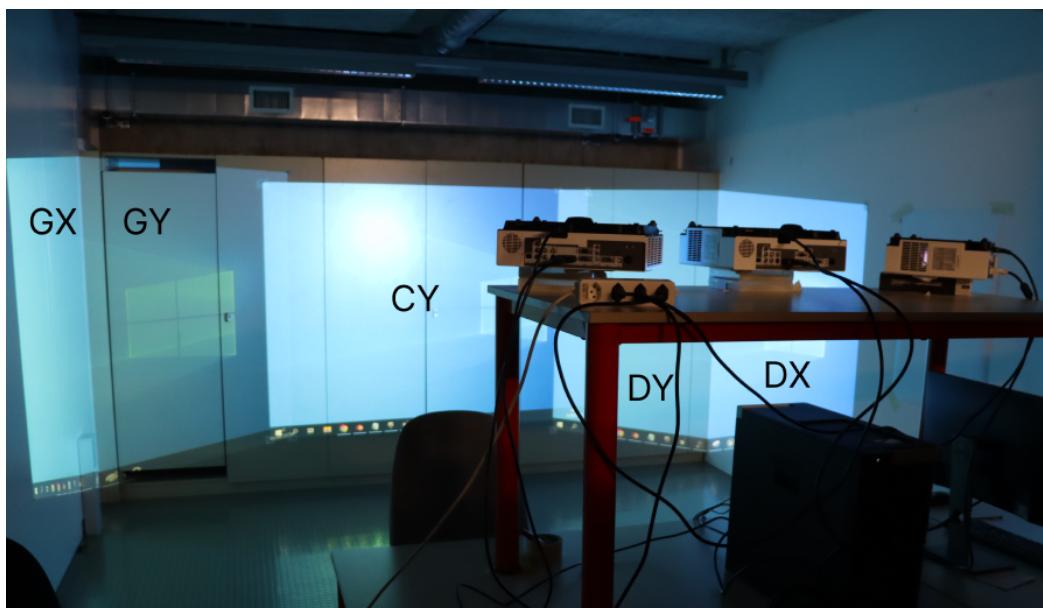
- Ensuite, il faut réaliser autant de DLT qu'il y a de projecteurs. Pour cela, il faut déjà établir le repère que l'on va suivre tout au long du processus. Je conseillerai personnellement de choisir un repère main droite où X va vers le mur. Cela car dans Unreal cela sera plus simple si nos objets sont déjà dans ce sens.

Ensuite, les observations des points en X, Y, Z et leurs correspondances en coordonnées pixels sont regroupées d'abord regroupées dans un tableau excel puis dans une bibliothèque python.

A noter que les premiers tests ont été réalisés dans un repère où X allait vers nous et ici 20 observations ont été mesurées par DLT.

Pour des résultats plus propres, il est conseillé d'utiliser les Moindres Carrés.

Prenez aussi les positions de chaque coin des formes projetées ; elles vous serviront à créer les écrans virtuels dans la configuration nDisplay. Pour les identifier et comme nous les retrouverons partout, nommons les. Par exemple comme cela :



La première lettre est la position de la projection et non du vidéoprojecteur.

La deuxième est le plan sur lequel la projection est inscrite.

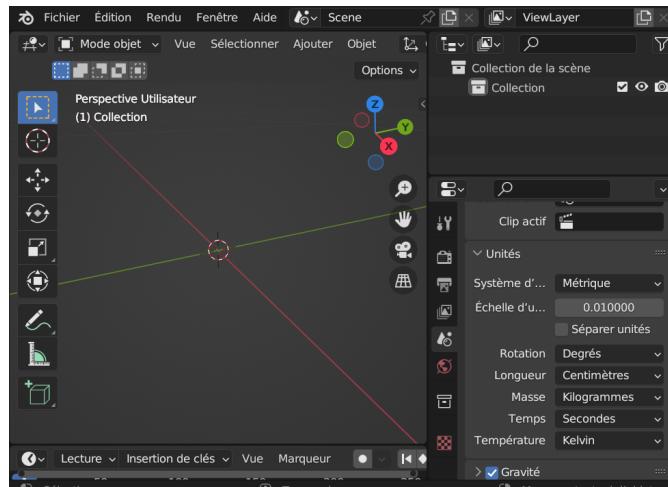
Ainsi, GX va être la forme de la projection gauche dans le plan des X.

III. Intégration de la configuration dans Blender

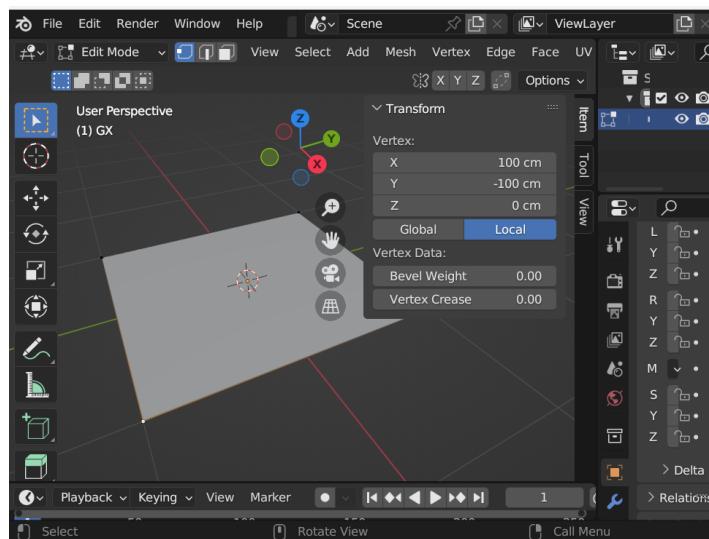
- Pour passer de ces formes aux écrans d'Unreal il faut d'abord passer par la création de plans sur Blender. Pour réaliser cela, ici en 3.6, commençons par créer un projet "Général" et retirons les éléments de la scène.

Nous pouvons déjà Ctrl + s pour sauvegarder et nommer le projet.

Changeons les unités des futurs objets en centimètres :



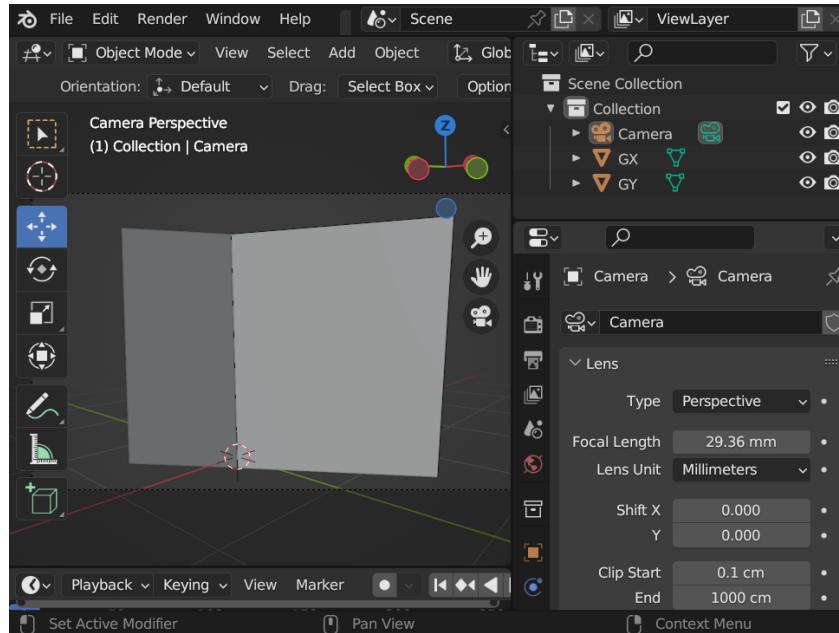
Créons le premier plan avec Add > Mesh > Plane, sélectionnons-le et passons en Edit Mode pour d'abord modifier ses sommets. Pour cela, cliquons sur un sommet, tapons n sur le clavier et rentrons les coordonnées mesurées correspondantes. Cela devrait ressembler à :



Attention : il y a un sens. La face visible du plan après sa création doit être celle qui correspond à la face visible depuis le projecteur après que l'on aura donné les coordonnées aux sommets.

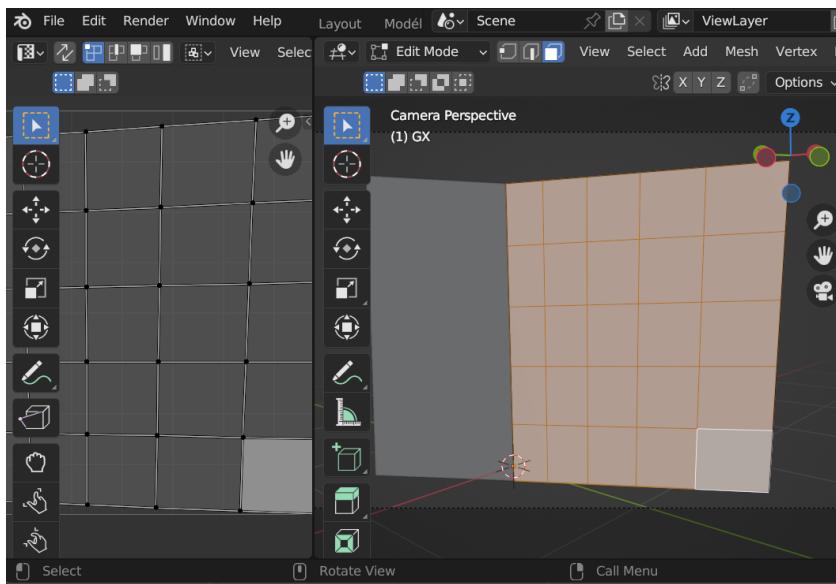
Une fois les quatres sommets placés, repassons en “Object Mode” et créons l’autre plan qui sera l’autre écran qui complète la forme et plaçons les sommets. Conseil : nommer bien vos plans.

Nous pouvons maintenant ajouter une caméra : Add > Camera et la placer aux coordonnées du projecteur en question. Plaçons nous ensuite dans la vue de celle-ci avec : View > Viewport > Caméra et réglons ses paramètres pour englober les deux plans :



Sélectionnons un des deux plans et repassons en Edit Mode et choisir la face cette fois-ci pour venir après un clic-droit sur la face utiliser l’outil subdivise comme cela :

Sans subdivise, la UV map serait séparée en deux triangles qui ne corrige pas de la même manière et c’est ce que l’on évite ici. Cliquons ensuite sur UV Editing et en sélectionnant toutes les faces, faisons un clic-droit > UV unwrap > Project from View (bounds). Cela devrait ressembler à :

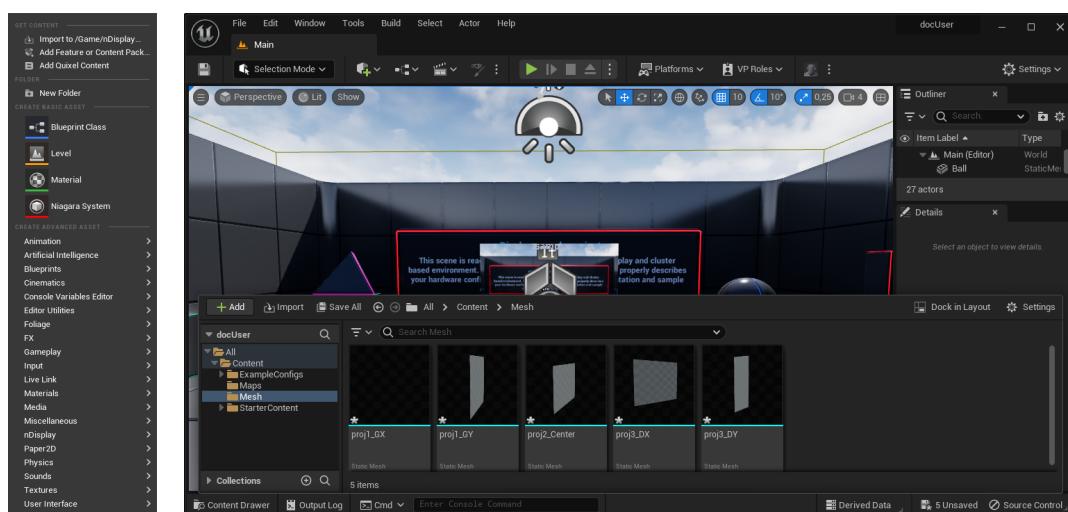


Repassons en Object Mode et faisons pareil pour l'autre plan, puis pour les autres formes. Nous pouvons d'ailleurs séparer tous les plans dans des fichiers différents, mais nous pouvons aussi tous les réunir en un fichier en faisant bien attention.

Maintenant, il ne nous reste qu'à : File > Export > FBX et nos plans pourront être utilisés dans Unreal en tant qu'écrans virtuels.

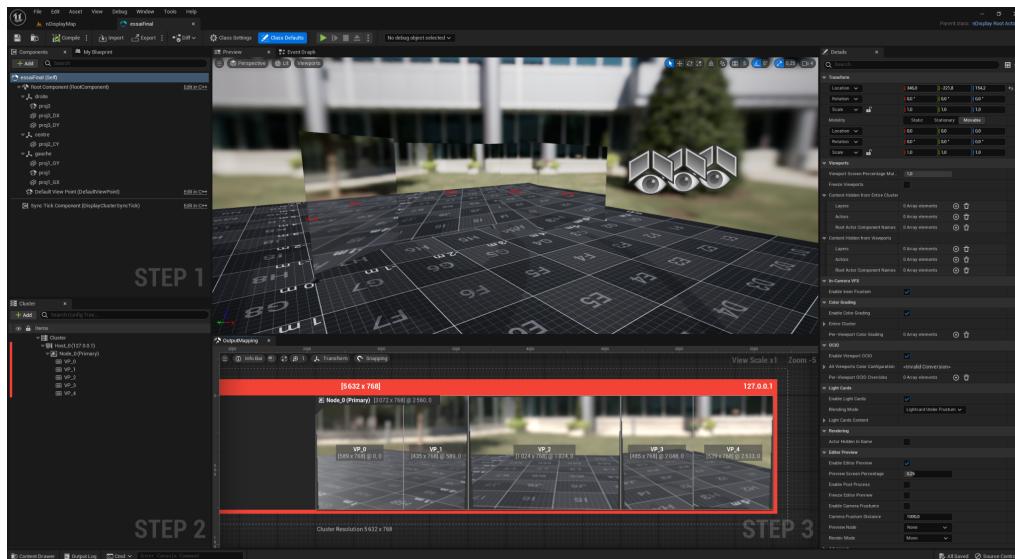
IV. Importation dans Unreal

- Repassons maintenant sur Unreal, tapons Ctrl + Espace pour ouvrir le Content Browser et commençons par importer ces plans dans un dossier "Mesh" avec clique droit > Import in.

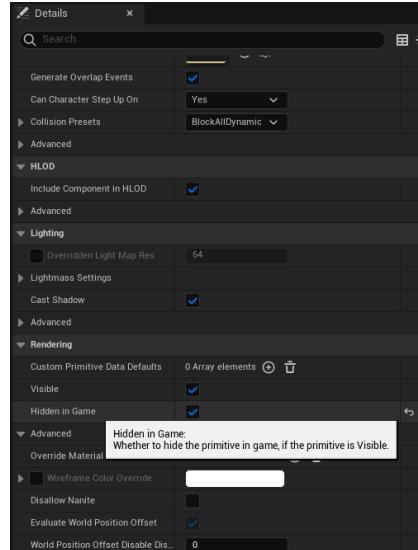


Attention à sélectionner tous ces objets et à effectuer un Ctrl + s pour bien les prendre en compte. Différence visible avec cette astérisque.

- Ensuite, dans un autre dossier qui comportera vos configurations, avec clique-droit, il est déjà possible d'en ajouter une en cliquant sur “nDisplay”. Après avoir décidé d'en créer un sans exemple, il ne reste plus qu'à placer les objets dans la scène. Nous pouvons d'ailleurs les organiser pour y voir plus clair. On peut laisser le DefaultViewPoint de côté et créer trois nDisplay origin avec le + Add qui représenteront ici nos trois projecteurs. Après les avoir placées, la scène ressemble à :



Pour chaque écran, cochez l'option “Hidden in Game” pour que les autres écrans ne soient pas visibles sur les projections.

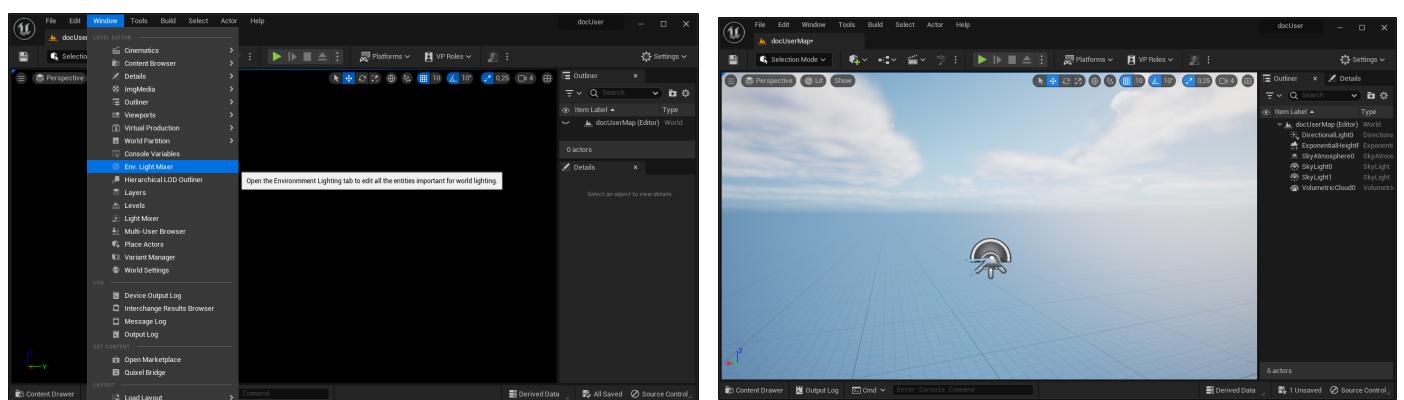


➤ Passons maintenant aux viewports. Sur cluster, faisons un clic-droit et ajoutons d'abord le cluster node qui sera la fenêtre qui contiendra tous les viewports. Ensuite, avec un nouveau clic-droit > add viewport, les écrans sont enfin associés à ce qui va être finalement projeté. Il faut aussi faire attention à bien associer le bon projecteur.

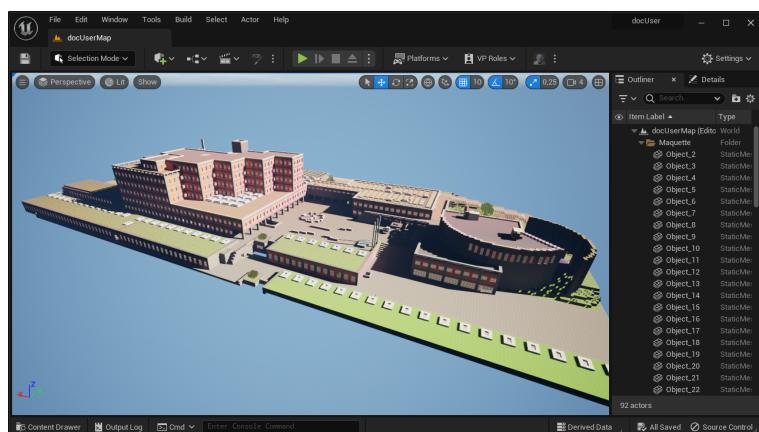
La capture d'écran de la configuration complète montre ce que donne la configuration des viewports réalisée dans mon cas.

V. Mise en place du projet

➤ Maintenant que l'on a notre configuration, il faut la placer dans notre scène pour montrer les données 3D souhaitées ainsi, pour l'exemple, créons un nouveau terrain "Empty" dans le dossier Maps pour voir comment se fait une scène avec clique droit > Terrain ou Level. Ensuite, pour faire simple, allons dans Window > Env Light et cliquons sur toutes ces options qui nous donneront un ciel et les lumières naturelles du projet.



De la même manière que nos plans, nous pouvons maintenant importer nos différents objets et placer les Mesh sur le terrain :



VI. Lancer la configuration

- Pour faire afficher ce que voient les écrans dans cette scène, il faut maintenant utiliser Switchboard. Pour cela, cliquons sur ce logo et lançons d'abord le Switchboard Listener et ensuite Switchboard.



Commençons par ajouter une “New Config” et rentrons simplement le chemin du fichier Uproject et nous pouvons continuer avec add device > nDisplay > chercher la configuration.

Node	Host	Connected	Driver	PresentMode	Gpus	Displays	SyncRate	HouseSync	Syn
Node_0	127.0.0.1	yes	536.23	n/a	n/a	n/a	n/a	n/a	n/a

Comme le but n'est pas ici de lier le jeu à la projection, décochons la auto-toggle multi-user session et si les adresses IP sont les bonnes, nous pouvons cliquer sur la petite prise et lancer ensuite la projection. Les ports peuvent aussi être vérifiés si rien ne s'affiche après le lancement. Si une fenêtre Unreal se lance mais que le résultat est noir, cela peut venir du fait que chaque élément doit bien être sauvegardé avant, notamment les plans venant de Blender qui nécessitent un Ctrl + s.

D'ailleurs, si rien ne change entre deux projections, vérifier que le terrain est bien sauvegardé lui aussi.

Finalement, après avoir placé la configuration dans le terrain et un certain temps qui dépend du nombre d'écrans à calculer et de la puissance de votre ordinateur, nous obtenons :

