La projection dans Unreal

Maintenant que les projections sont comprises, nous pouvons désormais voir comment cela se concrétise dans Unreal.

Notre objectif ici est alors 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, comme la suite sur la correction des déformations d’une projection ou la possibilité de voyager dans l'environnement qui sont des éléments essentiels de l’immersion de la salle.

1. Présentation rapide d’Unreal
2. Pourquoi un moteur de jeu vidéo ?

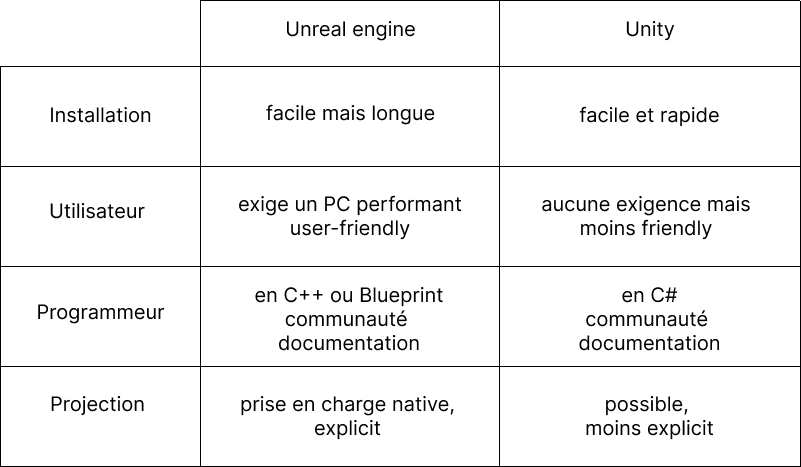
Avant de s’attaquer à cela, il est important de mettre au clair pourquoi nous avons besoin d’un moteur de jeu vidéo. Par exemple, posons nous ces questions :

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

C’est, en tout cas, bien grâce à un moteur comme Unreal que l’on va pouvoir faire cela. On pourrait même se dire que l’on est obligé d’en utiliser un car notre jeu de données qui représente notre environnement sera forcément des objets 3D et non de simples images, comme une ballade en street view.

De plus, comme nous l’avons vu dans la partie précédente, 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 ; d’où le besoin d’en utiliser un.

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 avant. Ensuite, le choix a été réalisé selon ces critères :



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.

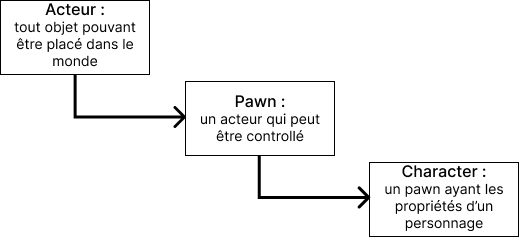
1. Les outils nécessaires à la projection ou comment projeter ?

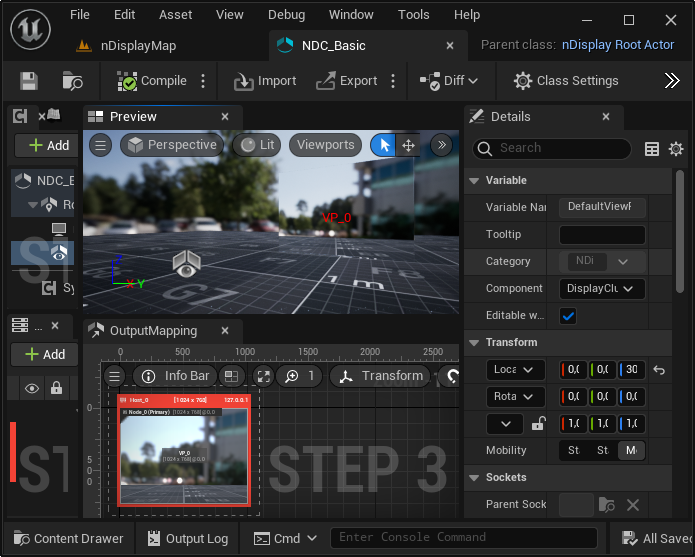
Avant de rentrer dans le vif du sujet, regardons d’abord à quoi ressemble une scène dans Unreal et ce que l’on a à disposition : 

Maintenant, la partie précédente sur les projections nous a montré que nous devons donner au vidéoprojecteur l’image de notre environnement comme si elle était déjà projetée sur la forme imaginaire qui compense le contexte spatial.

Pour comprendre comment cela se traduit 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 alors ce genre de hiérarchie :



Cette classification 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 : 

Dans la scène de la configuration, nous retrouvons classiquement ce qui va être le vidéoprojecteur 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.

Si l’on revient à notre besoin, nous retrouvons bien ce qui va nous servir d’une part à simuler la projection sur la forme corrective et ensuite, attribuer cette image au bon vidéoprojecteur.

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 :

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 alors 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 vidéoprojecteurs utilisés sont directement branchés sur le même PC et représentent alors une suite d’écrans, mais notons pour l’instant que cette option existe.

Gardons 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 ces objets. Comme par exemple celle-ci qui fait partie des exemples directement utilisables :

Nous pouvons d’ailleurs observer les 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. Nous pouvons alors noter que :

* si l’on projette sur un mur, nous aurons alors un seul viewport pour le vidéoprojecteur
* par contre, si l’on projette sur deux murs, nous pouvons diviser la fenêtre du vidéoprojecteur en deux viewports qui prendront leurs images depuis deux écrans différents

1. Intégration de nos plans dans Unreal

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

Pour rappel, suite à la DLT, nous avons désormais accès aux coordonnées de chaque coin de la forme corrective. Or, cette dernière doit alors être matérialisée dans la configuration nDisplay pour servir d’écran et ainsi pour parfaitement imiter la réalité. Petite parenthèse si nous projetons dans un angle, la projection est alors coupée en 2 formes, sur deux plans différents, mais qui partagent 2 mêmes coins :

Ainsi, la première étape est la création et l’importation de cette forme, ce que nous ferons grâce à Blender, un logiciel libre de modélisation, d’animation par ordinateur et de rendu en 3D. Pour cela, nous créerons un plan comme celui-ci que nous pouvons déformer :

Plusieurs choses à faire attention :

* la face visible ici devra être la même que celle visible depuis la caméra dans Unreal. Ainsi, il faut faire attention lorsque l’on attribue une coordonnées d’un coin de la forme en question à un coin du plan dans Blender.
* le repère de Blender par rapport au repère de la salle

Une fois le plan fini cela ressemble à cela :