Projection dans un angle

Objectif : trouver un protocole pour correctement projeter dans un angle ou autrement dit, comment projeter dans n’importe quelle situation et parvenir à obtenir le même résultat que si l’on projetait centralement sur un seul plan.

// Photos de la configuration la plus basique à laquelle on veut se ramener, une dans la vraie vie, une dans Unreal + partie plutôt en amont qui explique comment on affiche le jeu sur les écrans et configuration basique

Problématique : La mise en place des vidéoprojecteurs et la configuration de notre salle donnent lieu à différentes contraintes, des contraintes qui varient alors purement selon l’endroit. Par exemple, si l’on veut réaliser une projection plane mais non centrale (non orthogonalité entre plan du mur et la projection du vidéoprojecteur, voir image), on imagine et voit bien que cette configuration déformera le rendu.

// Photos de la config physique

Ainsi, comme l’adaptabilité et la reproductivité sont des critères importants pour la réalisation de cette salle, il est primordial de devoir s’affranchir du contexte dans lequel on projette.

On garde à l’idée que faciliter la mise en place de la salle va de soi mais qu’il faudra toujours savoir comment faire même dans des cas extrêmes pour réussir à projeter dans n’importe quel contexte. D’où l’idée de corriger une projection faite avec un vidéoprojecteur placé approximativement à 45 degrés entre deux murs formant un angle droit, ce que l’on a considéré être comme la configuration extrême et la plus visible pour tester et valider notre protocole.

Voyons maintenant à quoi ressemblerait une projection basique mais dans cette configuration :

**I - Introduction au protocole**

Commençons en remarquant le changement le plus frappant : la déformation des lignes droites en deux lignes bien distinctes sur les deux plans des murs formant le coin. En effet, ce sont elles qui vont nous donner des indices sur la correction à appliquer pour retrouver nos lignes droites même si l’on projette dans un angle. On imagine que si l’on trouve la raison de cette déformation, on comprendra alors ce qui est à corriger mathématiquement et donc comment le faire.

Dans cette optique et pour visualiser ce problème, il faut d’abord comprendre ce que l’on est en train de réaliser. Concrètement, on a utilisé la configuration nDisplay la plus basique d’Unreal, c'est-à-dire un plan et sa caméra puis on a physiquement placé le vidéoprojecteur dans l’angle. Présenté comme cela, avec cette configuration dans un coin, c’est comme si l’on avait en fait deux plans bien différents qui ont la même caméra et qui devront être rendus grâce à un seul vidéoprojecteur. Il semblerait alors que l’on puisse séparer les deux plans en deux cas distincts puisque tant que l’on arrive à bien les faire se rejoindre avec les viewports, alors leur réunion une fois corrigés devrait être correcte.

On peut maintenant considérer que projeter correctement dans un coin, c’est comme projeter sur deux plans bien distincts mais collés.

Isolons alors un des deux murs et imaginons comment le plan qu’il représente est placé et orienté par rapport à notre vidéoprojecteur ; il y a d’abord une orientation lié au fait que l’on projette vers un angle (pas centralement) mais il semble aussi y avoir une légère inclinaison vu la forme créée sur le mur et vu que les lignes semblent “remontées” lorsque l’on s’éloigne du vidéoprojecteur. En somme, le vidéoprojecteur et ses paramètres semblent alors cacher la clé du problème.

Comme on pouvait s’y attendre, c’est où et comment on place notre vidéoprojecteur qui définit ce qui va être déformé, mais à un point où connaître l’orientation globale de notre projecteur dans l’espace par rapport aux plans physiques des murs va nous montrer comment compenser ces paramètres.

Pour résumer la théorie, on doit alors faire en sorte que la combinaison de notre configuration physique et notre configuration dans Unreal doit se compenser pour former à nouveau nos lignes droites (dans la matrice rotation du vidéoprojecteur). Ainsi, si l’on crée un nouveau plan qui serait le symétrique du plan du mur par rapport au plan de rotation du vidéoprojecteur, et que l’on associerait celui-ci à la fenêtre du viewport qui va avec, alors on est censé atteindre notre objectif.

Ainsi, la première étape est d’obtenir les paramètres d’orientation du vidéoprojecteur par rapport aux plans physiques du mur, autrement dit il faut la matrice de rotation de la projection dans un repère où les plans du mur sont aussi consignés. La position du vidéoprojecteur dans ce repère sera aussi la bienvenue.

Dans cette optique, on se propose de réaliser une Direct Linear Transformation, un algorithme qui nous permettra de retrouver ces informations.

Rapidement, la DLT est un algorithme qui permet de relier les coordonnées des pixels dans une image aux coordonnées spatiales correspondantes, en utilisant l'équation de colinéarité. (\* maths ?)

Ensuite, en estimant les paramètres de transformation à partir d'un ensemble de correspondances connues, la DLT permet de retrouver nombre de paramètres et notamment ceux recherchés. Voici les grandes étapes de notre protocole :

* La première chose à faire est de régler physiquement le vidéoprojecteur à l’aide des vis pour que la projection soit au moins verticalement correcte. Cela n’est pas difficile et aidera au résultat puisque l’on corrige plan par plan et qu’on attribuera un plan à une place dans la fenêtre du rendu ; et si la projection n’est pas droite alors un plan risque forcément d'empiéter sur l’autre.

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

pour que nos deux plans (murs) soient visibles et atteignables facilement. Comme cela, on peut supposer que nos plans sont bien orthogonaux, ce qui implique par exemple que chaque position sur le plan (X,Z) a alors forcément un Y = 0.

* Sinon, concernant la DLT, la première étape a été de lier les coordonnées en pixel (x,y) de la projection avec leurs positions (X,Y,Z) dans l’espace. Pour ce faire, j’ai d’abord affiché dans l’angle et 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 me restait alors qu’à mesurer les coordonnées spatiales correspondantes. Voici la configuration de la salle et à quoi cela ressemble les points dans le tableau :
* Ensuite, passons aux calculs de l’algorithme implémentés ici en python. Une fois ce tableau passé sous forme de bibliothèque, il a d’abord