Rapport Mini-projet de modélisation géométrique : Trajectoire de caméra sous Unity

Younes Boudili
Benjamin Coupry

Département Sciences du Numérique - Filière Image et Multimédia
2020-2021

Table des matières

1 Introduction **3**

2 Différentes approches **3**

3 Rotations **3**

4 Interpolation Lagrange **3**

5 Approximation Bézier **4**

6 Splines fermées **5**

7 Utilisation **5**

1 Introduction

Nous avons choisi d'étudier les différentes méthodes d'interpolation et d'approximation de courbes dans l'objectif de modéliser des trajectoires de caméra. L'application de tels algorithmes peut s'avérer utile pour le rendu de cinématiques tout en demandant un travail minime pour la définition des trajectoires (placement de quelques points de contrôles)

2 Différentes approches

Nous avons approché le problème de deux manières différentes.

La première approche consiste a établir les équations temporelles des courbes : en effet, nous avons pu écrire des fonctions qui pour un temps t quelconque nous renvoient un vecteur position et un quaternion indiquant la rotation de la caméra. Cette approche présente l'avantage d'être très souple et modulable : il est aisé de modifier en temps réel la trajectoire, de modifier la position des points de contrôle, etc. De plus, l'utilisateur pourrait « manipuler » le temps en accélérant, revenant en arrière, etc. En revanche, cette méthode est très consommatrice en performance dans la mesure où la position doit être calculée chaque frame.
Voir `TrajectoireOld.cs`

La deuxième approche consiste à pré-calculer la position de la caméra pour un échantillonnage fixe défini. En termes de calculs, les points ne sont calculés qu'une seule fois, ce qui permet de faire tourner la simulation en temps constant quel que soit la machine. En revanche, là où l'approche précédente pourrait trouver des applications dans le jeu vidéo et la simulation en temps réel, cette approche ne peut que s'appliquer dans le cas d'une cinématique où la trajectoire est fixée et définie à l'avance. Néanmoins, elle présente le réel avantage de permettre la mise en place de schémas de subdivision de courbes, ce qui permet un calcul bien plus optimisé.
Voir `Trajectoire.cs` (version finale)

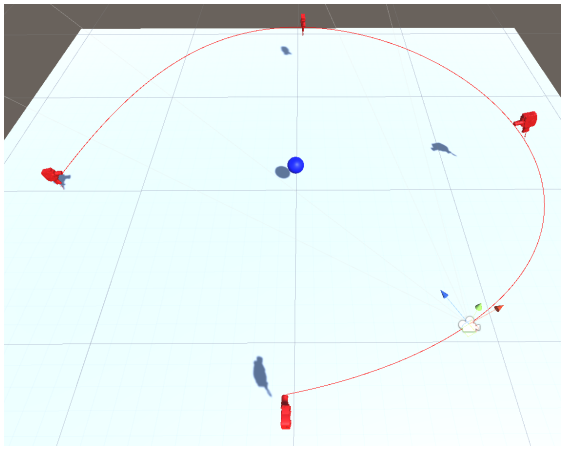
3 Rotations

Les rotations sont interpolées avec une interpolation sphérique entre deux points successifs. Dans le cas des splines, les rotations sont calculées par une approximation de spline au même titre que les positions.

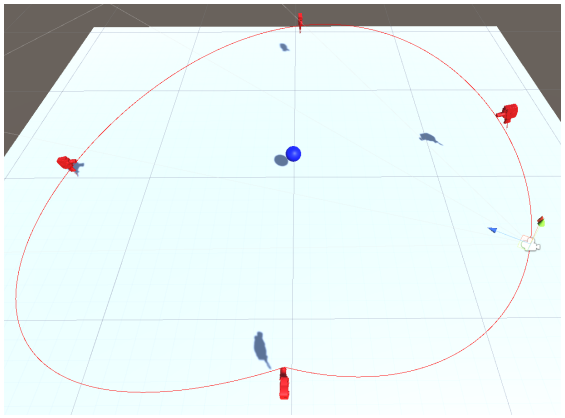
4 Interpolation Lagrange

Pour l'interpolation nous utilisons simplement une paramétrisation temporelle régulière entre les points. Nous avons implémenté l'algorithme de Neville, ainsi qu'une implémentation naïve utilisant l'expression des polynômes de Bernstein. Pour des raisons de performances, nous avons décidé de garder l'algorithme de Neville, même si l'algorithme naïf permet une bonne compréhension de la mécanique générale de l'interpolation.

Le choix d'interpoler les points implique l'utilisation de polynômes de très haut degré quand on utilise un nombre conséquent de points de contrôle. Ainsi, la courbe se met à adopter un comportement oscillatoire désagréable si l'on place trop de points. Malgré tout, si l'on doit passer exactement par certains points, il s'agit de la seule méthode valable.



(a) Lagrange appliqué à un polygone ouvert

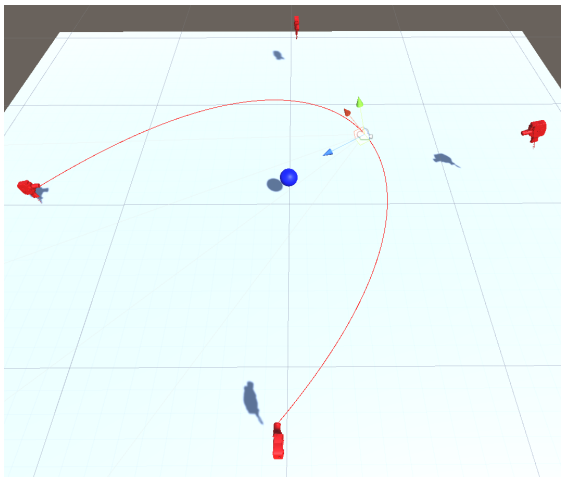


(b) Lagrange appliqué à un polygone ouvert

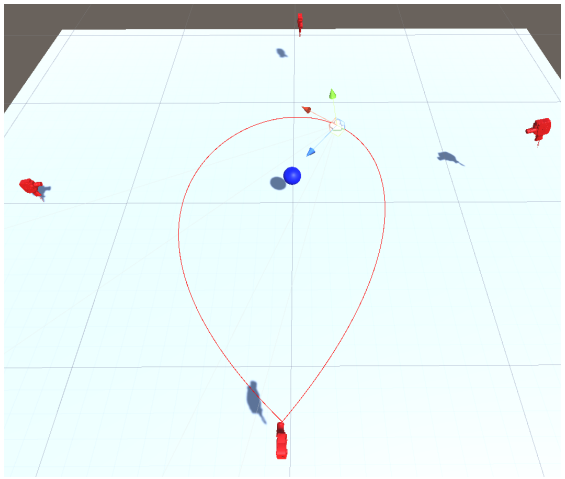
5 Approximation Bézier

Pour l'approximation de trajectoire par les courbes de Bézier, nous utilisons l'algorithme de DeCasteljau.

La méthode est particulièrement adaptée quand on souhaite obtenir une trajectoire entre deux points précis (le premier et le dernier) mais que l'on souhaite introduire une certaine dynamique dans la trajectoire. De plus, les rotations étant elles interpolées, on peut parfaitement contrôler le champ de vision au cours de la trajectoire.



(a) Bézier appliqué à un polygone ouvert



(b) Bézier appliqué à un polygone ouvert

6 Splines fermées

Enfin, nous avons implémenté la méthode la plus polyvalente, l'approximation par spline. Elle permet de conserver une bonne régularité tout en restant proche des points de contrôle. Plus le degré est élevé, plus la courbe s'éloigne des points de contrôle. Cette méthode est vraiment adaptée pour les longues trajectoires complexes. Son implémentation est celle de la subdivision de splines. Spécifiquement pour cet algorithme, les rotations sont calculées itérativement par subdivision sur un modèle identique à la position.

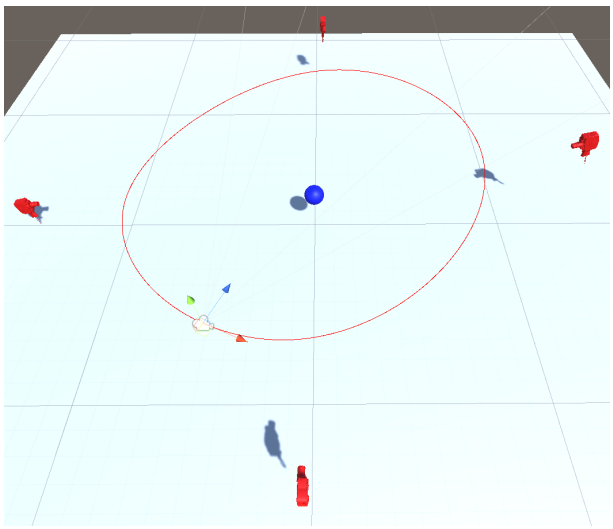


FIGURE 3 – Exemple de Spline

7 Utilisation

Pour utiliser notre solution, il faut d'abord placer des `CameraPos`, situées dans le dossier `Prefabs` du projet, et les orienter pour indiquer les endroits de passage de la caméra. Il faut préciser ensuite l'ordre de passage de chaque position, voir le champ `Ordre Passage` de l'inspecteur. Il y a une possibilité de cacher ces objets au niveau du rendu final en cochant la case `Hide`.
Après avoir attaché le script `Trajectoire.cs` à la caméra et avoir configuré les paramètres, il suffit de lancer l'application (`Play`).
Dans l'éditeur, il faut s'assurer que le bouton `Gizmos` est bien coché pour voir la trajectoire suivie par la caméra.