Binetti Aurélia

Gouin Alexandre

Souche Lucile

Info 5 – RéVA

Rapport PFE

Réalisation d’un jeu de Wire Loop en RV

Polytech Marseille Encadrants : Sébastien Mavromatis

2016-2017 & Peter Banton

# Sommaire

[Sommaire 3](#_Toc504492320)

[Introduction 4](#_Toc504492321)

[Contexte 4](#_Toc504492322)

[Présentation du sujet 4](#_Toc504492323)

[Organisation du travail 5](#_Toc504492324)

[Matériel utilisé 5](#_Toc504492325)

[Répartition des tâches 5](#_Toc504492326)

[Difficultés rencontrées et solutions apportées 6](#_Toc504492327)

[Environnement virtuel 6](#_Toc504492328)

[Génération de la courbe 6](#_Toc504492329)

[Les différentes classes 6](#_Toc504492330)

[Algorithme de génération de courbe 7](#_Toc504492331)

[Résultats 8](#_Toc504492332)

[Ouverture & conclusion 9](#_Toc504492333)

[Annexes 10](#_Toc504492334)

[Code source : 10](#_Toc504492335)

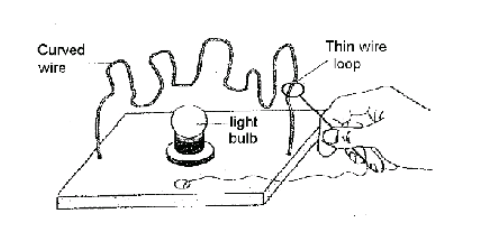
# Introduction

## Contexte

Ce projet s’inscrit dans l’enseignement de l’école d’ingénieur Polytech Marseille. Il s’agit d’un projet donné aux étudiants l’option RéVA (Réalité virtuelle et augmentée) du département INFO (Informatique) de l’école.

## Présentation du sujet

L’objectif de ce projet est de réaliser un jeu de Wire Loop en réalité virtuelle, avec comme interface un casque de réalité virtuelle et un capteur magnétique de position/orientation qui servira à contrôler l’anneau.



Wire Loop Game

Le jeu montré ci-dessus consiste en un fil de fer (wire) autour duquel se trouve une boucle de fer (loop). L’objet du jeu est de faire passer la loop le long du wire sans le toucher. Chaque fois que la loop touche le wire, c’est une erreur.

La réalisation de ce projet a été découpé en deux partie, attribués à deux groupes différents. Nos objectifs sont, pour notre groupe :

* Permettre l’utilisation du casque de RV
* Réalisation de l’environnement 3D qui facilitera l’immersion du joueur
* Génération de la courbe (le fil de fer) de manière aléatoire, avec plusieurs niveaux de difficultés possibles.

Les autres tâches nécessaires à la finalisation du jeu sont réalisées par l’autre groupe. Cependant nous travaillerons ensemble afin de combiner nos travaux et créer un seul jeu au final.

Matériel à notre disposition :

* Un Oculus Rift (casque de réalité virtuelle)
* L’autre groupe dispose d’un Polhemus Patriot (capteur magnétique de position/orientation)

# Organisation du travail

Le travail à réaliser est divisé en deux sous partie, chacune attribuée à un groupe. Ce rapport ne traite que du travail réalisé par notre groupe.

## Matériel utilisé

Nous avons choisi d’utiliser un dépôt Git afin de synchroniser notre travail et ainsi pouvoir communiquer avec le deuxième groupe.

Nous avons choisi de développer en C#, sous l’environnement Unity. Nous utilisons donc la version 2017.3.0f3 mise à disposition gratuitement dans un cadre d’enseignement.

## Répartition des tâches

Afin de maximiser notre productivité, nous avons décidé de nous distribuer les différentes tâches. Un membre se concentrera donc sur la génération de la courbe (wire) tandis que les deux autres modéliseront l’environnement et permettrons l’utilisation d’un Oculus.

Une fois chaque partie indépendamment terminée, nous nous occuperons, avec l’autre groupe, d’assembler toutes ces fonctionnalités pour créer un build du jeu.

# Difficultés rencontrées et solutions apportées

## Environnement virtuel

La problématique que nous avons rencontré ici est la suivante : « Comment rendre l’immersion la plus naturelle et agréable possible ». De plus, en VR, nous n’avons aucun retour haptique ou collision possible avec l’environnement.

Pour répondre à ces questions et justifier le fait que l’anneau (loop) passe à travers la courbe (wire) sans retour utilisateur autre qu’un retour sonore et l’arrêt du chronomètre, nous avons choisi de modéliser un environnement futuriste riche ou la courbe sera modélisé comme un hologramme. Nous justifions ainsi l’impossibilité de collisions lié à la VR, et permettons une meilleure immersion.

## Génération de la courbe

La première question à résoudre était de définir un algorithme afin de générer la courbe. Plusieurs possibilités ont été envisagées :

* Assembler aléatoirement des cylindres afin de former une droite brisée continue (en diminuant la taille des cylindres on peut envisager un aspect lisse).
* Assembler des objets courbés ou des morceaux de courbes, en utilisant des intégrales pour les définir.
* Utiliser des courbes de Bézier ou des B-Splines afin de définir la courbe, et générer par la suite des éléments texturés le long de cette courbe pour permettre une détection des collisions et une épaisseur.

La solution qui a été retenue est la dernière. Elle nous permet de mettre en pratique des connaissances que nous avons pu acquérir lors de notre formation et semble bien plus intéressante que les deux autres. De plus, elle permet une manipulation plus simple, et bien que plus complexe sur le plan mathématique, elle permet une simplification algorithmique non négligeable.

### Les différentes classes

Bezier

getPoint(p1,p2,p3,p4,t)

getDerivate(p1,p2,p3,p4, t)

BezierSplineInspector

OnInspectorGui()

OnSceneGui()

BezierSpline

Set/GetControlPoint()

Set/getControlPointMode()

EnforceMode()

CurveCount()

GetPoint()

AddCurve()

Reset()

**RandBezier()**

BezierControlPointMode

<enum>

Free, Aligned, Mirrored

SplineDecorator

Awake()

Diagramme de classes simplifié

Nous avons donc créé un ensemble de 5 classes afin de pouvoir générer la courbe finale. Nous allons donc coller des courbes de Bézier quadri linéaires.

La classe « Bezier » servira à calculer la position d’un point sur la courbe en lui donnant les quatre points de contrôle qui la définissent et une valeur t entre 0 et 1 qui définit quel point de la courbe calculer. Elle permet également de calculer la dérivée de la courbe en un point t. Cette fonctionnalité nous permet d’orienter les objets que l’on place le long de la courbe.

La classe « BezierSplineInspector » permet de manipuler cette courbe dans l’interface Unity. On y défini comment la paramétrer, la manipuler et l’afficher. Grâce à cela, on peut placer les points de départ et d’arrivé manuellement (même en « glissant » dans l’environnement de développement) ou choisir différents paramètres.

L’énumération « BezierControlPointMode » permet de définir la façon de raccorder deux Béziers. Si on choisit « Free » elles seront raccordées avec une continuité C0, C1 pour « Aligned » et C2 pour « Mirrored ». Si on choisit « Free » on verra des angles au niveau des changement de courbes qui seront complètement invisible en « Mirrored ».

La classe « SplineDecorator » permet dès son réveil (au lancement de la scène donc) de positionner des objets le long de la courbe afin de pouvoir la visualiser à la caméra et gérer des collisions avec l’anneau (loop).

La classe « BezierSpline » est la classe principale de l’algorithme. Elle stocke les points de contrôles de la courbe finale et possède des fonctions qui permettent de modifier ces points, d’en ajouter, de les manipuler, ou de les générer aléatoirement selon l’algorithme décris par la suite. On notera qu’elle dispose d’une fonction reset qui permet de réinitialiser l’objet via Unity pour qu’il ne se constitue que d’une Bézier droite au centre du repère.

### Algorithme de génération de courbe

Les points de départ et d’arrivés étant définie lors de la création de la scène, on va donc utiliser le vecteur qui part du départ pour aller à l’arrivé, le paramètre « width » qui définira la largeur que l’amplitude que l’on veut donner à la courbe ainsi que le nombre de Bézier que l’on veut utiliser.

On notera de plus que l’on peut définir cette courbe manuellement via l’outil créé pour manipuler des courbes, ou cocher l’option « génération aléatoire » qui s’occupera de la générer.

Algorithme (fonction « RandBezier() »:

* Calcul du vecteur normalisé donnant la direction du vecteur « début->fin » et calcul du pas de parcours, en fonction du nombre de point de contrôle à créer.
* Le premier point étant celui de départ et le dernier celui de l’arrivé, on créé une boucle qui va générer des points de contrôles entre ceux-ci, le long du vecteur directeur mais avec un espacement aléatoire du vecteur principal.
  + On calcul un point du vecteur sur lequel on se place en multipliant le vecteur normal par le pas, et en ajoutant le résultat au point de départ.
  + On définit deux composantes d’un vecteur grâce à une fonction d’aléatoire, puis on calcule la dernière composante de façon à ce que le produit scalaire entre le vecteur créé et le vecteur « début->fin » soit égal à zéro. On veut effectivement éviter que la courbe fasse demi-tour pour éviter les « nœuds », donc on cherche à générer des vecteurs orthogonaux à ce dernier.

*(U\*V) = u1\*v1+u2\*v2+u3\*v3*

*Si (U\*V) = 0 -> v3 = - (u1\*v1+u2\*v2) / u3*

On fera cependant attention aux divisions par zéro, et aux imprécisions des « float » en définissant un delta qui permettra de considérer qu’une valeur vaux zéro si elle s’en rapproche trop. On adapte le calcul en fonction.

* Une fois la direction aléatoire choisie, on tire la distance de manière aléatoire. On définit comme maximum la variable « Width » choisit par l’utilisateur.
* Pour finir on ajoute ce point de contrôle à la liste des points de la courbe.
* On modifie ensuite les points de façon à ce que les courbes ainsi générées soient connectées C0, C1 ou C2 en fonction du choix utilisateur. On notera que si on choisit C1 ou C2 (aligned/mirrored) il se peut que des retours en arrière se fassent suite au repositionnement des points. On a donc des risques de créer des « nœuds » qu’on n’a pas en C0 (free).
* Une fois tous les points générés, la classe « SplineDecorator » se chargera de modéliser des éléments de long de la courbe en utilisant le calcul de la classe « Bezier ».

# Conclusion

L’objectif du projet a donc été atteint. Nous avons créé un outil puissant permettant de manipuler des courbes sur Unity de manière graphique, ou via des scripts, ainsi que d’en générer aléatoirement. Nous avons également construit un environnement 3D riche dans Unity en mettant en pratique ce que nous avons pu apprendre lors de notre formation.

On peut cependant imaginer différentes améliorations comme la possibilité de créer manuellement des courbes directement via le jeu. Cependant, des Oculus Touchs seront certainement indispensable à la manipulation en VR.

On peut dès alors imaginer diverses applications possibles pour ce projet. En dehors d’être un simple jeu il peut permettre d’étudier des performances humaines en fonction des différents paramètres tel que la largeur de l’anneau ou la complexité du chemin.

Ce projet nous à permit de découvrir l’environnement de développement Unity ainsi que de mettre en pratique différentes notions venant de différents cours que nous avons pu suivre. Il nous a également permis de parfaire notre méthodologie et notre façon de travailler en équipe.

# Annexes

## Code source :

Fonction RandBezier() :

