UNIVERSITÉ SAVOIE MONT BLANC

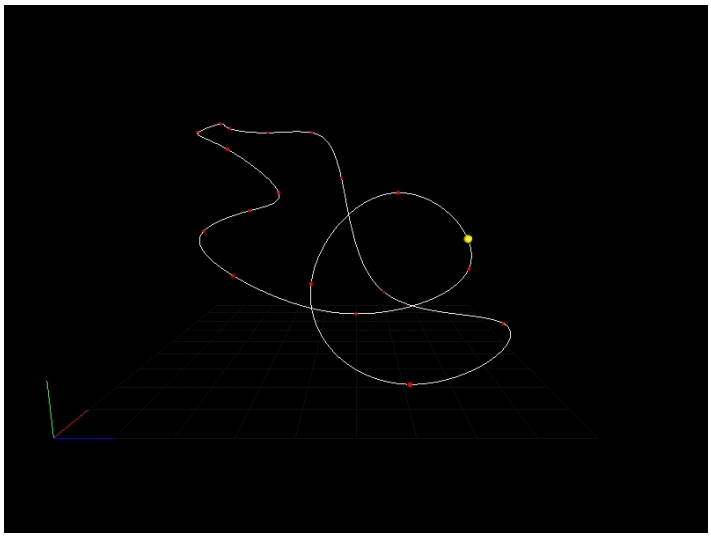
Projet de 8 heures

Python - conception

— M1 MMAA —

Roller Coaster

En très bref, je vous propose de mettre en place une simulation de roller coaster avec NumPy et Matplotlib !



1 Roller Coaster

Le programme que vous devez implémenter est une simulation de roller coaster (montagne russe, grand huit, ...).
Ce programme est divisé en 3 parties :

- **la piste** : elle est générée par une spline 3D passant par une série de points de contrôle contenus dans un fichier passé en argument dans votre programme.
- **la cabine** : elle est modélisée par une masse ponctuelle et représentée par un point ou une sphère (matplotlib). Elle doit se déplacer sur la piste en accord avec les principes de la physique.

— **les caméras** : vous devrez développer **au moins** deux caméras dont l'une est statique et représente représente une vue d'ensemble.

2 Organisation

Vos groupes seront composés de 2 ou 3 étudiants (maximum).

3 Travail demandé

Durant ce projet de 6-8 heures, vous devrez :
— Organiser votre équipe.
— Ecrire vos programmes.
— Utiliser le plus possibles les classes en Python !
— Générer un rapport sur votre travail.
Vous utiliserez les bibliothèques NumPy pour la partie génération de la spline 3D et matplotlib pour la partie visualisation et animation. Votre programme devra fonctionner sans modification sur les machines de l'Université.

4 Le circuit

La création du circuit est l'une des difficultés de ce projet. Le circuit est défini par une spline 3D passant par une série de points de contrôle stockés dans un fichier (format du fichier : voir annexes). La seule contrainte de ce circuit est qu'il doit boucler sur lui-même.

4.1 Spline : non uniforme

Nous utilisons des splines pour générer notre circuit pour des questions de simplicité. Cependant, les splines n'ont pas une "densité" uniforme. En effet, pour parcourir une spline, on utilise une variable t qui varie de 0 à 1 : 0 correspond au début de la spline (premier point de contrôle) et 1 correspond à la fin de la spline (dernier point de contrôle). A toutes les valeurs intermédiaires de t correspondent les points intermédiaires sur la spline. Mais voilà, comme le montre la Figure 1, ces points ne sont par répartis uniformément sur toute la spline. Ceci signifie que avancer de $t = 0.1$ au début de la spline, au milieu de la spline ou à la fin ne correspondra pas nécessairement à la même distance. Cette propriété est problématique pour une utilisation avec un roller coaster qui doit avancer à chaque nouvelle image d'une distance bien précise.

Une solution consiste à transformer notre spline en une liste de points équidistants. Pour cela, nous allons procéder en plusieurs étapes :

1. générer les paramètres de la spline à partir de la liste de points de contrôle.
2. parcourir la spline, même de façon non linéaire, pour en estimer la longueur.

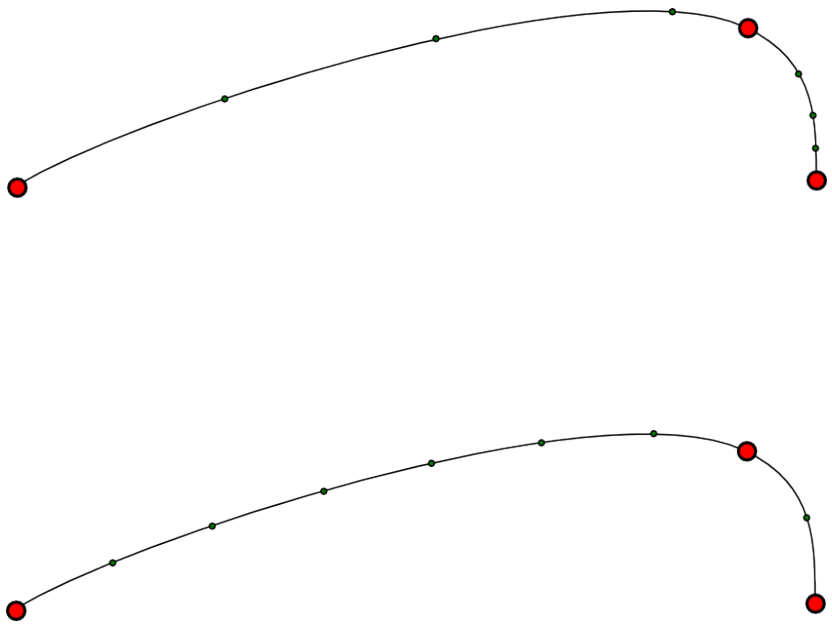


FIGURE 1 – Densité non uniforme dans une spline.

3. trouver un intervalle *step* pour nos nouveaux points qui divise la longueure estimée à l'étape précédente.
4. reparcourir la spline avec un pas *dt* assez petit : le premier point inséré est le premier point de contrôle. Ensuite, vous avancez tout doucement sur la spline. Quand la distance vous séparant du premier point de contrôle est tout juste supérieure à *step*, vous ajouter un nouveau point et vous recommencez jusqu'à ce que toute la spline soit parcourue.

4.2 Spline 3D

Le modèle de spline proposé est un peu inhabituel puisque le calcul des splines s'effectue en utilisant, non pas les points de contrôle 4 par 4, mais en utilisant tous les points du circuit simultanément.

Soient $P_{ii=1...n}$ les n points de contrôle définissant notre courbe. A l'aide de ces points de contrôle, nous pouvons construire les 3 systèmes linéaires suivants :

$$\begin{bmatrix} 4 & 1 & 0 & & & 0 & 1 \\ 1 & 4 & 1 & 0 & & & 0 \\ 0 & 1 & 4 & 1 & 0 & & \\ & 0 & 1 & 4 & 1 & 0 & \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \\ 0 & & & 0 & 1 & 4 & 1 \\ 1 & 0 & & & 0 & 1 & 4 \end{bmatrix} \begin{pmatrix} Dx_1 \\ Dx_2 \\ \vdots \\ \vdots \\ \vdots \\ Dx_n \end{pmatrix} = \begin{pmatrix} 3(Px_2 - Px_n) \\ 3(Px_3 - Px_1) \\ \vdots \\ \vdots \\ \vdots \\ 3(Px_1 - Px_{n-1}) \end{pmatrix}$$

$$\begin{bmatrix} 4 & 1 & 0 & & & 0 & 1 \\ 1 & 4 & 1 & 0 & & & 0 \\ 0 & 1 & 4 & 1 & 0 & & \\ & 0 & 1 & 4 & 1 & 0 & \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \\ 0 & & & 0 & 1 & 4 & 1 \\ 1 & 0 & & & 0 & 1 & 4 \end{bmatrix} \begin{pmatrix} Dy_1 \\ Dy_2 \\ \vdots \\ \vdots \\ \vdots \\ Dy_n \end{pmatrix} = \begin{pmatrix} 3(Py_2 - Py_n) \\ 3(Py_3 - Py_1) \\ \vdots \\ \vdots \\ \vdots \\ 3(Py_1 - Py_{n-1}) \end{pmatrix}$$

$$\begin{bmatrix} 4 & 1 & 0 & & & 0 & 1 \\ 1 & 4 & 1 & 0 & & & 0 \\ 0 & 1 & 4 & 1 & 0 & & \\ & 0 & 1 & 4 & 1 & 0 & \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \\ 0 & & & 0 & 1 & 4 & 1 \\ 1 & 0 & & & 0 & 1 & 4 \end{bmatrix} \begin{pmatrix} Dz_1 \\ Dz_2 \\ \vdots \\ \vdots \\ \vdots \\ Dz_n \end{pmatrix} = \begin{pmatrix} 3(Pz_2 - Pz_n) \\ 3(Pz_3 - Pz_1) \\ \vdots \\ \vdots \\ \vdots \\ 3(Pz_1 - Pz_{n-1}) \end{pmatrix}$$

Une fois ces trois systèmes résolus, vous obtenez trois vecteurs Dx, Dy et Dz. Ceux-ci vont nous permettre de générer une spline par intervalle de 2 points de contrôle consécutifs.

Pour générer la spline entre le point k et le point $k+1$, nous allons définir 3 polynômes $f_x(t) = ax_0 + ax_1 t + ax_2 t^2 + ax_3 t^3$ (respectivement $f_y(t)$ et $f_z(t)$) de la façon suivante :

$$\begin{aligned} ax_0 &= Px_k \\ ax_1 &= D_k \\ ax_2 &= 3(Px_{k+1} - Px_k) - 2D_k - D_{k+1} \\ ax_3 &= 2(Px_k - Px_{k+1}) + D_k + D_{k+1} \end{aligned}$$

Pour parcourir la spline entre les points k et $k+1$, il suffit de faire varier t entre 0 et 1. Pour un t donné, nous obtenons un point 3D

$$\mathbf{X} = (f_x(t), f_y(t), f_z(t))$$

Pour parcourir tout le circuit, il faut pour chaque intervalle de 2 points de contrôle consécutif calculer les 3 polynômes $f_x(t)$, $f_y(t)$ et $f_z(t)$ puis faire varier t de 0 à 1.

Pour plus de détails sur cette méthode, vous pouvez vous reporter sur :
`http://mathworld.wolfram.com/CubicSpline.html` .

Une fois la spline créée, vous pouvez passer à la création de point équidistants pour représenter le circuit.

4.3 Affichage

Pour afficher le circuit sous matplolib, il suffit de dessiner des lignes entre les points équidistants générés pendant la création du circuit.

A noter : Il faut aussi pouvoir afficher les points de contrôle.

5 La cabine

Le circuit est à présent représenté par un ensemble de points équidistants. La cabine, supposée ponctuelle, doit se déplacer sur ce circuit. Pour cela, elle avance de façon rectiligne entre deux points consécutifs du circuit. Le mouvement de la cabine doit être généré en accord avec le principe fondamental de la dynamique. La figure 2 représente la somme des forces appliquées à la cabine.

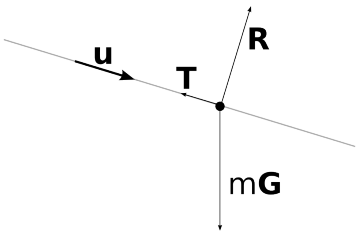


FIGURE 2 – Forces appliquées à la cabine.

Le vecteur **u** est le vecteur unitaire représentant la direction du circuit au niveau de la cabine. Le vecteur **T** représente les forces de frottement, le vecteur $m\mathbf{G}$ représente la force du point et le vecteur **R** la réaction du câble sur la cabine. Le principe fondamental de la dynamique se résume ainsi :

$$m.\mathbf{a} = m\mathbf{G} + \mathbf{T} + \mathbf{R}$$

Si l'on projette ces forces sur l'axe du circuit, nous obtenons :

$$ma_u = -T + mG\mathbf{down}.\mathbf{u}$$

où a_u représente l'accélération de la cabine sur l'axe **u** du circuit et **down** représente le vecteur unitaire vertical dirigé vers le bas. On peut considérer que les forces de frottement s'écrivent sous la forme : $T = -kv_u$ où v_u est la vitesse de la cabine et k un coefficient de frottement.

L'accélération a_u s'écrit alors :

$$a_u = G.\mathbf{down}.\mathbf{u} - \frac{k}{m}v_u$$

Sur un intervalle de temps dt, la vitesse de la cabine pourra donc être approximée par :

$$v_u(t + dt) = v_u(t) + a_u.dt$$

et la position

$$\mathbf{p}_u(t + dt) = \mathbf{p}_u(t) + v_u.dt\mathbf{u}$$

Notez que ces équations impliquent que l'on se souviennent de la vitesse et de la position de la cabine à l'instant précédent. Pour la mise à jour de la position, il faudra faire très attention au changement d'intervalle : au temps t, la cabine est entre les points k et $k + 1$. Au temps

$t + dt$, il faut qu'elle avance d'une distance d. Si cette distance d est inférieur à la distance séparant la cabine du point $k+1$, il suffit alors d'avancer de façon rectiligne entre k et $k+1$. Dans le cas contraire, après avoir dépassé le point $k+1$, il faut penser à changer de direction et continuer d'avancer dans la direction de la droite $(k+1, k+2)$.

Lors du parcours du circuit, il se peut que la cabine n'ait pas assez d'élan et s'arrête sur une montée, puis commence à reculer. Il est recommandé de tenir compte du fait que votre cabine peut reculer, et donc parcourir le circuit à reculons.

Remarque 1 : ce dernier point vous sera exigé ou non selon l'avancement général de l'ensemble des groupes.

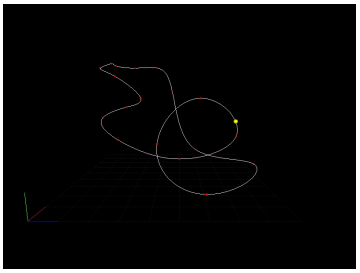
Remarque 2 : l'animation devra se faire avec matplotlib.

6 La caméra

Il existe de multiples façon de filmer un roller coster. Par défaut, vous pouvez utiliser une caméra statique filmant l'ensemble de la scène. Pour la suite, vous pouvez mettre en place au moins deux systèmes de gestion des caméras.

6.1 Vue d'ensemble

La caméra est statique et filme l'intégralité du circuit.

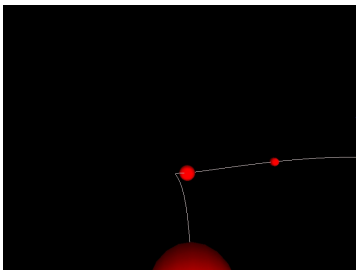


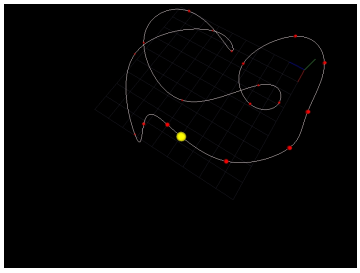
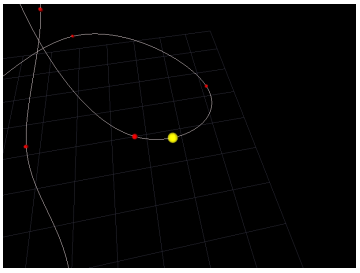
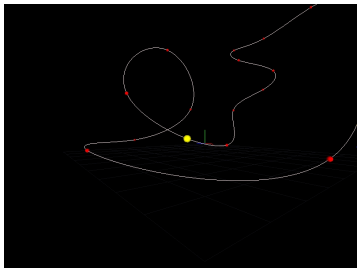
6.2 Caméra à la 3ème personne

Dans cette situation, la caméra se trouve à la verticale de la cabine, éventuellement un peu derrière (ce qui implique de connaitre sa direction à tout moment).

6.3 Autres

Cette approche nécessite plusieurs caméras disposée le long du circuit. Ces caméras ne peuvent pas se déplacer mais peuvent s'orienter pour suivre la cabine. La camera qui filme est la caméra la plus proche de la cabine.



7 Le rapport

Votre rapport comprendra les éléments suivants :
— **un manuel** : il explique comment utiliser votre programme.
— **une liste de ce qui marche** : avec des captures d'écran.
— **une liste de ce qui marche presque** : avec ce qui manque pour que ça marche.
— **une partie gestion de projet** : comprenant entre autres la répartion des tâches et un planing prévisionnel.
— **autres** : toutes autres parties que j'aurais oublié et qui vous parait pertinente.

8 Annexes

8.1 Format de fichier des points

Les points de contrôle à charger seront stockés dans un fichier sous la forme suivante :

```
nombre de points de contrôle

X1
Y1
Z1

X2
Y2
Z2
```

...

Pour être valide, un circuit doit comporter au moins 4 points de contrôle.