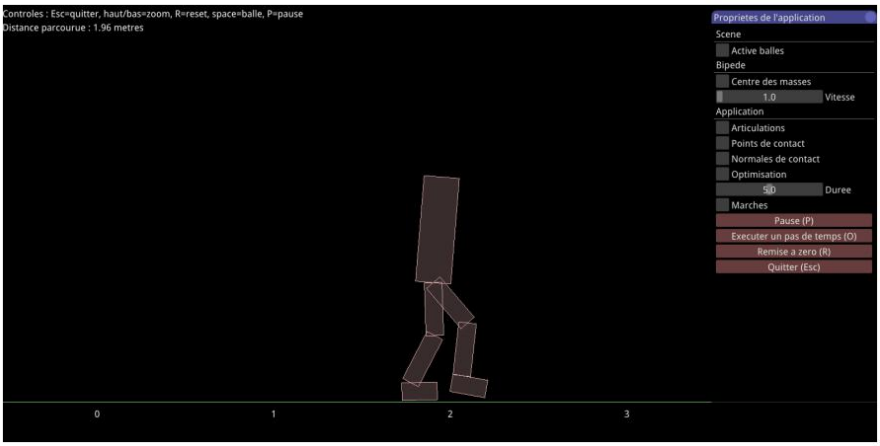
Grégoire Gobert, Daniel Douheret (11927231 , 11933390)

20/01/2020

Master ID3D – Animation, corps articulés et moteurs physiques

TP3 - Contrôleur de mouvement



On souhaite réaliser un contrôleur de mouvement d'un bipède articulé à partir d'une base de code déjà fournie. L'algorithme consiste à faire passer le bipède par une série d'états (des poses clés prédéfinies), afin de le faire avancer avec éventuellement des contraintes extérieures. Les poses clés, définies dans une machine à états, permettent d'interpoler les angles cibles lorsque le bipède se trouve entre deux états.

A chaque pas de temps, on souhaite donc minimiser l'écart entre la pose courante et la pose cible (target), par le biais de l'application d'un moment τ à l'articulation, obtenu par le régulateur Proportionnel Dérivé suivant:

$$\tau = K_p(\theta_{target} - \theta_{courant}) + K_v(\dot{\theta_{target}} - \dot{\theta_{courant}})$$

Où K_p et K_d sont des constantes à définir, et θ, $\dot{\theta}$ les angles et vitesses angulaires des articulations.

Les transitions entre chaque état se feront lorsque le temps de pose inhérent à l'état sera écoulé.

On dispose de 3 classes,:

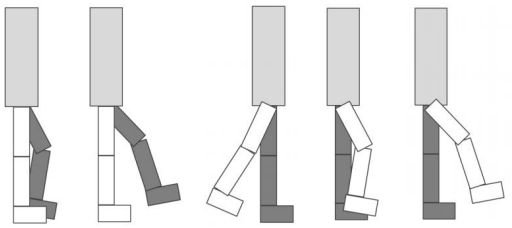
- *biped.cpp/.h* : le bipède
- *FSM.cpp/.h* : la machine à états
- *PDControlleur.cpp/.h* : le régulateur qui fournit le moment à appliquer

Partie 1 – 1 Etat

Dans cette partie, on ne prendra qu'un état, qui consiste à faire tenir debout le bipède. On implémente la fonction *Biped::KeyPoseTracking()* qui va permettre de récupérer les angles cibles et affecter les bons angles courants grâce au PDControler. On ajoute la création de l'attribut *m_PDControleur*, en affectant des valeurs de gain K_p et K_d permettant au bipède de tenir debout.

Partie 2 - Marche

Dans cette partie nous créons la machine à états et les poses clés permettant au bipède de marcher. Pour cela, on se base sur l'animation approchée d'une marche classique,afin de déterminer les angles associées à chaque état.



Le plus important est de trouver les meilleurs coefficients K_p et K_d pour obtenir un résultat satisfaisant. L'opération est manuelle et répétitive, mais permet d'obtenir un certain résultat : le bipède avance, et se remet bien des perturbations extérieures comme les balles ou les faibles marches.

Partie 3 - Optimisation

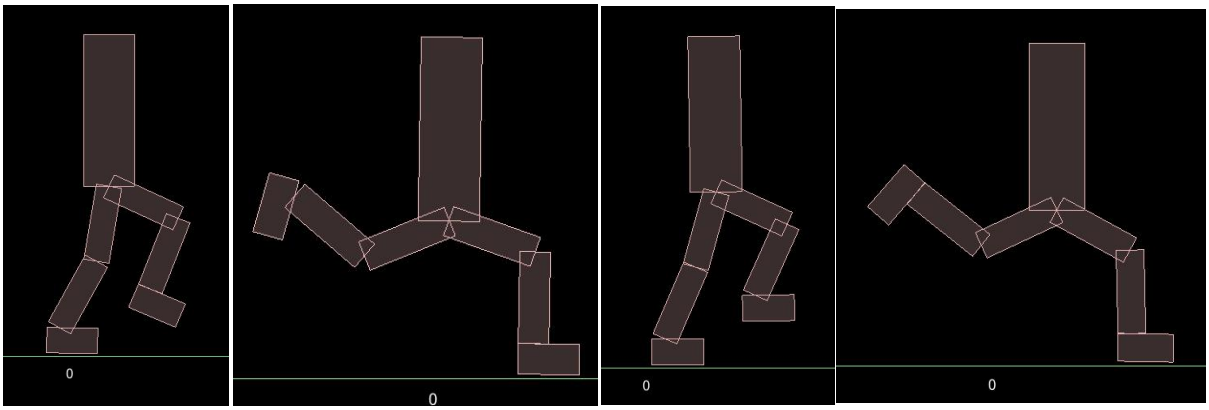
Afin de se passer de cette phase manuelle laborieuse, on recherche des bonnes valeurs de gain automatiquement. Pour cela on utilisera la classe Application déjà fournie. Les principaux apports se situent dans la fonction *main*. L'optimisation se fait en partant de la meilleure configuration connue (dans notre cas, la mesure utilisée est la somme des vitesses angulaires), en choisissant aléatoirement une configuration plus ou moins proche, en évaluant la qualité de cette nouvelle configuration, et en la conservant si elle est plus performante. Cela permet de se rapprocher d'une configuration optimale.

Partie 4 - Amélioration

Nous avons ensuite apporté une amélioration à l'application. Nous avons choisi de :

- Créer une machine à états finis représentant un cycle de course
- Créer ensuite une machine à états finis permettant de démarrer au repos, puis de marcher de plus en plus rapidement pour finalement courir.

La création de la machine a états pour un cycle de course se fait de manière similaire au cycle de marche. Ce cycle de course ne comporte cependant que 4 états et le temps de transition entre deux états est plus court. Ces choix sont volontaires, puisqu'ils engendrent de grandes différences entre deux états successifs, ce qui va imposer un moment plus important et aussi une inertie plus grande,ce qui est nécessaire pour bien représenter un cycle de course. Les états sont représentés ci-dessous (état 1-2-3-4):



- Etat 1 : le jambe gauche est en flexion vers l'avant
- Etat 2 : la jambe droite s'élance vers l'arrière
- Etat 3 : la jambe droite est en flexion vers l'avant
- Etat 4 : la jambe gauche s'élance vers l'arrière

Le grand écart entre les états successifs 2 et 3 et les états 4 et 1 permettent d'obtenir cette inertie globale vers l'avant.

Une quatrième machine a états finis a été créée afin de proposer une transition du repos à la marche puis de la marche à la course. Cette machine à états est un peu particulière puisqu'elle s'appuie sur les machines à états *Stand*, *Walk* et *Run* mais ne comporte pas d'états propres. Les méthodes *update*, *getCurrentTargetAngles* et *getCurrentTargetLocal* ont été réécrites afin d'appeler les méthodes correspondantes des 3 machines à états précédentes au bon moment :

- de t=0s à t=3s : les méthodes de *Stand* sont appelées.

- de t=3s à t=13s : le temps de transition entre deux états diminue progressivement (linéairement de 0.3s à 0.2s) et les méthodes de *Walk* sont appelées.

- de t=13s à t=23s : le temps de transition entre deux états diminue progressivement (linéairement de 0.28s à 0.18s) et les méthodes de *Run* sont appelées.

Précisons que le transition entre la marche et la course se fait au bout de 13 secondes et lorsque la marche est à son dernier état, puisqu'il n'est pas possible de passer d'un cycle à un autre n'importe quand, il est nécessaire que les poses successives soient assez proches.

Les gains des contrôleurs PD sont aussi modifiés en adéquation avec les transitions.

Conclusion

Cet algorithme basé sur des principes simples et intuitifs semble bien fonctionner, sous réserve de l'utilisation de combinaisons de gains efficaces. Il est de plus robuste et résistant aux perturbations extérieures. Comme nous avons pu le voir dans l'amélioration apportée, il est assez simple de rajouter des machines à états et de réaliser des transitions entre ces différents cycles. Cependant, il permet difficilement d'obtenir un résultat « naturel » de marche, et la recherche de gains est laborieuse.