PLANIFICATION EN INTELLIGENCE ARTIFICIELLE

IFT 702

CamUS

Planification dynamique d'une

caméra interactive à la 3e personne

Travail présenté à

M. Frodualt Kabanza

Par

Marc-Alexandre Côté	07 166 997
Julien Filion	07 177 770

Université de Sherbrooke

Hiver 2010

Table des matières

Introduction 1
Difficultés 2
Instructions pour démarrer l'application 3
- Mise en garde : 3
- Touche clavier 3

Évaluation d'une prise de vue 5
- Travaux existants 5
- Système de récompense 5
- Occlusion 6
- Profil des objets 6
- Distance des objets 6
- Optimisations 7

Planification 8
- Travaux existants 8
- Value Iteration 8
  - États 9
  - Récompense 9
  - Transitions 9
- Optimisations 10

Trajectoire de la caméra 11
- Temps réel 11
- Lissage de trajectoire 12

Prochaine étape 13
- Parallélisations 13
- Prédiction 13
- Raffiner la précision 14

Conclusion 14

Table des figures

Figure 1 : Rendu original de la scène (Haut) comparé au rendu avec volume englobant (Bas) 7
Figure 2 : Géosphères avec transition originale (Droite) et élaguée (à Gauche). 10
Figure 3 : Courbe Catmull-Rom Spline. 12

Introduction

Olympus est un engin graphique développé par un groupe d'étudiants de l'université de Sherbrooke. Cet engin encore en développement cible la conception de jeu vidéo. Un jeu vidéo est composé d'une multitude d'éléments qui visent tous à rendre le jeu amusant. Un de ces éléments est la caméra. La caméra est particulièrement importante, car elle dicte ce que le joueur voit ou ne voit pas. Nous avons donc décidé d'implanter un système de caméra pour l'engin Olympus.

Plusieurs types de caméra ont été utilisés dans les jeux.
La caméra fixe est une des plus simples, les propriétés de cette caméra sont déterminées lors de la création du jeu par les programmeurs et ne changent pas durant en cours de partie.

Un système de caméra de tracking offre une solution plus sophistiquée à la prise de vue. Cette caméra suit le personnage partout où il va avec une vue à la 3e personne. Le joueur ne peut pas contrôler manuellement la caméra. Ce genre de caméra provoque souvent des prises de vue indésirables si un objet cache le personnage ou si la caméra ne montre pas ce que le joueur souhaite voir.

Finalement, la caméra interactive est une amélioration de la caméra de tracking. Cette caméra suit un personnage, mais le joueur peut influencer la prise de vue de la caméra. Ce type de caméra offre généralement un système d'intelligence artificielle qui contrôle automatiquement la caméra tout en laissant au joueur la possibilité de modifier sa configuration. Par contre, même avec une caméra interactive, il n'est pas rare que des points d'intérêt ne soient pas visibles à la caméra et que le joueur soit forcé de modifier lui-même la position de la caméra.

Difficultés

Notre objectif est de programmer une caméra interactive qui fournit toujours la meilleure prise de vue pour un jeu en trois dimensions. Ceci demande de surmonter plusieurs défis complexes.

Une des plus grandes difficultés est le caractère imprévisible du joueur. On ne connaît pas à l'avance quelles actions fera le joueur, ce qui nous empêche de planifier tous les paramètres de la caméra en un seul moment, tout doit se faire au fur et à mesure que le jeu se déroule.

Un autre défi est d'être en mesure d'évaluer la qualité d'une prise de vue, on doit déterminer quels critères permettent de dire qu'une vue est meilleure qu'une autre.

Aussi, un jeu vidéo est généralement un environnement assez complexe avec plusieurs objets qui se déplace. La caméra doit être positionnée dans un monde en trois dimensions, mais on doit aussi lui assigner une direction à regarder, on se retrouve donc avec six dimensions pour représenter l'état de la caméra. Sans compter qu'il s'agit d'un environnement dynamique et que le monde évolue au fil du temps.

De plus, nous somme limité dans le temps de calcul, car la planification de la caméra ne doit pas affecter la rapidité du jeu. Il faut aussi être rapide dans le calcul, car le monde peut changer rapidement et nous devons nous y adapter.

Finalement, nous devons aussi porter attention aux obstacles dans le jeu, il ne faut pas que la caméra puisse passer au travers des objets.

Instructions pour démarrer l'application

Pour démarrer l'application, il suffit de lancer CamUS.bat situé à la racine du CD-rom.

Mise en garde :

Pour utiliser notre application, l'ordinateur doit posséder DirectX et le Framework .NET de Microsoft. De plus il est préférable de ne pas utiliser la souris car sur certaines versions de Windows, son comportement n'est pas adéquat.

Touche clavier

Voici la liste des touches clavier qui ont un effet sur le jeu :

Esc : Termine l'application.

<u>Déplacement du personnage</u>

W : Déplace Mario vers l'avant.

A : Déplace Mario vers la gauche

S : Déplace Mario vers l'arrière

D : Déplace Mario vers la droite

Flèche droite : Change la direction de Mario vers la droite

Flèche gauche : Change la direction de Mario vers la gauche

<u>Déplacement de la caméra</u> (En mode statique seulement)

2 (Clavier Numérique) : Déplace la caméra vers le bas

4 (Clavier Numérique) : Déplace la caméra vers la gauche

6 (Clavier Numérique) : Déplace la caméra vers la droite

8 (Clavier Numérique) : Déplace la caméra vers le bas

+ (Clavier Numérique) : Zoom in

- (Clavier Numérique) : Zoom out

Enter (Clavier Numérique) : Réinitialise le Zoom

Fonctions

1 : Change le mode de calcul du plan ; mode Statique ou temps réel

2 : Change la technique d'interpolation de trajectoire ; mode Catmull-Rom Spline, Bezier ou linéaire.

3 : Change le mode de calcul d'occlusion; mode Occlusion Query ou Ray Tracing

4 : Change le test de visibilité, mode normal ou algébrique

Récompense de la pièce d'or

Flèche haut : augmente la récompense de la pièce d'or

Flèche bas : diminue la récompense de la pièce d'or

F12 : Réinitialise la récompense de la pièce d'or

Taux de rafraichissement du plan (Mode Temps réel seulement)

Home : Augmente le taux de rafraichissement du plan

End : Diminue le taux de rafraichissement du plan

Slash (clavier numérique) : Réinitialise le taux de rafraichissement du plan

Visualisation du plan (Géosphère intérieur)

Page Up : augmente le rayon de la géosphère

Page Down : diminue le rayon de la géosphère

* (clavier numérique) : Réinitialise le rayon de la géosphère

F1 : Montre les points de contrôle en bleu pale

F2 : Montre les points de contrôle dans la trajectoire en rouge

F3 : Montre la trajectoire en bleu

Visualisation du plan (Géosphère extérieur)

F5 : Montre les points de contrôle en jaune

F6 : Montre les points de contrôle dans la trajectoire en rouge

F7 : Montre la trajectoire en bleu

F8 : Montre les transitions possibles (lignes jaunes)

Évaluation d’une prise de vue

Le premier problème auquel nous avons fait face était de pouvoir comparer la qualité des prises de vue.

Travaux existants

Plusieurs travaux ont déjà été effectués pour évaluer la qualité d’une scène, mais la plupart des approches sont basées sur un système de contrainte. Par exemple, Christianson[1] analyse le niveau d’occlusion d’un cadrage à l’aide de volume englobant ou Tomlinson[2] qui utilise une technique de ray casting pour évaluer la visibilité.

Par contre, nous considérons que de n’utiliser qu’un système de contrainte n’est pas suffisant pour connaître la qualité d’une scène. Selon nous, certains objets de la scène sont plus importants à filmer que d’autres, il nous faut donc un système qui analyse cette importance.

Système de récompense

Notre idée est d’utiliser un système de récompense afin que les objets intéressants aient plus de poids dans le calcul de la qualité. Par exemple, comme il s’agit d’une caméra à la 3e personne, le personnage observé aura généralement la plus grande récompense. Par contre, pour rendre ce système possible, le programmeur doit lui-même spécifier la valeur de cette récompense pour tous les objets intéressants. Nous pouvons donc évaluer la qualité d’une scène en additionnant la récompense des objets visibles.

---

[1] D. Christianson, S. Anderson, L. He, D. Salesin, D. Weld, and M. Cohen. Declarative camera control For automatic cinematography. At ThirteenthNational Conference on Artificial Intelligence, 1996. 2, 3
[2] B. Tomlinson, B. Blumberg, and D. Nain. Expressive autonomous cinematography for interactive virtual environments. In C. Sierra, G. Maria, and J.S.Rosenschein, editors, Proceedings of the 4th International Conference on Autonomous Agents (AGENTS00), pages 317–324, NY, June 3–72000. ACMPress. 2, 3

Occlusion

Par contre, cette technique ne tient pas compte du pourcentage visible des objets. Nous avons donc raffiné le calcul à l'aide d'un ratio du nombre de pixels que devrait prendre l'objet avec le nombre de pixels réel affiché. On calcul ces valeurs en prenant deux photos successives de la scène. La première ne fait le rendu que de l'objet et on obtient le nombre de pixels qu'il devrait prendre. La deuxième est composée de toute la scène et nous permet de retrouver le nombre de pixels réel.

Profil des objets

Nous avons donc une évaluation qui optimise la visibilité des objets importants de la scène, mais il nous reste encore certains détails à régler, car certains angles de vue sont plus intéressants que d'autres. Par exemple, dans un jeu à la 3e personne, il est préférable de placer la caméra derrière le personnage plutôt que d'être face à celui-ci. Pour remédier à ce problème, nous avons implémenté un facteur d'atténuation propre à chaque objet qui dépend de l'angle d'affichage de ceux-ci. Donc le personnage principal aura une meilleure récompense si on le voit de dos alors que les personnages secondaires peuvent avoir une meilleure récompense si on les voit de face.

Distance des objets

Une dernière amélioration à notre fonction est d'utiliser un deuxième facteur d'atténuation qui dépend de la distance entre un objet et la caméra. Chaque objet a une distance idéale à la caméra, par exemple, plus un objet est gros, plus il doit être vu de loin.

Optimisations

Le calcul des récompenses est une opération particulièrement couteuse lorsque l'on doit en faire une grande quantité. Pour cette raison, nous avons dû faire quelques optimisations à notre procédé.

La première optimisation intervient lors du calcul de visibilité des objets. Il s'agit de réduire la résolution des images que nous prenons de la scène. Utiliser une résolution plus petite est une technique simple qui nous permet d'avoir une estimation de la vraie scène. On perd un peu de précision, mais on gagne en rapidité. Voir la figure 1 pour une démonstration.

Une deuxième optimisation, encore dans le calcul de la visibilité, est d'utiliser des volumes englobants plutôt que les objets originale lorsque nous prenons une photo de la scène. Nous avons une perte de précision, car un volume englobant peut cacher un objet alors que celui-ci ne serait pas caché par l'objet original.

Finalement, bien que cette technique n'a pas été implantée, il est possible de pré calculé le nombre de pixels que devrait prendre un objet avant de démarrer le jeu. Ainsi, le nombre de photos prises de notre scène serait diminué de moitié.

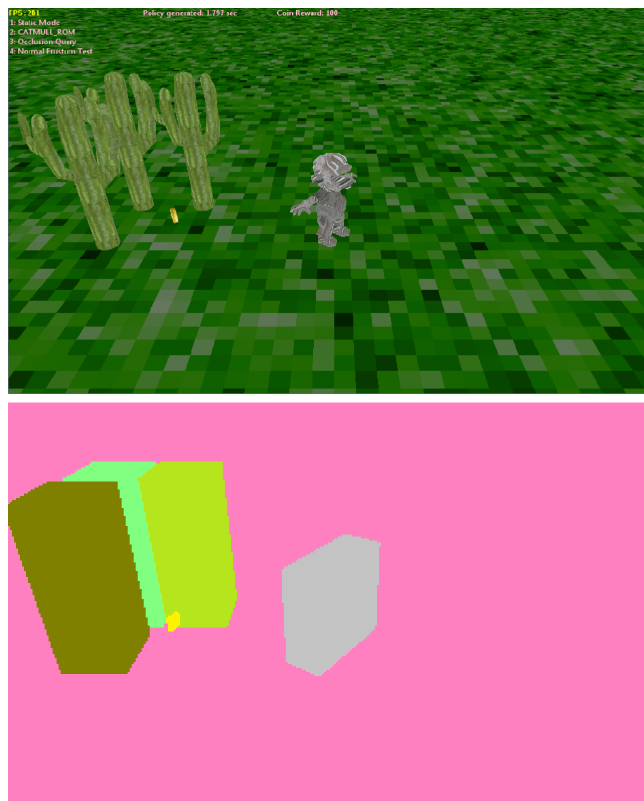

Figure 1 : Rendu original de la scène (Haut) comparé au rendu avec volume englobant (Bas)

Planification

Maintenant que nous pouvons comparer les différentes prises de vue, nous devons automatiser la caméra pour obtenir la meilleure séquence de scènes.

Travaux existants

Il existe déjà quelques travaux sur la planification de caméra, mais la plupart d'entre eux se concentrent sur une planification à priori du plan à suivre. Par exemple, Kabanza[3] utilise l'algorithme TLPLAN pour déterminer à l'avance la trajectoire de la caméra pour observer un bras robotique en mouvement. Ou encore Halper[4] qui utilise un algorithme génétique.

Il existe aussi quelque approche pour planifier des caméras en temps réel. Ces approches utilisent généralement des techniques de satisfaction de contrainte ou des scripts prédéfinis. Par contre, il est difficile d'obtenir de l'information sur l'implantation de caméra dans des jeux connus, car leurs détails restent cachés par les développeurs.

Value Iteration

Pour notre projet, nous nous sommes tournés sur une approche markovienne, car elle s'adapte particulièrement bien à notre système de comparaison des prises de vue. Nous avons donc implanté l'algorithme Value Iteration.

---

[3] F. Kabanza, K. Belghith, P. Bellefeuille et B. Auder. Planning 3D Task Demonstrations of a Teleoperated Space Robot Arm.

[4] Nicolas Halper and Patrick Olivier. CAMPLAN: A Camera Planning Agent**.** In *Smart Graphics. Papers from the 2000 AAAI Spring Symposium (Stanford, March 20-22, 2000)*, pages 92-100, Menlo Park, 2000. AAAI Press.

États

Chaque nœud de l'algorithme représente une configuration de caméra unique. Idéalement, la caméra devrait pouvoir se déplacer librement dans l'environnement tridimensionnel et pouvoir regarder dans toutes les directions. Par contre, laisser autant de liberté à notre caméra n'est pas réalisable dans le cadre d'une planification en temps réel. Afin de réduire le nombre d'états, nous avons donc fixé l'orientation de la caméra sur le personnage principal. Nous avons aussi fixé la distance entre la caméra et le personnage, ce qui nous laisse deux degrés de liberté pour l'état de la caméra qui correspond à sa position sur une géosphère située autour du personnage principal. Voir la figure 2 pour une vue graphique de la géosphère.

Récompense

Maintenant que nous avons nos états, nous devons en calculer la récompense. Cette récompense représente la qualité de la prise de vue et nous utilisons le calcul expliqué dans la section précédente pour obtenir cette valeur. Le calcul de la récompense est particulièrement gourmand en temps de calcul, nous devons donc réduire au maximum la quantité de récompenses à calculer.

Transitions

Ne reste plus qu'à déterminer les transitions entre nos états. Pour ce faire, on considère que l'on peut passer d'un état à un autre s'ils sont voisins verticalement, horizontalement ou diagonalement. Chaque nœud possède également une transition qui revient sur lui-même afin de s'assurer que la caméra puisse rester dans une vue optimale. Par contre, nous devons supprimer des liens qui sont obstrués par un obstacle. Pour détecter s'il y a un obstacle entre deux nœuds, nous utilisons une technique de lancer de rayons (Ray Tracing). Voir la figure 2 pour une vue graphique des transitions sur la géosphère. De plus, l'effet d'un déplacement de caméra est déterministe, ce qui nous permet d'utiliser une probabilité de 100% pour toutes nos transitions.

Optimisations

Puisque nous sommes dans un système en temps réel, la rapidité du calcul est très importante. Pour cette raison, nous appliquons une phase d'élagage qui enlève du calcul tous les nœuds non accessibles à partir de la position initiale de la caméra. Par exemple, si le personnage se trouve sur un plancher, tous les angles de vues qui sont dessous le plancher seront ignorés. Voir la figure 2 pour le résultat de l'élagage sur les transitions.

Un des paramètres importants de notre algorithme est le nombre de nœuds calculé. Si la discrétisation est trop précise, le plan sera trop lent à calculer, mais si le nombre d'états est trop petit, ne seras pas assez souple.

De plus, il serait intéressant d'essayer Policy Iteration et de le comparer à Value Iteration afin de n'utiliser que l'algorithme le plus rapide.

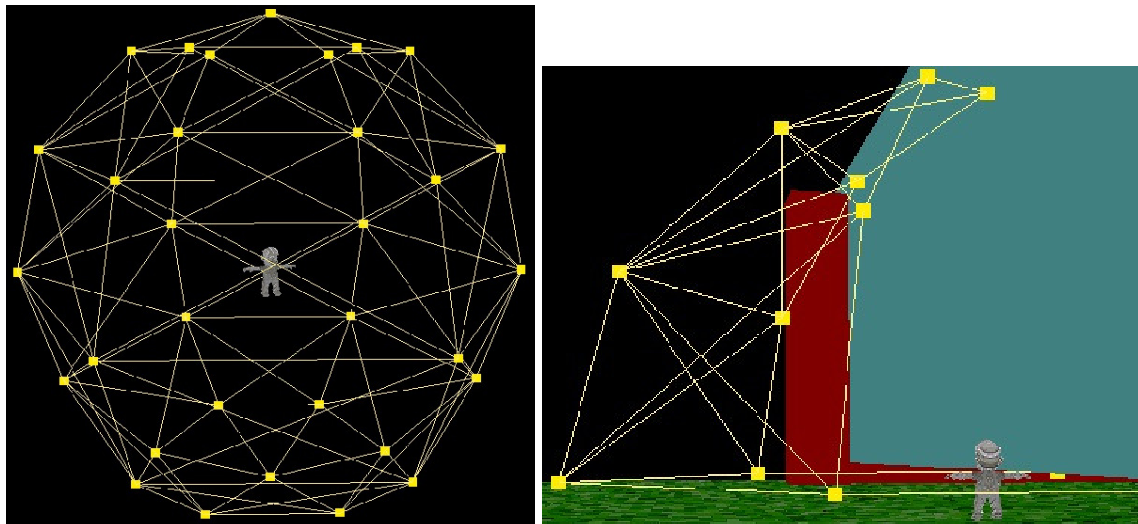

Figure 2 : Géosphères avec transition originale (Droite) et élaguée (à Gauche). Les points jaunes représentent les états et les lignes jaunes les transitions.

Trajectoire de la caméra

Grâce à Value Iteration, nous sommes capables de calculer une trajectoire optimale pour notre caméra. L'avantage d'utiliser Value Iteration est qu'il ne cherchera pas à trouver le chemin le plus rapide entre la position initiale de la caméra et l'état optimal, mais il va aussi prendre en compte la qualité des scènes lors de tout le trajet.

Temps réel

Puisqu'il est impossible de prédire comment un jeu évolue dans le temps, nous ne pouvons pas construire un seul plan. Pour utiliser Value Itération en temps réel, nous avons donc décidé de recalculer le plan à intervalle régulier, ce qui nous permet de tenir compte des modifications à l'environnement. Par contre, cette approche n'est pas optimale, en effet nous avons pensé à utiliser des techniques plus appropriés qui seront discutés en détail dans la section « Prochaine étape ».

Lissage de trajectoire

Afin de rendre notre trajectoire plus fluide, nous avons utilisé des trajectoires courbes pour passer d'un point de contrôle au prochain. Deux fonctions de courbe sont implantées, soit des courbes de Béziers et Catmull-Rom Spline. L'algorithme Catmull-Rom Spline est très utilisé pour le lissage de trajectoire de caméra et donne un meilleur résultat visuel. Voir la figure 3 pour la représentation graphique d'une trajectoire de caméra avec Catmull-Rom Spline.

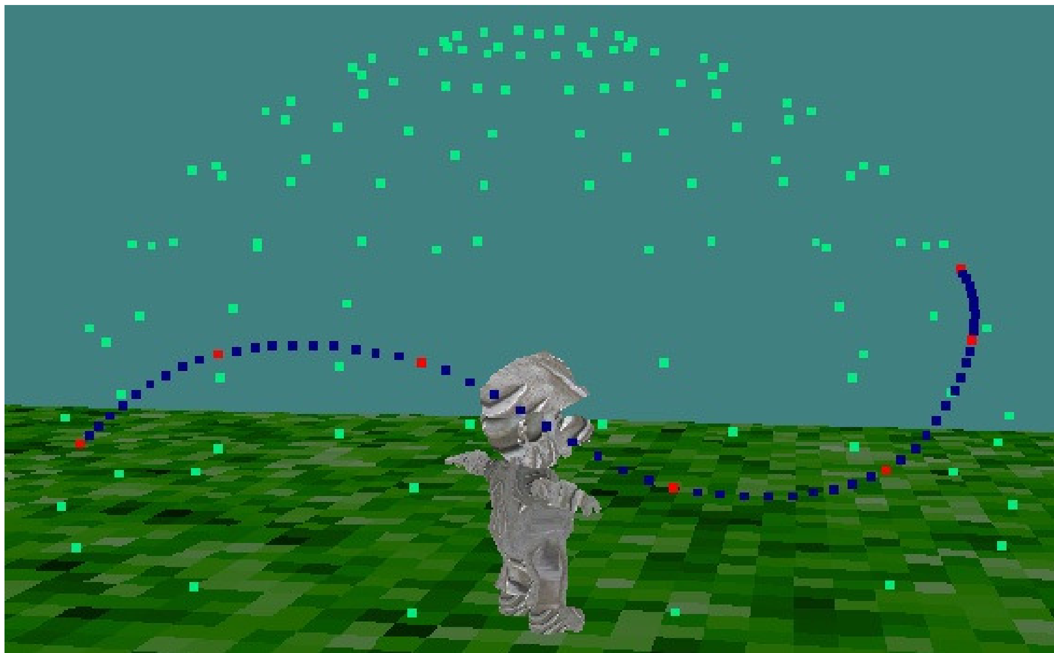

Figure 3 : Courbe Catmull-Rom Spline.
Les points rouges représentent les points de contrôle et les points bleus la trajectoire de la caméra.

Prochaine étape

Notre projet est un projet ambitieux, mais nous avons dû couper certaines fonctionnalités en raison du temps disponible pour le réaliser.

Parallélisations

Une grande amélioration de notre processus serait d'utiliser deux processus séparés. Soit un processus qui planifie la trajectoire à l'aide de Value Iteration et un deuxième processus pour l'exécution du jeu. Cette amélioration permettrait de calculer des plans plus complexes tout en conservant un bon taux de rafraichissement dans l'application. On pourrait aussi tirer profit des ordinateurs multiprocesseurs.

Prédiction

Une modification importante de notre algorithme nous permettrait d'améliorer la trajectoire de la caméra lorsque le personnage principal se déplace. Il s'agit de considérer la direction et la vitesse du déplacement de notre personnage pour générer les états successeurs. On ne se retrouverait plus avec une sphère, mais avec une forme conique pour les différentes positions de caméra. Avec cette technique, tant que le personnage conserve la même direction, on peut conserver le même plan. De plus, au fur et à mesure que le personnage et la caméra se déplacent, on peut élaguer les nœuds qui ne sont plus accessibles. Par contre, dès que le personnage change de direction, on doit refaire la planification.

On peut aussi utiliser ce concept pour des trajectoires de personnage qui ne sont pas linéaires. Par exemple, si le personnage marche sur un chemin, on peut prédire quand il devrait tourner et l'inclure dans notre calcul. Il serait même possible de faire de la reconnaissance de plan pour deviner la trajectoire du personnage.

Raffiner la précision

Une autre amélioration intéressante serait d'utiliser une précision itérative pour la précision. C'est-à-dire de faire un premier plan avec peu de nœuds, puis en faire un deuxième avec un peu plus de nœuds en continuant jusqu'à ce que l'on doive calculer un nouveau plan.

Conclusion

Pour conclure, le domaine de la planification de caméra est un domaine intéressant et rempli de défis. Il existe une multitude de méthodes pour résoudre ce problème et nous avons démontré que l'utilisation de Value Itération est une technique qui donne de bons résultats. Un des défis de notre application est de calculer les récompenses des nœuds afin de conserver une bonne précision, mais aussi une bonne rapidité. Bien que notre projet donne de bons résultats, il y a encore place à amélioration, par exemple, en faisant la planification dans un autre processus.