N° d'ordre : 181-2012 Année 2012

THESE DE L'UNIVERSITE DE LYON
Délivrée par
L'UNIVERSITE CLAUDE BERNARD - LYON 1
Ecole Doctorale Informatique et Mathématiques
DIPLOME DE DOCTORAT

Procedural Locomotion of Multi-Legged Characters in Complex Dynamic Environments: Real-Time Applications

Soutenue publiquement le 17 Octobre 2012
par **Ahmad ABDUL KARIM**

Composition du jury:

Rapporteurs	
Mr. François FAURE	Professeur à l'Université Joseph Fourier
Mr. Jean-Pierre JESSEL	Professeur à l'Université Paul Sabatier
Examinateurs	
Mme. Dominique FAUDOT	Professeur à l'Université de Bourgogne
Mr. Serge MIGUET	Professeur à l'Université Lumière Lyon 2
Mr. Axel BUENDIA	Directeur de Spir.Ops, PAST au Cédric/CNAM
Directeur	
Mme. Saïda BOUAKAZ	Professeur à l'Université Claude Bernard Lyon 1
Co-encadrant	
Mr. Alexandre MEYER	Maître de Conférences à l'Université Claude Bernard Lyon 1

I would like to dedicate this thesis to my loving parents.

Walid, Joumana, I would have never achieved this grade
without your support.

I hope I have made you proud of me.
Love you

Remerciements

Mes remerciements vont tout d'abord à Mme Saïda Bouakaz qui a dirigé cette thèse. Malgré l'éloignement, elle a toujours été disponible pour prodiguer des conseils et des orientations pertinentes.

Mes plus vifs remerciements vont également à M. Alexander Meyer qui a co-dirigé cette thèse. Ses conseils et ses nombreuses relectures de mes articles et de ce manuscrit m'ont permis de faire évoluer mes connaissances et mon point de vue sur ce travail.

Mes remerciements également à M. Axel Buendia pour m'avoir accepté au sein de l'entreprise Spir.Ops, pour sa présence tout au long de ce travail, pour la confiance qu'il m'a accordée et pour ces conseils formateurs et enrichissants.

Je tiens à remercier les rapporteurs de cette thèse M. François Faure et M. Jean-Pierre Jessel pour la lecture, pour les retours et pour l'intérêt qu'ils ont porté à mon travail. Je remercie également Mme Dominique Faudot et M. Serge Miguet qui ont bien voulu accepter de participer au jury de cette thèse comme examinateur.

Je ne trouve pas les mots pour exprimer ma reconnaissance envers M. Thibaut Gaudin, pour toutes les connaissances pratiques que j'ai acquises grâce à lui et qui a su orienter mes travaux dans les bonnes voies. Merci à mes collègues : Florian, Patrick, William, Jérôme et Denis pour m'avoir intégré au sein de l'entreprise ou je me suis sentis comme dans une grande famille, merci pour leur retour autour de la table de déjeuner.

J'adresse un grand merci à tous les amis que j'ai croisés pendant ces trois ans à Paris, à Lyon et au laboratoire LIRIS qui ont rendu mon séjour dans ces deux villes agréable, intéressant et chaleureux.

Du fond du coeur, je remercie ma famille, et plus particulièrement mes parents et ma petite soeur, pour leur formidable soutien.

Abstract

Multi-legged characters like quadrupeds, arachnids, reptiles, *etc.* are an essential part of any simulation and they greatly participate in making virtual worlds more life-like. These multi-legged characters should be capable of moving freely and in a believable way in order to convey a better immersive experience for the users. But these locomotion animations are quite rich due to the complexity of the navigated environments and the variety of the animated morphologies, gaits, body sizes and proportions, *etc.* Another challenge when modeling such animations arises from the lack of motion data inherent to either the difficulty to obtain them or the impossibility to capture them.

This thesis addresses these challenges by presenting a system capable of procedurally generating locomotion animations for dozens of multi-legged characters in real-time and without any motion data. Our system is quite generic thanks to the chosen **Procedural-Based** techniques and it is capable of animating different multi-legged morphologies. On top of that, the simulated characters have more freedom while moving, as we adapt the generated animations to the dynamic complex environments in real-time. The main focus is plausible movements that are, at the same time, believable and fully controllable. This controllability is one of the forces of our system as it gives the user the possibility to control all aspects of the generated animation thus producing the needed style of locomotion.

Keywords: Locomotion, Procedural Animation, Multi-Legged Characters, Complex Dynamic Environment, Real-Time, User Style.

Résumé

Les créatures à n-pattes, comme les quadrupèdes, les arachnides ou les reptiles, sont une partie essentielle de n'importe quelle simulation et ils participent à rendre les mondes virtuels plus crédibles et réalistes. Ces créatures à n-pattes doivent être capables de se déplacer librement vers les points d'intérêt de façon réaliste, afin d'offrir une meilleure expérience immersive aux utilisateurs. Ces animations de locomotion sont complexes en raison d'une grande variété de morphologies et de modes de déplacement. Il convient d'ajouter à cette problématique la complexité des environnements où ils naviguent. Un autre défi lors de la modélisation de tels mouvements vient de la difficulté à obtenir des données sources.

Dans cette thèse nous présentons un système capable de générer de manière procédurale des animations de locomotion pour des dizaines de créatures à n-pattes, en temps réel, sans aucune donnée de mouvement préexistante. Notre système est générique et contrôlable. Il est capable d'animer des morphologies différentes, tout en adaptant les animations générées à un environnement dynamique complexe, en temps réel, ce qui donne une grande liberté de déplacement aux créatures à n-pattes simulées. De plus, notre système permet à l'utilisateur de contrôler totalement l'animation produite et donc le style de locomotion.

Mot Clés : locomotion, animation procédural, créatures à n-pattes, environnement dynamique complexe, temps réel, style d'utilisateur.

Locomotion procédurale de créatures à n-pattes dans des environnements complexes et dynamiques : vers des applications en temps réel

Résumé étendu de la thèse en français

Chapitre 1 : introduction

Motivation : Les animaux réels ou imaginaires font de fréquentes apparitions dans les jeux vidéo, les serious games [DAJ11], les films et les mondes virtuels. L'animation de ces créatures virtuelles à n-pattes est essentielle pour rendre ces mondes virtuels plus crédibles et plus réalistes. Des créatures comme les quadrupèdes, les arachnides, les insectes, les reptiles ou les robots imaginaires à n-pattes, *etc.* La tâche la plus commune que ces créatures à n-pattes effectuent est la locomotion : la capacité de se déplacer librement dans des mondes virtuels vers des points d'intérêt. Ces mouvements de déplacement sont essentiels pour rendre les simulations virtuelles plus convaincantes et immersives, surtout quand ils se rapprochent des animations des vrais animaux dans le monde réel. Si la qualité des animations produites reste déterminante, dans le cas étudié l'efficacité des techniques et leur application au contexte du temps réel sont des facteurs tout aussi importants.

La richesse de ces animations de locomotion due à la variété des morphologies animées (un chien et une araignée), à leurs démarches (galop et trot), *etc.* est l'un des premiers défis lors de la création d'un système générique réutilisable capable d'animer ces différentes créatures avec une configuration minimale. De plus, ces créatures à n-pattes naviguent dans des environnements complexes (des escaliers, carte de hauteur, *etc.*), avec différents types d'obstacles (stable, instable, dynamique, *etc.*). Pour cela et afin de générer une simulation crédible, le système d'animation doit s'adapter à l'environnement et produire des animations de locomotion avec des franchissements d'obstacles, des traversées des terrains irréguliers, des évitements d'objets dynamiques, *etc.*

Un autre défi lors de la modélisation de tels mouvements provient du manque des données habituellement fournies par les systèmes d'acquisition (Motion Capture (MoCap) data) inhérents à la difficulté de les obtenir (les insectes) ou à l'impossibilité de les capturer (mammouths, créatures à 5-pattes). Et même lorsque des données de mouvement existent comme pour la course d'un chien, il est difficile d'adapter l'animation à un environnement différent de celui de la capture.

Contexte industriel : Cette thèse est une collaboration entre l'équipe SAARA du LIRIS et une entreprise parisienne Spir.Ops. Le but de cette collaboration est de profiter à la fois du savoir-faire de l'équipe SAARA dans les systèmes d'animations et mondes virtuels, et des compétences de Spir.Ops dans les systèmes d'intelligences artificielles décisionnelles et simulations de foules.

Contribution : Nous présentons un système procédural qui génère des animations de locomotion pour des dizaines des créatures à n-pattes, en temps réel et sans données de mouvement (MoCap). Notre système est générique et capable d'animer différentes morphologies. Il est capable d'adapter l'animation générée à des environnements complexes et dynamiques, en temps réel, ce qui donne la liberté de déplacement aux créatures simulées. Notre système est hautement contrôlable afin de permettre à l'utilisateur de configurer tous les aspects de l'animation finale pour générer le style de locomotion voulu.

Chapitre 2 : techniques d'animation - état de l'art

Les créatures virtuelles sont souvent représentées par un maillage de polygones et animées par un squelette. Un squelette articulé est constitué de plusieurs corps rigides reliés par des articulations (figure 1).

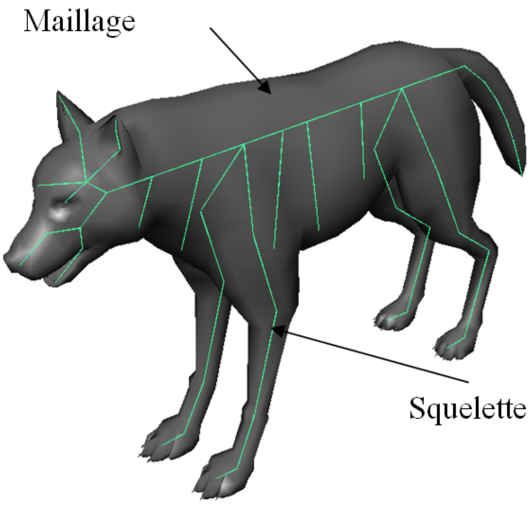


Figure 1: *Example d'une créature virtuelle avec son maillage et squelette.*

Il existe de nombreuses techniques pour animer ce squelette. Multon et al. [MFCD99] et van Welbergen et al. [vWvBE+10] ont identifié deux groupes principaux: **les techniques pilotées par les données** et **les techniques procédurales**. Chacune de ces techniques (et leurs sous-catégories) présente des avantages et des inconvénients et offre un compromis entre : le contrôle, la naturalité du mouvement généré et le temps de calcul.

Les techniques pilotées par les données utilisent des données de mouvement pour produire l'animation des personnages virtuels. Ces données d'animation sont généralement acquises (MoCap) ou créées manuellement (Keyframe data). Ces techniques sont les plus naturelles, car elles gardent le style du mouvement implicite du sujet capturé. Et elles sont bien adaptées à des applications temps réel puisque les données d'animation sont rejouées en temps réel sans coûts de calcul supplémentaires. Néanmoins, **les techniques pilotées par les données** souffrent d'un manque de contrôle, car les données du mouvement sont liées à une morphologie particulière et à un contexte spécifique. Plusieurs techniques sont proposées pour résoudre ces problèmes.

- Motion Blending : fusionner plusieurs données de mouvement pour produire une nouvelle animation. Ce mélange est en fonction des poids qui varient dans le temps [HKG06, BMJC01, KG04].
- Retargeting : le processus de transfert des données de mouvement entre de différentes morphologies [Gle97, PW99, CC10].
- Motion Warping : l'utilisation des commandes d'utilisateur et/ou l'environnement pour modifier les trajectoires absolus des données d'animation [Gle01, CKHL11, KHKL09].

La nécessité de plusieurs couches de traitement lorsque le contexte ou la morphologie change et le manque de données d'animations sources pour les créatures visées dans cette thèse nous a fait dévier de cette catégorie de techniques.

En revanche, **les techniques procédurales** n'utilisent pas de données d'animation. L'animation finale est générée en utilisant soit les méthodes cinématiques (l'utilisation des formules mathématiques et des algorithmes qui sont généralement inspirés des données empiriques) soit les méthodes physiques (l'utilisation des équations physiques pour calculer les forces et les torsions qui dirigent le mouvement d'une figure articulée).

Les techniques procédurales sont plus génériques et s'adaptent mieux au contexte et aux morphologies. L'utilisation des équations physiques dans les méthodes physiques a permis à de nombreux systèmes d'animation de capturer le réalisme de mouvement en simulant la physique du monde réel. Les animations générées sont crédibles, avec des effets émergents naturels [KKI02, dLMH10]. Mais le nombre élevé des équations utilisées dans ces simulations (plusieurs degrés de liberté dans un squelette articulé et plusieurs contraintes à satisfaire) nécessite l'utilisation de *méthodes d'optimisation* avec des fonctions de coût agissant sur une ou plusieurs contraintes (minimiser l'énergie, minimiser les moments angulaires, *etc.*) [LP02, PH05, dL11]. D'autres systèmes utilisent des méthodes de *simplification* afin de réduire la complexité de ce problème non linéaire : IPM, SLIP model [KKK^{+}02, PT06, MdLH10], Feedback-Based (comme SIMBICON) [YLvdP07, CBvdP10], système de vibration naturelles [KRFC09, NKZ12].

Les méthodes physiques souffrent donc de problèmes de temps de calcul, car elles ne peuvent pas animer plus de deux personnages en même temps, ainsi que de problèmes de contrôlabilité, car l'animation finale est contrôlée d'une manière implicite en utilisant seulement les forces et les torsions.

Finalement, les méthodes cinématiques sont le meilleur choix dans le contexte de cette thèse. En effet, elles offrent un bon compromis entre la quantité de contrôle sur le mouvement, la crédibilité de l'animation résultante et le temps de calcul. Des systèmes cinétiques comme le Versatile Walk Engine [BUT04] génèrent l'animation de plusieurs bipèdes en même temps et avec plusieurs styles, ou le PODA Animation System [GM85, Gir87] qui est capable d'animer des créatures de différentes morphologies avec un minimum de contrôle utilisateur. Dans les méthodes cinématiques, l'animation produite est contrôlée explicitement, ce qui signifie la production de mouvement exacte pendant le temps attribué. L'utilisation de la biomécanique et des données empiriques garantit la plausibilité de la locomotion produite. Et en se concentrant sur la crédibilité du mouvement [BHW96] plus que sur la précision de la simulation, et par rapport aux techniques basées sur la physique, ces techniques sont capables d'animer un nombre important de créatures, en temps réel.

Chapitre 3 : animation de créatures à n-pattes

La plupart des animaux terrestres se déplacent d'un endroit à l'autre en mettant un pied devant l'autre de manière successive jusqu'à atteindre le point d'intérêt (la cible). Lors du mouvement normal d'un pied, nous pouvons distinguer deux phases principales : phase au sol et phase de vol [INM66]. Pendant la phase d'appui, le pied reste bloqué au sol. Sinon, le pied vol selon une courbe parabolique sans aucun contact avec le sol. Un cycle de locomotion (démarche) est l'acte de répéter ces mouvements des pieds selon un certain rythme ou tempo (figure 2).

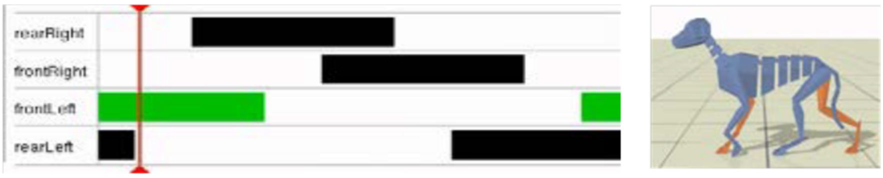


Figure 2: *Cycle de locomotion: les barres représentent la phase de vol.*

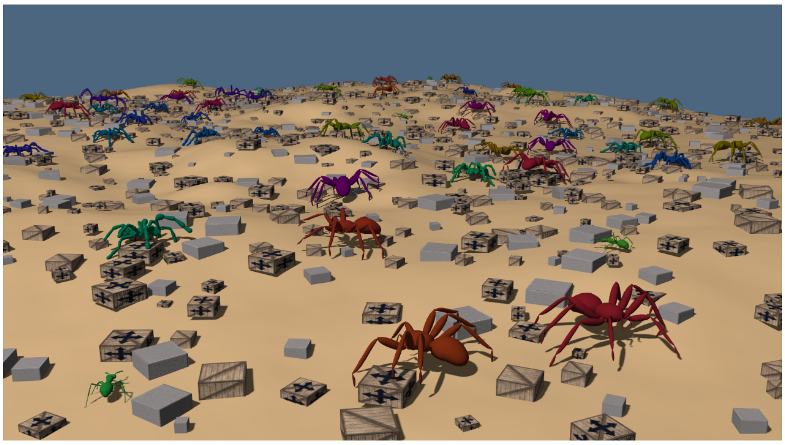


Figure 3: *Simulation finale.*

Notre système (figure 3) suit ces principes lors de la génération, procédurale et en temps réel, d'animation de locomotion des créatures à n-pattes. Son but principal est de satisfaire les quatre objectifs suivants :

- **Adaptabilité** : générer des animations qui s'adaptent aux différentes morphologies et aux enivrements complexes.

- **Contrôlabilité** : donner des outils aux utilisateurs pour générer le style de locomotion souhaitée.
- **Crédibilité** : la simulation finale doit être crédible pour créer une expérience immersive.
- **Efficacité** : un système capable d'animer en temps réel des dizaines de créatures.

Schéma général

Le système est composé de trois blocs principaux (figure 4):

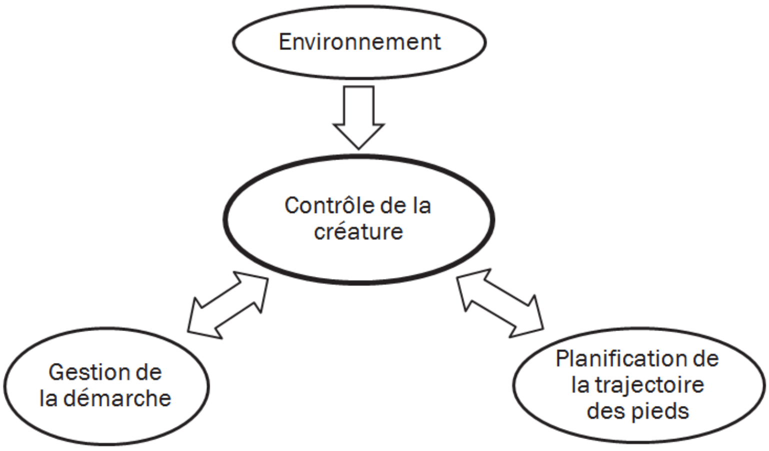


Figure 4: *Schéma général.*

- **Contrôle de la créature** : la structure centrale qui gère la locomotion globale de la créature à n-pattes. Elle repose sur les deux autres structures pour calculer le mouvement des pieds et le déplacement du bassin.
- **Gestion de la démarche** : cette structure impose le rythme du mouvement des pieds selon le tempo défini par l'utilisateur.
- **Planification de la trajectoire des pieds** : cette structure évalue pour chaque pied toutes les cibles et trajectoires possibles en temps réel, et choisit parmi toutes les possibilités le meilleur couple (trajectoire, cible), c'est-à-dire

la meilleure trajectoire 3D qui traverse l'environnement vers la meilleure cible (empreinte du pied). Elle utilise une structure de **construction de trajectoire en 3D** pour calculer la trajectoire 3D d'un pied à l'aide d'une représentation efficace discrète de l'environnement. Cette représentation est maintenue en temps réel et mise à jour en prenant en compte les objets dynamiques.

Nous utilisons le système de cinématique inverse (Inverse Kinematics (IK)) Cyclic-Coordinate Descent (CCD) (figure 5) pour calculer la position des articulations intermédiaires des jambes. En animant le bassin, les pieds et les jambes avec ces trois blocs et le CCD, nous générons une animation complète du bas du corps, ce qui correspond à une animation du corps entier pour la plupart des créatures que nous traitons.

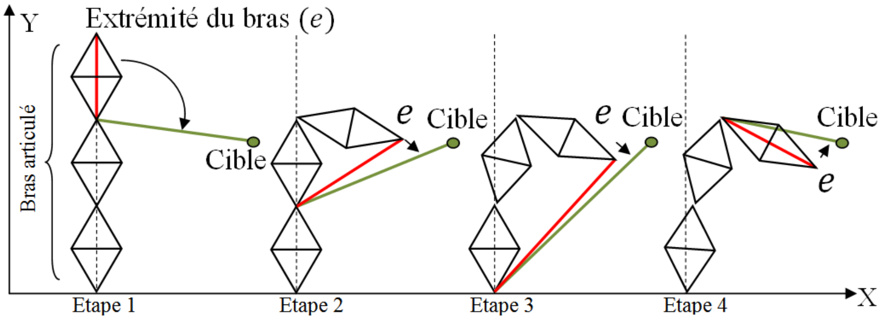


Figure 5: *De gauche à droite : résolution d'un problème de cinématique inverse en utilisant le CCD. Le but est de calculer les angles pour que l'extrémité du bras (e) atteigne la cible.*

Contrôle de la créature

Elle est en charge de deux tâches principales.

- Gestion du mouvement des pieds : à chaque pas de simulation et selon la démarche, certains pieds vont entrer en phase de vol. Pour chacun de ces pieds, cette structure calcule une empreinte préférée selon la vitesse de la créature et l'environnement. Ensuite, elle appelle la **planification de la trajectoire des pieds** pour trouver le meilleur couple. Pour les pieds en phase au sol, cette structure les bloque au sol ou sur un obstacle.

- Calcul du mouvement 3D du bassin : le mouvement 2D du bassin sur le plan horizontal (Y est vers le haut) est calculé en utilisant la vitesse et l'orientation. L'élévation et l'inclinaison du bassin sont calculées à partir de l'environnement en dessous de la créature.

Gestion de la démarche

Cette structure organise et visualise le tempo des pieds et effectue les transitions entre différents styles de mouvements. Comme la locomotion est cyclique, il nous semblait naturel de représenter la démarche avec des cercles. Comme illustré sur la figure 6, chaque cercle représente un pied et sa phase de vol est représentée par le secteur coloré. L'aiguille active et désactive les secteurs en fonction de sa position actuelle. L'activation d'un secteur signifie que le pied correspondant devrait entrer dans sa phase de vol.

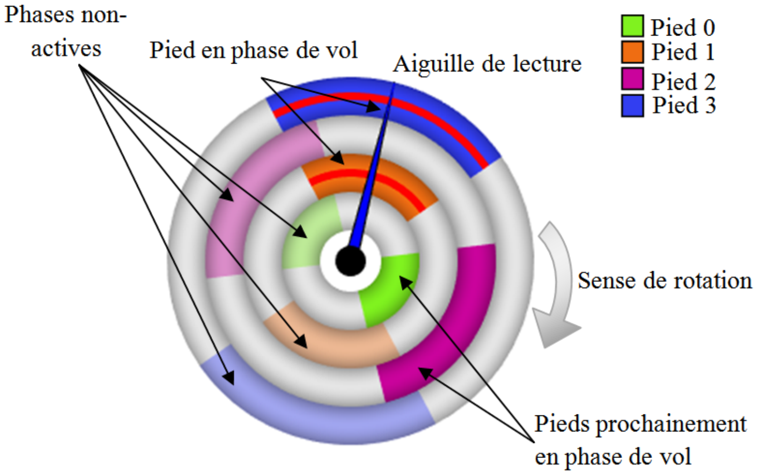

Phases non-actives
Pied en phase de vol
Aiguille de lecture
🟩 Pied 0
🟧 Pied 1
🟪 Pied 2
🟦 Pied 3
Sense de rotation
Pieds prochainement en phase de vol

Figure 6: *Gestion de la démarche : avec cette interface, l'utilisateur peut créer le tempo souhaité.*

Construction de trajectoire en 3D

L'environnement de travail est assez complexe comme le montre la figure 7. Il est généré en utilisant une carte de hauteur avec plusieurs types d'obstacles, ce qui signifie une énorme quantité de triangles pour représenter et rendre cet environnement.

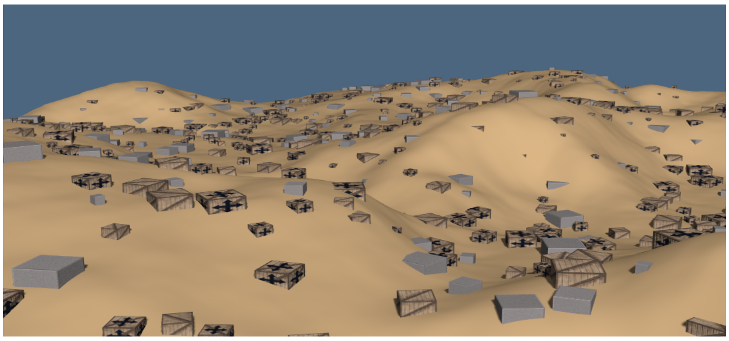


Figure 7: *Un exemple d'un environnement complexe et dynamique que le système peut avoir comme entrée.*

Nous discrétisons l'environnement 3D en deux grilles 2D. La carte d'obstacles décrit les zones navigables de l'environnement pour les pieds, comme illustré sur la figure 8. Les cellules noires représentent les obstacles et sont appelées les cellules interdites. La carte d'élévations contient l'élévation de l'obstacle le plus haut dans chaque cellule (figure 9).

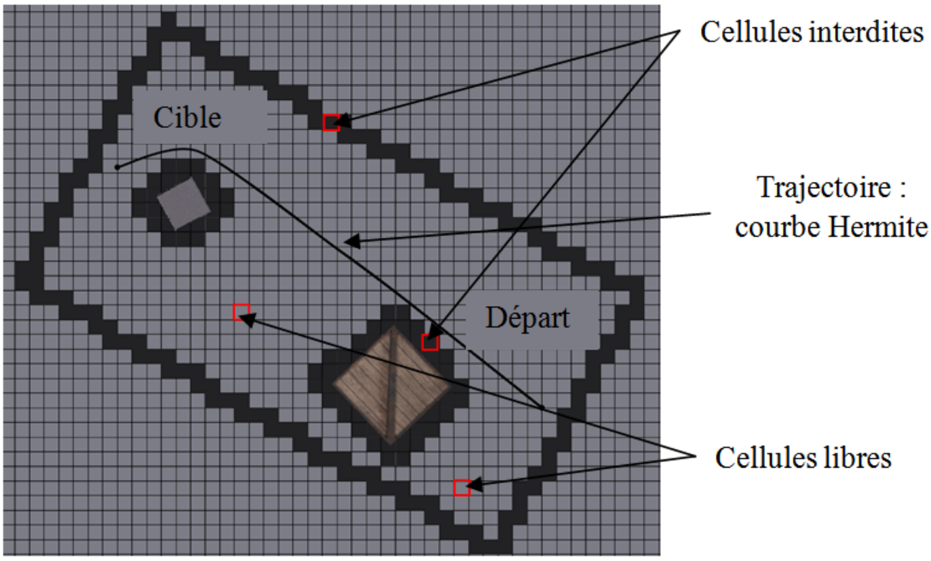


Figure 8: *Carte d'obstacles.*

Nous calculons d'abord la trajectoire en 2D sur le plan horizontal à l'aide des projections de la source et de la cible sur ce plan. Notre planificateur discrétise le plan et trouve le plus court chemin avec l'algorithme WaveFront [Kha86], couplé avec une courbe B-spline (la courbe d'Hermite dans la figure 8). En

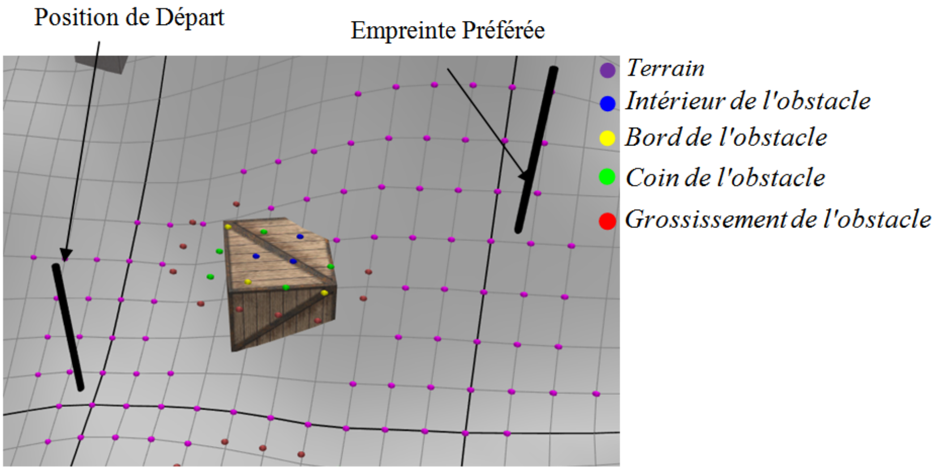


Figure 9: *Carte d'élévations.*

échantillonnant cette courbe 2D et en utilisant la carte d'élévations, nous calculons l'élévation de chaque échantillon, ce qui nous donne la trajectoire 3D qui traverse l'environnement sans collision (figure 10).

Planification de la trajectoire des pieds

C'est l'une des contributions importantes de cette thèse. Chaque trajectoire 3D peut contourner ou passer au-dessus de chaque obstacle, ce qui génère de multiples trajectoires vers une seule cible. En plus, notre algorithme évalue toutes les cibles (empreintes de pied) autour de l'empreinte préférée désignée par le **contrôle de la créature**. Notre espace de recherche est l'ensemble des trajectoires possibles qui vont du point de départ vers toutes les cibles possibles. Notre algorithme choisit le meilleur couple (empreinte, trajectoire), en temps réel. L'algorithme explore l'espace des couples solutions et utilise les scores partiels des trajectoires et des cibles pour limiter le nombre de solutions explorées. Le résultat final est illustré sur la figure 10. Le temps d'exécution de cet algorithme est facilement contrôlable ce qui nous permet d'implémenter des techniques de niveau de détail (Level of Detail (LOD)) pour accélérer les calculs pour les créatures loin ou derrière la caméra. En implémentant ces techniques de LOD, nous arrivons à augmenter le nombre de créatures à n-pattes animées en temps réel.

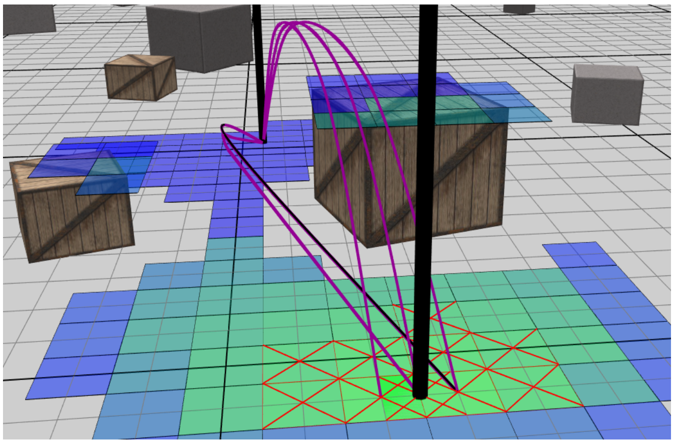


Figure 10: *Les cibles potentielles : la couleur varie du vert (meilleure cible) au bleu (pire cible). Les cellules marquées d'une croix sont les cellules traitées par notre algorithme, en rose les trajectoires possibles et en noir la trajectoire choisie.*

Performance et résultat

Notre système est capable d'animer une grande variété des créatures à n-pattes (figure 11). L'animation finale est générée automatiquement avec un contrôle total sur plusieurs paramètres de locomotion comme par exemple la vitesse de déplacement souhaitée, l'orientation, la démarche, *etc.* Le temps de calcul moyen, sur chaque étape de simulation, pour 100 créatures à 8-pattes est de l'ordre de 0.029 seconde dans un environnement composé d'une carte de hauteur de taille 100m2 et d'obstacles (3000 boîtes, 40% dynamiques). La simulation finale (figure 3) est réalisée en temps réel, 30 images par seconde (Frames Per Second (fps)).

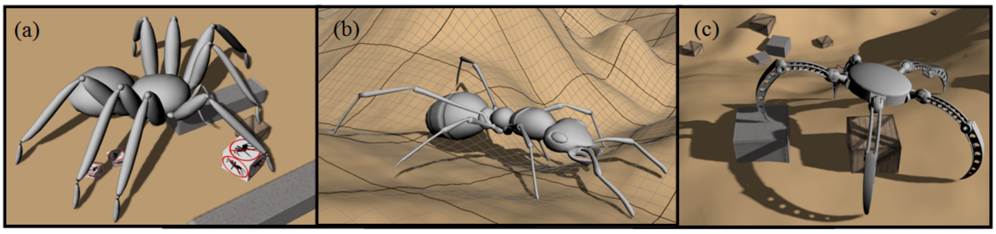


Figure 11: *Types de morphologie simulés.*

Chapitre 4: vers des animations plus naturelles

Le système proposé produit des animations crédibles, mais en comparaison avec des créatures naturelles il manque certains effets indispensables au réalisme. L'objectif principal de ce chapitre est d'améliorer le réalisme en ajoutant les composants suivants : mouvement pseudo physique du bassin et une colonne vertébrale flexible.

Mouvement pseudo physique du bassin

La trajectoire du bassin chez les humains [INM66] et la plupart des animaux [Muy57] est sinusoïdale. Pour générer automatiquement ce mouvement sinusoïdal, nous utilisons une particule pour représenter le bassin afin de simplifier les équations physiques. Ce mouvement sur l'axe Y est régi par la force de gravité qui pousse vers le bas et les forces des pieds qui poussent vers le haut. Nous traitons chaque pied indépendamment des autres, comme si la particule du bassin se trouvait sur un seul pied avec un ressort à la place de la jambe (un *pogostick* sur la figure 12) et ce pied supporte la masse (m) entière du bassin. Ce pied pousse vers le haut avec une certaine quantité de force pendant la phase au sol.

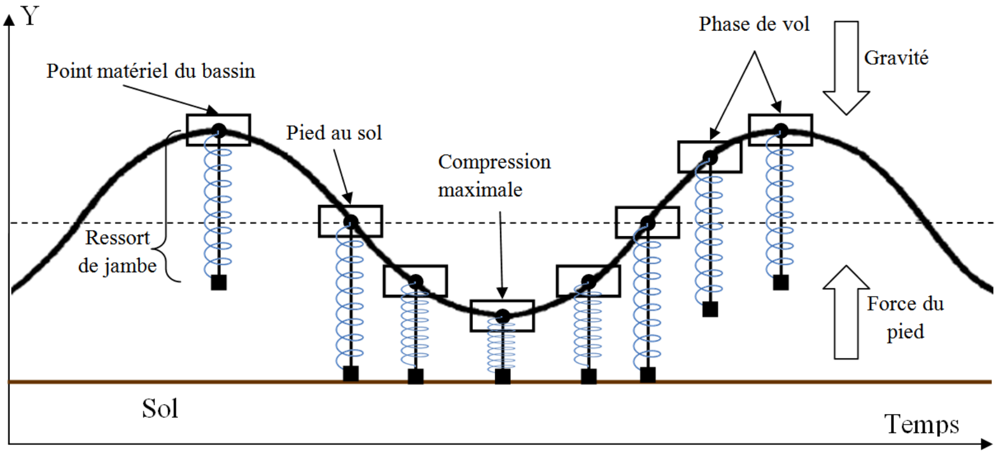


Figure 12: *La trajectoire de la particule du bassin quand un pogostick est utilisé pour représenter sa relation avec chaque jambe.*

Nous utilisons les lois du mouvement de Newton pour calculer la force exercée par chaque pied (i) sur l'axe Y:

$$m \cdot a_{\ bassin} = W + F_{\ pied} \tag{1a}$$

$$m \cdot a_{\ bassin} = -m \cdot g + m \cdot a_{\ pied} \tag{1b}$$

$$\tag{1c}$$

$a_{\ pied}$ est l'accélération du pied (i). Comme g est une force négative, l'équation 1b devient :

$$a_{\ bassin} = -g + a_{\ pied} \tag{2}$$

Nous savons que : $a_{\ bassin} = \frac{v_T - v_C}{T}$ avec T une durée, v_T la vitesse souhaitée après la durée et v_C la vitesse actuelle. En remplaçant dans l'équation 3 :

$$a_{\ pied} = g + \frac{v_T - v_C}{T} \tag{3}$$

L'accélération $a_{\ pied}$ représente la force réelle que le pied va exercer sur la particule du bassin. v_T et T sont définis par la démarche. Cela signifie un contrôle transparent pour l'utilisateur comme il n'a pas besoin d'ajuster des paramètres supplémentaires. Notre système calcule la force de poussée pour chaque pied de manière indépendante. Ensuite, la particule du bassin intègre toutes ces forces des pieds (a_i) pour générer son mouvement sinusoïdal (figure 13).

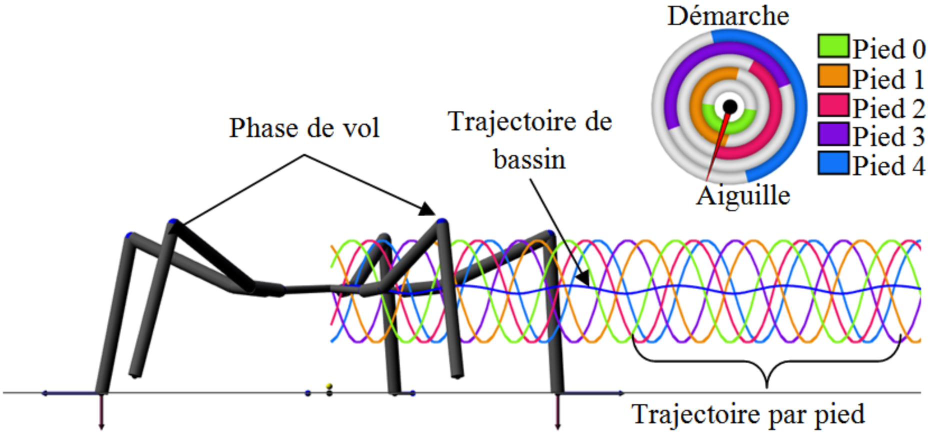


Figure 13: *Trajectoire finale du bassin.*

Colonne vertébrale flexible

Les quadrupèdes ont une colonne vertébrale flexible, leur octroyant une grande agilité, et donc une variété de mouvement étendue. Nous allons d'abord décomposer le bassin de la créature en plusieurs noeuds virtuels de bassin (des épaules), illustré sur la figure 14. Chaque pied est relié à l'un de ces noeuds, à l'exception de la tête. Ces noeuds de bassin sont indépendants (hauteur, inclinaison, emplacement des pieds, *etc.*) et connectés avec notre modèle de colonne vertébrale flexible.

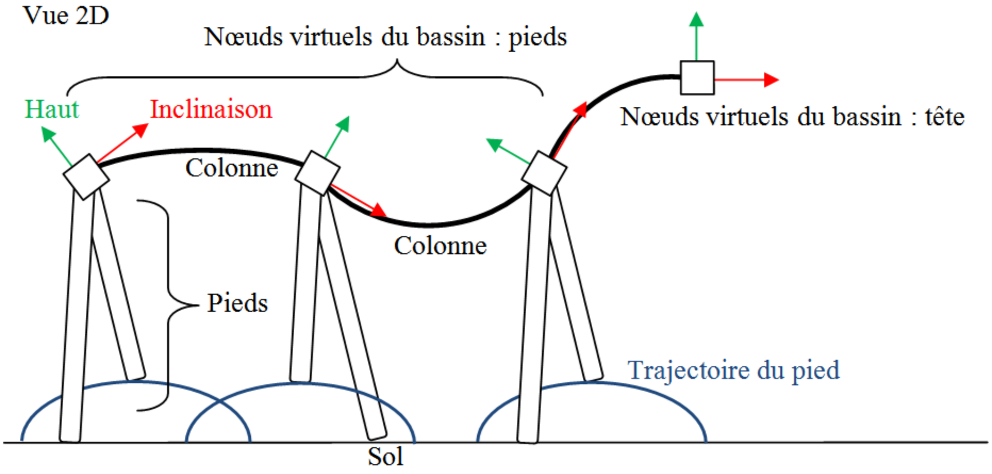

Vue 2D
Nœuds virtuels du bassin : pieds
Haut
Inclinaison
Colonne
Colonne
Nœuds virtuels du bassin : tête
Pieds
Trajectoire du pied
Sol

Figure 14: *Les noeuds virtuels du bassin et la colonne vertébrale.*

La colonne vertébrale est calculée par quatre étapes successives (figure 15). Tout d'abord sur le plan horizontal, puis sur le plan sagittal pour la simplification. Sur le plan horizontal, nous nous concentrons sur l'orientation 2D et la translation, tandis que sur le plan sagittal, nous nous concentrons sur l'élévation (avec notre système pseudo physique) et l'inclinaison.

Dans l'étape finale, nous combinons les positions 2D calculées dans les étapes précédentes et les élévations calculées dans l'étape 3 pour obtenir les positions 3D pour chaque noeud du bassin virtuel. En utilisant les orientations 2D calculées précédemment et l'inclinaison calculée dans l'étape 3, nous obtenons les tangentes dans le plan sagittal. Une courbe B-Spline (Hermite) est construite avec ces données pour générer la colonne vertébrale finale (figure 16 et figure 17).

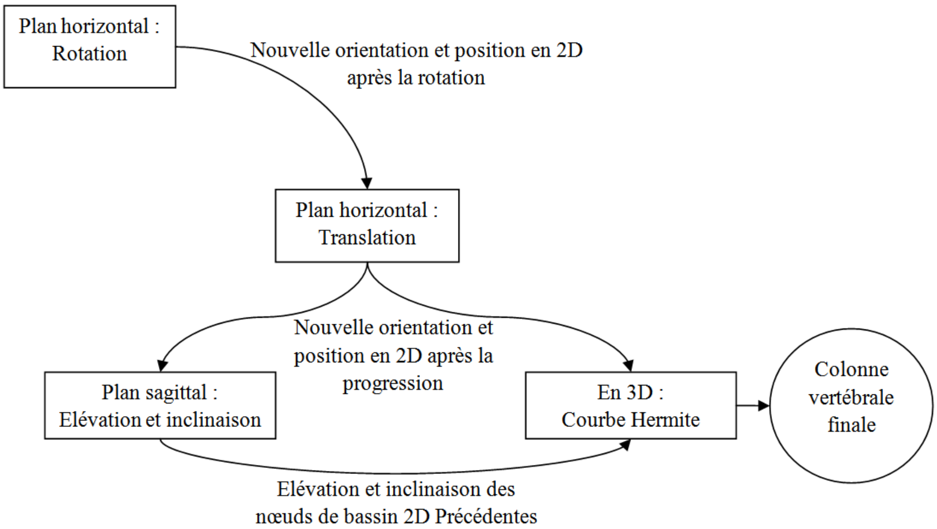

Plan horizontal :
Rotation
Nouvelle orientation et position en 2D
après la rotation
Plan horizontal :
Translation
Nouvelle orientation et
position en 2D après la
progression
Plan sagittal :
Elévation et inclinaison
En 3D :
Courbe Hermite
Colonne
vertébrale
finale
Elévation et inclinaison des
nœuds de bassin 2D Précédentes

Figure 15: *Les étapes de calculs de la colonne vertébrale.*

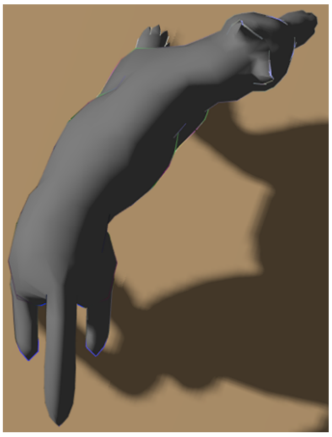


Figure 16: *Colonne vertébrale d'un loup.*

Performance et résultat

Notre système reste temps réel après l'ajout de ces composants. De plus, l'animation générée devient plus crédible pour les quadrupèdes avec colonnes vertébrales flexibles (figure 18).

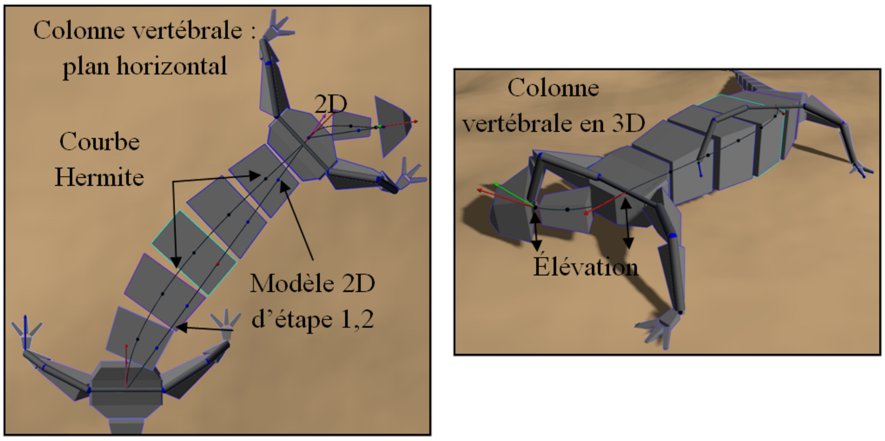

Colonne vertébrale :
plan horizontal
2D
Courbe
Hermite
Modèle 2D
d'étape 1,2
Colonne
vertébrale en 3D
Élévation

Figure 17: *Colonne vertébrale d'un lézard.*

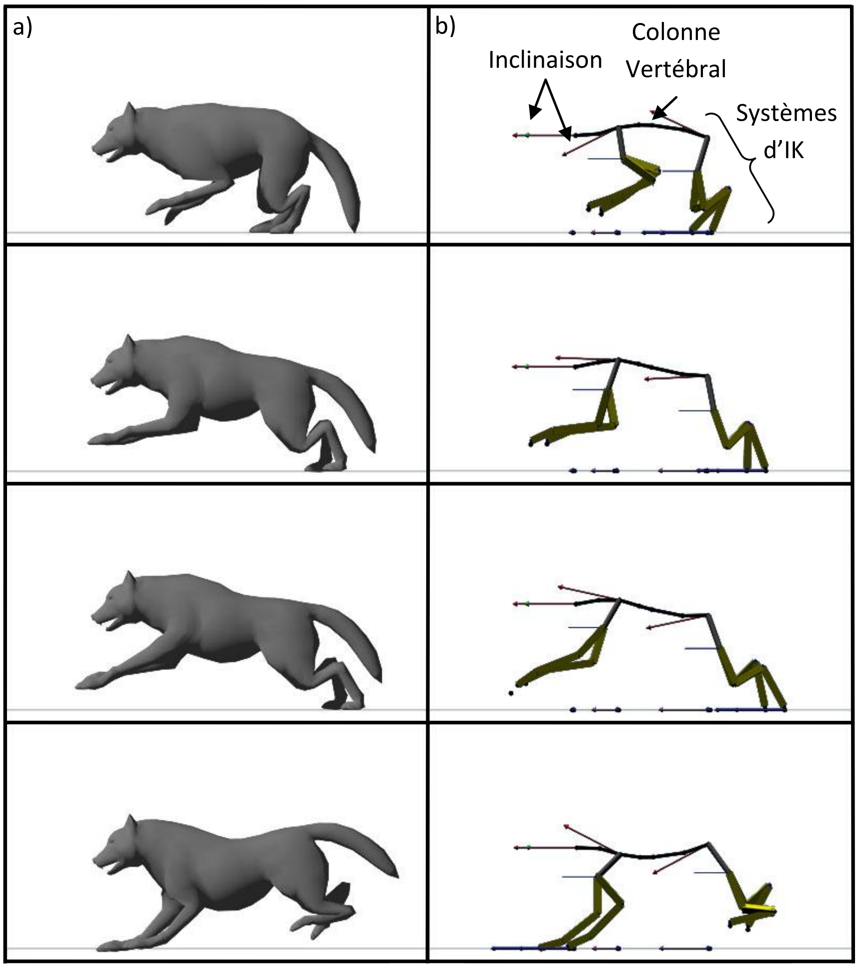

a)
b)
Inclinaison
Colonne
Vertébral
Systèmes
d'IK

Figure 18: *Un loup qui court avec sa colonne vertébrale flexible.*

Chapitre 5 : effet d'oscillation contrôlé

Nous proposons un système original qui ajoute des effets d'oscillation à n'importe quel objet à base de squelette, en temps réel avec un contrôle complet en utilisant des pendules 3D (figure 19). Notre système génère ces oscillations en utilisant seulement la trigonométrie et sans recours à des équations physiques, ce qui le rend plus contrôlable et stable. Les pendules que nous proposons ont trois degrés de contrôle : temps de réaction, amortissement et direction repos (figure 19(a)). Ils sont utilisés afin d'ajouter des mouvements secondaires [HOZ97] : effets d'oscillation et de vibration de la queue d'un loup par exemple. Un pendule est une barre avec de longueur fixe L, attiré vers sa direction de repos par un ressort. Nous découpons ces pendules 3D en 2 ressorts sur chaque plan 2D (voir figure 19(b)). Nous avons choisi ce système pour éviter le mouvement de rotation en spirale autour de la direction de repos.

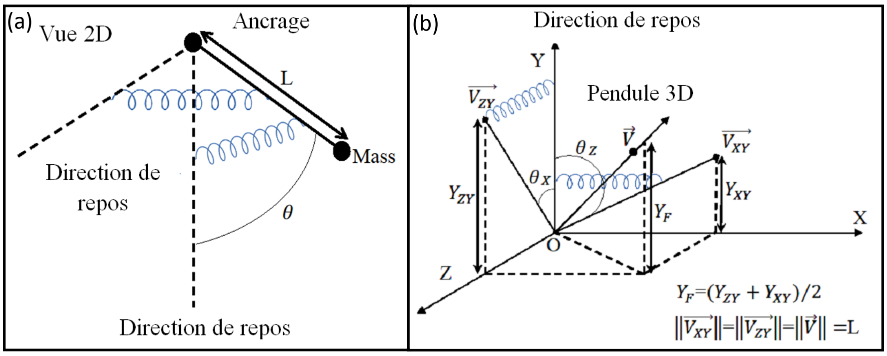

(a)
Vue 2D
Ancrage
L
Mass
Direction de repos
θ
Direction de repos
(b)
Direction de repos
Y
Pendule 3D
$\overrightarrow{V_{ZY}}$
$\vec{V}$
θz
θx
$\overrightarrow{V_{XY}}$
Y_{ZY}
Y_{XY}
Y_F
X
O
Z
$Y_F=(Y_{ZY}+Y_{XY})/2$
$\|\overrightarrow{V_{XY}}\|=\|\overrightarrow{V_{ZY}}\|=\|\vec{V}\|=L$

Figure 19: *(a) Représentation simplifiée des pendules, avec un ressort et une direction de repos. (b) Pendule 3D.*

Contrôle temporel des ressorts

L'équation de mouvement d'un ressort est :

$$\ddot{x} = -(k(x - x_0) + c\dot{x})/m \tag{4}$$

Pour une position donnée x de masse m, le ressort va osciller autour de la position de repos x_0, en cherchant à minimiser l'erreur $(x - x_0)$ jusqu'à atteindre zéro.

Cette oscillation dépend directement des constantes (k, c, m). Pour le contrôle temporel, nous utilisons le principe de temps d'établissement (*settling time*) : le temps nécessaire pour que la position x de masse m atteigne son amplitude maximale à l'intérieur d'un intervalle d'erreur donné (figure 20). Avec le principe de temps d'établissement et en supposant que $m = 1$, nous arrivons à calculer les constantes (k, c) en utilisant l'amortissement et le temps de réaction seulement, ce qui nous permet de contrôler parfaitement le mouvement du ressort.

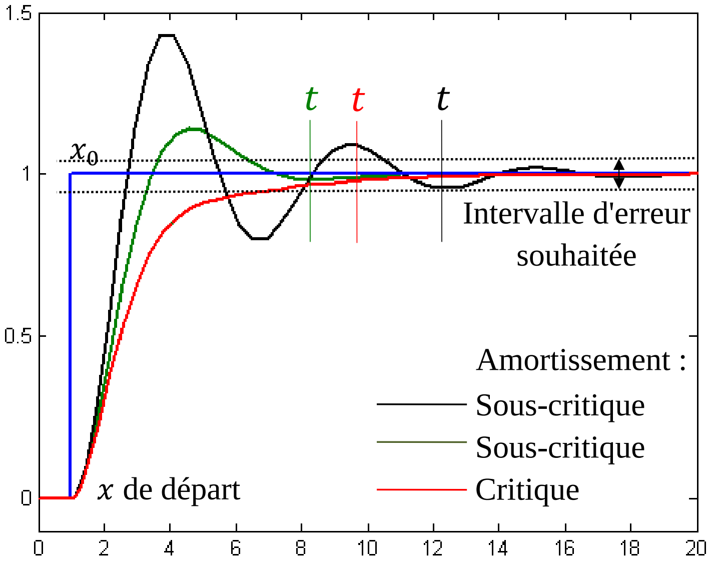


Figure 20: *Oscillation des ressorts différents avec des amortissements différents.*

Les stratégies d'interaction

Le squelette est un arbre d'articulations. En connectant un pendule 3D entre chaque articulation et en définissant la direction de repos de chacun de ces pendules, nous obtenons un arbre de pendules (voir figure 21). Certains pendules 3D agissent comme un noeud père pour d'autres pendules. Quand ils se déplacent, les points d'ancrage de leurs fils se déplacent.

Ces pendules ont besoin d'interagir les uns avec les autres afin d'avoir une réaction crédible. Par exemple les noeuds fils doivent se déplacer selon le mouvement de leur noeud père. Nous définissons deux stratégies utilisées en conjonction pour atteindre notre objectif :

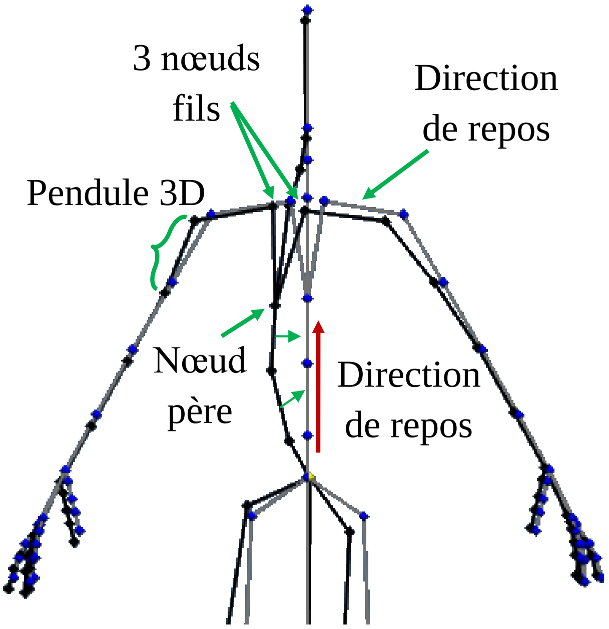


Figure 21: *Arbre des pendules 3D.*

- Stratégie de poursuite du père. L'objectif de cette stratégie est de propager le mouvement du pendule père vers ses enfants, pour intégrer ce mouvement dans leur propre mouvement.
- Stratégie de poursuite des fils. L'objectif de cette stratégie est de refléter la perturbation qui peut se produire sur le niveau fils vers son noeud père.

Notre algorithme final effectue les calculs de l'arbre des pendules en deux passes linéaires. Une passe de bas en haut pour propager les perturbations au niveau des noeuds fils vers les noeuds pères et une autre passe de haut en bas pour refléter les mouvements des noeuds père vers leurs fils. Avec ce système simple, nous arrivons à générer des mouvements crédibles et contrôlables en temps réel.

Résultat

Nous avons ajouté ces effets d'oscillation aux antennes de nos créatures (fourmis, araignées, *etc.*) et à la queue du loup ou d'un lézard comme l'illustre la figure 22. Nous avons également ajouté ces oscillations au corps de fourmis/araignées pour leur donner un effet moins rigide lors des déplacements.

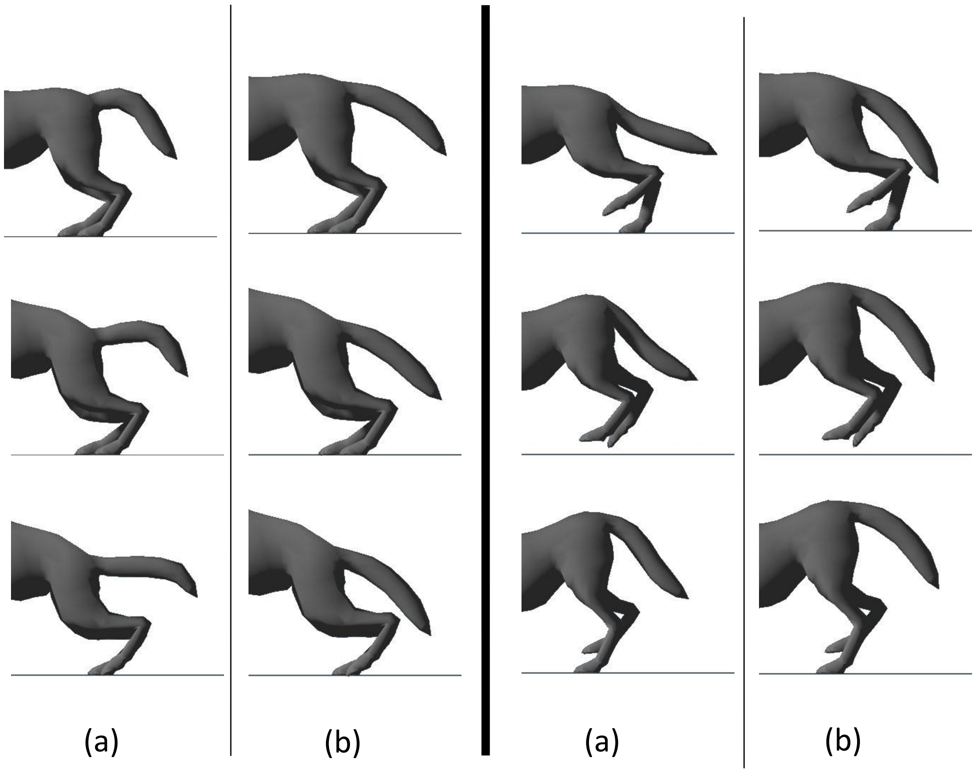


Figure 22: *Image par image de l'animation de la queue de loup après l'ajout de mouvement secondaire. (a) Simulation d'une queue souple. (b) Simulation d'une queue plus rigide.*

Chapitre 6 : conclusion

Dans cette thèse, nous avons présenté un système de locomotion capable d'animer, en temps réel, une grande variété de morphologies de créatures à n-pattes. Notre système de locomotion satisfait quatre objectifs principaux. Il est capable d'adapter l'animation générée à un environnement complexe dynamique et à de différentes morphologies. L'utilisateur a un contrôle total sur la locomotion finale et peut concevoir le style de locomotion désiré à travers nos interfaces. Le système génère des animations crédibles et réalistes. Enfin, il est suffisamment efficace pour simuler des dizaines de créatures à n-pattes en temps réel.

Contents

Contents xxxi

List of Figures xxxv

1 Introduction 1
- 1.1 Motivation 1
- 1.2 Contribution 3
- 1.3 Industrial Context 4
- 1.4 Thesis Organization 5

2 State Of The Art 7
- 2.1 Data-Driven Techniques 9
- 2.2 Procedural-Based Techniques 14
- 2.3 Physics-Based Techniques 19
- 2.4 Hybrid Techniques 27
- 2.5 Multi-Legged/Non-Human Characters 30
- 2.6 Path and Motion Planning 32
- 2.7 Conclusion 38

3 Animating Multi-Legged Characters 41
- 3.1 System Overview 43
- 3.2 Character Controller 47
  - 3.2.1 Feet Movement Task 47
  - 3.2.2 3D Pelvis Movement Task 48
- 3.3 Gait Manager 51

3.4 3D Path Constructor 53
  3.4.1 Environment Grid-Based Representation 54
  3.4.2 3D Path Construction 57
3.5 Footprints and Feet Path Planning 61
  3.5.1 Potential Footprints and Trajectories scoring 62
  3.5.2 Finding Best Couple 63
3.6 System Loop 66
  3.6.1 Sequence 1 66
  3.6.2 Sequence 2 67
  3.6.3 Sequence 3 68
  3.6.4 Sequence 4 69
3.7 Level of Details 70
3.8 Performance and Results 71
3.9 Conclusion 78

4 Adding Naturality and Realism 79
4.1 Pelvis Movement Using Pseudo Physics 80
4.2 Spine Model 86
  4.2.1 Step 1: Spine Orientation 89
  4.2.2 Step 2: Spine Advancing 90
  4.2.3 Step 3: Spine Elevation and Pitch 91
  4.2.4 Step 4: Final 3D Spine 92
4.3 Other Visual Effects 93
4.4 Performance and Results 95
4.5 Conclusion 99

5 Controlled Oscillation Effects 101
5.1 Pendulums System Overview 104
5.2 Time Based Spring Dampers Control 105
5.3 Pendulums Strategies 107
  5.3.1 Father Pursuit Strategy 108
  5.3.2 Son Pursuit Strategy 109
  5.3.3 Final workflow 110

5.4 Advantages . . . 111
5.5 Implementation in Animation Systems . . . 113
5.5.1 Adding Physical Reaction Effects to any Skeleton-Based Bodies . . . 113
5.5.2 Cloth Simulation . . . 115
5.5.3 Secondary Motion in Locomotion System: Results and Performance . . . 117
5.6 Conclusion . . . 119

6 General Conclusions 121

Glossary 125

Appendices 131

A PD Controller 133

B Physics Engines Responses 135

C Depth-First vs Breadth-First Search 137

D Locomotion System Parameters 139

E Main Algorithm 141

F Hermite Curve 143

References 145

List of Figures

1 Modèle 3D xi
2 Démarche xiv
3 Simulation finale xiv
4 Schéma général xv
5 Cinématique inverse CCD xvi
6 Gestion de la démarche xvii
7 L'environnement de travail xviii
8 Carte d'obstacles xviii
9 Carte d'élévations xix
10 Trouver le meilleur couple xx
11 Variété des morphologies xx
12 Trajectoire de la particule du bassin xxi
13 Trajectoire finale du bassin xxii
14 Noeuds virtuels du bassin xxiii
15 Etapes de calculs de la colonne vertébrale xxiv
16 Colonne vertébrale d'un loup xxiv
17 Colonne vertébrale d'un lézard xxv
18 Image par image d'un loup xxv
19 Pendule 2D et 3D xxvi
20 Contrôle temporel des ressorts xxvii
21 Arbre des pendules 3D xxviii
22 Image par image de la queue d'un loup xxix

1.1 Multi-Legged Characters in Video Games 2

2.1 Skeleton Example: a Wolf 8

2.2 MoCap Example 8
2.3 Motion Blending-1 10
2.4 Motion Blending-2 11
2.5 Retargeting 12
2.6 Motion Path Editing 13
2.7 Motion Graphs 13
2.8 Example of a Gait Pattern 15
2.9 Anatomy Planes 16
2.10 IK Problem 17
2.11 Cyclic-Coordinate Descent Method 18
2.12 Example of a MoCap based IK System 19
2.13 Skeleton Examples 20
2.14 Walk Cycle FSM 21
2.15 QP Example 23
2.16 Inverted Pendulum Model 24
2.17 Spring Loaded Inverted Pendulum Model 25
2.18 Feedback Based Controller 26
2.19 PD Controller Introduced Delay 28
2.20 Adding Responses to motion data 29
2.21 Morphology Variations in Spore 31
2.22 Path Planning Problem 33
2.23 Path Planning Roadmap 33
2.24 Path Planning Grid 34
2.25 Grasp Based Planning 36

3.1 Simulation Snapshot 1 42
3.2 System Overview 43
3.3 System Overview Detailed 45
3.4 Spider and Wolf Morphological Parameters 46
3.5 Character Controller Feet Movement Task 47
3.6 Foot Preferred Footprint Target 48
3.7 Feet Spacing Interface 49
3.8 Pelvis Preferred Height Calculation 49

3.9 Feet Feedback 50
3.10 Gait Manager 51
3.11 Gait Interpolation 52
3.12 Environment from Heightmap 53
3.13 Obstacles Types 53
3.14 Complex and Dynamic Environment 54
3.15 3D Path Construction Steps 55
3.16 Obstacles' Map 56
3.17 Elevations Map 56
3.18 Obstacle Size Increase 57
3.19 Wavefront Algorithm 58
3.20 Wavefront Modification 59
3.21 Bresenham's algorithm 60
3.22 Final 2D plan 60
3.23 Final 3D plan 61
3.24 Potential Footprints 63
3.25 Main Algorithm Slices 65
3.26 Finding Best Couple 66
3.27 Sequence 1 Initial Position 67
3.28 Sequence 1 - 1 67
3.29 Sequence 1 - 2 68
3.30 Sequence 2 68
3.31 Sequence 3 69
3.32 Sequence 4 69
3.33 Color Coded Simulation LOD 71
3.34 Example of Animated Characters 72
3.35 Simulation Snapshot 2 72
3.36 Frame By Frame: Spider 73
3.37 Frame By Frame: Robot 74
3.38 Performance Char - 1 76
3.39 Performance Char - 2 76

4.1 Fixed Pelvis Problem and Solution 80

4.2 Human Walking Pelvis Movement 81
4.3 Dog Galloping Pelvis Movement 81
4.4 Pogo Stick Trajectory 82
4.5 Foot Pushing Phases 83
4.6 Pelvis Sinusoidal-Like Ballistic Movement 84
4.7 Per Foot Sinusoidal-like Ballistic Movement 86
4.8 Spine Deformation while Galloping 87
4.9 With and Without a Spine Results 88
4.10 Pelvis Virtual Nodes 88
4.11 Spine Model Calculation Workflow 89
4.12 Spine Rotation Steps 90
4.13 Spine Advancing Steps 91
4.14 Pelvis Virtual Nodes Preferred Height Calculation 92
4.15 Abstract Lizard Spine Model 93
4.16 Non-Linear Character Progression 94
4.17 Random Gait Examples 95
4.18 Frame By Frame: Lizard 96
4.19 Frame By Frame: Wolf 97
4.20 Average Calculation Time with Pseudo Physics 98
4.21 Average Calculation Time with Spine 98

5.1 Pendulums Uses Examples 102
5.2 A 2D Pendulum 103
5.3 The 3D Pendulum 105
5.4 Time Based Spring Dampers Control 106
5.5 3D Pendulums Tree 107
5.6 Father Pursuit Strategy 108
5.7 Son Pursuit Strategy 109
5.8 Animating a Lifeless Octopus Model 114
5.9 Adding Oscitation to MoCap 115
5.10 Cloth 3D Pendulums System 116
5.11 Cloth Reaction Sequence 116
5.12 Examples of Pendulums Integration 117

5.13 Wolf Tail Example 118
5.14 Pendulums Tree Added Calculation Time 118

6.1 Frame By Frame: Biped 124

A.1 PD Controller Loop 133

B.1 Example of Physics Engines Responses 135

C.1 Depth-First Search 137
C.2 Breadth-First Search 138

F.1 Hermite Curve 144

Chapter 1

Introduction

Contents

1.1 Motivation **1**

1.2 Contribution **3**

1.3 Industrial Context **4**

1.4 Thesis Organization **5**

1.1 Motivation

Real or imaginary animals make frequent appearances in video games (Figure 1.1), serious games [DAJ11], films and virtual world simulations. Animating these virtual multi-legged characters like quadrupeds, arachnids, insects, reptiles or any imaginary n-legged robots or creatures make these virtual worlds believable and more life-like. The interest in animating these multi-legged characters is gaining momentum in recent years in computer graphics and in robotics [SRH$^+$08, SRH$^+$09], even if most of the proposed systems focus on human-like (biped) characters [vWvBE$^+$10].

The most common task that these virtual multi-legged characters perform is locomotion: the ability to move freely in virtual worlds toward the points of interest. These displacement movements are quite essential in order to deliver a convincing simulation to the users. And, these locomotion animations should be as close as possible to their real-life counterparts, to create a better immersive experience. The produced animation quality is important, as well as the efficiency of the techniques with more and more real-time simulations (*e.g.* video games) populate their virtual worlds with many creatures, as shown in Figure 1.1. Artists in these simulations want to have a total control over the produced motion, in order to generate the needed and imagined user experience.

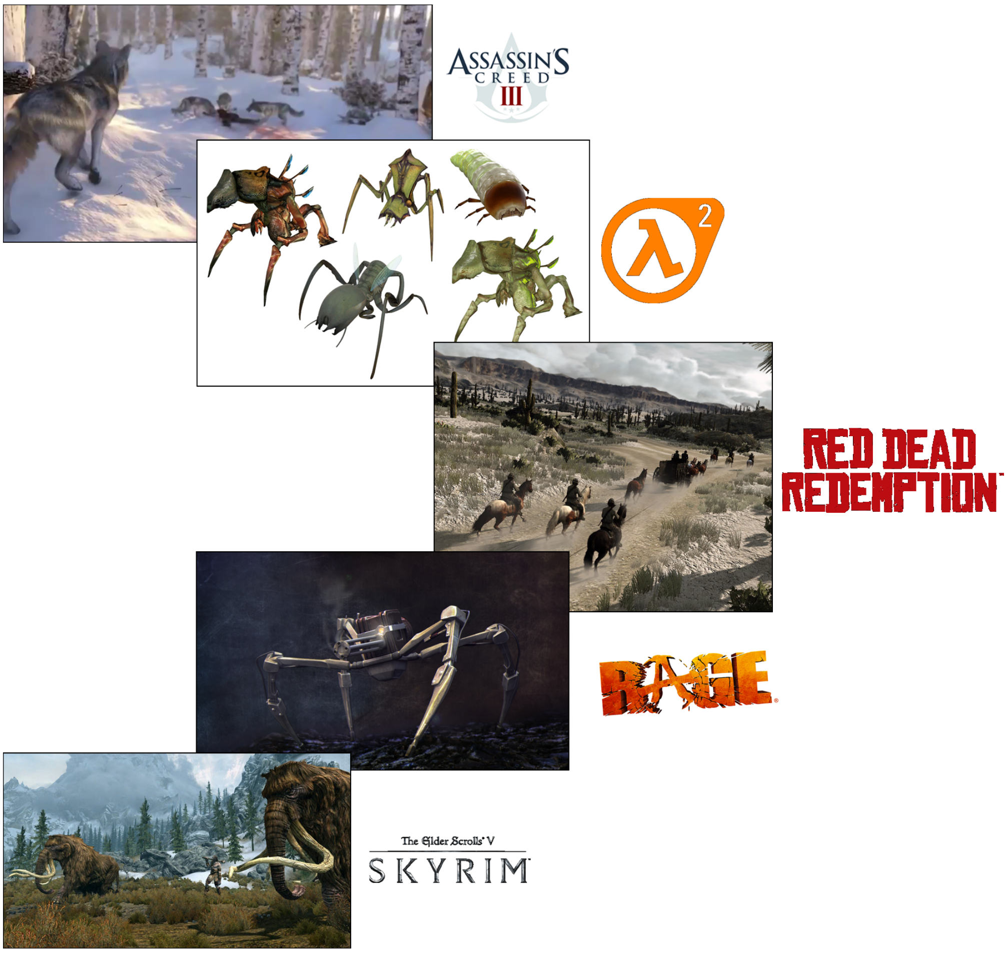


Figure 1.1: *Example of multi-legged characters in video games, From top to bottom: wolf packs in Assassin's Creed III, the imaginary antlion bugs in Half Life 2, horses in Read Dead Redemption, sentry bots in RAGE and finally mammoths in Skyrim.*

These locomotion animations are quite rich due to the variety of animated morphologies (a dog vs. a spider), gaits (galloping vs. trotting), body sizes and proportions (a wolf vs. a horse), *etc.* which is one of the first challenges when creating a generic reusable system capable of animating these different creatures with minimum configuration and tweaking. On top of that, these multi-legged characters normally navigate through complex environments with many obstacles often dynamic. So to produce a believable simulation, any animation system should adapt the motion to the surrounding environment and produce locomotion animations with obstacles overcoming, uneven terrain crossing, dynamic objects avoidance, *etc.*

Another challenge when modeling such motions arises from the lack of Motion Capture (MoCap) data inherent to either the difficulty to obtain them (insects) or the impossibility to capture them (like mammoths or 5-legged creatures). Even when motion data exists, it needs to be recaptured when the context changes. For example, motion data for a dog running on a plain terrain can not be used in an another simulation with irregular terrain: a recapture is needed.

1.2 Contribution

Locomotion animation is an essential part of any simulation and it is the main goal in this thesis, with four essential objectives. It needs to **Adapt** to complex environments and different morphologies. It needs to be **Controllable**, allowing the user to configure all aspects of final animation, giving him the tools to generate the desired locomotion style. Final simulation should be **Believable**, to create a better immersion experience. And, the system needs to be **Efficient**, to be used in real-time simulations.

Considering these four objectives, we present a system that procedurally generates locomotion animations for dozens of multi-legged characters, in real-time, without any motion data. Our system is quite generic as it is capable of animating different morphologies: varied number of legs, different body sizes, different number of leg sections, *etc.* and it is capable of adapting the generated animations to dynamic complex environments, in real-time, giving the simulated multi-legged characters more freedom when moving. The controllability is one of the advantages of the chosen **Procedural-Based** techniques (see Section 2.2 for more details), as it allows the user to control the generated animation thus

producing the needed style of locomotion.
Our system uses a combination of techniques to achieve these four objectives:

- Terrain analysis and 3D trajectory construction, to give to the animated characters the tools to navigate in a complex environment.
- Feedback control, in order to adapt the locomotion to moving obstacles.
- Biomechanics parameters, to define different styles of locomotion and to add realism.
- Intelligent feet placement, that mimics the choices that real-life creature takes when placing their feet.
- Pseudo physics and secondary motions, to generate more plausible and believable multi-legged characters animation.
- User-friendly interfaces, to edit and customize the final animation in real-time and ensure the controllability of our system.

Final simulation is rendered using OpenGL with *C++*. We use GLSL Shaders in order to improve 3D visuals and rendering speed, thus enhancing immersion and believability.

1.3 Industrial Context

The presented work is a collaboration between *SAARA*[1] team at the *LIRIS*[2] in Lyon and *Spir.Ops*[3] a company based in Paris. It is funded by the CNRS[4] and *Spir.Ops*.

The team *SAARA* is part of the image department of the *LIRIS*. Their research topics concern the simulation, analysis and animation of complex scenes containing virtual characters (like humans and any other multi-legged characters), with an orientation on augmented reality environments. Their main goal is to enrich movements in virtual worlds by proposing tools based on motion capture or, like in the case of this thesis, on procedural approaches based on physics or geometry.

---

[1] Simulation, Analysis, Animation for Augmented Reality

[2] Laboratoire dÍnfoRmatique en Image et Systèmes d'information, UMR5205 CNRS, F-69622, France

[3] http://www.spirops.com/

[4] Centre National de la Recherche Scientifique

Spir.Ops is a private research company in Artificial Intelligence (AI), founded in 2003, that develops tools and propose services in decisional AI. Their research topics concern evolved AI behaviors that bring simulated agents to a new level of believability, crowd simulations and most importantly procedural animation. They have a broad domain of applications like video games, serious games, robotics and industrial simulations (crowd simulations in entertainment parks and public transportation).

1.4 Thesis Organization

In **Chapter 2**, we present a thorough overview of the existing animation techniques in general, and locomotion controllers in specific. We list their advantages and disadvantages in order to justify the choices made in this thesis. In **Chapter 3**, we detail the first contribution in this thesis: our multi-legged characters locomotion system. The generic adaptive animation system that satisfies the four objectives: adaptability, controllability, believability and efficiency. **Chapter 4** focuses more on the realism and naturality of the locomotion produced by the system, as we add more control layer (pseudo physics, flexible spine model, *etc.*) to make the generated animation even more plausible and life-like. **Chapter 5** focuses also on the realism of the generated animation, as it presents a system of 3D pendulums that adds secondary motion to the final simulation. Secondary motions are indirect motions derived from the primary motion, but essential in order to add more realism to the final simulation [HOZ97]. Finally, **Chapter 6** contains final thoughts, general conclusion, future work and short/long term ambitions concerning the presented animation system.

Chapter 2

State Of The Art

Contents

2.1 Data-Driven Techniques **9**
2.2 Procedural-Based Techniques **14**
2.3 Physics-Based Techniques **19**
2.4 Hybrid Techniques **27**
2.5 Multi-Legged/Non-Human Characters **30**
2.6 Path and Motion Planning **32**
2.7 Conclusion **38**

Virtual creatures are often represented by a 3D polygon mesh and animated through a skeleton as illustrated in Figure 2.1; An articulated skeleton consists of several rigid links connected by joints. After animating this skeleton, the 3D mesh animation and deformation is typically achieved using a skinning method. We use the classical linear blend skinning in this thesis [CHP89, TSC96, Tur95, GB08] concentrating only on the skeleton deformation of the articulated figure.

There are many techniques for animating this skeleton. Multon *et al.* in [MFCD99] and more recently van Welbergen *et al.* in [vWvBE+10] identified two main groups: **Data-Driven** and **Procedural-Based** techniques. Each of these techniques and their subcategories have advantages and disadvantages as they offer a trade-off between: the amount of control over the motion, the naturalness of the resulting motion and calculation time.

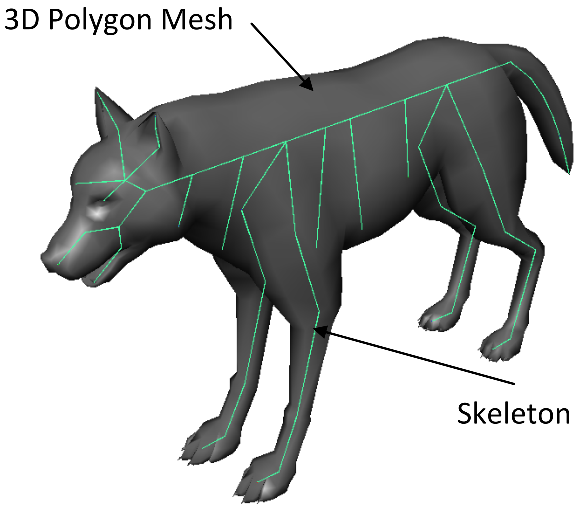


Figure 2.1: *Example of a 3D mesh with its skeleton.*

Data-Driven techniques use input data to produce the virtual characters animation in the virtual world. These input data are usually captured, called Motion Capture (MoCap) data (Figure 2.2) or manually created, called Keyframe data.

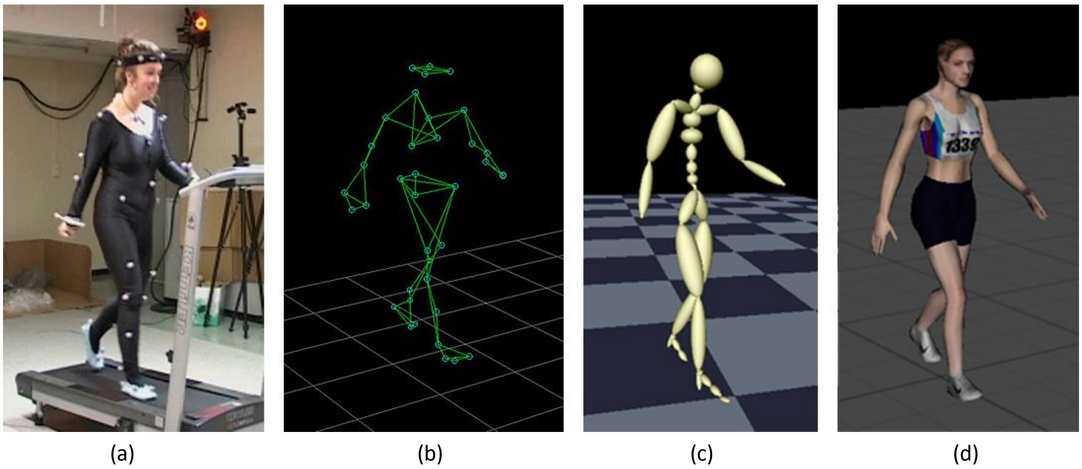


Figure 2.2: *Example of optical motion capture, courtesy of [McC10]. a) Performer wears reflective dots that are followed by several cameras. b) Raw captured data. c) Final skeleton is animated using the motion data. d) Final 3D mesh is animated using the MoCap data.*

On the other hand **Procedural-Based** techniques do not use any motion data. Final animation is generated using either *kinematics* or *physics*:

- *Kinematics*: the use of mathematical formulas and algorithms that are usually inspired by biomechanics and/or empirical data (Section 2.2).
- *Physics*: the use of actual physics (dynamics) to calculate the torques/forces that govern the articulated figure animation (Section 2.3).

In the following sections we discuss in details these techniques plus their advantages and disadvantages, in order to explain and justify the choices that we made in the system proposed in this thesis.

We must note that the interest in legged creatures locomotion can be found in the robotics domain also, as legged robots provide better mobility than wheeled ones. They can choose the best footprints in the reachable terrain, in a way that optimize support and traction [Rai86]. Most importantly, legged robots decouple the path of the body (which is smooth) from the path of the feet (that adapts to the terrain). Nevertheless, with these advantages comes a whole new type of problems like balance controls, complex articulated figures control, *etc.* Stability is the most important thing in the robotic domain (as discussed by Van de Panne *et al.* in [VdPLHF00]), as there are problems and constraints that does not exist in computer graphics like the mechanical capabilities of the actual joints (motor) or actual physics constraints from the actual world. While in computer graphics the most important thing is visual quality, in order to generate a believable immersive virtual world.

2.1 Data-Driven Techniques

Data-Driven techniques use motion data as input (Figure 2.2), preserving the naturalness and the visual fidelity of the captured or created animation data. They are more suited for real-time applications, as playing motion data is usually negligible in term of calculation time. But these techniques suffer from several issues related to a lack of control, as the motion data are fixed to a specific morphology and a specific context. For example, motion data for a character going-up stairs do not tolerate any small change of steps height, neither a change of body parts proportions. Moreover, to achieve the multitude of possible movements (different walk styles, walking on uneven terrain, *etc.*) a good amount of MoCap or keyframe data are needed and stored normally in a database, presenting even more problems to use and combine these motion data. Finally, any new movement demands the re-capture or re-creation of a new animation.

Several techniques are proposed to overcome these shortcomings and create a natural looking animations. Like Motion Blending: the process of producing new motion by blending multiple motion clips according to some time-varying weights. Retargeting: the process of transferring motion data between different morphologies. And many others.

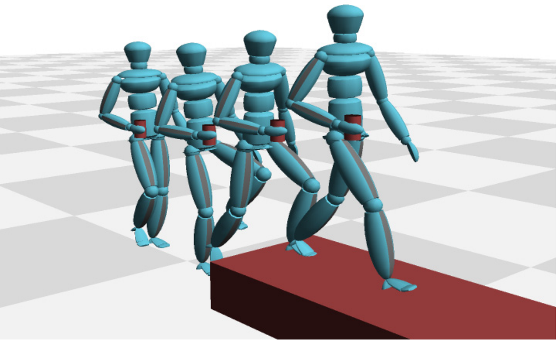


Figure 2.3: *A motion generated by blending the upper body of a person carrying a cup with the lower body of a person stepping up onto a platform, courtesy of [HKG06].*

Motion blending is used to create seamless transitions between motion data, allowing the generation of lengthy and complicated animations out of simpler clips. And is also used to combine these clips, like blending the upper body movement in a MoCap clip with the lower body movement from another MoCap clip in [HKG06] (Figure 2.3). Motion blending can be complex like in [BMJC01] as they blend several motion clips cooperatively and concurrently based on the needed tasks. A typical method to achieve the blending is *Linear Blending*: the i^{th} motion frame resulting from linear blend is the weighted average of the skeletal parameters (like joints angles) at the i^{th} frame of each input motion. But this type of blending can fail when motions have different timing and corresponding events occur at different absolute times (Figure 2.4). A solution is to *timewarp* the input motions (like the system proposed by Kovar and Gleicher in [KG04]), so corresponding events occur simultaneously and transition between motions occur in a logical way. *Timewarping* can also be used to change the duration of a MoCap clip in a natural way, which means slowing down or accelerating the motion data while respecting some physical rules [HdSP07].

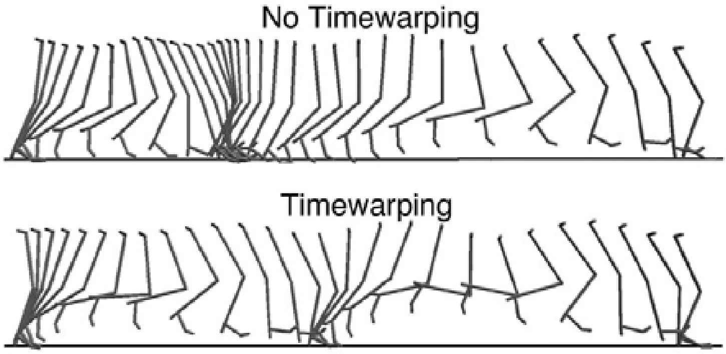


Figure 2.4: *A transition between walking and jogging that spans two locomotion cycles. Only the right leg is shown. Without* timewarping *(Top), out-of-phase frames are combined and the character floats above the ground with its legs nearly straight, courtesy of [KG04].*

Retargeting (Figure 2.5) is the process of adapting an existing motion data to a new morphology. The main goal is to always preserve as much as possible the look and feel of the original motion clips. A common use of retargeting is computer puppetry [SLGS01]: mapping the movements of a performer (MoCap) on an animated character in real-time. Several methods are used to achieve this retargeting like numerical constraints (spacetime and motion signal) [Gle97, GL98, Gle98, PW99], Inverse Kinematics (IK) solvers [LS99, SLGS01] (more about IK systems in Section 2.2), real-time stochastic calculations [CC10], the use of actual physics (dynamics) to account for the change in the body parts proportions and weights while preserving contextual information (contacts and collision with the environment) [ZH99, OM01, SKG03, PW99, MKHK08]. Retargeting becomes more difficult when motion data contains many contact constraints as final animation should maintain the spatial relationship between the new character and its environment and other characters [HKT10]. Or when retargeting between morphologies with drastically different kinematic structure [PW99]. Several papers propose the use of an intermediate model in order to facilitate the process [PW99, KMA05, MKHK08]. This model decomposes the general movement of the human body into syntax-based movement of its main parts (arms, legs, trunk, *etc.*).

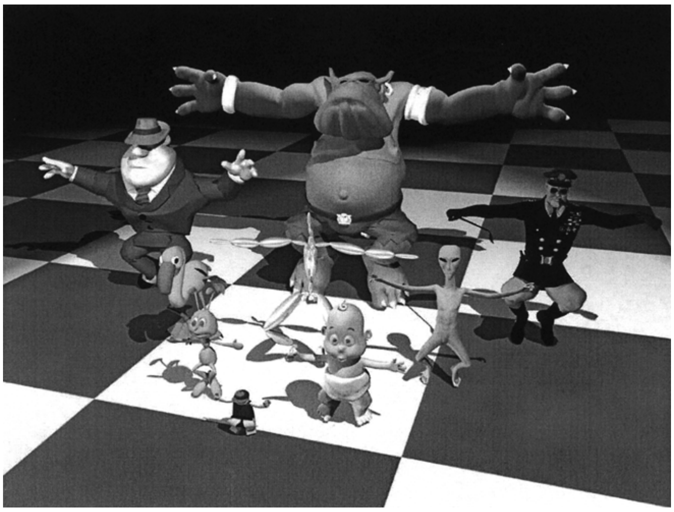
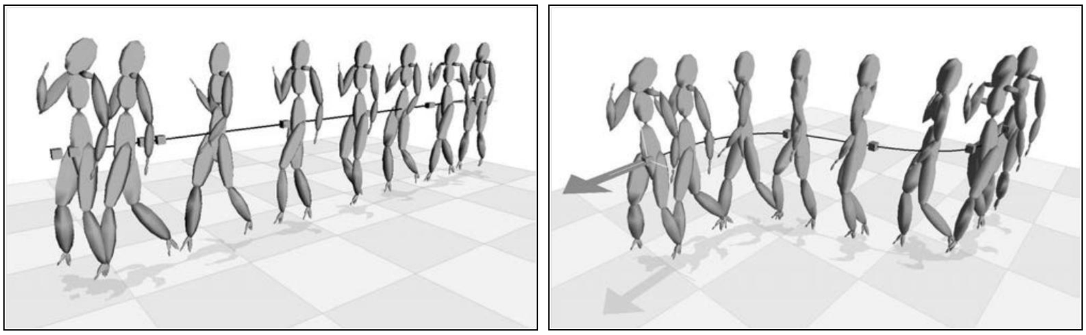


Figure 2.5: *Example of motion data retargeting possibilities, courtesy of [Gle01]. The original skeleton from the MoCap data is shown in the center.*

Motion Path Editing or Motion Warping. Normally MoCap data occurs on a certain absolute path in the virtual world, Like a walk animation on a straight line in Figure 2.6. But what if we need the same animation to occur on a curved path? These techniques (like the system proposed by Gleicher in [Gle01]) have a goal of altering these paths based on external user control and/or environment, while ensuring several constraints [CKHL11]: non-penetration, contact preservation, continuity and motion constraints. An interesting use of motion warping is to synchronize the motion between several characters, which demands the ability of inserting and deleting motion clips based on the user manipulations, as seen in [KHKL09].

As we mentioned before, a typical simulation can have a database with many motion data (MoCap or keyframe). To manage this huge amount of motion data several papers use Motion Graphs: graphs that manage the transition between different motion data clips. Final motion can be simply generated by building walks on the motion graph (Figure 2.7). These graphs can be simple [KGP02, AF02, GSKJ03, RP07] or parametric [HG07].



(a) (b)

Figure 2.6: *An example of motion warping, courtesy of [Gle01]. a) Original motion data. b) Edited MoCap after altering the original path.*

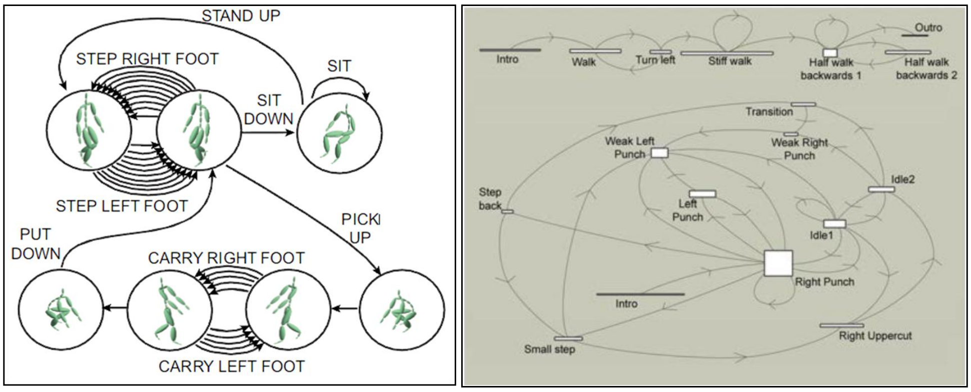

STAND UP
STEP RIGHT FOOT
SIT
SIT DOWN
STEP LEFT FOOT
PICK UP
PUT DOWN
CARRY RIGHT FOOT
CARRY LEFT FOOT
Intro
Walk
Turn left
Stiff walk
Half walk backwards 1
Outro
Half walk backwards 2
Transition
Weak Left Punch
Weak Right Punch
Idle2
Step back
Left Punch
Idle1
Right Punch
Intro
Small step
Right Uppercut

Figure 2.7: *Example of motion graphs, courtesy of [SKG05, BPP07].*

These motion graphs can be quite complex containing predefined avoidance motions [OM11] and can be manually created or generated automatically. A typical technique is to calculate the postural similarities between motion clips, to identify the best transition between them, thus automatically generating the motion graphs [KGP02, GSKJ03]. Reinforcement learning (RL) techniques (with immediate and long-term rewards) are also used to construct these motion graphs. Like in [TLP07, LKL10] they explore all the possibilities of sequences to be applied starting from all the possible starting positions (poses) in order to achieve predefined goals: walking, running, turning, obstacle avoidance, *etc.* RL techniques can be used to navigate through these motion graphs in an optimal way [AFO05]. Lee *et al.* in [LLP09] identifies the subset of motion data that covers a given task (like navigating an environment, going up-stairs, down-stairs, *etc.*) to use less MoCap clips when constructing the motion graphs. Finally, some papers generate responses animations (a biped responding to an external perturbation) without using any dynamics. Like Yin *et al.* in [YPP05] with their system that blends and edits a database of MoCap data containing animations of people responding to random pushes.

A common artifact from any attempt of modifying or blending motion data is *FootSkating*: where a foot slides on the ground when it should be planted firmly. This can be solved using IK systems [LLP09]. Mixing IK solvers with offline analysis [IAF06] or allowing small and smooth changes in the bone lengths of the skeleton [KSG02]. Some systems concentrated more on the smoothness of the resulting animation while allowing some of this *FootSkating* [BUT04, CKJ+11].

So **Data-Driven** techniques are the most naturally looking techniques as they are directly captured from real life subjects or created (tweaked) by hand. But they lack the control, as a change in the animated morphology or the context requires, most of the time, the use of many processing layers in order to effectively use motion data in the virtual worlds. On top of that, it is quite hard to capture the movement of the creatures that we aim to animate in this thesis: spiders, ants, wolves, lizards, *etc.* and even sometimes it is impossible to do the capture: imaginary 5-legged robots. That is what pushed us toward the **Procedural-Based** ones.

2.2 Procedural-Based Techniques

Procedural-Based techniques are gaining momentum in recent years specially in computer-based applications (simulations, video games) as these applications are becoming more interactive, narrative and AI-intensive, requiring simulated characters to perform at run-time a wider range of actions and gestures that are difficult to anticipate (sometimes even impossible) during conception. With this added complexities, capturing or creating all motion data in advance is becoming quite difficult and time consuming. Indeed, more animation systems are starting to use **Procedural-Based** techniques as they are more generic than **Data-Driven** ones, adapt better to the environment and can integrate external perturbations. These criteria are what made us orient our work toward the procedural approaches. The essential reasons that got us interested in these techniques. In this section we concentrate on *kinematics-based* techniques, while in Section 2.3 we detail *physics-based* ones.

When it comes to locomotion, **Procedural-Based** techniques in general and *kinematics-based* ones in specific use biomechanics studies and observations as a source for their control policies. For instance, Muybridge *et al.* in [MT55, Muy57] studied the walk and run behavior of humans and over 40 animals. Inamn in [INM66] propose a study about the sinusoidal movement of the pelvis in humans and the general phases of the feet: *stance* and *swing*. There are also interesting studies on gait transitions, like the transition from run to walk in humans [Hod91] or from trotting to pacing in quadruped [Rai90]. In [HWC00], Hoyt *et al.* studied the effect of limb and step length on running speed for a variety of species, during different situations: load carrying, inclined terrain, *etc.* Bertram and Ruina in [BR01] show the relationship between walking speed and step frequency (gait) in humans. While Youn *et al.* in [YPS07] studied phases and gait mechanics in animals, like: galloping, trotting, pacing, *etc.* Most studies show that humans and animals move in a way that minimizes the energy consumed as much as possible.

Many systems (like ours) use mathematical parametrization and algorithms coming from biomechanics experimental data, to generate for example bipeds locomotion [BMtT90]. *PODA* is one of the earliest procedural-centric locomotion controllers by Girard *et al.* [GM85, Gir87]. They use it to animate a wide range of multi-legged characters on planar terrain (bipeds, quadrupeds, *etc.*). The user needs only to specify the gait pattern (an example of a gait pattern is shown in Figure 2.8), everything else is calculated automatically.

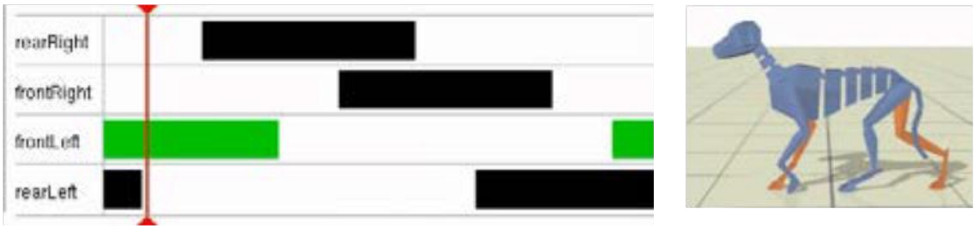


Figure 2.8: *Example of a gait pattern: a dog walk cycle, solid bars represent the legs flight phase (swing phase), courtesy of [CKJ$^+$11].*

They add pseudo-physics calculations to the pelvis of the multi-legged character, making it reacts to the feet movement in the horizontal and sagittal plane in a more believable way. The anatomy plans are illustrated in Figure 2.9.

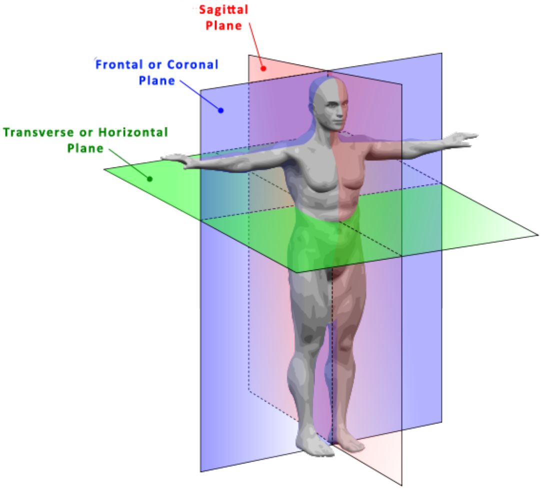


Figure 2.9: *Anatomy planes for humans, which can be generalized to all other creatures.*

Many *kinematics-based* techniques are based on **Inverse Kinematics** (IK systems). The problem can be explained as follow: the generalized location of an *end-effector* e (the joint at the end of a chain of joints, shown in Figure 2.10) is a function of the rotations of all joints in that chain q.

$$e = f(q) \tag{2.1}$$

A typical IK problem is calculating these joints rotations using the location of the *end-effector* only: given the desired position of a skeleton's hand (Target), what must be the angles of the skeleton's joints?

$$q = f^{-1}(s) \tag{2.2}$$

Equation 2.2 may not always have a (unique) solution. Indeed, there are multiple and sometimes endless combinations of joint Degrees of Freedom (DOF) values that put the *end-effector* in the right location. Van Welbergen *et al.* in [vWvBE^{+}10] identify several numerical techniques proposed to solve this problem, like the

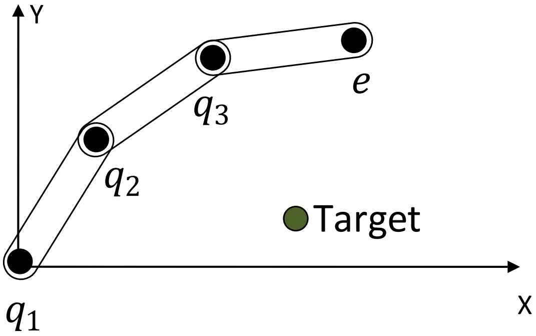


Figure 2.10: *Example of an IK problem: given the desired position of a skeleton's hand (Target), what must be the angles of the skeleton's joints?*

Jacobian Inverse method used in [GM85, Gir87] which is an iterative method that tries to approximates a good solution using the relation between the joint velocities and the velocity of the *end-effector*:

$$\dot{e} = J\dot{q} \textit{ with } J = \frac{\partial f}{\partial q} \tag{2.3}$$

J is an $m \times n$ matrix, with m the dimension of the *end-effector* (three for the position only, six for the position and rotation) and n is the number of joint variables. Inverting equation 2.3 gives the joint velocities:

$$\dot{q} = J^{-1}\dot{e} \tag{2.4}$$

Then, an iterative approach is used to find q. Find the derivative of $e : \dot{e} = \dot{f(q)}$, calculate J, invert J, use equation 2.4 to calculate $\dot{q}$, integrate $\dot{q}$ to obtain q and finally repeat until $f(q)$ is close enough to e. Typically, J is non-square. J^{-1} then has to be replaced by the *pseudo-inverse* of J^{+} (which is computationally expensive). **Optimization Based** methods converts the IK problem into a minimization problem using the distance between $f(q)$ and e as an error measurement. The goal is then to find the DOF vector q that minimizes the error, which is a classical non-linear constraint optimization problem [Wel94] that can be solved using standard numerical methods for which several toolkits are available. In our system, we use the Cyclic-Coordinate Descent (CCD) method proposed in [Lue84, WC91]. The CCD iterates through the joints, typically starting with the one closest to the *end-effector*, and varies one joint variable at a time based on a heuristic. An example of such a heuristic is shown in Figure 2.11 where the goal is to minimize the angle between the vector originating from the current joint toward e and the vector from the current joint towards $f(q)$. Unlike the **Jacobian Inverse** method, which distributes joint rotation changes equally along the chain, CCD has a preference of moving distal links first. We chose this CCD IK system over the others thanks to its performance: in the following chapters we are going to use it to calculate the position of over a 500 limb in real-time. We chose it also thanks to its simplicity and controllability: calculations time can be easily controlled (Section 3.7).

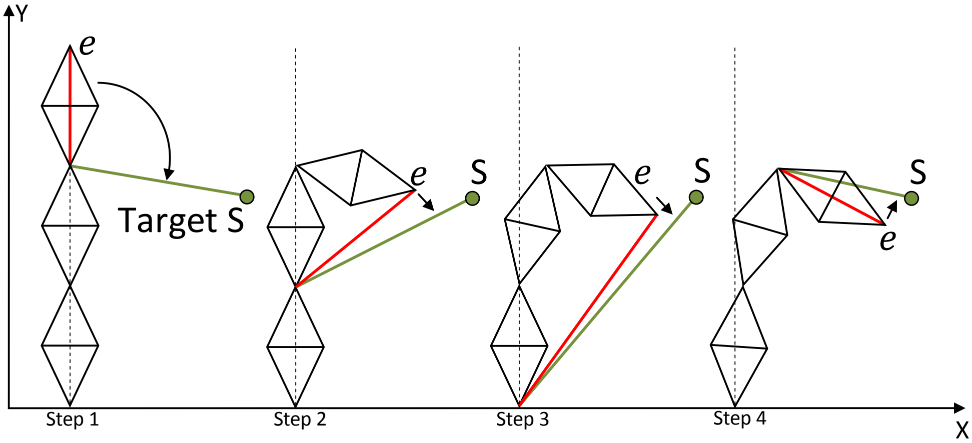


Figure 2.11: *From left to right: typical steps for the CCD IK method.*

We must note that it exists other types of IK systems beside the *analytical* ones. These systems use motion data (MoCap or keyframe) to automatically learn a model of logical and natural poses [GMHP04, Kal08, WTR11, KG04]. The goal of these systems is to generate the most natural poses: poses that are most similar to the space of poses in the training data (Figure 2.12). Finally, there is also *Mesh Based* IK techniques, like in [SZGP05, DSP06], that directly moves the vertices and polygons of the 3D model in order to deform the mesh toward the needed position.

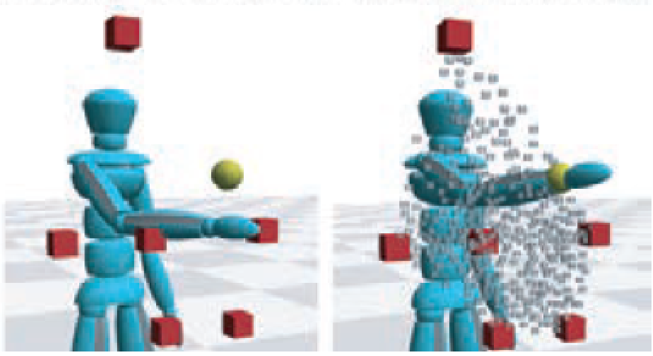


Figure 2.12: *Left: cubes in red represent the possible 3D hand positions from raw MoCap Data and the yellow sphere represent the needed position. Right: the use of motion blending and sampling to achieve the non-existent needed position, courtesy of [KG04].*

An important aspect in IK systems is joint constraints: in humans the shoulder joint has 3-DOF while the knee has only 1-DOF, *etc.* Integrating these constraints in the IK process is really important in order to generate logical movements that satisfies the specifications of the needed morphology . There is a good amount of papers (specially in the medical field) about the range of motion of joints either in humans [BA79, MT98] or in multi-legged characters. We use these data as an input constraints to our CCD based IK system. There is also many techniques to enforce these constraints on the virtual joints, like using reach cones in [WG01] to constrain ball-and-socket joints (the human shoulder).

2.3 Physics-Based Techniques

Many motion effects observed in real-life like balance control, momentum propagation, *etc.* are due to real world physics. *Physic-based* techniques concentrate on using dynamics and virtual body characteristics in order to produce physically realistic motions. These techniques simulate the actual physics forces that act on the articulated body parts, then they calculate the forces and torques that should be applied in order to achieve the desired movement. These techniques are also used for secondary motions like simulating the clothes of the animated characters, animating the tail of a dog, *etc.*

An articulated figure has links connected with each other by different kind of joints, resulting in many Degrees of Freedom (DOF) (*e.g.* a human or a horse, Figure 2.13). The computation of the physics laws that govern this articulated figure is quite expensive, especially with the increase of DOF.

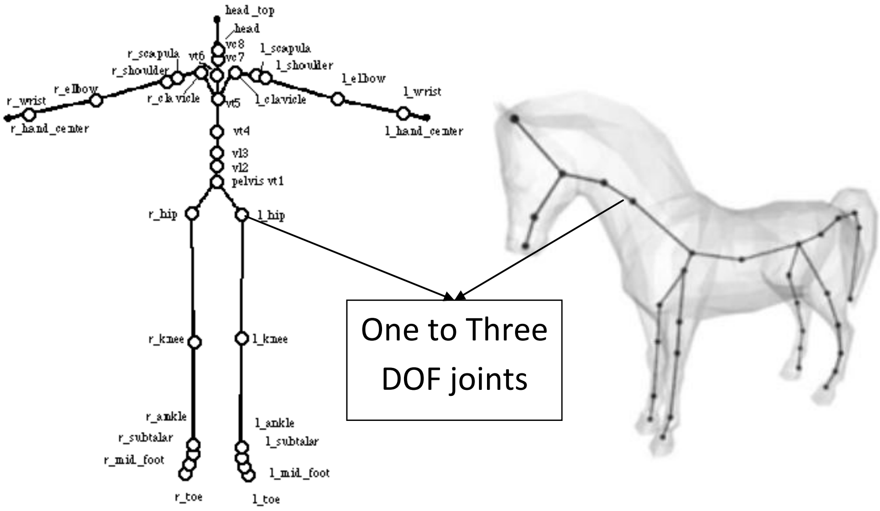

head_top
head
vc8
vc7
r_scapula
vt6
l_scapula
r_shoulder
l_shoulder
r_elbow
l_elbow
r_wrist
l_wrist
r_clavicle
vt5
l_clavicle
r_hand_center
l_hand_center
vt4
vl3
vl2
pelvis vt1
r_hip
l_hip
One to Three DOF joints
r_knee
l_knee
r_ankle
l_ankle
r_subtalar
l_subtalar
r_mid_foot
l_mid_foot
r_toe
l_toe

Figure 2.13: *Example of articulated figures and their many DOF.*

Kokkevis in [Kok04] classifies the methods that simulate these articulated bodies physics (commonly called a ragdoll) into: maximal coordinate methods [Bar96] and reduced coordinate methods [Fea99a, Fea99b]. Maximal coordinate methods, like in [Bar96, Fau99], treat each bone as a separate rigid body and use explicate constraints to remove the extraneous degree of freedom. These methods are reputed to be easy to implement but they are computentially expensive. Another drawback is that they operate in cartesian space which makes it difficult to guaranty length constraints. Reduced coordinate methods, like the one proposed by Featherstone in [Fea87], eliminates any cyclic calculation problems and breaks the computation into several linear passes (whether from the root of the skeleton toward the leaf joints or the inverse). In [Fea99a, Fea99b] Featherstone extends his work, providing stable solution to animate articulated bodies with branches and loops (Divide-And-Conquer algorithm), a method used in many systems [RGL05, MZ90].

Beside articulated body physics equations, a *Physic-Based* locomotion controller needs to move the simulated creature based on the simulation needs. A typical way to do that is to use a gait pattern (manually created or automatically generated) represented by a Finite-State Machine (FSM) (See the walking pattern represented by an FSM in Figure 2.14). Each state in this FSM consists of a body pose: body part positions or target angles for all joints in respect to their parent links. Transitions between these states can occur after a fixed durations of time, after a new foot contact or any other criteria.

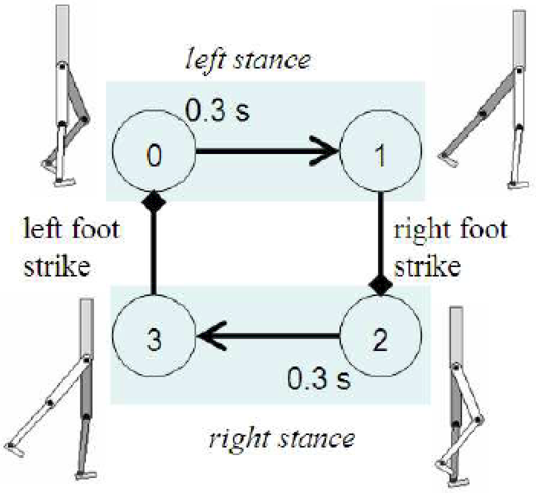


Figure 2.14: *An example of a Finite-State Machine (FSM) of a walk cycle, courtesy of [YLvdP07].*

During simulation, each body link attempts to drive towards its target position using external calculated forces in order to generate the needed locomotion. Or every individual joint attempts to attain its target angle using internal torques calculated by systems like the Proportional Derivative (PD) controllers [1].

A common source to create this gait patterns are empirical and biomechanics data, like general locomotion data [LvdPE96, LvdPF97, RH91], or more specific gait patterns like athletes running, bicycling, vaulting [HWBO95] or swimming (with fluid dynamics) [YLS04]. Most previous controllers are simulated in a fully dynamic world, which means that legged characters need to maintain their equilibrium to counter gravity. Otherwise these simulated characters will fall off to the ground even while not moving. To counter gravity, most systems use a balance controller, an essential part of any *physics-based* technique. Many biology and biomechanics-based research papers studied this balance control behavior, mostly in humans. Like the studies about balance strategies in humans while standing on a moving platforms under several conditions [PLB95, Rob06, VHB^{+}08]: lights on/off, holding a cup, holding a safety bar, *etc.* Faure *et al.* in [FDCM97] studied torques and actuator data during a human walk. Other studied human Center Of Mass (COM) trajectory while balancing, walking or running [PP97, LF98, SvABV09]. Zordan in [Zor10] shows the importance of angular momentum in balance control strategies (like windmilling) and during locomotion (like the movement of the arms and the trunk in response to the legs movement). Popovic *et al.* in [PHH04] studied the main principles of human locomotory function in order to generate natural looking walking animation, like joints spin angular momentum, the total sum of joint torque squared, *etc.*

---

[1] A PD controller is a feedback control mechanism that calculates a torque based on the difference between the current state and the desired one. It uses the proportional and derivative constants (K_d, K_p), that are normally tweaked by hand, in order to output this torque (Appendix A).

As we can see, in addition to the complexity of the articulated figure physics (due to their numerous DOF) there are objective functions that need to be satisfied (minimized or maximized like the balance controller) plus constraints that should be respected in order to have a logical simulation. Constraints imposed by the simulation itself (collision, joints range of angles, *etc.*) or imposed by the user (gait pattern, a planted foot should not skate, symmetry of the movement, *etc.*). The complexity of such problem is what pushed many systems to use optimization methods with intuitive fitness functions and what pushed others to use simplification methods in order to make this non-linear problem more tractable.

Optimization Methods

Many systems use optimization techniques to find a solution to the non-linear equations and constraints that govern the articulated figure physics, as it is quite intuitive to find a fitness function that summarize how close a given solution is to achieving the set aims. Fitness function like minimizing energy consumption, minimizing torques and angular momentum, *etc.* Quadratic programming (QP) based systems are an example of optimization techniques, like the one used by Liu and Popovic in [LP02] (and similarly by Fang and Pollard in [FP03]). Their system focus on the synthesis of highly dynamic movement such as jumping, kicking, running, and other gymnastics (Figure 2.15). The animator defines key poses in the 3D space (keyframes) with their timing, then a QP solver is used to minimize the objective functions: minimum mass displacement, minimal velocity of DOF and static balance. While respecting the constraints: the poses fixed by the animator, the transition poses estimated between those poses and environmental constraints. The final results of their QP solver are the orientation of each joint and position of the COM at each frame (the final animation).

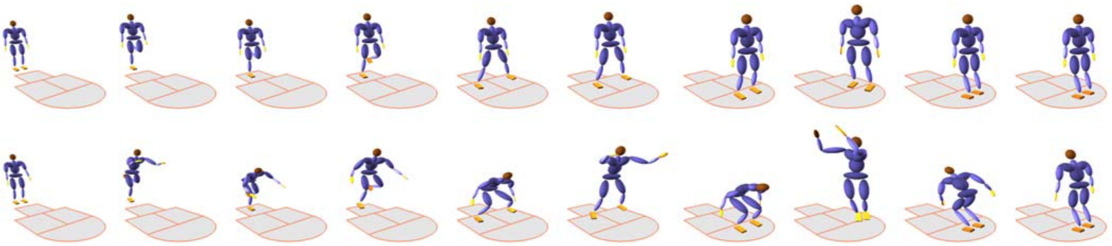


Figure 2.15: *Example of an animation generated by a QP solver, courtesy of [LP02]. Top: simple input animation. Bottom: synthesized realistic animation.*

Balance control can be achieved with optimization methods in response to external perturbations [PH05] or ground friction (slippery ice-like terrain) [AdSP07]. One of the advantage of these systems is their ability to generate human-like strategies for balance controls like arm swinging and bending in [KKI02], stepping strategies in [KKI06], *etc.* which were not coded explicitly in the controller. This allows for complex animations, like in the system proposed by Jain *et al.* in [JYL09], where the character can decide to take several small steps then supports itself on a wall in response to an external push.

Objective functions can be either weighted [AdSP07, KKI06, PH05, JYL09] or prioritized [dLH09, dLMH10, dL11] allowing interesting animation results with the simulated creatures sacrificing comfort in order to achieve a needed goal.

The main advantage of optimization methods (and the main advantage of the *physics-based*/**Procedural-Based** techniques) is the ability to adapt the locomotion animation to the environment: rough terrain with large drops and gaps, uneven terrains, slops, *etc.* [MdLH10, WZ10]. They do so more easily than the **Data-Driven** techniques as they are not constrained by the original motion data context, allowing more varieties in the simulated environments. On the other hand, *physics-based* techniques are not real-time or cannot simulate more than two characters at the same time.

Simplification Methods

The strategy of these methods is to represent complex dynamic characters by simpler models with less DOF. The Inverted Pendulum Model (IPM) is a typical example and it is used to represent the whole body when it is supported by a single leg. An IPM is a massless telescopic leg that connects the COM of the body with the needed foot (Figure 2.16). This simple model is used to avoid complex kinematics and dynamics calculations on the original multi-segments skeleton [KKK+01a, Rat05, PT06]. The use of the IPM is not limited to computer graphics, it is also used in robotics like in [KKK+02, SA09, KKK+03]. One of the main uses of the IPM is in predicting the best foot position to maintain balance (stepping strategy) [KKI06] specially after an external perturbation (a push) [PT06, RCP07].

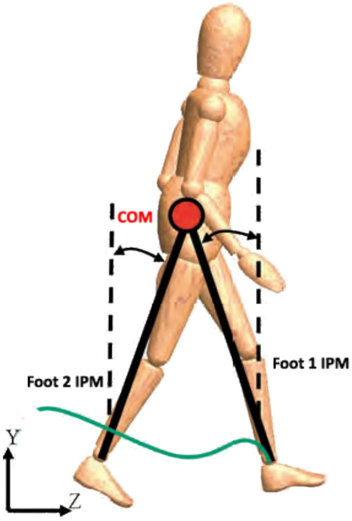


Figure 2.16: *An example of an IPM model, courtesy of [TLC+10].*

An IPM is only valid for locomotion with low inertia change (like walking) that is why other papers use Spring-Loaded Inverted Pendulum (SLIP) (Figure 2.17) that generalizes the IPM by replacing the fixed length leg with a spring, thereby capturing energy storage and release during running [SK00, BSG⁺07]. The SLIP (and the IPM) can be used to generate the actual gait pattern like in [MdLH10], instead of using the fixed gait patterns previously represented by an FSM.

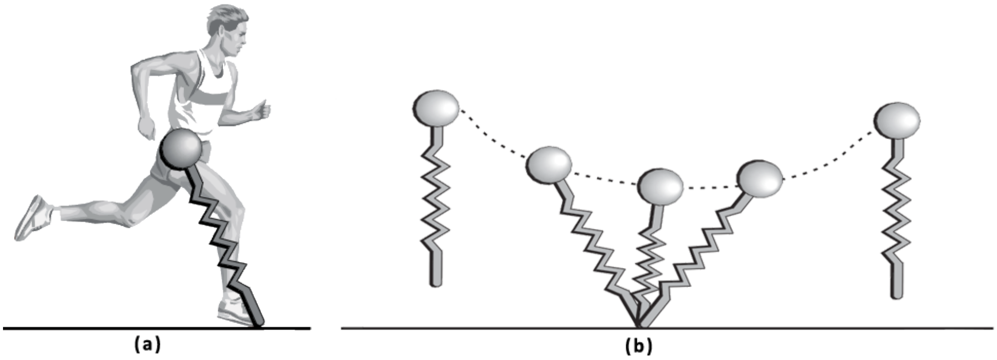


Figure 2.17: *An example of a SLIP model, courtesy of [SK00]. a) The actual model. b) SLIP in action.*

Many biomechanics-based papers studied these two models (like the energy changes in [KDR05]). Specially, how an IPM can be successfully used to predict the preferred walking and running speed for test subjects [SR06] and how it is a good approximation of the human walking mechanism [Kuo02], using relationships between speed and step length [Kuo01]. Although simple models like the IPM and the SLIP are good approximations of the actual legs in multi-legged characters, Full and Koditschek in [FK99] show how these models fail in capturing the diversities in locomotion animations. They show also the importance of having realistic models based on the actual morphology and physiology of an animal (legs, joints, muscles, *etc.*) in order generate more natural looking and realistic locomotion animation. These realistic models can integrate these simple models (IPM, SLIP, *etc.*).

Feedback (feed forward) systems are another type of simplification methods and one of the inspirations for our final locomotion controller. Instead of calculating everything offline, these methods uses an initial controller that generate the animation, then these controllers adapt themselves based on the feedback of the environment (changing the control strategy, anticipating a fall, *etc.*). A typical feedback loop is shown in Figure 2.18. By doing so, these systems are more responsive and adapt better to uncertain simulations where it is nearly impossible to anticipate everything in advance. The previously presented PD controller, the IPM and the SLIP are common components of these kind of systems. The main system, explained in this thesis (Chapter 3), is a **feedback-based** one, but we use kinematics instead of dynamics.

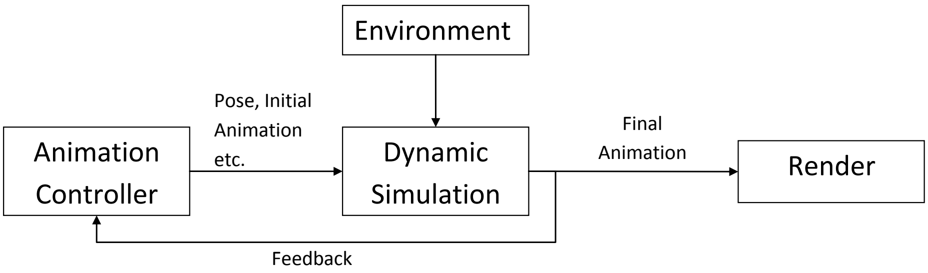


Figure 2.18: *Feedback (feed forward) based animation controller.*

A good example of **feedback-based** systems is SIMple BIped CONtrol (SIMBICON) proposed by Yin *et al.* in [YLvdP07]: a system capable of generating locomotion animation for a variety of bipeds on irregular terrains while undergoing external perturbations. Their controller synthesizes animations in real-time and produces locomotion by using feedback strategies: continuously adapting the motion to the real environment by changing the PD controllers default target angles (shown in Figure 2.14) based on the environment feedback.

Many systems couple this SIMBICON (or a SIMBICON-like) controller with other techniques in order to achieve more. Like adding a footprint constraint during each step that the biped should satisfies (achieve) in order to anticipate next steps [CBYvdP08]. Using Genetic Algorithm. techniques in order to produce naturally looking walking animation [WFH09]. Making the previous controller more robust by using exploratory Reinforcement learning (RL) techniques [CBvdP09], or by optimizing it based on certain scenarios (walking on a narrow passage, changing walking speed etc) [WFH10]. In [CBvdP10], Coros *et al.* couple a SIMBICON inspired motion generator with an IPM in order to generate in real-time a walk animations for a biped that can keep its balance, reach objects, lift and push boxes.

Finally, there are interesting simplification systems like in [KRFC09, NKZ12] that do not animate each joint by itself. They identify the subset of joints, using modal analysis, that are sufficient to animate the whole virtual creature. Final animation is generated by making this subset of joints vibrates with specific periods and with low frequencies, which induces passive movement in other connected joints and rigid bodies. They aim at simulating the regular pushes from the muscles that are necessary to re-inject back to the system lost energy from the movement itself.

2.4 Hybrid Techniques

The systems in this section marry the advantages of the **Data-Driven** techniques (naturalness) with the **Procedural-Based** ones (controllability and adaptability) in order to generate more realistic animations. The benefit of this mix can be directly observed in *physics-based* techniques. As it is always challenging to manage the functions that leads to the successful completion of the desired task while ensuring the generation of a visually appealing animation that meets the user requirements. Some *physics-based* techniques, use motion data (MoCap or keyframe) as an input to refine the generated animation, making it more close to reality.

A good example is the system in [ZH02]. Zordan *et al.* propose a boxing simulator using MoCap data and dynamics. At each simulation time step, the system determines a desired state for the simulation from the motion data, then the Proportional Derivative (PD) controllers compute torques based on those desired values and based on the balance controller output. The simulation is integrated forward in time, the state variables are updated, and the process repeats. The idea is to calculate the needed torques in order to drive the articulated figure toward the MoCap data. During this tracking process, this articulated figure can undergo external forces and have responses motions that were not present in the original motion data. The previous torques can be calculated offline [MLPP09] or directly in run-time using these PD controllers [ZH02, LKL10, ZMCF05] or any other system. But one of the main problems in this tracking technique is the possibility of introducing a delay in the produced animation between the target pose and the response even if there is no external perturbation, as shown in Figure 2.19. Our pendulums system [AKGM$^+$11] overcomes this problem while adding more control (Chapter 5).

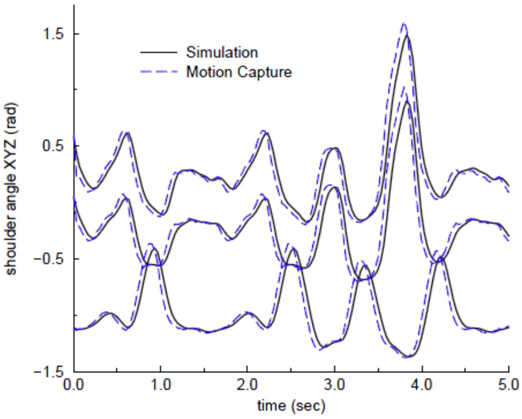


Figure 2.19: *An example of the time delay that can be introduced by a PD controller, courtesy of [ZH02].*

Using this simple tracking idea [ZMCF05] and by adding specific motion data like response animations [ZMM$^+$07, NWB$^+$10] and anticipation animations [1] [ZMM$^+$07, MZH$^+$08] found in biomechanics studies [GRVT06], these systems are capable of creating more rich simulations than pure MoCap ones with more predictable and controlled results compared to *physics-based* one. A typical control loop in these simulations is illustrated in Figure 2.20 with the articulated figures tracking MoCap data, blending-out toward dynamic reactions (pure physics or more controlled dynamics) when there is an external perturbation, then blending-in toward the closest MoCap data when the external perturbation is finished or the best motion data is found.

Tracking motion data is only one example of the methods that mixes **Data-Driven** and **Procedural-Based** techniques as there are other systems that take advantage of this mix benefits, like adding balance control to MoCap

---

[1] Leaning or pulling away from the threat, turning away with the goal of orienting the face away from the threat, pulling the free extremities in towards the body, *etc.*

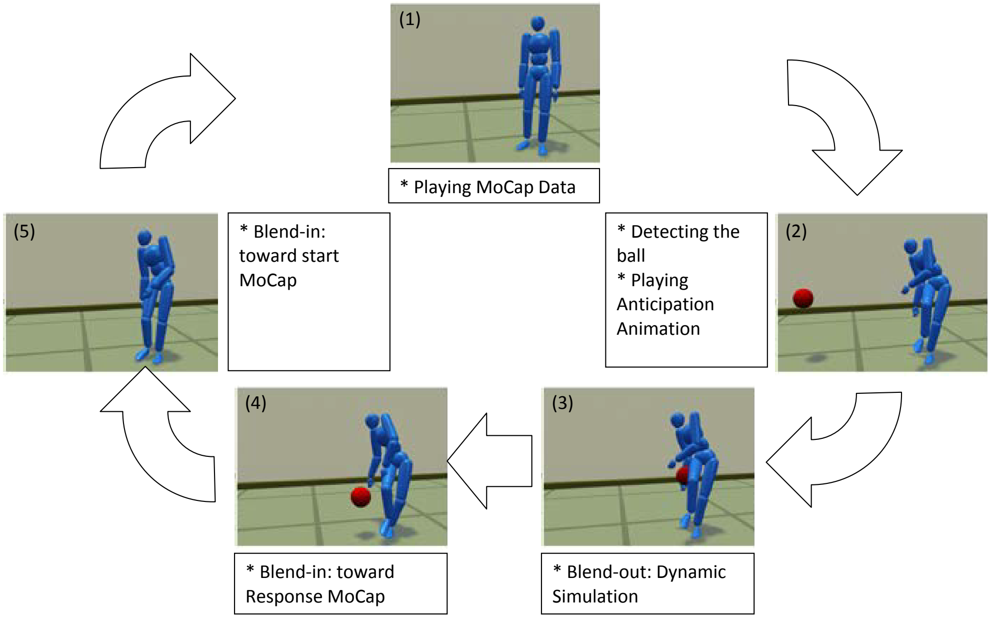

(1)
* Playing MoCap Data
(2)
* Detecting the ball
* Playing Anticipation Animation
(3)
* Blend-out: Dynamic Simulation
(4)
* Blend-in: toward Response MoCap
(5)
* Blend-in: toward start MoCap

Figure 2.20: *Example of a system that adds responses to motion data, original image courtesy of [ZMM+07].*

data using an Inverted Pendulum Model (IPM) [WMZ08, KH10, TLC+10] or by using optimization methods [MZS09, dSAP08, MPP11], editing motion data using dynamics and optimization methods (like altering the landing position of a jump clip) [ALP04, SP05]. And the main goal of most of these techniques is to extend the existing motion data by integrating extra motions (responses to external perturbations, balance control, *etc.*) that are either context dependents: changing the weight of the manipulated box in MoCap data [AP06], or hard to predict during capture or creation: external perturbation [YP03, AFO05].

Finally, most of the previously mentioned **Data-Driven** and **Procedural-Based** systems can be categorized as **Foot Placement Driven Techniques**. Techniques with the feet driving the locomotion and the overall trajectory of the COM, not the opposite. These techniques either adapts existing MoCap data to satisfy footprints fixed in the environment, using IK systems [BK96], greedy optimizations techniques [vBPE10], IPM [SKRF11], *etc.* Or they use these footprints as active constraints in dynamic systems [Tor97, TvdP98] in order to simplifies calculations. But these techniques suffer from a biomechanics problem: normally, the lower body (the legs) is not the one controlling the locomotion. On the contrary, the upper body (*e.g.* COM or pelvis) imposes a logical trajectory that the lower body tries to follow as discussed by Berthoz in [Ber09]. Our system like [GM85, Gir87, CR06] respects this concept: the pelvis can adapt its movement based on the feet feedback, but it is always the one imposing the overall trajectory.

2.5 Multi-Legged/Non-Human Characters

More and more studies started exploring locomotion and animation of non-human (non-biped) characters, like animating birds during flight phase using aerodynamics and biomechanics observations [WP03]. Creating valid swimming animation for aquatic creatures using fluid simulations and QP solvers [TGTL11]. Even generating locomotion animation for box-based fictional creatures using genetic algorithm [Sim94], where the locomotion was optimized using genetic algorithm, as well as the morphology of the creatures itself in order to generate the best combination (morphology - locomotion) for a given task. In [FRDC04, CKJ$^+$11, HRE$^+$08, WP09] they generate animation for legged characters, which consists of a wide panel of creatures like quadrupeds, six-legged characters, imaginary three-legged creatures *etc.* All these non-human systems use the same previous animation techniques as shown by Skrba *et al.* in [SRH$^+$08, SRH$^+$09]: **Data-Driven** methods, *physics-based* models, IK systems, or some combination of the above. In this thesis, we have mostly animated n-legged creatures with $n > 2$, which is related to this family of methods.

Data-Driven based systems use a wide variety of input data to generate the multi-legged character animation. Like using gait patterns observed in biology and dynamics (oscillatory based) to animate six-legged characters (a cockroach) that adapts to planar and uneven terrain [MZ90], extracting 3D cyclic motion of animals from video sequences using image processing techniques [FRDC04], using gait patterns and dynamic values (forces/torques) extracted from MoCap coupled with PD controllers in run-time, to animate a dog capable of a wide range of locomotion on a straight line [CKJ$^+$11], and so on. These systems are, most of the time, morphology specific meaning that they can animate only a specific morphology like a dog in [CKJ+11], a horse in [TCHL12] or a quadruped robot (Called BigDog by Boston Dynamics ™) in [RBNP08].

We are more interested in morphology independent systems like the previous *PODA* system [GM85, Gir87] as they can animate a multitude of morphologies with nearly no constraints. One of the most interesting morphology independent systems is the one used in the game Spore ™ (by Maxis Studio ™). Where Hecker *et al.* in [HRE+08] created a system capable of animating multi-legged characters whose morphologies are unknown when creating the actual animation system. These characters can be created by the user in run-time, an example of the possible morphologies can be seen in Figure 2.21. Their animation database contains semantically generalized keyframe data, for example: moving the links that connect the right most grasper and the spine, in a certain way (created by the animator), relatively to the creature limb length and to the ground. In run-time these movements are specialized (real-time retargeting) based on the actual new morphology, using a special iterative numeric IK system. The animators need only to create gaits for six and less legged characters, then their system generates plausible locomotion animation for any legged character (6-legs or more). Different leg groups could be in different gait styles simultaneously on the same character. For example, two short legs could be running while four long legs are trotting.

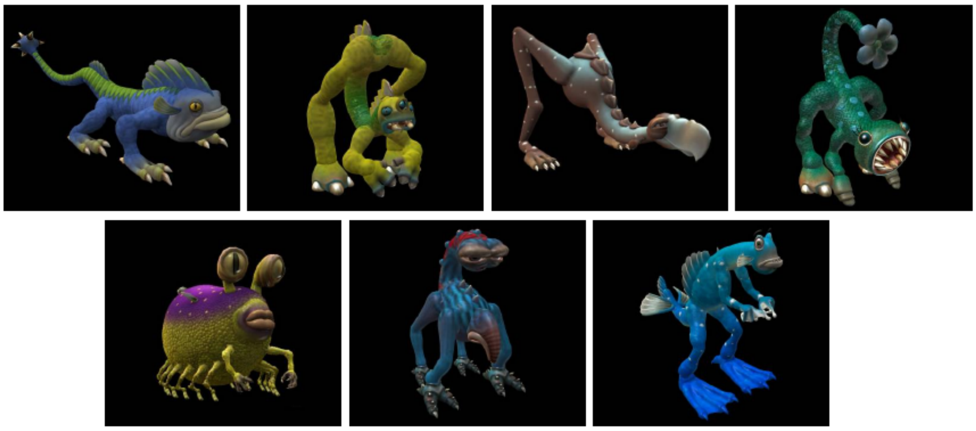


Figure 2.21: *Example of the possible user created morphologies in the game Spore*^{TM}*, courtesy of [HRE+08].*

In [WP09], Wampler and Popovic present a system capable of generating plausible locomotion animation for multi-legged characters (biped, 4-legged, 5-legged, *etc.*), with totally different morphologies (long legs, short legs, long body, *etc.*) and single point contact feet. Using dynamics and offline spacetime constraints optimization process, their system is capable of generating a plausible locomotion animation by using as input the masses of the body links, the desired velocity and a user defined feet tempo (the time, in a gait cycle, when a given foot should be in contact with the ground). Their system can optimize the morphology also, but they differ from other techniques in that they only vary the lengths and radii of the animal's limbs (they do not add a new leg), while in [Sim94] the whole morphology is optimized and changed. So their system is capable of adapting the morphology to the previously mentioned user constraints and new ones like keeping the head on a certain height. And by assuming that each foot touches the ground for only a single interval during a gait cycle, they are capable of optimizing the actual gait based on the creature morphology.

Finally, most of the systems discuss the importance of adding a spine-like model while animating multi-legged characters (specially quadrupeds) in order to generate more natural results [CKJ$^+$11, CR06]. An important issue that we elaborate in more details in Section 4.2.

2.6 Path and Motion Planning

Path planning in general is the method of finding a trajectory (a path), in 2D or 3D, that connects a start position S with a goal G, while avoiding collision with known obstacles (Figure 2.22). A typical use of a path planner is to calculate characters route in the virtual world. In our case, we use it to plan feet trajectories that navigates the environment in 3D, an always present problem that needs to be solved in real-time (Section 3.4).

There are two main methods for path planning: **Geometric-Based** or **Grid-Based**. In **Geometric-Based** algorithms a graph is generated based on the environment, called a *roadmap*: it is a graph that connects nodes/waypoints that exist in the free space or on the edge of the obstacles, as shown in Figure 2.23. There are different ways for generating this *roadmap* graph. Manually, which is a

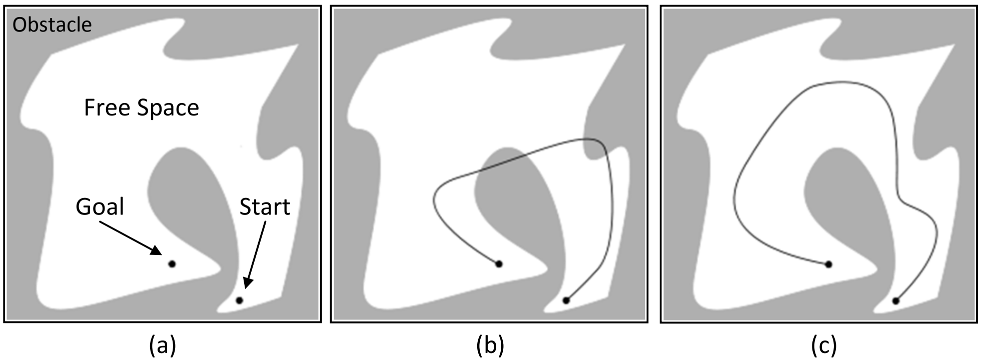


Figure 2.22: *a) Example of a path planning problem. b) Invalid path (collides with an obstacle). c) Valid path.*

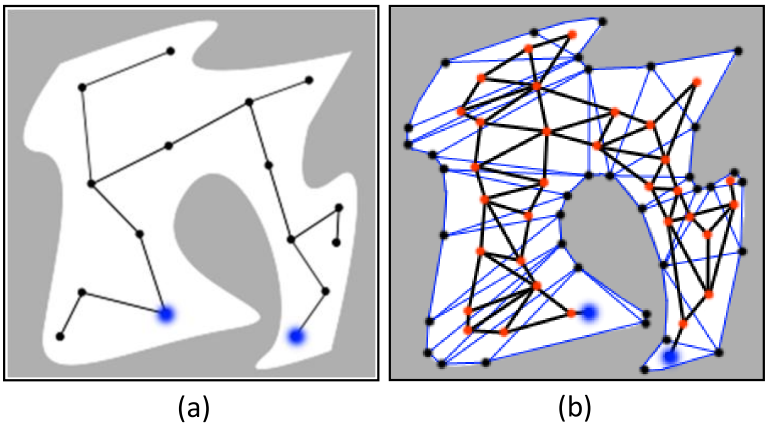


Figure 2.23: *Example of possible* roadmaps *(In 2D for simplification). a) Sample based. b) Triangulation based.*

tedious approach. Sampling techniques (Figure 2.23(a)), which samples the free space then connects the selected points in a way that does not intersect with any obstacle. Triangulating the environment (Figure 2.23(b)), then connecting the center of each triangle and S and G. And many other techniques for *roadmap* construction.

These *roadmaps* can be in 3D with waypoints positioned in the 3D space and connected using 3D segments. Or they can be in 2D (Figure 2.23), coupled with 3D elevation information like the ceiling height in [Lam09] or the *elevations map* in our system (Section 3.4.1). By doing this coupling, the path planner is capable of doing pseudo 3D planning (2.5D) without the complexity of generating and updating 3D *roadmaps*.

After generating this *roadmap*, final path can be calculated by traveling from the S node toward G. There are several algorithms to optimize this search and calculate the optimal path. Naive techniques like the Depth-First Search (DFS) and Breadth-First Search (BFS) explore all possibilities until finding the first solution (Appendix C). There are more elaborate techniques like the Dijkstra's algorithm [Dij59] that evaluates the cost of moving between waypoints in order to chose the lowest cost path, the A* (A-Star) algorithm [HNR68] that optimizes the Dijkstra's algorithm calculation time and memory usage through the use of heuristics, the HPA* (Hierarchical Path-Finding A*) that splits huge *roadmaps* into hierarchical ones in order to accelerate path calculation and many other techniques for graph navigation.

In **Grid-Based** algorithms, a grid with fixed cell size is overlaid on the environment which helps in discretizing the environment (shown in Figure 2.24). Final path can be calculated using the same techniques of the *roadmap* traveling, or more specialized ones like the wavefront algorithm porposed by Khatib in [Kha86] and used in our final system (explained in details in Section 3.4). Actually the main locomotion system presented in this thesis (Chapter 3) uses a hybrid **Grid-Based** 3D path planning algorithm, as the **Grid-Based** algorithms adapts better

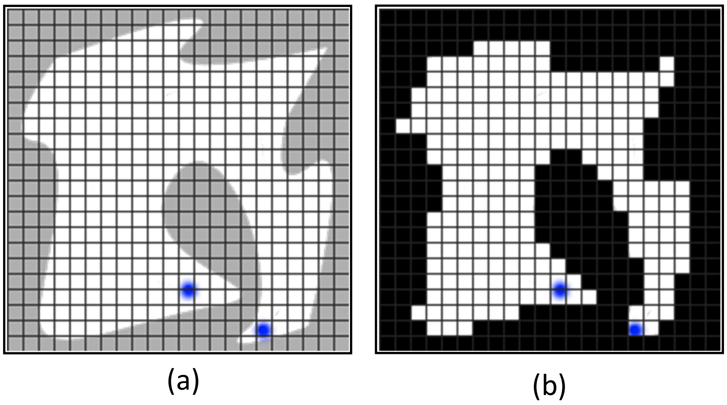


Figure 2.24: *Example of possible planning grid. a) Overlaying the grid. b) Discretizing the environment.*

to the complex environment that we simulate (a heightmap with moving obstacles, shown in Figure 3.14).

In animation systems, path planning can be more complicated as it needs to take into account the actual abilities of the virtual creatures being animated. This complexity is more apparent in robotics as controllers have to accommodate for the capabilities of each robot and the real-physical world constraints. A typical solution for this problem in robotics is to use the current robot configuration plus a discrete set of feasible, statically-stable footstep placement positions with their associated pre-computed and balanced stepping motions. Using these data, a set of stable footsteps are generated, that connects the starting position with the goal position while satisfying extra constraints: comfort constraints, allowing/disallowing stepping on obstacles, *etc.* Obstacles can be either static [KKK+01b, KKN+03] or dynamic [CLC+05] (ASIMO™ Humanoid robot).

Finding the collision-free trajectory that navigates through the environment for a character with many DOF while generating the animation itself can be quite a challenging task. The problem is the near infinite number of the 3D creature poses, which means that a certain pose could easily invalidate a path that could have been valid with another pose. This problem is called **Motion Planning** and several methods were proposed to solve it. Most notably the Randomized Path Planning (RPP), the Probabilistic Path Planning (PPP), the Reinforcement learning (RL) techniques and Greedy-based motion planners.

In the RPP, the planner tries *randomly* several configurations (joint angles, chosen grasps, *etc.*) and several 3D/2D position in the environment (near randomly sampled *roadmap*) in a way that moves the character closer to its goal configuration, then the planner converges toward the best step based on the gradient descent method. As this technique can fall deep into a local minima or enter in collision, a common solution is to use *backtracking*: the planner rolls back the current motion plan to a chosen backtracking point and then restarts the planning process from there. PPP techniques differ only in their *roadmap* generation as they use probability calculations when sampling the environment (waypoints picking) [CLS03, SKG05]. RPP and PPP motion planners are used to generate a collision free animation for articulated figures. Like when manipulating objects: grasping an object from inside a box while walking [SKF07]. Or to generally plan the locomotion of a human model with many DOF in a complex environment [PSL02, YKH04, PZLM10]. Kalisiak and van de Panne in [KvdP00] (Figure 2.25) generate locomotion for a human model that can walk, crawl, climb and swing in a constrained environment using RPP. They used a simple representation of the human body with only 10-links (Figure 2.25(a)) and manually placed grasp points (Figure 2.25(b)) that the character can use as a foothold, handhold or both. They have an FSM (Figure 2.25(c)) to represent the four modes of locomotion, possible transitions among them, as well as their relative preference. Posture correction step is introduced at key points in the solution as a mean of modeling preferences for particular posture characteristics. Trajectory filters are added to ensure the fluidity of the final synthesized motion.

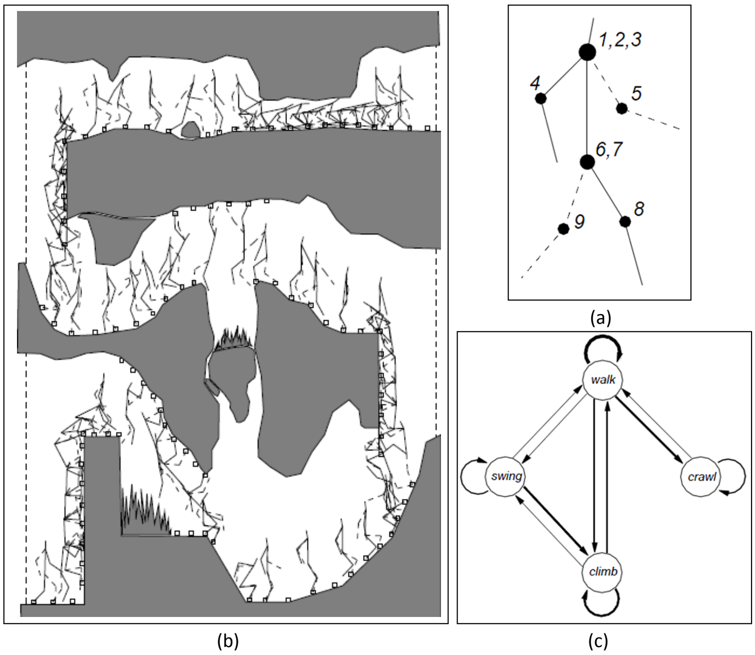


Figure 2.25: *Grasp based planning, courtesy of [KvdP00]. a) Simplified representation of the human body. b) Manually placed grasp points and the final animation. c) The FSM that represents the transition between the locomotion modes.*

RL based motion planner use controllers that learns the optimal policy, in offline, using the immediate and long-term reward notion: selecting the next action (motion clip) so that the long term reward is maximized (reaching a point in space or grasping), at every time step and from any possible state. In run-time they only need to define a goal (for example a needed 3D position), and their controller planes everything based on what it learned [LZ08]. This type of systems can be successfully used to generate in real-time locomotion animation agents with dynamic obstacles, hostile enemies and needed target positions [IAF05].

Finally there is Greedy-Based motion planner that tries all possibilities before choosing the good one, like the one proposed by Lau and Kuffner in [LK06]. They use a **Grid-Based** planning technique to generate the actual path between the start position and the goal position. Then they use a typical gait FSM to generate trees (of fixed depth) that represent all possible transitions starting from each FSM state. Like after walking, the character can turn left, continue walking or turn right, then starting from the previous left turn node the character can turn left again, continue walking or turn right and so on. They decomposed the generated path into intermediates waypoints. Then, by placing the pre-computed tree on each intermediate start pose, they are able to connect the waypoints using simple tree search techniques. Final animation is generated by simply traveling the chosen path in the tree and playing the corresponding clips associated to each tree node.

This **Motion Planning** problem is solved implicitly in our thesis. The system presented in the following chapters allows the user to control the speed and direction of the simulated creatures in real-time (video games approach), and it reacts near instantly to his commands. While at the same time, the generated locomotion is always adapting to the environment: collision avoidance, heightmap adaptation, *etc.* So our system is capable of following any logical trajectory (that does not go through walls for example) that navigates through the environment without any explicit **Motion Planning** steps.

As simulated environments can be dynamic, more and more path planning techniques are starting to integrate the temporal component when generating the final trajectory [LLKP11, LLL11]. They use dynamic space-time *roadmaps* to generate a valid path that avoids obstacles (static and dynamic) and uses moving platforms in order to achieve the target goal. We solve this dynamic path planning for the feet trajectories using a different approach that we detail in Chapter 3.

2.7 Conclusion

In the previous sections we presented a detailed overview of the animation techniques in general, and locomotion controllers in specific. Which are classified into two main categories [MFCD99, vWvBE$^+$10]: **Data-Driven** techniques and **Procedural-Based** techniques with two subcategories *kinematics-based* and *physics-based* .

Choosing **Procedural-Based** techniques over the **Data-Driven** ones was intuitive in our case as motion data for the multi-legged characters that we target are either rare or non-existent. Although that **Data-Driven** techniques are the most natural ones, as they capture the essence and implicit style of the captured subjects. Plus, they are well adapted to real-time applications, as the motion data is replayed with no extra calculation cost. Nevertheless, the difficulty sometimes to capture motion data and the need for many processing layers when the context or morphology changes made us deviate from this category of techniques.

Procedural-Based techniques are more generic and adapt better to the context and morphologies, which are important points to the recent and ever demanding complex simulations (and video games). The use of dynamics in the *physics-based* techniques allowed many systems to capture this realism through the simulation of real world physics, generating animations that are more realistic than the *kinematics-based* ones. The generated locomotion are quite believable with immigrant life-like movements not coded explicitly in the controller [KKI02, KKI06, dLMH10, MdLH10] (to name a few). But these techniques have several drawbacks. One of the most important ones is the lack of controllability. *Physics-Based* techniques lack the predictability on the resulting animation, since the interaction with the system occurs only by indirect physical forces. Although that in computer graphics visual fidelity is more important than physical accuracy. Another drawback is the bad performance of the *physics-based* techniques, they cannot animate more than one or two characters. They are computationally expensive due to the number of equations to solve (even after simplification): equations of external forces acting on the articulated body, equations of internal torques plus constraints equations. That limits the possibility of animating several characters at the same time, a problem that made us deviate from these techniques.

Appendix B shows some of the problems when simulating real world physics using State-of-The-Art commercial and open-source physics engines.

Kinematics-Based techniques are the best choice in the context of this thesis, as they offer a good compromise between the amount of control over the motion, the naturalness of the resulting animation and calculation time. The produced animation is controlled explicitly, which means producing the exact needed movement at the exact assigned time. Their use of biomechanics and empirical data ensure the naturalness of the produced locomotion. And by concentrating on the plausibility of the movement more than the accuracy of the simulation, these techniques are capable of animating a significant number of creatures in comparison to the *physics-based* one. Brazel *et al.* introduces the idea of plausible motion in [BHW96], they show that end-users are more interested in the visual plausibility of the movement than the actual calculations. Brazel extends this plausibility of motion concepts to fake the dynamics of ropes and springs in [Bar97], his techniques were successfully used to animate *Slinky Dog* in the movie Toy Story™. Most *kinematics-based* systems (like the one proposed in Chapter 3) aim at satisfying this plausibility of the movement while staying as accurate as possible.

Sure that the chosen *kinematics-based* techniques offer lots of advantages but in the same time there is many problems that needs to be addressed and solved. The Main problems are creating a morphology independent animation system, the need for a real-time dynamic feet trajectory planner as the simulated environment in this thesis is dynamic (Chapter 3) and the system should be capable of animating dozens of creatures in real-time (main focus throughout this thesis). Also, there is a small gap between plausibility and naturality which is sometimes difficult to achieve, a problem that we address in Chapter 4 and one of the main contributions.

We hope that our final system pushes the industry and developers of real-time simulators (*e.g.* video games) toward relaying less on **Data-Driven** techniques, making virtual worlds less repetitive and more rich when it comes to unique characters locomotion styles.

Chapter 3

Animating Multi-Legged Characters

Contents

3.1 System Overview 43

3.2 Character Controller 47

3.2.1 Feet Movement Task 47

3.2.2 3D Pelvis Movement Task 48

3.3 Gait Manager 51

3.4 3D Path Constructor 53

3.4.1 Environment Grid-Based Representation 54

3.4.2 3D Path Construction 57

3.5 Footprints and Feet Path Planning 61

3.5.1 Potential Footprints and Trajectories scoring 62

3.5.2 Finding Best Couple 63

3.6 System Loop 66

3.6.1 Sequence 1 66

3.6.2 Sequence 2 67

3.6.3 Sequence 3 68

3.6.4 Sequence 4 69

3.7 Level of Details 70
3.8 Performance and Results 71
3.9 Conclusion 78

The goal of our system is to generate locomotion animations for multi-legged characters. Locomotion is the act of moving from one place to another. For most terrestrial animals that means putting one foot in front of the others in a successive way until reaching the designated point of interest (target). During a normal foot movement there are two main phases *Stance* and *Swing* phase [INM66]. During the *stance* phase a foot is blocked on the ground. While in *swing* phase (flight phase) the foot flies in a parabolic-like curve toward its target without any ground contact. Locomotion cycle is the act of repeating these feet movements, based on a certain rhythm or tempo, called a Gait Pattern (Figure 2.8).

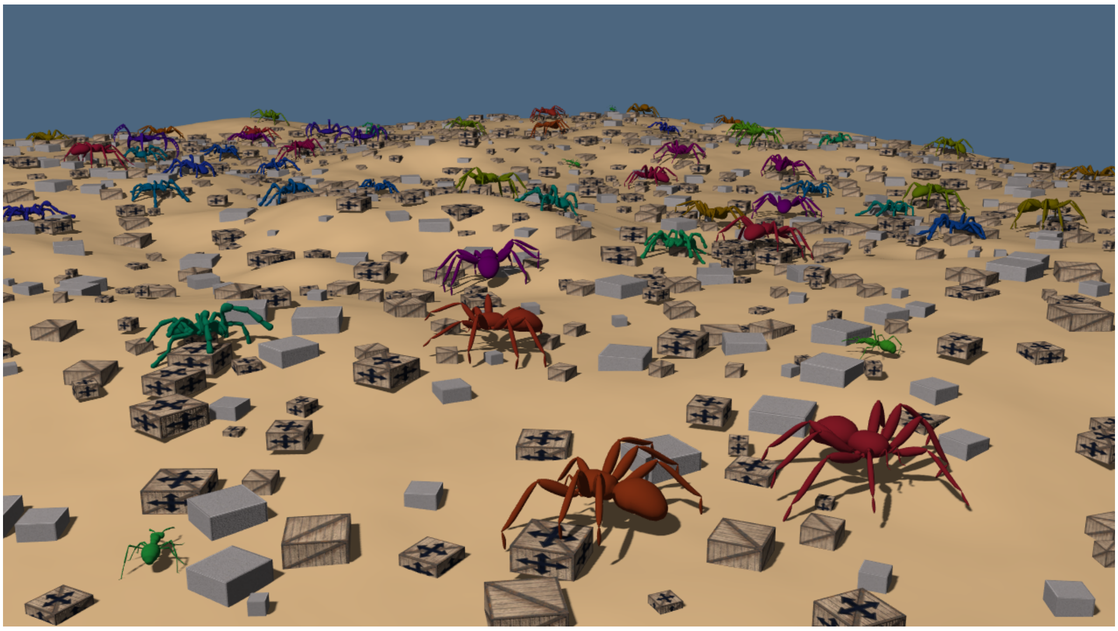


Figure 3.1: *Snapshot from final simulation.*

Our system, shown in Figure 3.1, follows these principles when generating, procedurally and in real-time, multi-legged characters locomotion. Its main goal is to satisfy the four objectives: adaptability, controllability, believability and efficiency. Plus, it covers a huge panel of real-life creatures like insects: ants, beetles, crickets, *etc.* quadrupeds: dogs, horses, wolves, *etc.* arachnids: spiders, scorpions, *etc.* reptiles: lizards, crocodiles, Gecko's, *etc.* and also covers other imaginary legged creatures like 3-legged or 5-legged robots.

In the following sections we will explain in details our system (Section 3.1), and its main components: character controller (Section 3.2), gait manager (Section 3.3), 3D path constructor (Section 3.4) and footprints and feet path planning (Section 3.5). Section 3.6 gives a more in-depth step-by-step examples of our system. In Section 3.7 we explain the implemented Level of Detail (LOD) techniques that allowed the animation of even more multi-legged characters in real-time. And finally we speak about the performance of the system and compare it to other proposed systems in Section 3.8.

3.1 System Overview

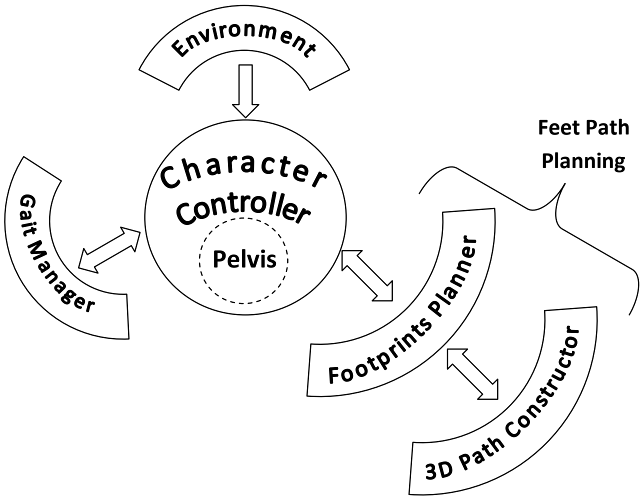


Figure 3.2: *Animation/Locomotion system overview.*

The overall locomotion process is computed by four main blocks as shown in Figure 3.2:

- The character controller is the central main structure that manages the overall locomotion of the multi-legged character (see Section 3.2). It relies on the three other structures to compute the motion of the feet and the overall pelvis displacement.

- The gait manager regulates the feet tempo according to the movement patterns defined by the user (see Section 3.3).
- The 3D path constructor that computes the 3D trajectory for a foot using an efficient discrete representation of the environment that can be easily maintained and updated with dynamic objects (see Section 3.4).
- The footprints planner evaluates for each foot all the possible targets and trajectories in real-time, and chooses the best **couple**: best 3D trajectory that navigates through the environment toward the best footprint target (see Section 3.5).

These four blocks work together (Figure 3.3) in order to generate the movement of the feet and the pelvis in 3D. In a complex and dynamic environment as shown in Figure 3.14, that can be defined by a heightmap (Figure 3.12) and contains many obstacles (Figure 3.13) with different types: moving or static.

First, the character controller on Figure 3.3 moves the creature pelvis based on several criteria's as explained in Section 3.2.2. Based on the pelvis movement and the gait manager (Section 3.3), the character controller queries the footprints planner about the best **couple** for each foot in *swing* phase (Section 3.5). The footprints planner uses the 3D path constructor (Section 3.4) in order to calculate these best **couples**. Then, the character controller moves the feet in *swing* phase based on their best **couple** and blocks the feet in *stance* phase (Section 3.2.1). We use the CCD IK system to calculate the position of the intermediate legs joints that connect the creature pelvis with its feet, we chose it over other IK systems because of its performance and simplicity. By animating the pelvis, the feet and the legs, we generate a complete lower body animation which is a full body animation for most of the multi-legged characters that we address. Finally, the feet send feedback information about their movement to be incorporated in the pelvis movement in the next simulation step.

Our system generates the locomotion animations using several input parameters that the user provides or controls in real-time. All the parameters (except the morphological ones) can be edited at any time by the user or by any automatic character controller when the focus is to animate a crowd of creatures. An overview of all the parameters follows and can be found on Appendix D.

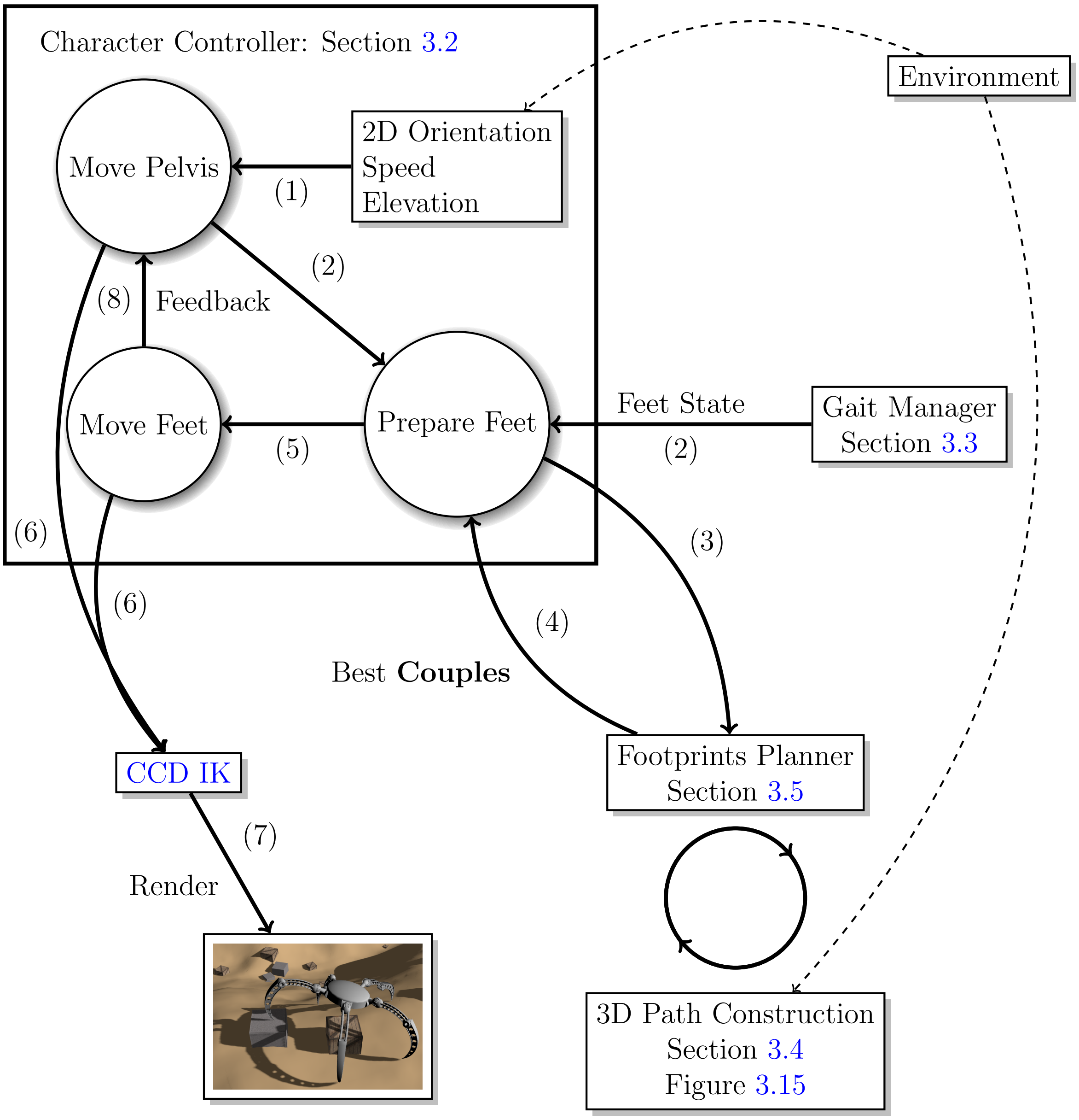

Character Controller: Section 3.2
Environment
Move Pelvis
2D Orientation
Speed
Elevation
(1)
(2)
(8)
Feedback
Move Feet
Prepare Feet
(5)
Feet State
(2)
Gait Manager
Section 3.3
(6)
(3)
(6)
(4)
Best **Couples**
CCD IK
Footprints Planner
Section 3.5
(7)
Render
3D Path Construction
Section 3.4
Figure 3.15

Figure 3.3: *The general steps taken by our locomotion system in order to calculate the creature final locomotion animation.*

- **Multi-legged character morphology**: the user provides an initial static skeleton that can be associated with a skinned mesh. To automatically map the morphology, the user provides the system with the name of each leg hip joint and the name of its *end-effector* (*e.g.* the joint before the foot). Out of these inputs the system detects the number of feet, leg sections, the relative hip positions, spine model (if it exists) and the initial feet relative positions. The user also needs to provide the system with the desired joint limits (for the legs CCD IK system and the spine model (if it exists)), real body center, projected body bounding box, body thickness, and finally the projected foot shape (if it exists) an example is shown in Figure 3.4.

- **Gait/Tempo**: using our interface, the user designs each foot cycle (*stance* and *swing* phases). These cycles describe the tempo (pattern) of the feet movement. The final gait can be symmetrical or asymmetrical.
- **Locomotion speed**: speed of the movement in *meters per second*.
- **Locomotion direction**: the needed orientation on the ZX-plane.
- **Step height**: the preferred foot step height in *meter*.
- **Feet spacing**: the preferred position of each foot, relative to the pelvis.

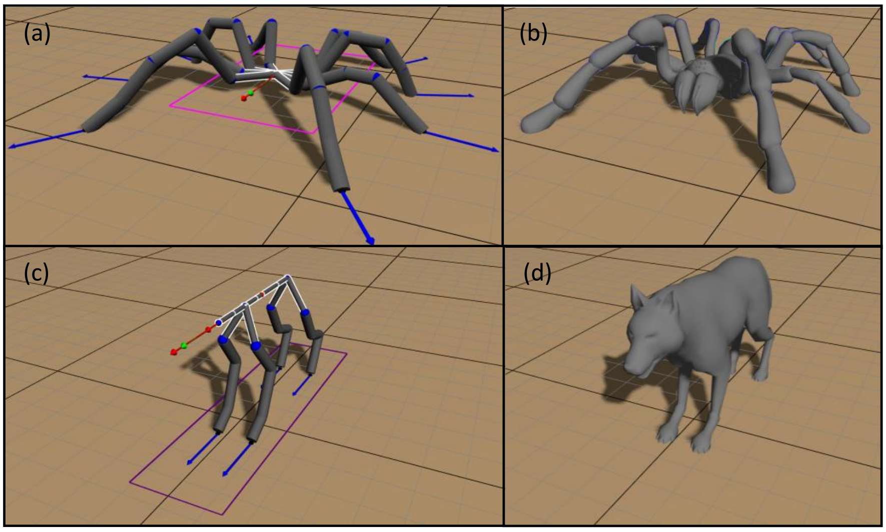


Figure 3.4: *Top: spider model. Bottom: wolf model. (a,c) Internal representation with the IK systems, in pink is the projected bounding box of the creature body, in blue the feet orientation. (b,d) The actual 3D mesh.*

3.2 Character Controller

The character controller is the coordinator of the overall system. It is in charge of two main tasks: managing the movement of the feet (Section 3.2.1) and computing the pelvis 3D movement (Section 3.2.2). Everything is generated based on the user needs, the environment and the feet feedback.

3.2.1 Feet Movement Task

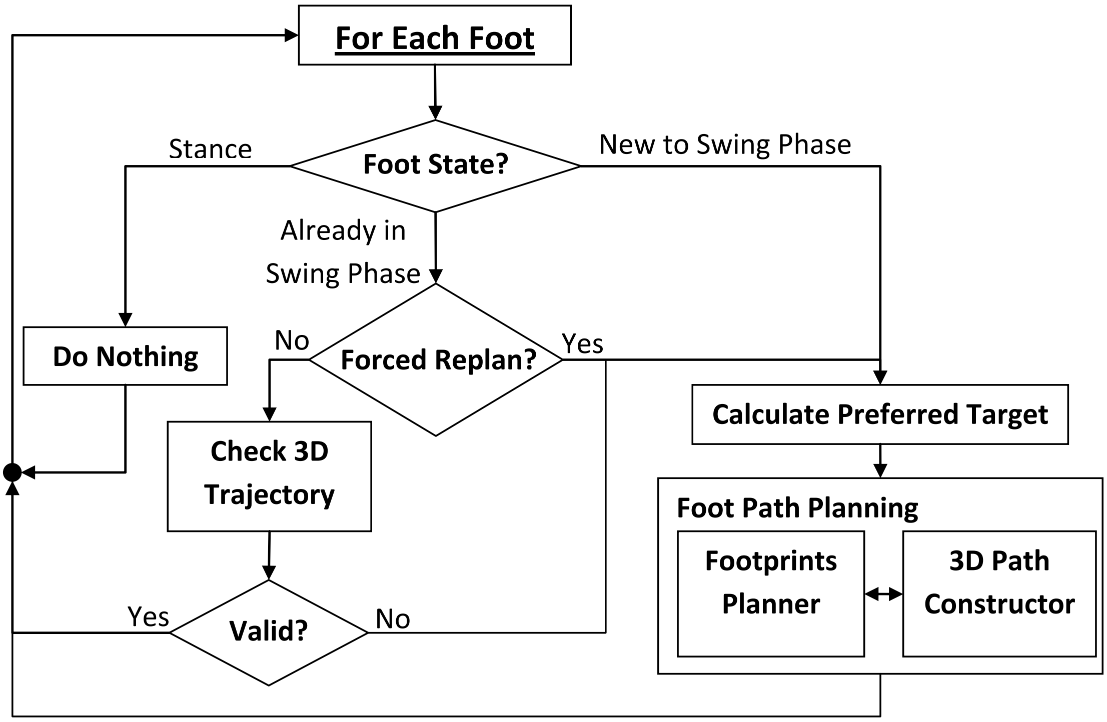

For Each Foot
Stance
Foot State?
New to Swing Phase
Already in Swing Phase
Do Nothing
No
Forced Replan?
Yes
Calculate Preferred Target
Check 3D Trajectory
Foot Path Planning
Footprints Planner
3D Path Constructor
Yes
Valid?
No

Figure 3.5: *Main job of the character controller per foot: based on the current foot state it can take different decisions.*

Figure 3.5 explains how the character controller manages the movement of the feet. At each time step, the gait manager informs the character controller about the feet that are going to enter in *swing* phase (detailed in Section 3.3). For each one of these feet, the character controller calculates a preferred footprint target, shown in Figure 3.6. Where the foot step length is calculated based on the current speed, multi-legged character morphology and the foot relative position that the user can impose as illustrated and explained directly in Figure 3.7. After calculating this foot step length, we use the initial position of the foot to calculate this preferred 3D target position based on the environment (see Figure 3.6). Finally, the character controller calls the foot path planner to compute the 3D trajectory that this foot will follow during its flight phase, as described in Section 3.5.

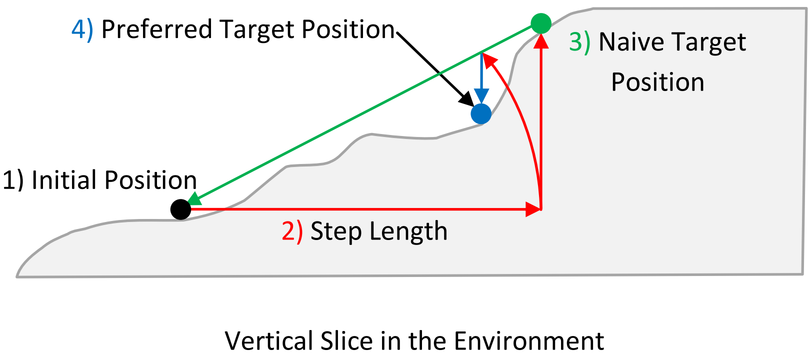

4) Preferred Target Position
3) Naive Target Position
1) Initial Position
2) Step Length
Vertical Slice in the Environment

Figure 3.6: *How the computation of the preferred footprint target for each foot is done. The vertical slice in the environment can contain objects and obstacles. The naive target position is calculated by querying the environment about the elevation of the 2D point resulted from the initial position + the step length.*

Feet that were already in *swing* phase may need a new a 3D trajectory (path) for several reasons: a new pelvis orientation (when turning), a new desired relative position of the foot (Figure 3.7), a new overall speed, or something changed in the environment thus invalidating its current 3D trajectory or its current target. In these cases the character controller processes this foot as if it just started its *swing* phase, and therefore it is redirected to the previously explained foot path planning phase. For the *stance* feet, the character controller simply blocks their 3D position on the ground or on an obstacle.

3.2.2 3D Pelvis Movement Task

The 2D movement of the pelvis on the ZX-plane (assuming the Y-axis is up) is calculated using only the speed and orientation, The computation of the pelvis height is more complex, as shown in Figure 3.8: we first construct a convex hull based on the height of the environment underneath the creature (using its bounding

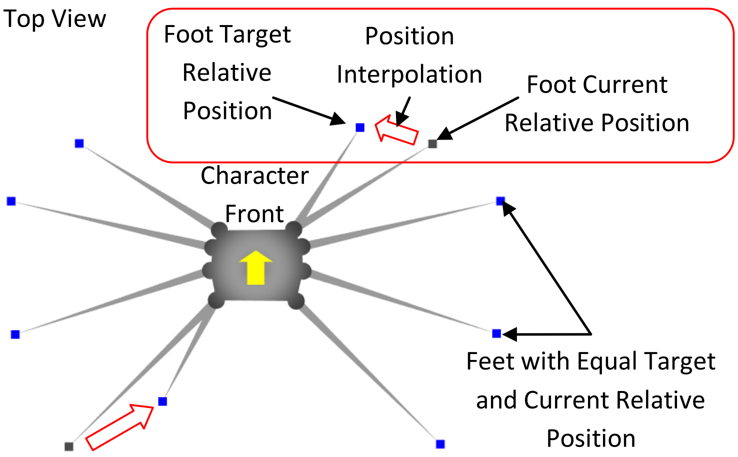

Top View
Foot Target Relative Position
Position Interpolation
Foot Current Relative Position
Character Front
Feet with Equal Target and Current Relative Position

Figure 3.7: *8-Legged Character Feet Spacing Interface*

box) and the actual position of the feet. By projecting the multi-legged character center on this convex hull we get the needed pelvis height and by adding the body thickness we get the actual pelvis height. The pitch angle of the multi-legged character body is calculated directly from the convex hull: it is the slope of the line segment that contains the projected center of the multi-legged character.

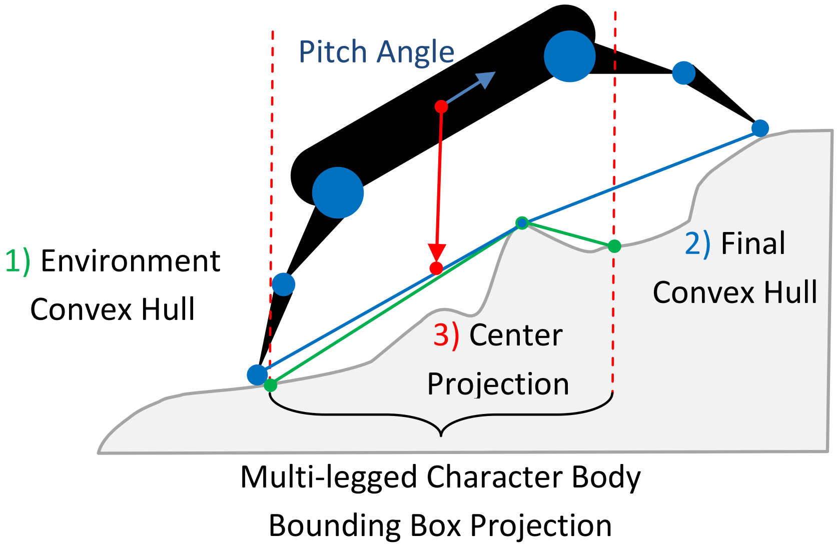

Pitch Angle
1) Environment Convex Hull
2) Final Convex Hull
3) Center Projection
Multi-legged Character Body Bounding Box Projection

Figure 3.8: *Again: a vertical slice in the environment can contain objects and obstacles. Computation of the pelvis' height using several convex hulls that covers the environment slice.*

After moving the feet, the character controller refines the pelvis movement based on the feet error feedback (shown in Figure 3.9). This error comes from the fact that each foot can choose a better target (different from the previous preferred target) that better adapts to the context (Section 3.5). For that, the character controller averages the offset between the preferred footprint targets (Figure 3.9 in blue) and the effectively chosen one (in green). The controller incorporates this offset in the pelvis movement when it is quite significant, based on the multi-legged character morphology, otherwise it discards it. By doing so, the upper body is always the one imposing the final trajectory while the lower body tries to follow [Ber09].

As we do not add any other movement to the pelvis, final animation can look rigid and un-natural. Which is an important point that we discuss and resolve in Chapter 4.

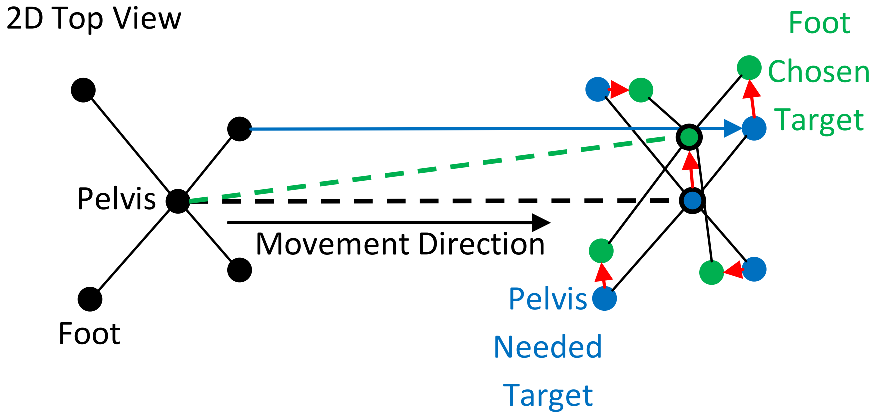


Figure 3.9: *Feet feedback: during the computation of the pelvis displacement, we use the feet feedback.*

3.3 Gait Manager

The role of the gait manager is to organize and visualize the pattern of the feet's cycle and to perform the transition between patterns. Since locomotion is cyclic, it seemed natural to represent the gait with circles. As illustrated in Figure 3.10, each circle represents a foot, with the colored sectors representing the *swing* phase portion of the foot movement. The feet needle activates sectors and deactivates others based on its current position, while turning clockwise. Activating a sector means that the corresponding foot should enter its *swing* phase.

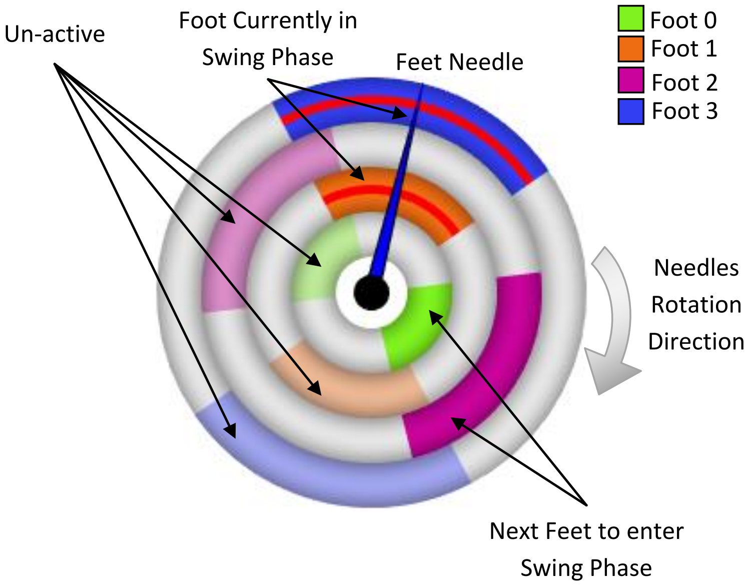

Un-active
Foot Currently in Swing Phase
Feet Needle
Foot 0
Foot 1
Foot 2
Foot 3
Needles Rotation Direction
Next Feet to enter Swing Phase

Figure 3.10: *An example of a four feet gait as shown by the gait manager. With this interface, the user can edit the pattern to create the desired gait. In this figure each foot cycle (swing and stance phase) is repeated twice to show the capabilities of our gait manager.*

Gaits interpolation/transition. During a locomotion cycle, most characters change their gait constantly to adapt to the environment, to change movement style and while turning, slowing down, accelerating *etc.* To accommodate for this changes in the gait and to introduce more variety in the locomotion styles, the gait manager allows the transition between any needed gaits on the fly using our interpolation system.

Each *swing* phase sector has a start and a finish, we compute the transition to another (destination) gait by simply interpolating the start of the source sector toward the start of the destination sector using a clockwise or counterclockwise direction. To choose this interpolation direction, we consider the area covered by the source and destination gait sector (the union) as the allowed area of interpolation and all the rest of the circle is forbidden. The postulate that this forbidden zone (Figure 3.11) is the zone where the user does not want the interpolation to pass through. Based on that, we chose the direction of interpolation in the direction that does not pass by the finish of the destination sector. By doing so, the interpolation does not pass by this forbidden zone. In Figure 3.11(a) we chose the clockwise direction while in Figure 3.11(b) we chose the counterclockwise one. The interpolation of the duration of the *swing* phase (the size of the sector) is quite straightforward, and the speed of all these processes (transition time) can be fixed in advance by the user. To effect the new gait on the foot, we replace its disk when it is in *stance phase* (not active) with the new disk calculated in the interpolation process, resulting in seamless and logical interpolation.

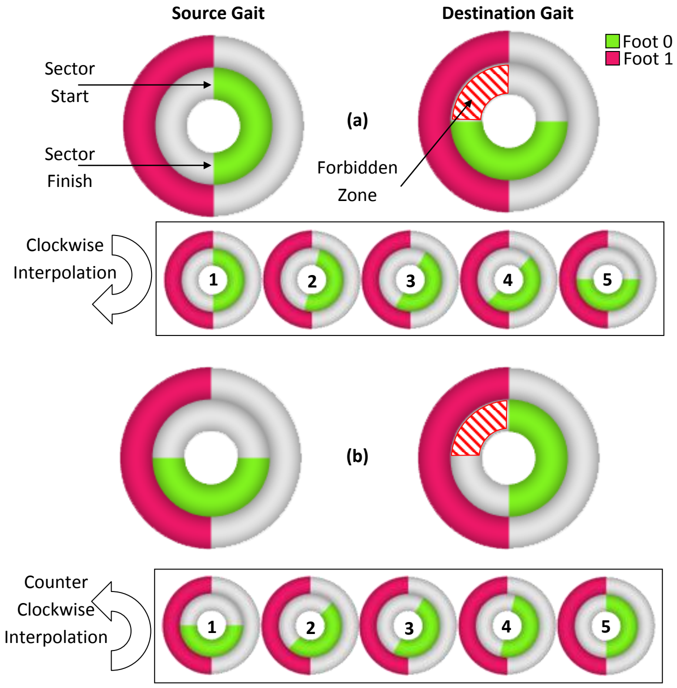

Source Gait
Destination Gait
🟩 Foot 0
🟥 Foot 1
Sector Start
Sector Finish
(a)
Forbidden Zone
Clockwise Interpolation
1
2
3
4
5
(b)
Counter Clockwise Interpolation
1
2
3
4
5

Figure 3.11: *Interpolations options. a) Deciding to do a clockwise position interpolation based on the actual disposition of the gait disks. b) Deciding to do a counter-clockwise position interpolation.*

3.4 3D Path Constructor

During the *swing* phase, each foot has a current position (source) and a footprint target. The role of the 3D path constructor is to compute the foot 3D trajectory that navigates through the environment, from this current position toward any footprint target without colliding with any obstacle. 3D path construction is requested several times with different footprint targets by the foot path planner as it will be explained in Section 3.5. That is why there is a huge emphasis on performance.

The working environment is quite complex as shown in Figure 3.14 and generated using a heightmap (Figure 3.12) with several type of obstacles (Figure 3.13). Meaning that we use a huge amount of triangles to represent this environment and render it.

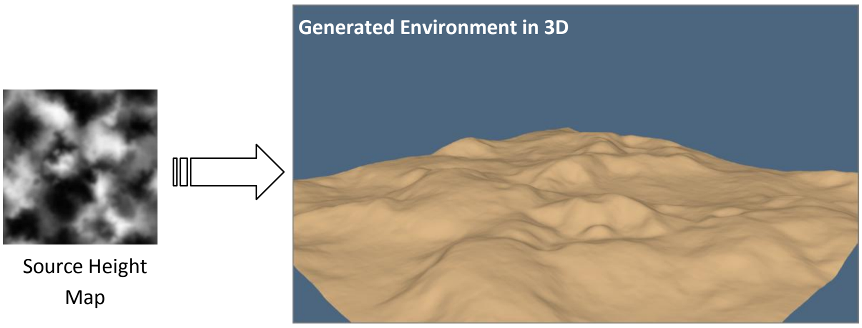


Figure 3.12: *Example of a heightmap image (left) and the corresponding generated environment (right).*

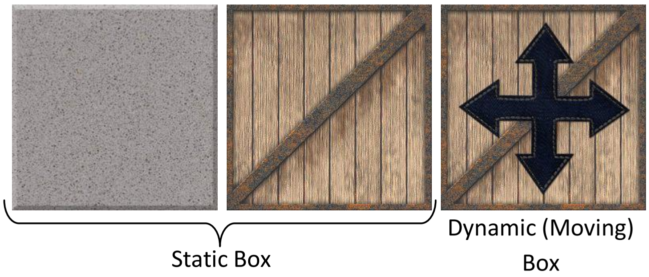


Figure 3.13: *Obstacles (boxes) types: static and moving.*

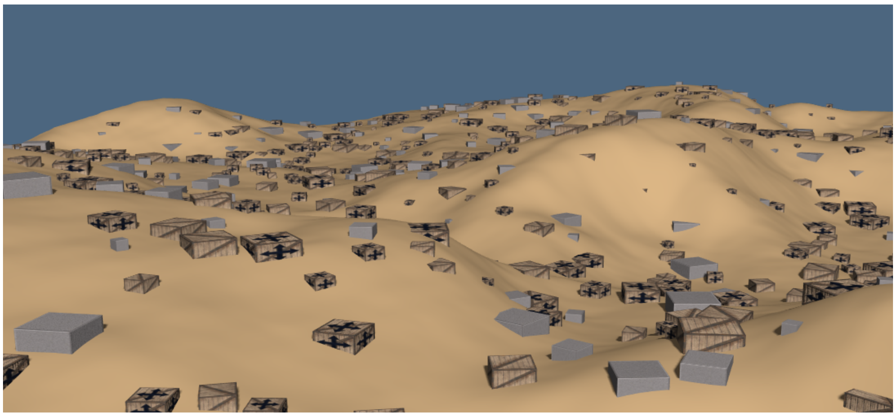


Figure 3.14: *An example of a complex and dynamic environment that the system can have as input.*

That is why we have oriented this path construction process toward a discrete grid-based planning approach, as it will be explained in this section. Actually, once the environment is converted and represented by our grids/maps (Section 3.4.1), the path construction (Section 3.4.2) becomes independent of the object's complexity. Moreover, discrete algorithms are easier to implement compared to vectorial-based ones.

Figure 3.15 illustrates the steps needed in order to construct the final 3D path. The algorithm starts by discretizing the environment (Section 3.4.1). Then, it constructs the 2D path on the ZX-plane using the wavefront algorithm, bresenham's algorithm and a b-spline curve (Hermite). After that the 2D path curve is discretized, elevated and the final 3D path is constructed (Section 3.4.2).

3.4.1 Environment Grid-Based Representation

We convert the 3D environment near the animated multi-legged character into two 2D grids: the *obstacles' map* and the *elevations map*. The *obstacles' map* describes the areas of the environment where the feet are allowed to pass, as illustrated in Figure 3.16, where black cells represent the obstacles and called forbidden cells. While the *elevations map* contains the elevation of the highest obstacle in each cell as shown in Figure 3.17. These maps are relatively small as they are only computed around the animated multi-legged character, speeding up calculations.

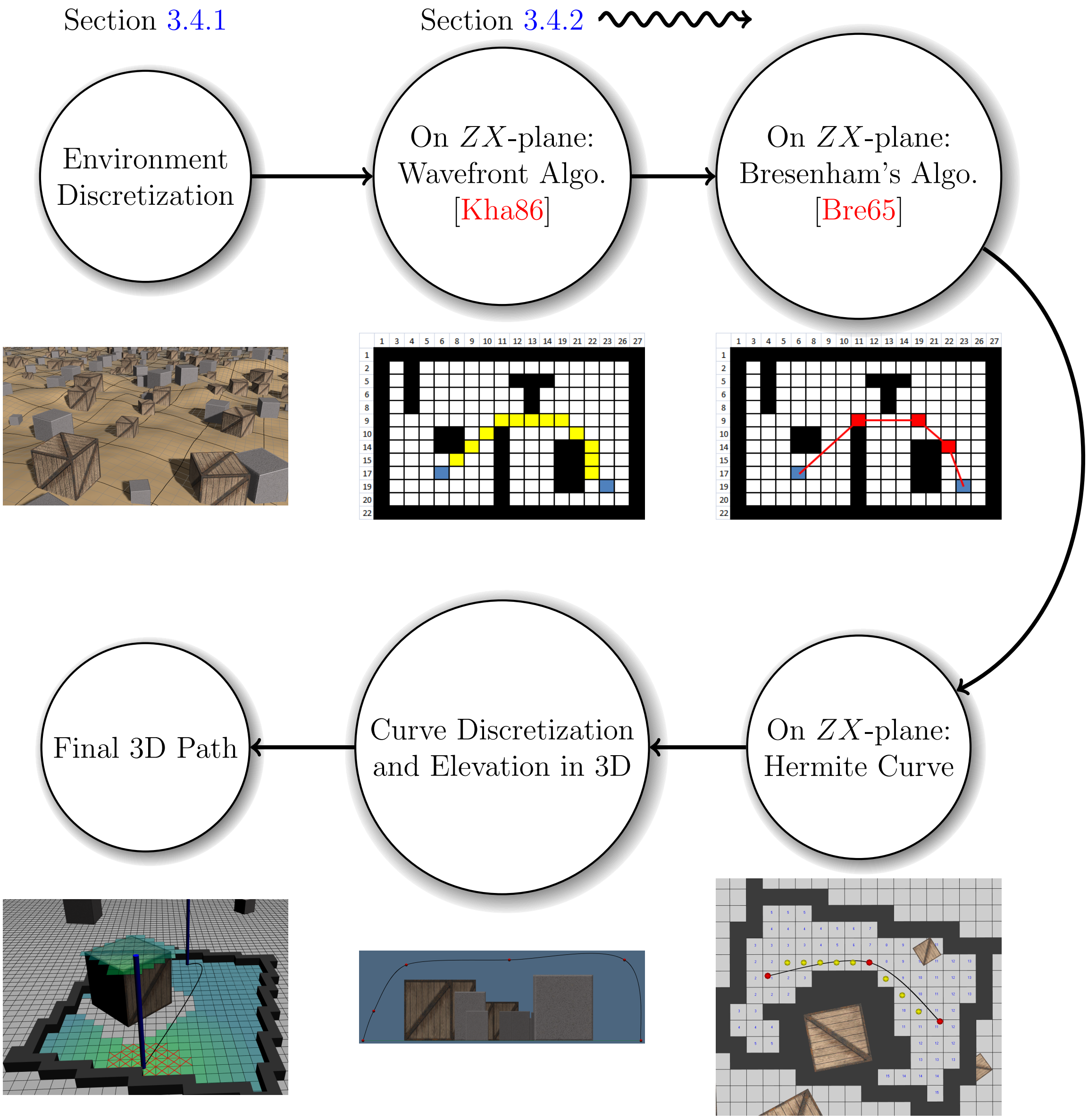

Section 3.4.1
Section 3.4.2
Environment Discretization
On ZX-plane: Wavefront Algo. [Kha86]
On ZX-plane: Bresenham's Algo. [Bre65]
On ZX-plane: Hermite Curve
Curve Discretization and Elevation in 3D
Final 3D Path

Figure 3.15: *A step by step illustration that shows how the final 3D path is constructed. The algorithm starts by discretizing the environment. Then, it constructs the 2D path on the ZX-plane using the wavefront algorithm, bresenham's algorithm and a b-spline curve (Hermite). After that the 2D path curve is discretized, elevated and the final 3D path is constructed.*

These two maps are computed using the terrain heightmap and the obstacles (objects): the bounding box of each object is voxelized onto the maps. Concave objects are subdivided into convex ones. We pre-compute the map's representation of static objects, while we compute the map's representation of dynamic objects

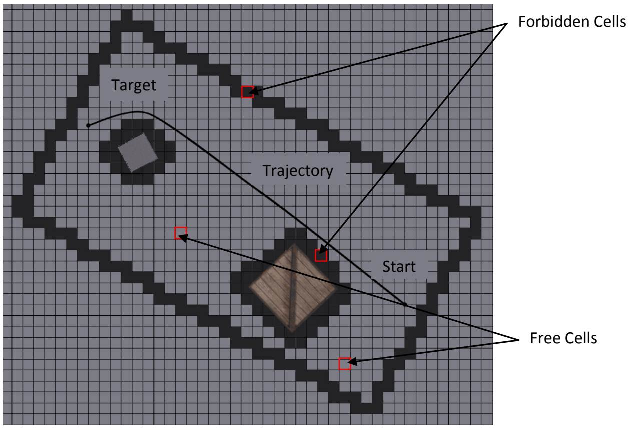

Forbidden Cells
Target
Trajectory
Start
Free Cells

Figure 3.16: *Example of the obstacles' map.*

when they move near the multi-legged character in real-time. In order to avoid legs crossing, we add for each foot the projection of other legs into the *obstacles' map* as forbidden cells. We do not need to add the trajectory of other feet nor the trajectory of dynamic obstacles as the character controller verifies the validity of each already calculated foot trajectory on each simulation step (Section 3.2.1). When it detects a collision, a new trajectory is calculated starting from the actual position of the foot.

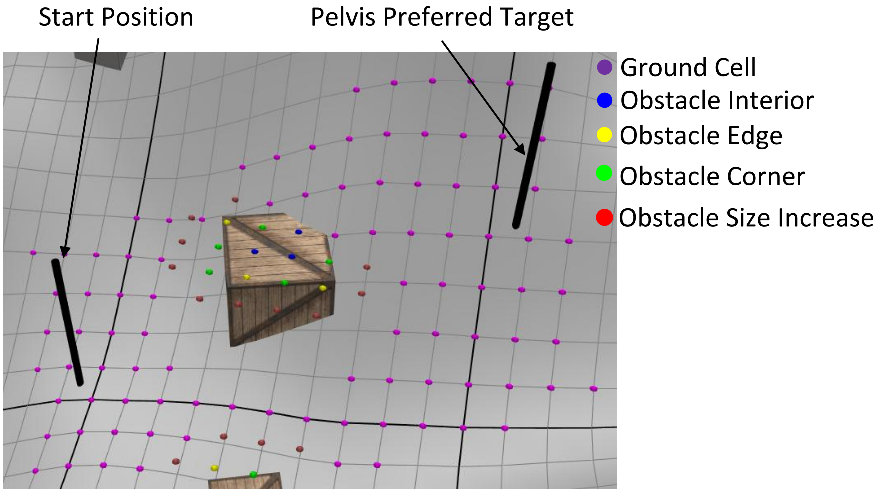

Start Position
Pelvis Preferred Target
Ground Cell
Obstacle Interior
Obstacle Edge
Obstacle Corner
Obstacle Size Increase

Figure 3.17: *An example of the environment discretization: pink spheres represent the ground, yellow spheres contain the corner of an obstacle, green spheres contain an obstacle edge, blue spheres are an obstacle interior and red spheres are an obstacle size increase.*

Since a foot is not punctual, we use the bounding box of the foot shape to increase the size of the obstacles in the opposite direction of the foot, as shown in Figure 3.18. This ensures the non penetration of the foot with the obstacles and satisfies the creature comfort: most real-life creatures, as observed by Alexander in [Ale03, Ale96], prefer to place their foot at a certain distance of an obstacle to avoid unpredicted collision or uncomfortable movement when trying to clear the obstacle during the next step.

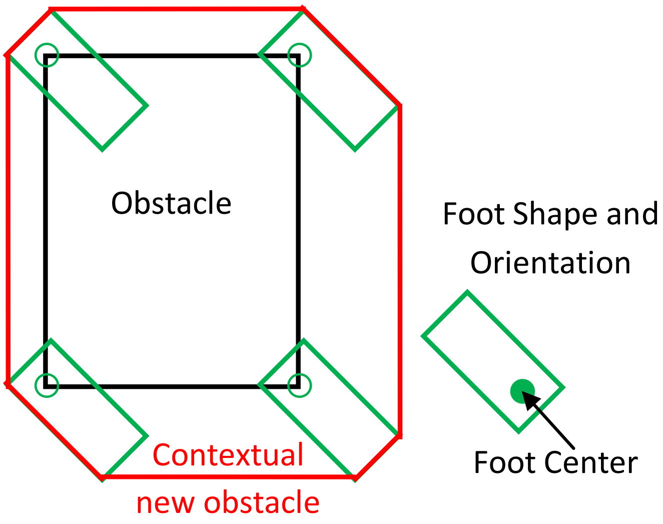
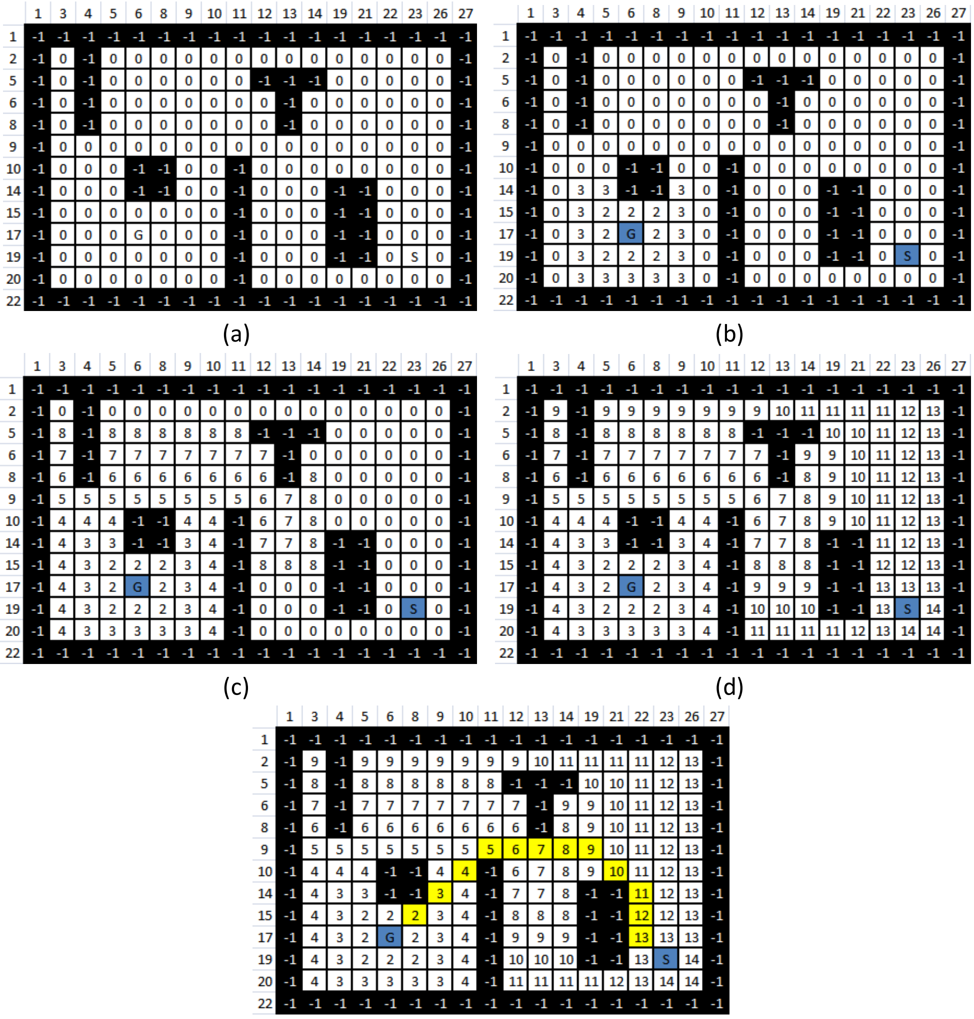


Figure 3.18: *We increase each obstacle size using the bounding box of the foot and its orientation in order to better avoid collisions.*

3.4.2 3D Path Construction

Always with a performance concern, we firstly compute the trajectory from the actual foot position toward the preferred footprint in 2D on the ZX-plane using the projections of the source and the target on this plan (in black in Figure 3.17, in blue in Figure 3.20).

Our path planner is grid-based and uses the wavefront algorithm proposed by Khatib in [Kha86] (used mainly in robotics like in [Nat11]). It is a, real-time, artificial potential field algorithm that avoids collisions using a Breadth-First Search (BFS) approach (Appendix C). The wavefront algorithm goal is to construct a path from point S (start in blue) to point G (goal in blue) through a discretized workspace, as shown in Figure 3.19(a). 0 designates a cell of free space, -1 designates a cell fully occupied by an obstacle in black. Starting at G, which is assigned the value 1 at the beginning, each cell is assigned a value which corresponds to the number of moves required for the shortest path from that cell to the goal (in a BFS way). To plan an optimal path (not always the shortest), the algorithm starts from S and picks on each step the next cell that has the minimum value of all adjacent cells until reaching the goal G. These picked cells (called the naive cells) construct the needed path, shown in yellow in Figure 3.19(e).



	1	3	4	5	6	8	9	10	11	12	13	14	19	21	22	23	26	27
1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
2	-1	0	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1
5	-1	0	-1	0	0	0	0	0	0	-1	-1	-1	0	0	0	0	0	-1
6	-1	0	-1	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	-1
8	-1	0	-1	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	-1
9	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1
10	-1	0	0	0	-1	-1	0	0	-1	0	0	0	0	0	0	0	0	-1
14	-1	0	0	0	-1	-1	0	0	-1	0	0	0	-1	-1	0	0	0	-1
15	-1	0	0	0	0	0	0	0	-1	0	0	0	-1	-1	0	0	0	-1
17	-1	0	0	0	G	0	0	0	-1	0	0	0	-1	-1	0	0	0	-1
19	-1	0	0	0	0	0	0	0	-1	0	0	0	-1	-1	0	S	0	-1
20	-1	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	-1
22	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

(a)

	1	3	4	5	6	8	9	10	11	12	13	14	19	21	22	23	26	27
1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
2	-1	0	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1
5	-1	0	-1	0	0	0	0	0	0	-1	-1	-1	0	0	0	0	0	-1
6	-1	0	-1	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	-1
8	-1	0	-1	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	-1
9	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1
10	-1	0	0	0	-1	-1	0	0	-1	0	0	0	0	0	0	0	0	-1
14	-1	0	3	3	-1	-1	3	0	-1	0	0	0	-1	-1	0	0	0	-1
15	-1	0	3	2	2	2	3	0	-1	0	0	0	-1	-1	0	0	0	-1
17	-1	0	3	2	G	2	3	0	-1	0	0	0	-1	-1	0	0	0	-1
19	-1	0	3	2	2	2	3	0	-1	0	0	0	-1	-1	0	S	0	-1
20	-1	0	3	3	3	3	3	0	-1	0	0	0	0	0	0	0	0	-1
22	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

(b)

	1	3	4	5	6	8	9	10	11	12	13	14	19	21	22	23	26	27
1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
2	-1	0	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1
5	-1	8	-1	8	8	8	8	8	8	-1	-1	-1	0	0	0	0	0	-1
6	-1	7	-1	7	7	7	7	7	7	7	-1	0	0	0	0	0	0	-1
8	-1	6	-1	6	6	6	6	6	6	6	-1	8	0	0	0	0	0	-1
9	-1	5	5	5	5	5	5	5	5	6	7	8	0	0	0	0	0	-1
10	-1	4	4	4	-1	-1	4	4	-1	6	7	8	0	0	0	0	0	-1
14	-1	4	3	3	-1	-1	3	4	-1	7	7	8	-1	-1	0	0	0	-1
15	-1	4	3	2	2	2	3	4	-1	8	8	8	-1	-1	0	0	0	-1
17	-1	4	3	2	G	2	3	4	-1	0	0	0	-1	-1	0	0	0	-1
19	-1	4	3	2	2	2	3	4	-1	0	0	0	-1	-1	0	S	0	-1
20	-1	4	3	3	3	3	3	4	-1	0	0	0	0	0	0	0	0	-1
22	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

(c)

	1	3	4	5	6	8	9	10	11	12	13	14	19	21	22	23	26	27
1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
2	-1	9	-1	9	9	9	9	9	9	9	10	11	11	11	11	12	13	-1
5	-1	8	-1	8	8	8	8	8	8	-1	-1	-1	10	10	11	12	13	-1
6	-1	7	-1	7	7	7	7	7	7	7	-1	9	9	10	11	12	13	-1
8	-1	6	-1	6	6	6	6	6	6	6	-1	8	9	10	11	12	13	-1
9	-1	5	5	5	5	5	5	5	5	6	7	8	9	10	11	12	13	-1
10	-1	4	4	4	-1	-1	4	4	-1	6	7	8	9	10	11	12	13	-1
14	-1	4	3	3	-1	-1	3	4	-1	7	7	8	-1	-1	11	12	13	-1
15	-1	4	3	2	2	2	3	4	-1	8	8	8	-1	-1	12	12	13	-1
17	-1	4	3	2	G	2	3	4	-1	9	9	9	-1	-1	13	13	13	-1
19	-1	4	3	2	2	2	3	4	-1	10	10	10	-1	-1	13	S	14	-1
20	-1	4	3	3	3	3	3	4	-1	11	11	11	11	12	13	14	14	-1
22	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

(d)

	1	3	4	5	6	8	9	10	11	12	13	14	19	21	22	23	26	27
1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
2	-1	9	-1	9	9	9	9	9	9	9	10	11	11	11	11	12	13	-1
5	-1	8	-1	8	8	8	8	8	8	-1	-1	-1	10	10	11	12	13	-1
6	-1	7	-1	7	7	7	7	7	7	7	-1	9	9	10	11	12	13	-1
8	-1	6	-1	6	6	6	6	6	6	6	-1	8	9	10	11	12	13	-1
9	-1	5	5	5	5	5	5	5	5	6	7	8	9	10	11	12	13	-1
10	-1	4	4	4	-1	-1	4	4	-1	6	7	8	9	10	11	12	13	-1
14	-1	4	3	3	-1	-1	3	4	-1	7	7	8	-1	-1	11	12	13	-1
15	-1	4	3	2	2	2	3	4	-1	8	8	8	-1	-1	12	12	13	-1
17	-1	4	3	2	G	2	3	4	-1	9	9	9	-1	-1	13	13	13	-1
19	-1	4	3	2	2	2	3	4	-1	10	10	10	-1	-1	13	S	14	-1
20	-1	4	3	3	3	3	3	4	-1	11	11	11	11	12	13	14	14	-1
22	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

(d)

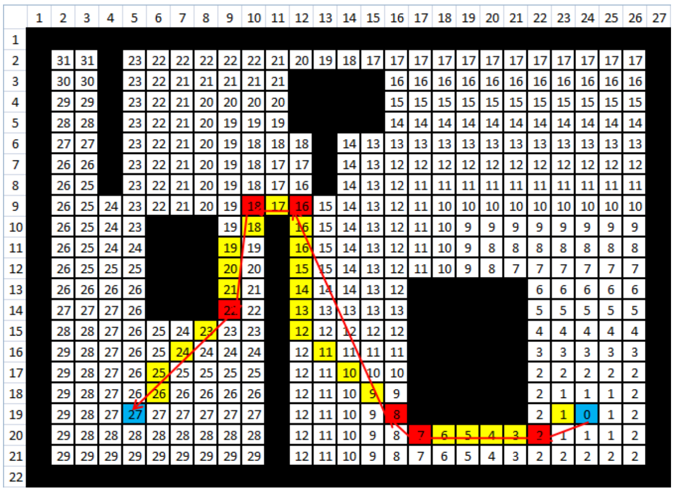
Figure 3.19: *Steps of the wavefront algorithm.*

Afterward, these naive cells undergo two processes:

1. We refine them using queries of line-of-sight in order to reduce the number of treated cells, refined cells are called final cells.

2. We connect the final cells using a curve in order to have a smooth 2D trajectory.

Our line-of-sight queries uses the *rasterization* algorithm proposed by Bresenham in [Bre65], in order to identify cells that can be connected without any intersection with a forbidden cell (obstacle). Thus eliminating extra cells and keeping only the final cells of the plan (waypoints in red in Figure 3.20).



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
1																											
2		31	31		23	22	22	22	22	22	21	20	19	18	17	17	17	17	17	17	17	17	17	17	17	17	
3		30	30		23	22	21	21	21	21	21					16	16	16	16	16	16	16	16	16	16	16	
4		29	29		23	22	21	20	20	20	20					15	15	15	15	15	15	15	15	15	15	15	
5		28	28		23	22	21	20	19	19	19					14	14	14	14	14	14	14	14	14	14	14	
6		27	27		23	22	21	20	19	18	18	18		14	13	13	13	13	13	13	13	13	13	13	13	13	
7		26	26		23	22	21	20	19	18	17	17		14	13	12	12	12	12	12	12	12	12	12	12	12	
8		26	25		23	22	21	20	19	18	17	16		14	13	12	11	11	11	11	11	11	11	11	11	11	
9		26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	10	10	10	10	10	10	10	10	
10		26	25	24	23				19	18		16	15	14	13	12	11	10	9	9	9	9	9	9	9	9	
11		26	25	24	24				19	19		16	15	14	13	12	11	10	9	8	8	8	8	8	8	8	
12		26	25	25	25				20	20		15	15	14	13	12	11	10	9	8	7	7	7	7	7	7	
13		26	26	26	26				21	21		14	14	14	13	12						6	6	6	6	6	
14		27	27	27	26				22	22		13	13	13	13	13						5	5	5	5	5	
15		28	28	27	26	25	24	23	23	23		12	12	12	12	12						4	4	4	4	4	
16		29	28	27	26	25	24	24	24	24		12	11	11	11	11						3	3	3	3	3	
17		29	28	27	26	25	25	25	25	25		12	11	10	10	10						2	2	2	2	2	
18		29	28	27	26	26	26	26	26	26		12	11	10	9	9						2	1	1	1	2	
19		29	28	27	27	27	27	27	27	27		12	11	10	9	8						2	1	0	1	2	
20		29	28	28	28	28	28	28	28	28		12	11	10	9	8	7	6	5	4	3	2	1	1	1	2	
21		29	29	29	29	29	29	29	29	29		12	11	10	9	8	7	6	5	4	3	2	2	2	2	2	
22																											

Figure 3.20: *To compute the path between the source and the destination cells (in blue). We use the wavefront algorithm (result in yellow). Then we refine the result with line-of-sight queries (in red). The black cells represent the obstacles (forbidden cells).*

Bresenham's algorithm is used normally to rasterize any 2D line projected on a grid. By default, it does not calculate all cells Figure 3.21(a), that why in this thesis we modify Bresenham's algorithm in order to calculate/activate **all** cells that any line passes through, as seen in Figure 3.21(b).

After applying the modified Bresenham's algorithm on the naive cells (in yellow on 3.22), we obtain the final cells of the plan (in red on 3.22). These red cells are used as control points for a parametric Hermite curve (Appendix F) in order to obtain a smooth 2D path, as shown on Figure 3.22. Hermite curve is a known

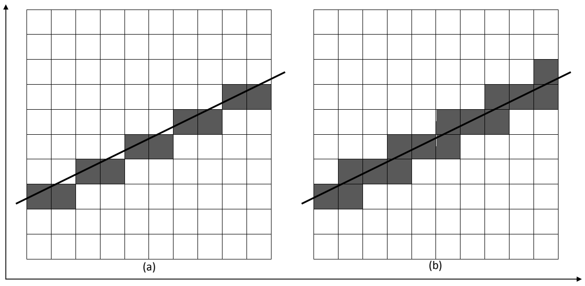


Figure 3.21: *a) Original Bresenham's algorithm for line rasterization, b) Modification on Bresenham's algorithm to activate all cells.*

b-spline curve, the tangents of this curve are calculated automatically using the matrix presented in Appendix F. In the following steps, the 3D path constructor verifies that this 2D curve does not collide with any obstacle.

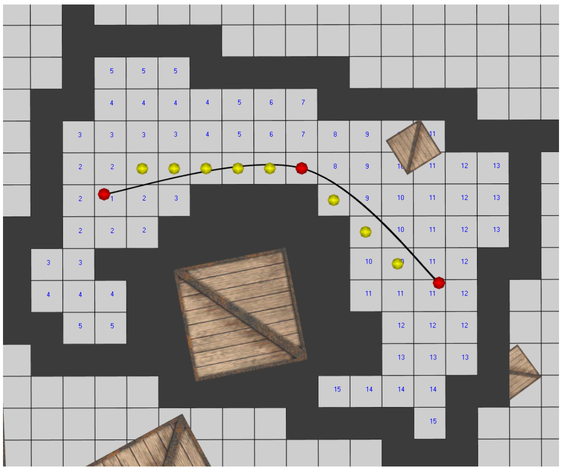


Figure 3.22: *First, we construct the path in a 2D voxel-based representation of the environment (yellow dots) and then generate the final curve (in black) with a selection of waypoints (red dots).*

In order to optimize this 2D path construction, we limit the search space using a virtual rectangle that contains the start position and the preferred target, with a width that varies based on the multi-legged character morphology (leg reach). By doing so, the previously explained wavefront algorithm processes less number of cells, accelerating the ZX-path construction. This virtual rectangle is shown in Figure 3.16 and in Figure 3.22.

So, in Figure 3.22 we obtained the 2D curve that navigates the complex environment on the ZX-plane. We sample this 2D Hermite curve using a fixed step and elevate it in 3D using the *elevations map*. During this sampling phase we make sure that this curve is collision-free. By constructing a convex hull out of these elevated sample points, we obtain the final 3D waypoints that navigates the environment collision-free (see Figure 3.23). Resulting waypoints are used as control points to define a 3D Hermite curve: this curve represents the final 3D trajectory that navigates through the environment in 3D, as shown in Figure 3.26. We optimize the previous waypoints by eliminating the ones that are too close to each others, in order to generate a smoother (less curvature change) b-spline curve.

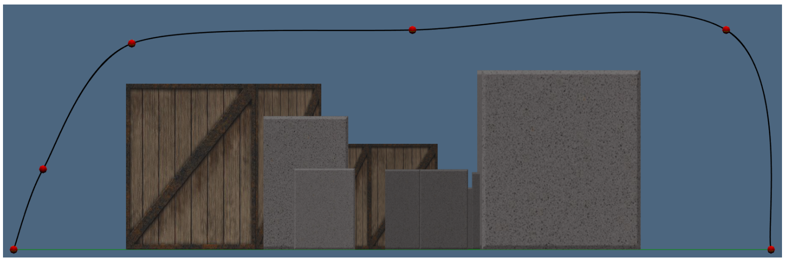


Figure 3.23: *Using the elevations map and the previous 2D trajectory we construct the 3D trajectory.*

3.5 Footprints and Feet Path Planning

In Section 3.2, we explained that starting from the initial foot position the character controller calculates a preferred footprint target according to the locomotion parameters (Appendix D) and the surrounding environment (Figure 3.14). But these calculations do not take into consideration the actual state of the foot and its preferences (*e.g.* the preferred target calculated by the character controller can be too close to an obstacle, on the edge of a crate, *etc.*). That is why in this step we calculate and assign to each foot the best trajectory toward the best target in the current environment. We call this trajectory-footprint combination a **couple**. Meaning that our main algorithm (Section 3.5.2), evaluates potential footprints and trajectories (Section 3.5.1) in order to find and assign the best **couple** (footprint-trajectory) to the current foot based on the current state of the environment.

3.5.1 Potential Footprints and Trajectories scoring

Our algorithm evaluates several footprint targets by exploring the potential cells around the preferred footprint (the one computed by the character controller in Figure 3.17). Several operations are done to do that using both discretization maps (the *obstacles' map* and the *elevations map*):

1. Increase the size of the obstacles using the foot shape (see Figure 3.18 and Figure 3.17 in red).
2. Eliminate possible intersection with other feet by filling the *obstacles' map* with the position of these other feet.
3. Identify all the potential cells that may accept a footprint: all the non-forbidden cells that are not a size increase ones (Figure 3.24).

For each potential target, we calculate a score based on several criteria's normalized between 0 and 1. Criteria's like:

- The distance to the preferred footprint: $value = 1 - Dist(CurrentTarget, PreferredTarget)/FixedDistance$, where the *FixedDistance* is based on the character morphology.
- Number of obstacle-free cells: $value = Num(FreeCells)/8$.
- Is it an obstacle edge? $value = 0.8$, in order to penalize cells on obstacle edge as they are less comfortable than other cells.
- Leading or not to a feet crossing: $value = 0$ *or* 1.

The combination of all these criteria's provides the final cell's score (footprint score) normalized between 0 (worst target) and 1 (best target), shown in Figure 3.24. Most of these criteria's were chosen based on biomechanics studies, like how most characters avoid footprints that are too close to an obstacle as it could lead to an uncomfortable and unrealistic movement on the next takeoff in order to avoid collision [Ale96, Ale03].

We also generate for each created 2D/3D trajectory (Section 3.4) a score based on the application specifications, like the total length of the path, the

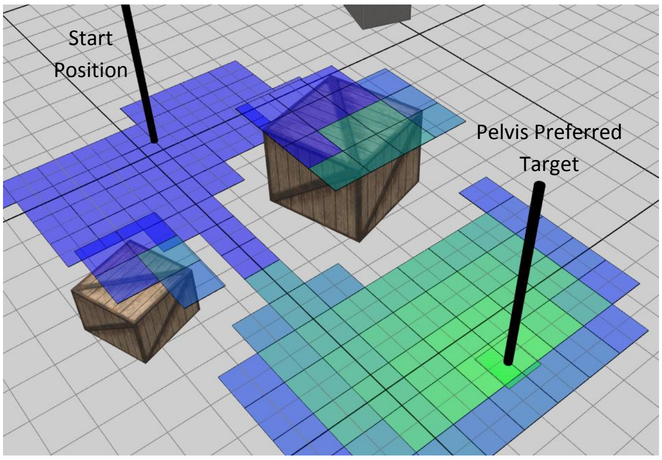


Figure 3.24: *Potential targets: the color varies from green (best target with score* $= 1$*) to blue (worst target with score* $= 0$*).*

acceleration and the curve tangents profile, *etc.* For instance, the current system prefers trajectories that are more straight (direct) with less curvature, a preference observed in biomechanics as it minimizes the energy cost.

3.5.2 Finding Best Couple

During the 3D path construction (see Section 3.4.2), the forbidden cells in the *obstacles' map* designate the parts of the environment that the 2D trajectory is going to avoid and go around, in order to avoid collision with these obstacles. This 2D trajectory passes through cells that have an elevation. The 3D path constructor uses this elevation to construct the final 3D trajectory. The system uses this *obstacles' map* to process an obstacle: either going around it or going over it.

So avoidance-wise, if the system wants to go around an obstacle, it just needs to add it to the *obstacles' map* as forbidden cells. In that way, the resulting 2D trajectory will definitely go around it. While if the system wants to go over an obstacle, it does not need to do anything special as the obstacle is already present in the *elevations map*. And the obstacle will be avoided accordingly based on its elevation.

In order to achieve this avoidance distinction in an optimal way in a complex environment (heightmap plus obstacles), our system discretize the *elevations map* on the Y-axis into slices (for instance slices of $10cm$ in Figure 3.25). This slice size is based on the simulated characters morphologies. For each slice, the system fills the *obstacles' map* with the cells that have higher elevation than this slice. In that way, all calculated trajectories will go over all obstacles, objects and pieces of the environment that have an elevation lower or equal to the slice's elevation. At the same time, all calculated trajectories will go around all obstacles, objects and pieces of the environment that have an elevation higher than the chosen slice's elevation. Everything can be visualized in Figure 3.25 with two obstacles. We must emphasize that the actual environment (the heightmap) is treated in the same way automatically through the *elevations map* . Meaning that during the process of defining slices, the system considers the part of the environment higher than the slice's height as forbidden cells.

So each trajectory can go around or over each obstacle, which generates multiple trajectories toward a target. Each one of these trajectories has a score. The search space is all the possible trajectories that go from the starting point toward all the possible targets. Out of this multitude of possibilities, our main algorithm picks up the best **couple** (target-trajectory) through an intelligent heuristic that uses partial scores to limit the number of explored solutions (not a simple max heuristic).

Our algorithm (results shown in Figure 3.26) loops on all the possible targets based on their score (in a descending way). For each one of these targets, the algorithm first generates the 2D trajectory and scores it, if the score of the **couple** (target-trajectory in 2D) is better than the current best **couple** found till now, it continues. Then it generates the 3D trajectory based on the 2D one and scores it, if the score of the new **couple** (target-trajectory in 3D) is better than the current best **couple**, then it tags this new **couple** as the best one and continues. The algorithm continues evaluating the **couples** until it reaches a **couple** (target-trajectory in 3D) with a score worse than the current best one, in this case it stops and the current best **couple** is the best one. So, at the end of the algorithm we obtain the best footprint target and 3D trajectory that this foot should follow. We calculate the length of this 3D trajectory. Using the needed time given by the character controller, we move the foot on the curve at a constant speed. A pseudo code of the algorithm can be found in Appendix E.

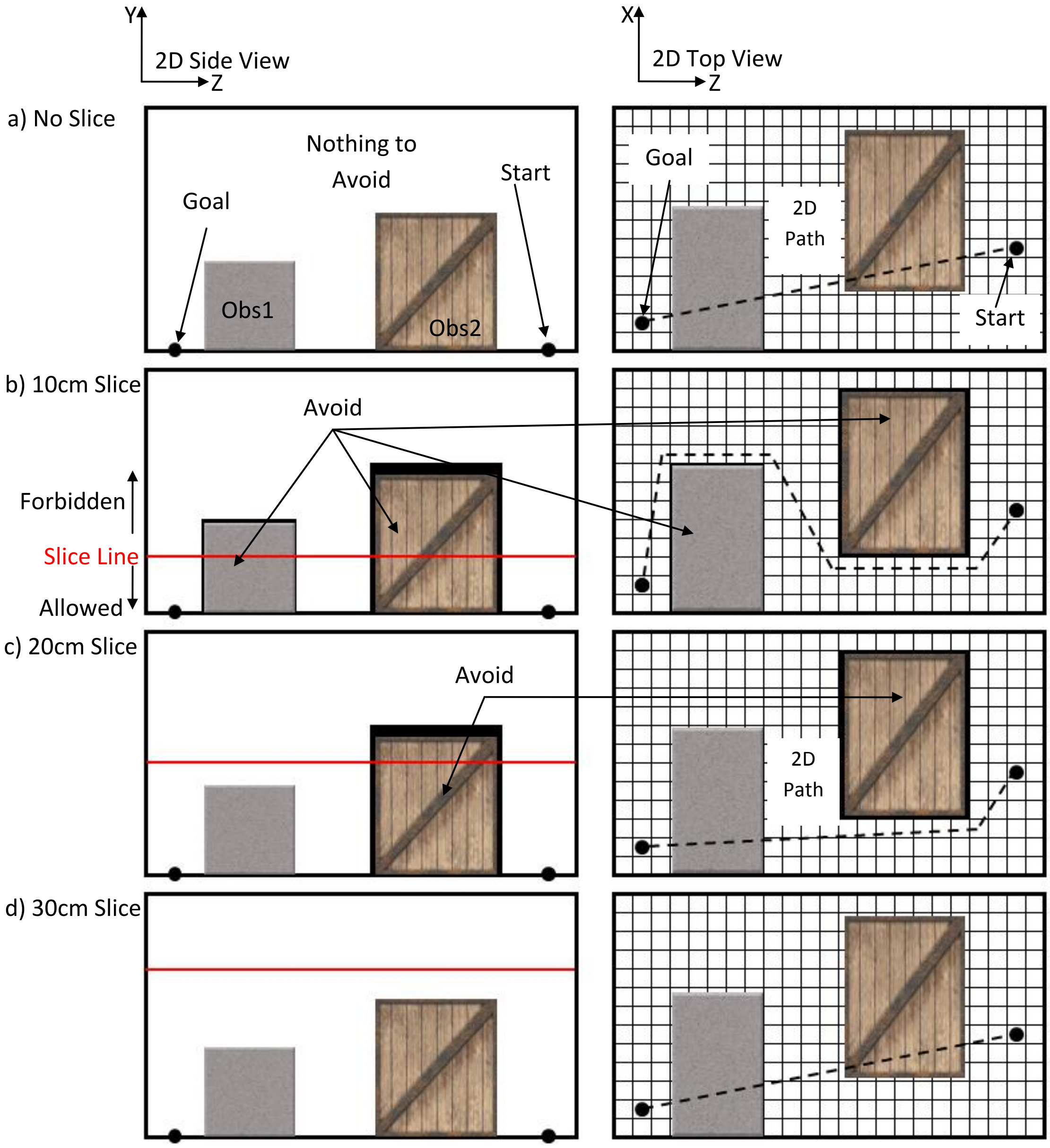

Y
2D Side View
Z
X
2D Top View
Z
a) No Slice
Nothing to Avoid
Goal
Start
Obs1
Obs2
Goal
2D Path
Start
b) 10cm Slice
Avoid
Forbidden
Slice Line
Allowed
c) 20cm Slice
Avoid
2D Path
d) 30cm Slice

Figure 3.25: *Visualized explantation on how to use slices in order to avoid or go around obstacles. a) No slice: the 2D trajectory goes over all obstacles. b) 10cm slice: avoid both obstacles by adding them to the obstacles' map. c) 20cm slice: avoid the second obstacle (Obs2) while going over the first obstacle (Obs1). d) 30cm: go over all obstacles also.*

The search space of this algorithm can be controlled easily (limiting exploration tree), yielding to a better performance. This aspect is discussed in details in Section 3.7.

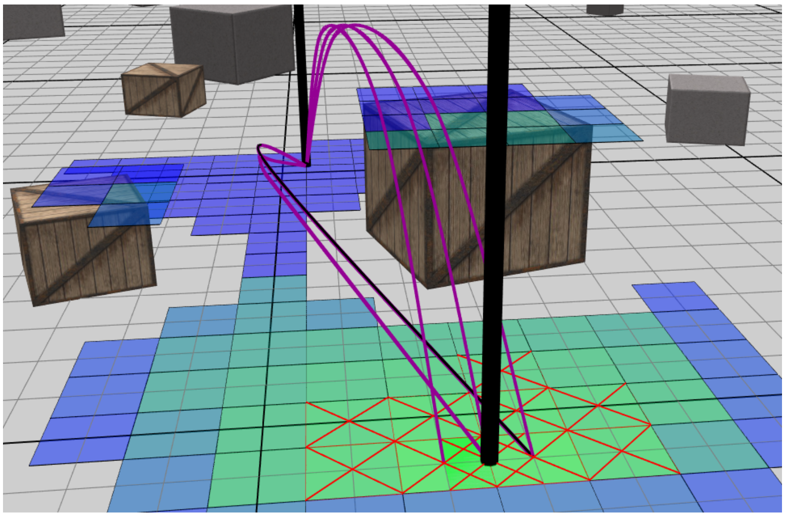


Figure 3.26: *Potential targets: the color varies from green (best target with score* = 1*) to blue (worst target with score* = 0*). Crossed cells are the cells processed by our algorithm, in pink possible trajectories, in black the chosen one.*

3.6 System Loop

This section contains a step-by-step diagrams that shows the previous system in action, the resulting curves and the reaction of the IK systems.

3.6.1 Sequence 1

In sequence 1, Figure 3.27, the highlighted leg (in green) is going to enter its *swing* phase on next simulation step, based on the gait manager.

In Figure 3.28 the preferred target assigned to the foot by the character controller (in yellow) and the chosen target by the feet path planning algorithm (in red) are the same. Meaning that the estimated target by the character controller was right and comfortable to the foot. The foot now follows the chosen 3D trajectory of the best **couple** throughout its *swing* phase. While this foot is following its curve, the CCD IK system is called to calculate the positions of this leg parts based on the foot *end-effector* position.

But the obstacle in Figure 3.28 is a moving one. When it moves the foot needs a new 3D trajectory. So in Figure 3.29 and each time the obstacle moves, the feet path planning is called to find the new best **couple**.

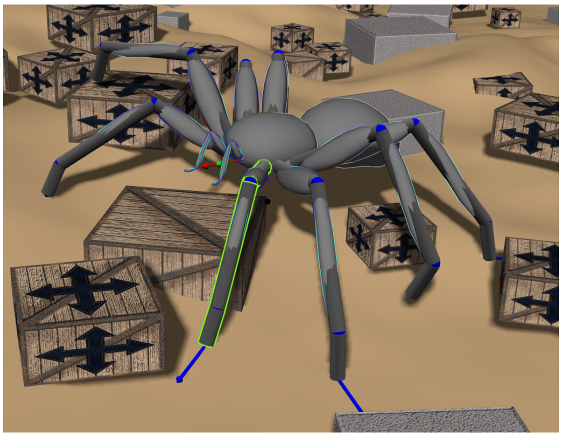


Figure 3.27: *Foot highlighted in green is entering its swing phase.*

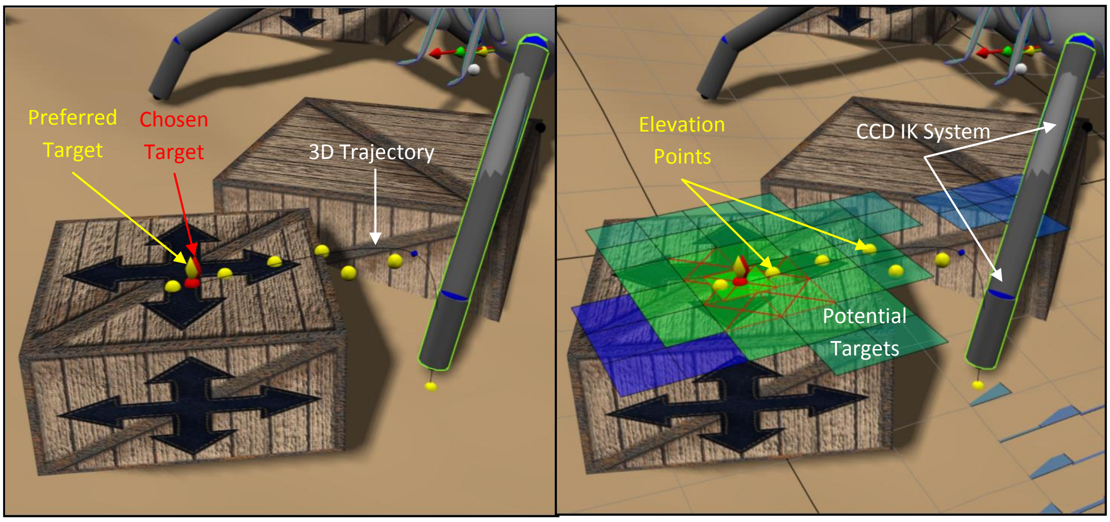

Preferred
Target
Chosen
Target
3D Trajectory
Elevation
Points
CCD IK System
Potential
Targets

Figure 3.28: *First planning choice made in sequence - 1 by the highlighted foot.*

3.6.2 Sequence 2

In sequence 2, Figure 3.30, the highlighted foot disregards completely the preferred target and chooses a new one, because the preferred target is right on the edge of the obstacle, which is not comfortable for the foot. So the best **couple** search yields a new target and a 3D trajectory that this foot will follow. Again, our IK system is called on each calculated position on this curve.

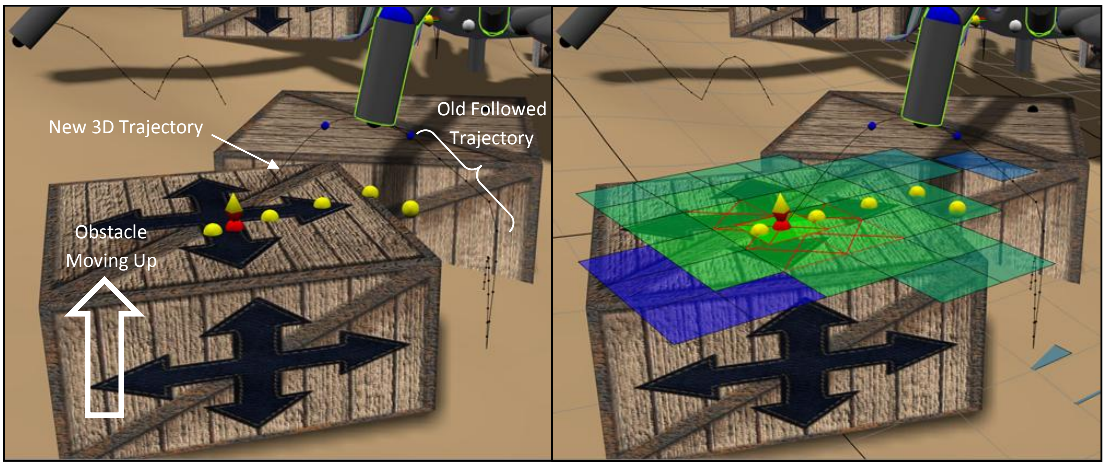

New 3D Trajectory
Old Followed Trajectory
Obstacle Moving Up

Figure 3.29: *Obstacle moves up which means a need for a re-plan.*

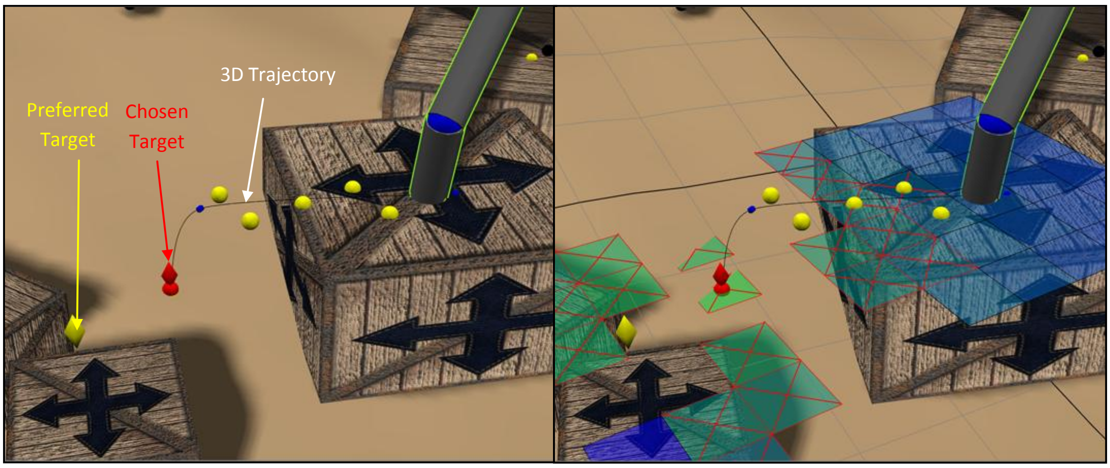

3D Trajectory
Preferred Target
Chosen Target

Figure 3.30: *Highlighted foot disregards the preferred target and chooses a new one.*

3.6.3 Sequence 3

In sequence 3, Figure 3.31, the character controller orders a re-plane for the highlighted foot because of a drastic change in the pelvis speed. In that case the feet path planning algorithm is called and a new best **couple** is found (new target in green). We can notice that the previously chosen target is no longer valid.

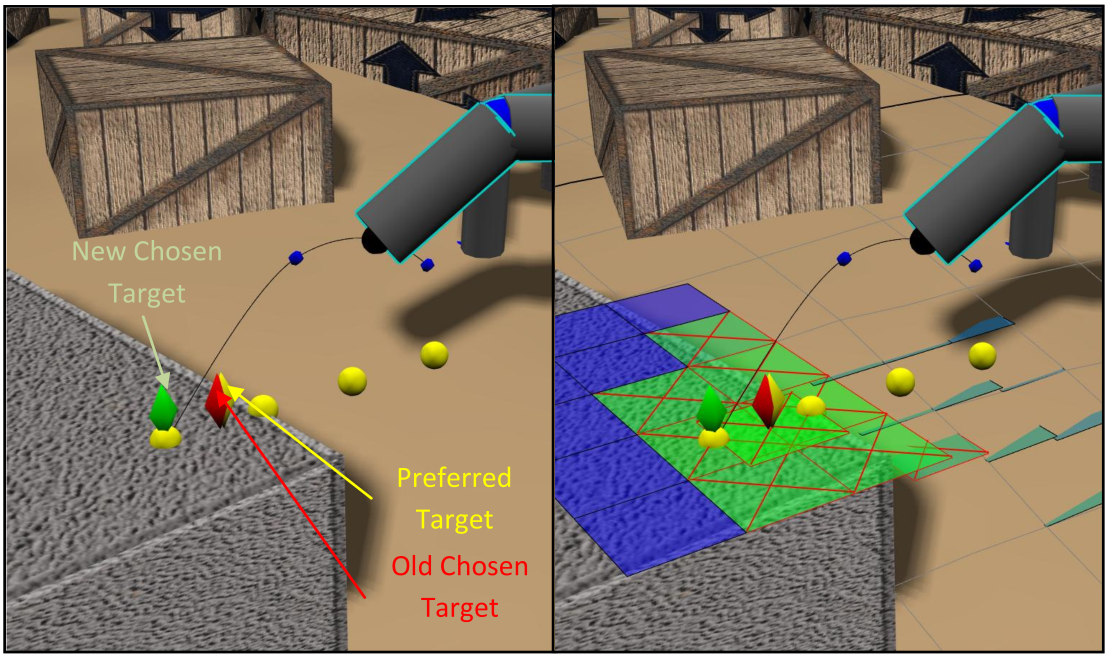

New Chosen
Target
Preferred
Target
Old Chosen
Target

Figure 3.31: *A new re-plan was ordered by the character because of a pelvis speed change.*

3.6.4 Sequence 4

In sequence 4, Figure 3.32, when a foot is in *stance* phase, it is blocked. But with the possibility of an obstacle movement, the character controller always ensures that the blocked foot 3D position is valid.

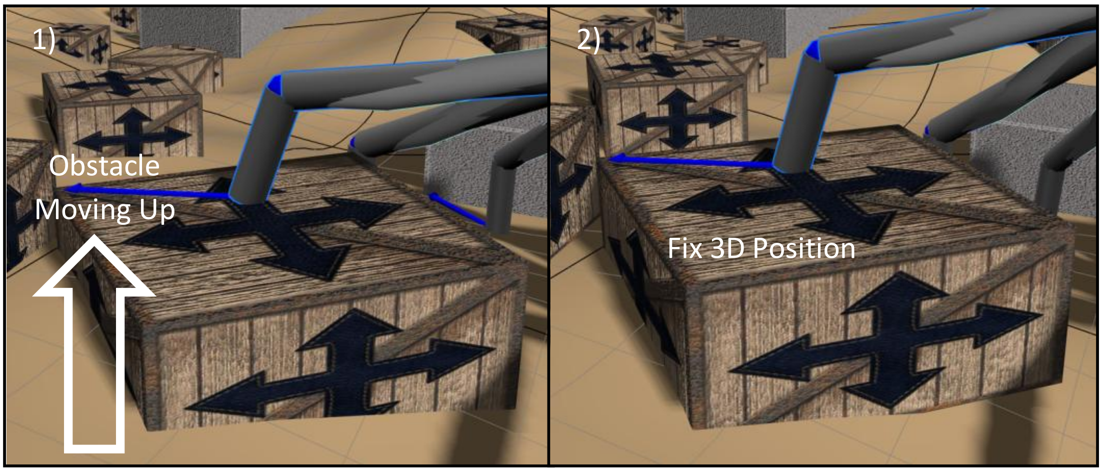

1)
Obstacle
Moving Up
2)
Fix 3D Position

Figure 3.32: *Fix the* stance *foot position based on the obstacle movement.*

3.7 Level of Details

In order to simulate dozens of multi-legged characters in real-time, we have developed a Level of Detail (LOD) techniques that we apply on the previously presented locomotion system. The main purpose of these techniques is to limit the search space for the off-screen and far away creatures in order to accelerate the computations. To ensure a believable simulation, these creatures will always takes coherent choices (Figure 3.33), even with a smaller search space, and the saved up Central Processing Unit (CPU) time allows the simulation off new multi-legged characters.

The execution time of the feet path planning algorithm explained in Section 3.5.2 can be controlled easily. This comes from the ability of controlling two main criteria's of this algorithm: the number of evaluated **couples** (target-trajectory) and the size of the slices used when avoiding obstacles (see Figure 3.25). But in order to always have this plausible and coherent locomotion animation, we start applying these LOD calculations after finding the first valid **couple** (target-trajectory), in this way we ensure that the algorithm assigns to each foot a valid trajectory toward a valid target before starting optimizations. The LOD techniques are applied as follow:

1. Controlling number of evaluated **couples**: for off-screen characters, the algorithm stops directly when finding this first **couple**. While for the on-screen characters, we limit the number of evaluated **couples** after finding this first **couple** in a linear way based on the distance of that multi-legged character from the camera. In this way for far on-screen characters the algorithm stops also when it finds the first **couple**.

2. Controlling the size of the slices: for off-screen characters the algorithm tries to go over any obstacle using no slices (not going around any obstacle). While for the on-screen characters, we increase the size of the slices explained in Figure 3.25 (for example $10cm$, $15cm$...no slices) in a linear way based on the distance of that character from the camera. This LOD process can be simply formulated as: the more the character is far away from the camera (or off-screen) the less our algorithm worries if the chosen **couple** is comfortable.

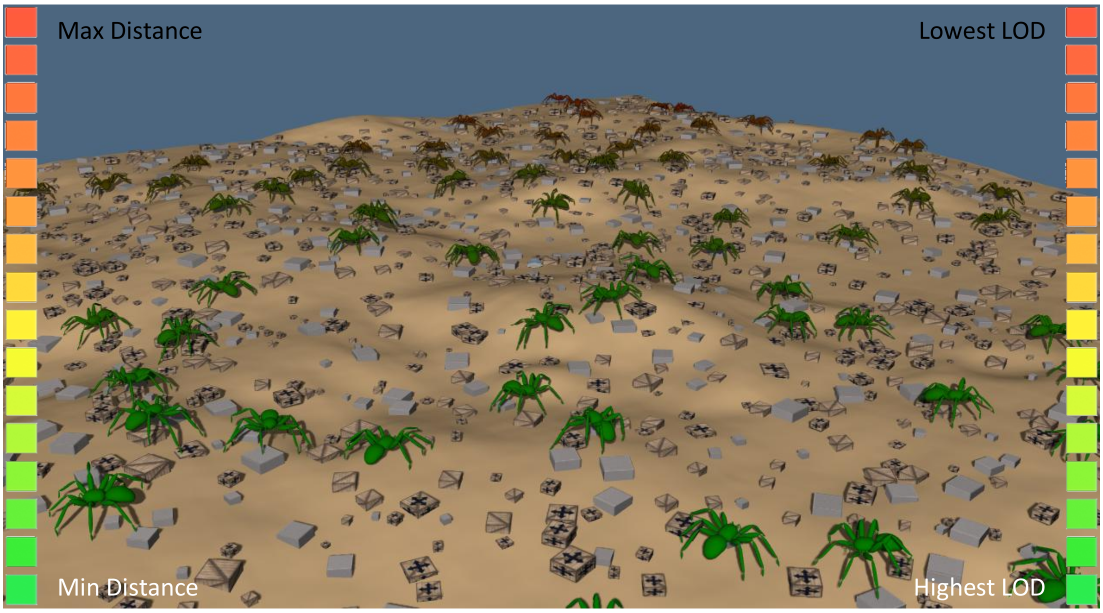


Figure 3.33: *Color coded LOD: far characters from the observer (in red) has lower level of details than near characters (in green).*

After applying this LOD, the final assigned **couple** to each foot is not always the best **couple**, especially for the off-screen and far away creatures. Because by limiting the number of evaluated **couples** and changing the slices size, the search space is not as exhaustive as previously explained.

Finally, we differ from other LOD techniques like in [RGL05] in that we do not change the number of joints in the IK systems, as the used CCD IK system is efficient enough for our real-time simulation. We only stop these IK calculations for off-screen characters. In the next section we discuss in details the performance of the real-time system and the effectiveness of these LOD techniques.

3.8 Performance and Results

Our system is quite generic as it animates a large range of multi-legged character models automatically (Figure 3.34) with a total control over many locomotion parameters (Section 3.1) like for instance the desired overall speed, the step height, the gait (interface illustrated in Figure 3.10), *etc.*

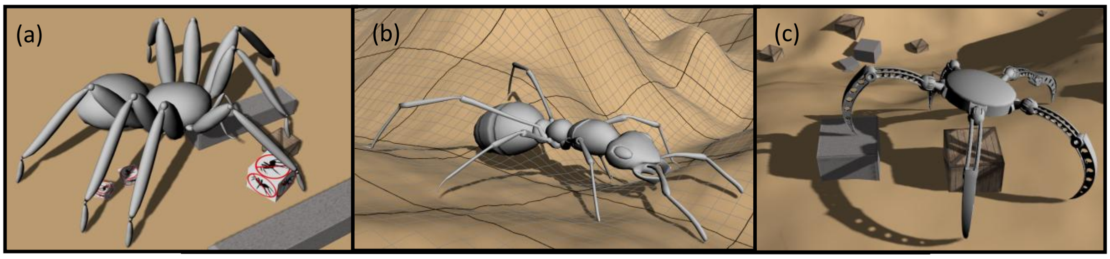


Figure 3.34: *(a) Spider model avoids crates flagged as forbidden and steps on others flagged as safe (b) Ant model on a heightmap (c) Imaginary 5-legged robot.*

The system was tested on different kinds of terrain (flat surface, smooth terrain, regular stairs, *etc.*) with different kinds of obstacles (static crates and moving ones). The size of the discretization maps used in these tests is 70 × 70 with $7cm$ cells, which consumes little memory and is precise enough since the maps describe only the environment close to each animated character. The animated multi-legged characters gaits are inspired from biology studies [Wil67, Bla05].

The following results show that our system is both generic and well adapted for real-time applications. In Figure 3.35 we show many morphologically different characters animated in real-time using an i7 Quad Core 2.7 GHZ, 8 GB RAM, 6870 ATI Radeon HD with 1GB vRam. We use **4 threads** in order to exploit the overall capabilities of the computer.

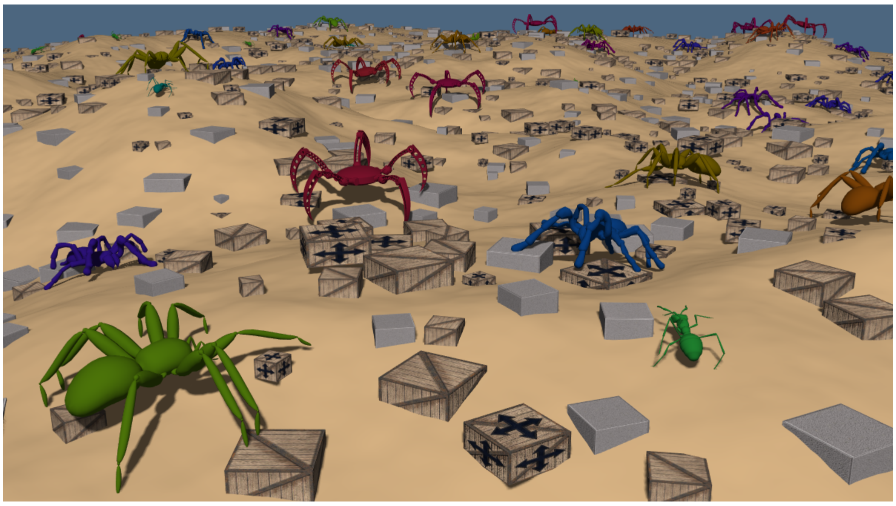


Figure 3.35: *Simulation snapshot showing morphological varieties.*

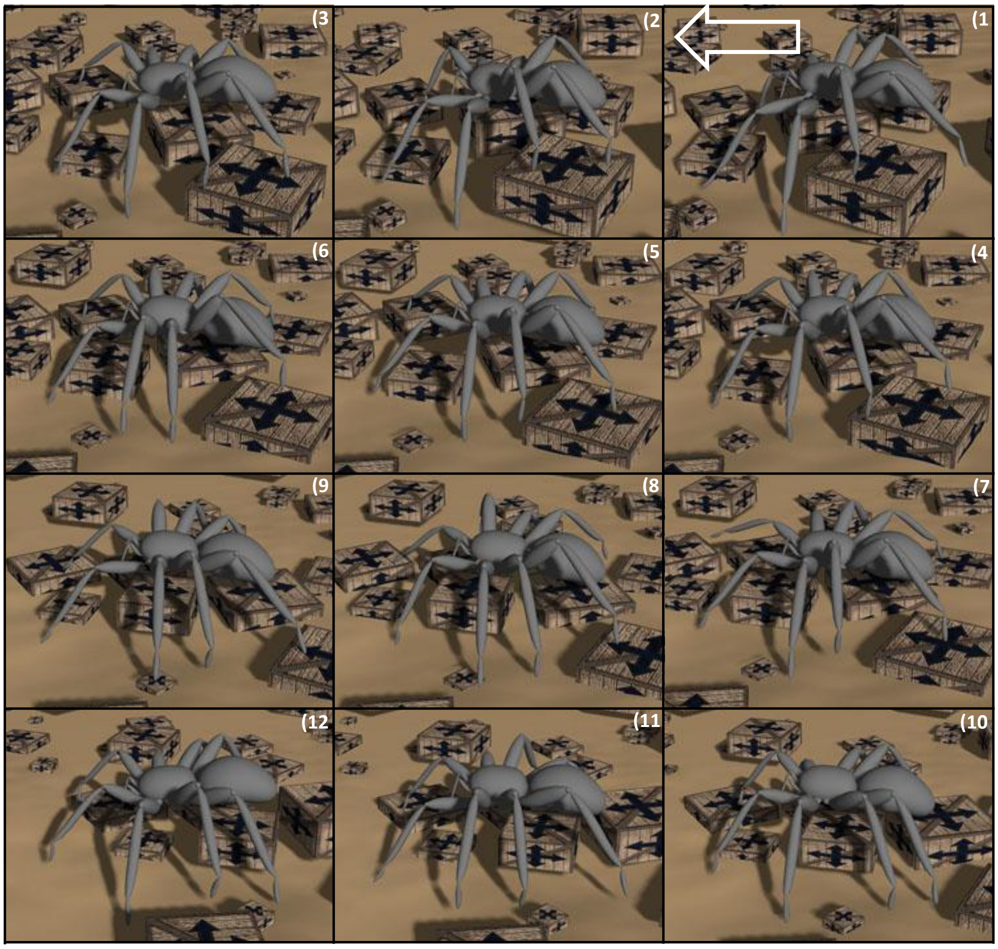


Figure 3.36: *A spider moving forward in an environment filled with moving obstacles.*

In Figures 3.36 and 3.37, we show a frame by frame snapshot of the simulation with a 5-legged robot going uphill with a robotic-inspired gait and a spider moving forward in an environment filled with moving obstacles. Other type of morphologies will be shown in next chapters.

Table 3.1 shows average computation time, on each simulation step, for 100 8-legged characters. The environment consists of a heightmap and obstacles (3000 crates, 40% dynamic). In the line called LOD, we make sure that all of the characters are in the camera field of view, while in during LOD Zoomed (Z-LOD), we zoom on one character and make sure that at least 50% of the characters are shown on the screen. As we can observe, the preparation time of the maps is

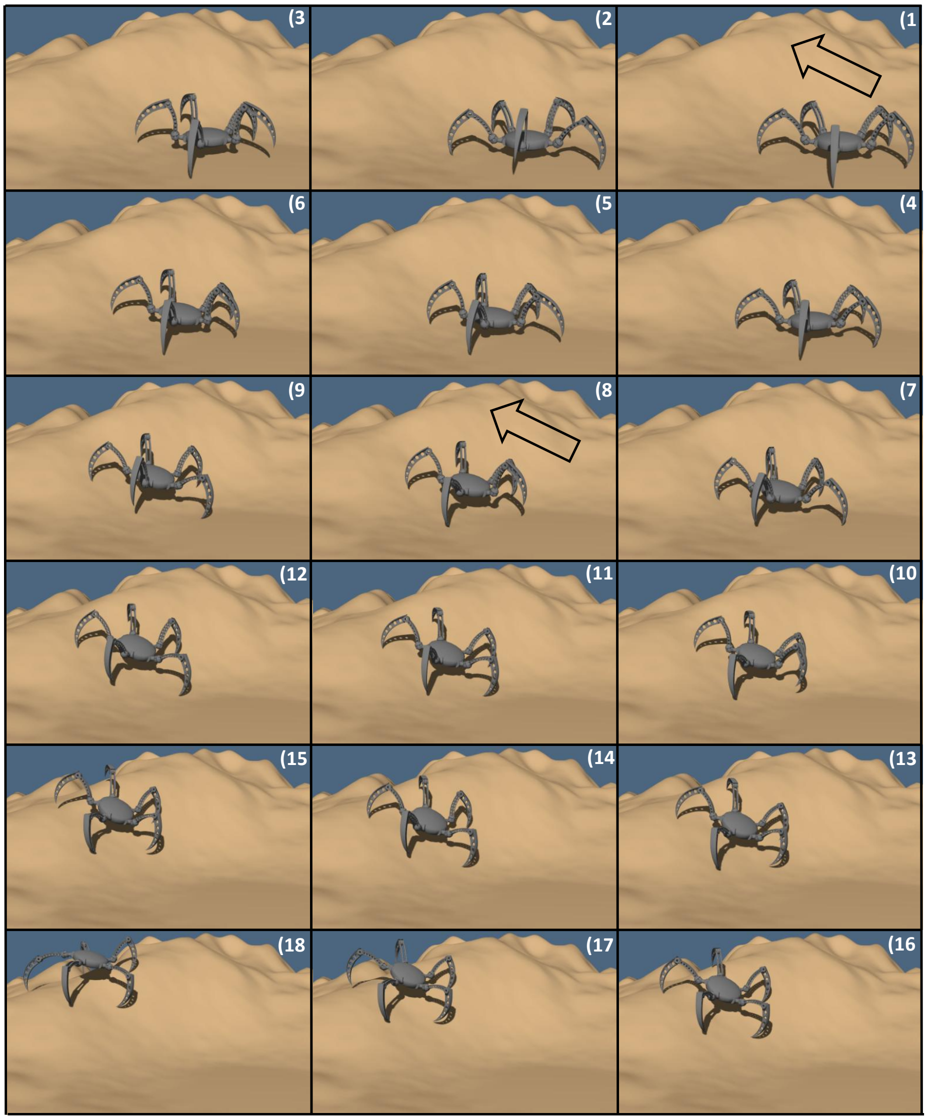


Figure 3.37: *A 5-legged robot going uphill.*

fixed. It is pre-computed once, on each simulation step, for all characters at the same time. In LOD Zoomed (Z-LOD) the system loses some computation time per character as there are more characters doing full search for the best **couple**, while in the same time gains computation time in the CCD IK systems as we do not calculate the IK for the off-screen characters. By using our previous LOD techniques, we maintain a simulation with ∼30 Frames Per Second (fps), which is qualified as real-time.

	Total for Characters	Maps Preparation	CCD IK Systems	Average FPS
LOD	0.029s	0.004s	0.013s	∼30 fps
LOD Zoomed	0.032s	0.004s	0.006s	∼30 fps
No LOD	0.051s	0.004s	0.013s	∼18 fps

Table 3.1: Average computation time for 100 8-legged characters

The following charts show the effect of the number of simulated characters and the environment complexity on our real-time system performance. Based on Table 3.1, we are only interested in the performance using the LOD techniques. All tests use 8-legged creatures (spider model). In Figure 3.38, we show the average calculation time for all characters on each simulation step, when a heightmap (H) exists versus a plain terrain (!H). There is no obstacles in the environment and values are in **seconds**. Our system calculation time increases in a linear way in relation to the number of simulated characters. And on a plain terrain it is more efficient, which is logical as the 3D trajectory construction is simpler on a flat surface (Section 3.4).

In Figure 3.39, we observe that our system is more efficient when populating the environment with obstacles (3000 crates, 40% dynamic), this is thanks to the obstacle size increase process. Again our main algorithm searches for the best **couple** using the process of defining slices explained in Figure 3.25. The obstacles and the actual heightmap are always processed equally during that process as everything is presented by the *elevations map* . As our system increases the size of each obstacle (Figure 3.18) for the previously explained comfort raisons, the cells tagged as an obstacle size increase are forbidden targets. Which decreases the evaluated targets in our feet path planning algorithm (Section 3.5). Decreasing the search space during the best **couple** exploration.

Based on the previous results, our locomotion system is quite efficient compared to recent ones. Systems like the one proposed in [CKJ+11] can only animate 2-3 quadrupeds, in real-time, in a fairly simple environment. Actually, by simulating complex real world dynamics like friction, body parts collisions, gravity (balance controller), *etc.* the number of characters that can be animated at the same time decreases drastically. Only one biped in [dLMH10] using QP solvers with 50%-

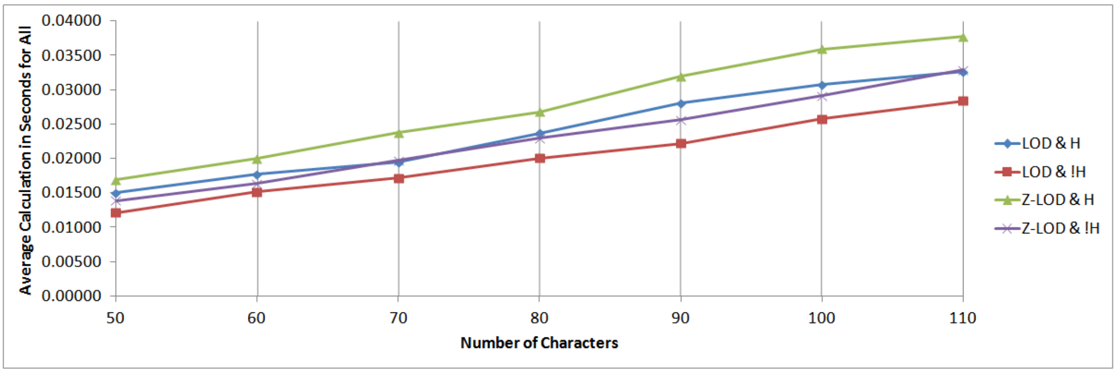


Figure 3.38: *Our system performance when a heightmap (H) exists versus a plain terrain (!H). There is no obstacles.*

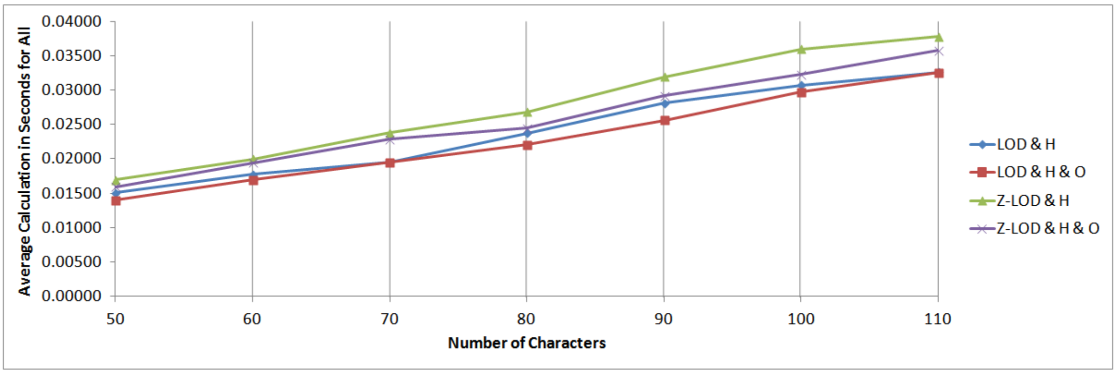


Figure 3.39: *Our system performance when a heightmap (H) exists versus a heightmap and obstacles (H & O, 3000 obstacles with 40% dynamic).*

100% real-time performance[1] (depending on the complexity of the morphology) on a plain terrain. 2-10 fps[2] for one biped in [JYL09], with a complex balance controller that can use the surrounding environment for its advantage. In complex environments, systems like in [WZ10] can simulate up to 4 characters in real-time with an 2.5 hours offline optimization process[3]. One biped in [MdLH10] with 15%-55% real-time results [4].

---

[1] DualCore 3.0 GHz Intel Xeon CPU, 4GB RAM.

[2] Single core of 2.93 GHz Intel Core 2 Duo Processor.

[3] Offline: 4 machines with dual quad-core 2.66 GHz Processor. Real-Time: 1 machine 2.10 GHz Intel Core 2 Duo Processor.

[4] Intel i7 Quad Core 2.7 GHz processor

By using simplified dynamics, systems like [TLC+10] (using an IPM) can simulate one biped[1] in real-time that adapts to a simple environment. 1-2 bipeds[2] in [CBvdP10] (using an IPM too). **Feedback (feed forward)** systems are more efficient. In SIMBICON [YLvdP07] they simulate up to 5 bipeds in 2D[3] and 1-2 bipeds in 3D[4]. The animation system in [CBvdP09] (based on SIMBICON) can animate from 1 to 3 bipeds[5] based on their morphology with an offline process that can take up to 9 hours based on the needed objectives.

The previously mentioned systems can look more realistic as they simulate real world physics, but in this thesis our main goal is real-time systems and real-time performance. On top of that, our locomotion animations stay quite plausible and believable using the *kinematics-based* techniques while being totally controllable.

By using motion data (like MoCap and keyframe), systems like the one proposed by Johansen in [Joh09] benefits from the **Data-Driven** techniques performance. His system can adapt, in real-time, the motion data of 10 bipeds/10 quadrupeds to a complex environment[6]. In [TLP07], Treuille *et al.* can animate 7-8 bipeds[7] in real-time in a near optimal way using an offline trained controller.

In industrial real-time applications (most notably video games) the motion data driven systems are the one used most of the times coupled with IK systems. Systems like [HRE+08] (Spore™) can animate 7-8 creatures with totally different morphologies on a not-so-much powerful machine (Single Core Pentium M, 1.7 GHz). In video games (like Crysis 2, Assassin's Creed Series, *etc.*) they can animate more than 20 character in real-time, with complex interactions between characters and a life-like results.

---

[1] Intel Core 2 Duo 3 GHz Processor
[2] 2.4GHz Dual Core processor, 2Gb RAM
[3] 1.8 Ghz CoreDuo Processor.
[4] Pentium 4 3.2 Ghz Processor.
[5] 2.4 GHz Intel Core 2 Duo Processor.
[6] 2.4 GHz Intel Core 2 Duo processor.
[7] Dual Core 3.5 GHz Intel Xeon processor

3.9 Conclusion

So far, we presented a system capable of procedurally generating believable locomotion animation of several multi-legged characters (like arachnids, insects, or any imaginary n-legged robots or creatures) in real-time with no *a priori* motion data nor any information about the environment. Our system is quite generic as we do not use any morphology specific calculations when generating the locomotion itself, at the same time the user can control many parameters (Section 3.1 and Appendix D) like the gait, the speed, the direction, *etc.* that can be changed in real-time in order to generate the desired locomotion. This system can serve as input to higher level characters' controllers that would like to provide more animations than only the locomotion, like touching objects for discovering the environment, studying insect's behavior, *etc.*

This procedural system [AKGM+12] is published in the Computer Animation and Virtual Worlds (CAVW) journal and was presented in the Computer Animation and Social Agents (CASA) Conference 2012 in Singapore.

But in its current state, the produced motion sometimes lacks realism and naturality because of things like the lack of natural pelvis movement, the non-existence of flexible spine model, *etc.* All of those missing components make the produced motion more robotic than life-like. Thus, in Chapter 4 we add several blocks to the character controller that help in generating more believable locomotion animations: more naturally-inspired pelvis movement, flexible spine-like structure, which is essential when simulating quadrupeds [CKJ+11]. In Chapter 5 we address secondary motions in final simulation: ants antennas, wolf/lizard tail, *etc.* To do so, we propose a **Pendulums** system.

Chapter 4

Adding Naturality and Realism

Contents

4.1 Pelvis Movement Using Pseudo Physics 80

4.2 Spine Model 86

4.2.1 Step 1: Spine Orientation 89

4.2.2 Step 2: Spine Advancing 90

4.2.3 Step 3: Spine Elevation and Pitch 91

4.2.4 Step 4: Final 3D Spine 92

4.3 Other Visual Effects 93

4.4 Performance and Results 95

4.5 Conclusion 99

In the previous chapter, we presented our main locomotion system. A generic real-time system capable of animating a wide range of multi-legged characters morphologies, with a total user control over the generated locomotion. But the proposed system, in its current state, tends to produce unrealistic (robotic-like) locomotion animations that are less plausible when compared to real-life examples. The main objective of this chapter is to add naturality and realism to the previously generated locomotion.

We achieve that by resolving two main problems: the fixed pelvis movement (Section 4.1) and the fixed rigid spine (Section 4.2). We add other visual effects to the final simulation to make it more realistic, like non-linear character progression in Section 4.3. We discuss our new system performance in Section 4.4. Like before, the user has a total control over the produced locomotion, as he can activate, deactivate or configure all the effects presented in the following sections.

4.1 Pelvis Movement Using Pseudo Physics

The character controller explained in Section 3.2 moves the pelvis based on the user needs (speed and orientation), feet error feedback (Figure 3.9) and also in a way that reacts to the environment underneath the multi-legged character. But by only doing so, the pelvis floats above the ground in an unnatural way, as shown in Figure 4.1(a).

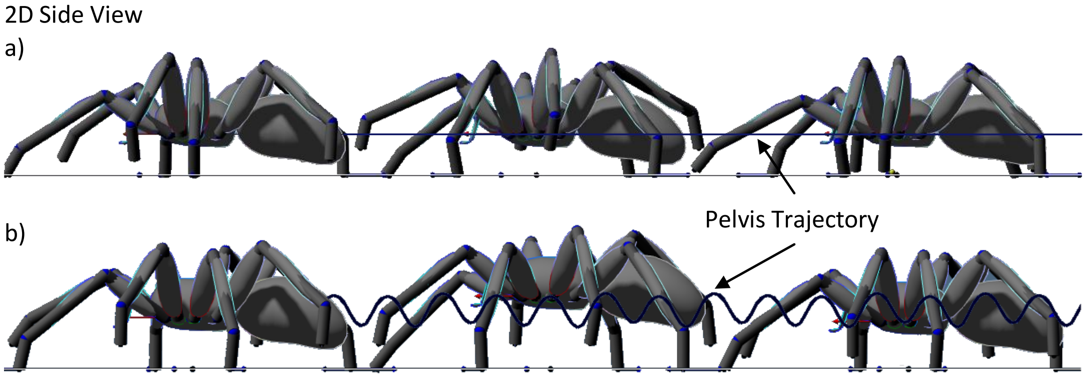


Figure 4.1: *Possibilities of pelvis movement for a spider model. a) A fixed pelvis movement producing a straight line. b) More realistic sinusoidal-like ballistic pelvis movement produced implicitly by the feet gait pattern. Frequency of oscillations is exaggerated on purpose in (b) to illustrate the controllability of our system.*

On the other hand, many biomechanics based studies and observations show that the pelvis trajectory in humans [INM66] (Figure 4.2) and in most animals [Muy57] (Figure 4.3) is sinusoidal-like. The amplitude and frequency of this movement vary based on several criteria's, like the creature morphology, overall speed, gait pattern, style and so on. The goal of this section is to **implicitly** generate this sinusoidal-like ballistic pelvis movement observed in nature, as shown in Figure 4.1(b).

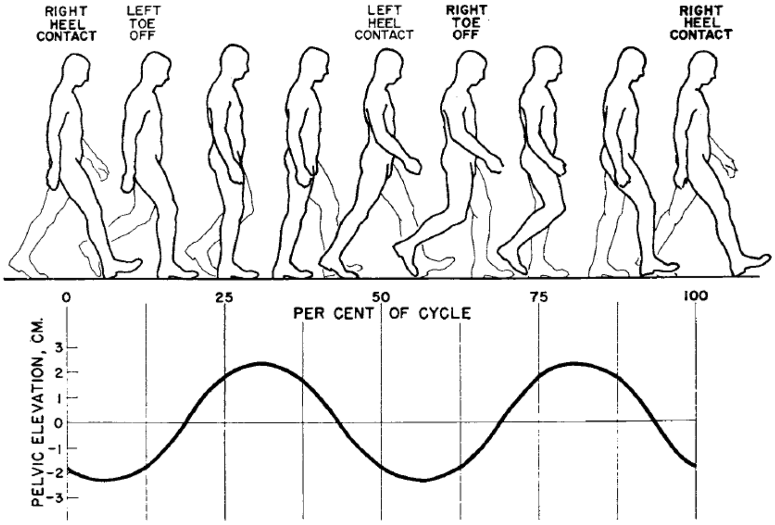


Figure 4.2: *The pelvis trajectory of a walking human, courtesy of [INM66].*

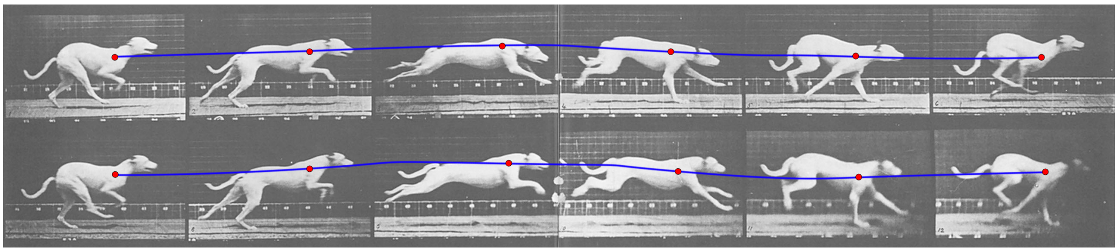


Figure 4.3: *The pelvis trajectory of a galloping dog, original image courtesy of [Muy57].*

We generate this movement by applying on the pelvis a pseudo particle-based physics, a technique inspired by the *PODA* animation system proposed by Girard *et al.* in [GM85, Gir87]. The main fundamental difference between our work and theirs is the feet force. In *PODA* the user needs to fix and tweak by hand the force of each foot in order to achieve the needed effect, while in our system we use the gait pattern, set by the user, to calculate this force. This means a transparent control for the user, with no need for any extra settings. We have deliberately chosen this simplified approach for ease of implementation, performance and controllability.

We use a particle to represent the pelvis in order to simplify the physics equations (neither rotation nor angular momentum equations). This particle motion on the sagittal plane (the up Y-axis in Figure 2.9) is governed by the gravity force (pushing downward) and the feet force (pushing upward), shown in Figure 4.4. While in the horizontal and coronal plane (ZX-plane) the pelvis particle movement is governed by the character controller commands explained in Section 3.2.

For the purpose of clarity, let us study the case of having only one foot ($n = 1$). In this case, the foot and the pelvis particle can be seen as a pogo stick[1], shown in Figure 4.4, with the foot (leg) supporting the whole mass m of the pelvis alone. This foot pushes the pelvis particle upward with certain amount of force when the pogo stick spring is compressed. We will later discuss the case of a multi-legged character.

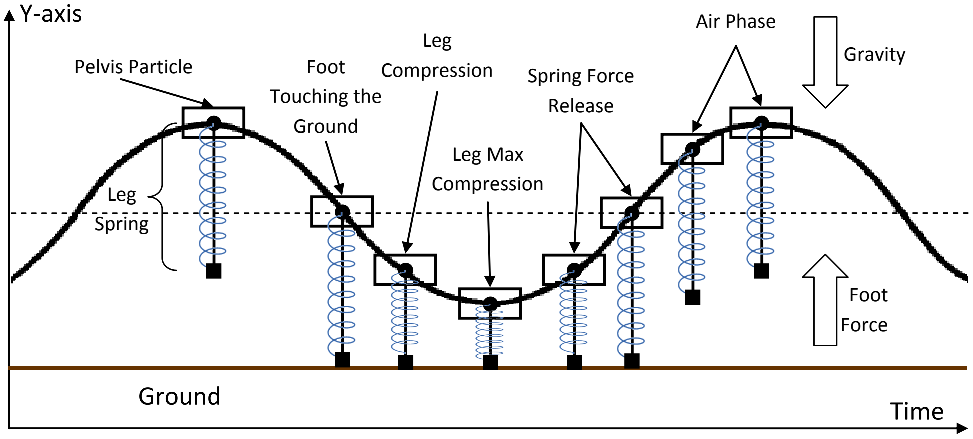

Y-axis
Pelvis Particle
Foot Touching the Ground
Leg Compression
Leg Max Compression
Spring Force Release
Air Phase
Gravity
Leg Spring
Foot Force
Ground
Time

Figure 4.4: *The trajectory of the pelvis particle when a pogo stick is used to represent its relationship with one leg.*

To calculate the force that this foot is exerting on the pelvis particle, we use the gait pattern set by the user as follows. Each foot has two phases: *stance* and *swing* phase. In *swing* phase the foot cannot participate in pushing the pelvis, as it does not have any contact with the ground. While in *stance* phase it can push the pelvis upward. We decompose this *stance* phase into two other distinctive phases, as shown in Figure 4.5: *reception* phase where the foot decelerates the downward movement of the pelvis to a stop, *propulsion* phase where the foot starts pushing the pelvis upward in order to prepare for the *swing* phase. The duration of the *reception* phase is equal to the duration of the *propulsion* phase and it is half of the *stance* phase duration ($T_{reception} = T_{propulsion} = \frac{1}{2}T_{stance}$), a choice we made based on biomechanics observations [Ale96, Ale03, Muy57].

---

[1] A pogo stick is a device for jumping off the ground in a standing position with the aid of a spring, used as a toy or exercise equipment. It consists of a pole with a handle at the top and footrests near the bottom, and a spring located somewhere along the pole.

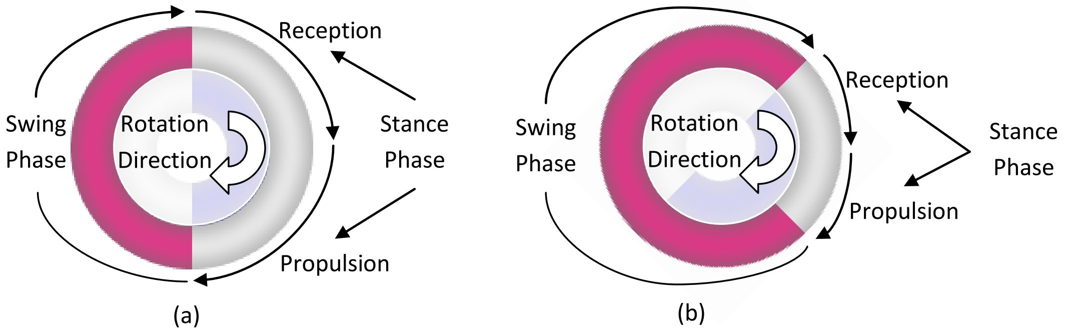


Figure 4.5: *Reception and propulsion phases during the foot stance phase. In (a) swing phase has the same duration of the stance phase while in (b) swing phase is longer.*

For now we are studying the case of having only one foot supporting the whole mass m of the pelvis. We use Newton second law's of motion to calculate the force which this foot is going to apply on the pelvis particle. We consider only the Y axis in our computations.

$$m \cdot a_{pelvis} = W + F_{foot} \tag{4.1}$$

a_{pelvis} is the pelvis acceleration, W is the weight force and F_{foot} is the foot pushing force, all on the Y axis.

$$m \cdot a_{pelvis} = -m \cdot g + m \cdot a_{foot} \tag{4.2a}$$

$$a_{pelvis} = -g + a_{foot} \tag{4.2b}$$

g is the gravity acceleration (which is negative) and a_{foot} is the foot acceleration on the Y axis. We now calculate this foot acceleration (a_{foot}) based on the gait pattern. This acceleration represents its actual upward force. We know that:

$$a_{pelvis} = \frac{v_T - v_C}{T} \tag{4.3}$$

With T a duration, v_T is the needed velocity to achieve at the end of that duration (T) and v_C is the current velocity. By replacing a_{pelvis} by its value from equation 4.2b, equation 4.3 becomes:

$$-g + a_{foot} = \frac{v_T - v_C}{T} \tag{4.4a}$$

$$a_{foot} = g + \frac{v_T - v_C}{T} \tag{4.4b}$$

In *reception* phase, the foot goal is to decelerate the pelvis into a stall ($v_{pelvis} = 0$). The beginning of this phase is the end of a *swing* phase, which means that the pelvis will be falling down under the gravity force. So the goal of this phase is to achieve $v_T = v_{pelvis} = 0$ with T is the *reception* phase duration. By replacing the values in equation 4.4b we can easily calculate the needed acceleration a_{foot} of this foot, which represents this foot pushing upward force (Figure 4.6).

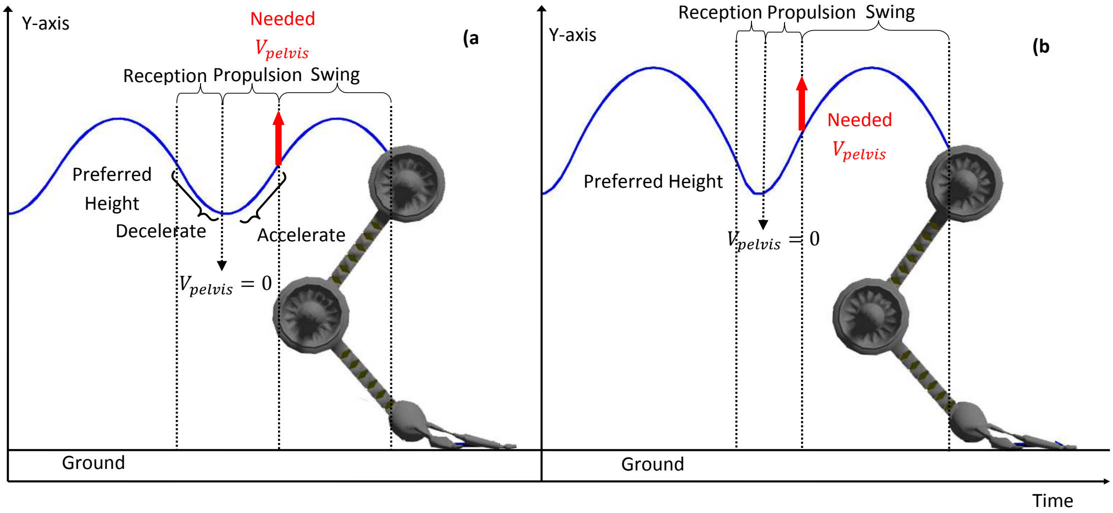

Y-axis
Needed
V_{pelvis}
(a
Reception Propulsion Swing
Preferred
Height
Decelerate
Accelerate
$V_{pelvis} = 0$
Ground
Y-axis
Reception Propulsion Swing
(b
Needed
V_{pelvis}
Preferred Height
$V_{pelvis} = 0$
Ground
Time

Figure 4.6: *Sinusoidal-like ballistic pelvis movement generated implicitly using the foot force calculated based on the reception and propulsion phases. Preferred height is the one fixed by the user. a) Trajectory is generated using the Gait Pattern (a) from Figure 4.5(a). b) Trajectory is generated using the Gait Pattern (b) from Figure 4.5(b).*

In *propulsion* phase, the foot goal is to achieve a v_{pelvis} at the end of the *stance* phase, that ensures the ballistic movement of the pelvis during this foot *swing* phase (Figure 4.6). To do so, we use the following basic motion equation:

$$y_t = \frac{1}{2}at^2 + v_0 t + y_0 \tag{4.5a}$$

$$a = -g \tag{4.5b}$$

$$v_0 = \frac{y_T - y_0}{T} - \frac{1}{2}gT \tag{4.5c}$$

$v_0 = v_{pelvis}$ is the needed velocity (Figure 4.6), T is the (*propulsion* + *swing*) phase duration, y_0 is the current height and y_T is the preferred height fixed by the user. By replacing the values in equation 4.4b we calculate the needed acceleration a_{foot} of this foot. These calculations are done on each simulation step, giving us the punctual force (acceleration) that this foot is going to exert on the pelvis particle on each simulation step. Again, during *swing* phase the foot acceleration is null ($a_{foot} = 0$).

In the case of a multi-legged character $n > 1$, the movement of the pelvis particle is influenced by several feet at the same time. Let a_i be the acceleration of the foot i calculated using the previous equations, which was denoted a_{foot} previously. This acceleration is calculated for each foot using the postulate that it is alone and using its gait pattern. So the pelvis particle needs to integrate all forces (a_i) of each foot to calculate its final a_{pelvis}. In our system we use the postulate that $m = n \times m_i$ where n is the number of feet and m_i is the share of mass that each foot supports. As if the feet share equally the mass of the pelvis particle. Thus, $a_{pelvis} = \sum_{i=1}^{n} a_i/n$ and the resulting sinusoidal-like ballistic movement of the pelvis is shown in Figure 4.7. A weighted average can also be used in order to give more importance to legs with more masses (heavy feet), back legs or any other user needs.

If the gait pattern is symmetrical as in Figure 4.7 (*swing* phase duration is equal to the *stance* phase duration) the resulting pelvis movement is a sinusoid. But with other gait patterns, the pelvis movement can be totally different as phases duration can be different. Like a gait pattern with $\frac{2}{3}$ *swing* phase and $\frac{1}{3}$ *stance* phase, illustrated in Figure 4.5(b) with the resulting trajectory in Figure 4.6(b).

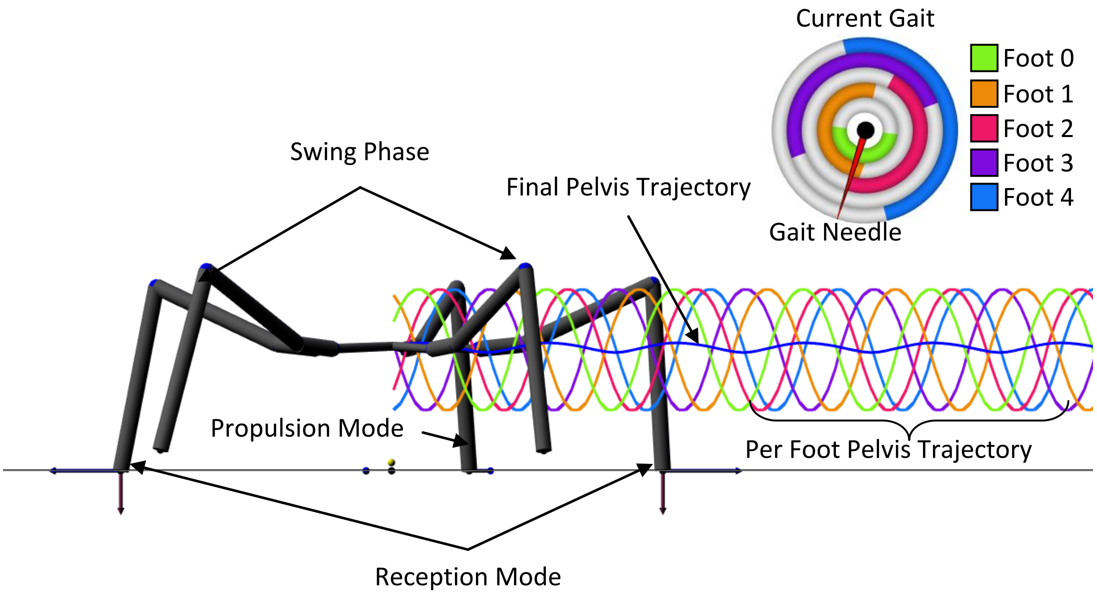

Current Gait
Foot 0
Foot 1
Foot 2
Foot 3
Foot 4
Swing Phase
Final Pelvis Trajectory
Gait Needle
Propulsion Mode
Per Foot Pelvis Trajectory
Reception Mode

Figure 4.7: *The trajectory per foot and the final pelvis sinusoidal-like ballistic movement.*

The user has a total control over this movement through the gait pattern with no need to set any extra values apart from the preferred height (Section 3.1). On top of that, these pseudo physics calculations are used to validate the plausibility of the gait pattern designed by the user. As with a badly designed gait pattern the forces calculated will be too large and not natural. We must note that any change in the character speed can occur only during the *propulsion* phase, as it is the only phase where a foot is virtually propelling the multi-legged character forward.

4.2 Spine Model

Quadruped animals like mammals (dog, horse, wolf, *etc.*) and reptiles (crocodile, lizard, gecko, *etc.*) have a flexible spine, which is an essential component for these type of animals during locomotion and other types of movements [Ale96, Ale03, Muy57]. They use the flexibility of this structure into their advantage (see Figure 4.8) to achieve a high variety of locomotion styles, as it gives them more agility (more DOF).

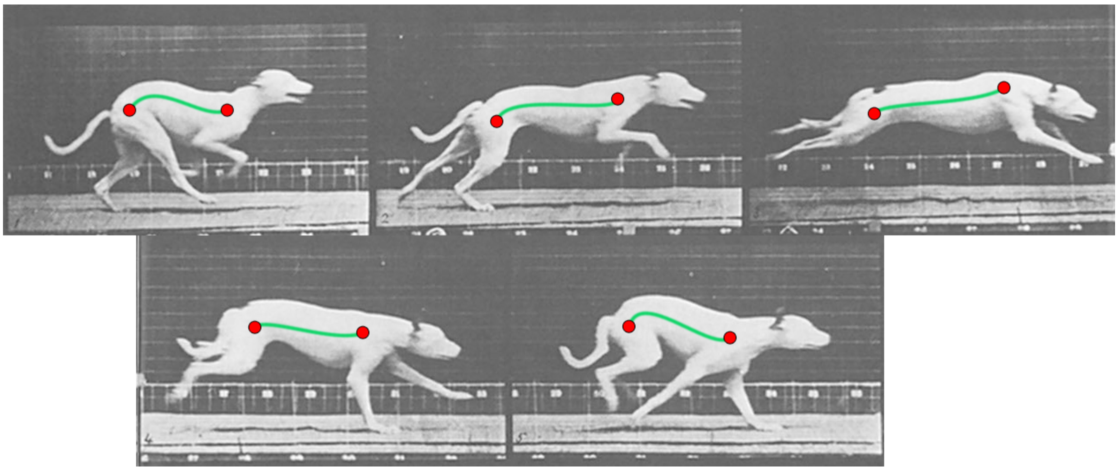


Figure 4.8: *The deformation of the dog flexible spine structure (in green) while galloping, original image courtesy of [Muy57].*

A variety of kinematic quadruped systems [SRH+08, SRH+09] implement a flexible spine in order to achieve more realism in the final generated animation. In [CKJ+11], Coros *et al.* show how the produced motion of their quadruped (a dog) loses its plausibility and naturalness when a rigid spine is used. By adding a flexible spine model, the animated wolf illustrated in Figure 4.9 looks more natural as it turns in a more realistic way.

To add this flexible spine model, we decompose the multi-legged character morphology into several virtual pelvis nodes (shoulders), seen in red in Figure 4.8 and in Figure 4.10. Each foot is connected to one of these nodes, except for the head node which has no foot connected to it. These virtual pelvis nodes (pelvis nodes for shortening) are quite independent in regard to height control, pitch control, footprint placement, *etc.* and are connected by the flexible spine model.

We calculate this spine model using four successive steps, described in Figure 4.11). Firstly, on the horizontal and coronal plane (ZX-plane) then on the sagittal plane (Y-axis) for simplification. On the ZX-plane we concentrate on the 2D orientation (Section 4.2.1) and translation (Section 4.2.2), while on the sagittal plane we concentrate on the elevation and pitch control (Section 4.2.3). In Section 4.2.4, we put everything together to calculate the final 3D spine model.

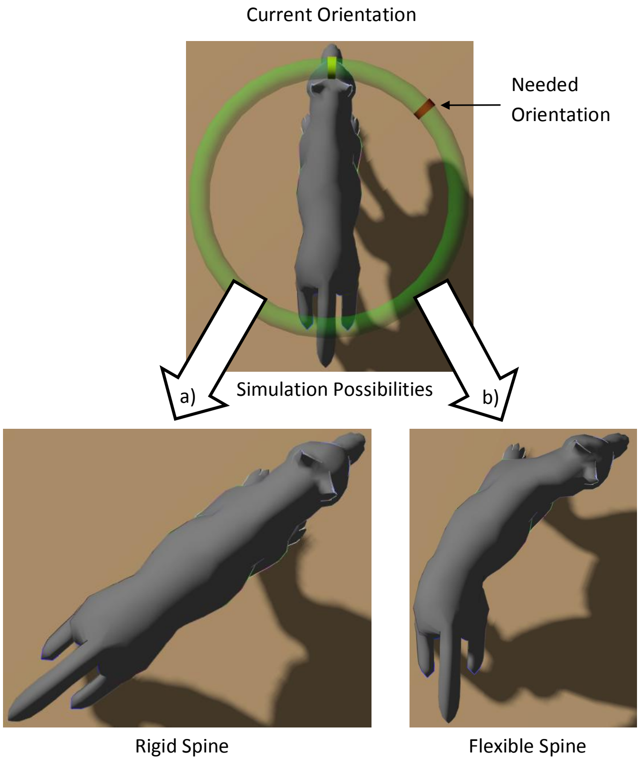

Current Orientation
Needed Orientation
Simulation Possibilities
a)
b)
Rigid Spine
Flexible Spine

Figure 4.9: *The visual difference of adding a spine model when executing a change in orientation command. a) Without a spine, the wolf model turns in a rigid way. b) With a flexible spine, the wolf model turns in a more natural way.*

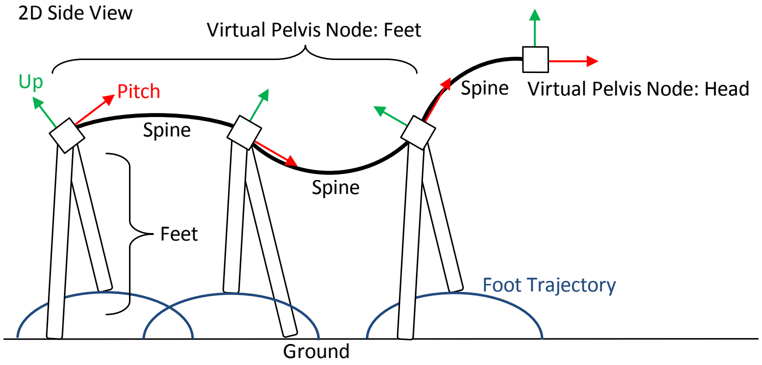

2D Side View
Virtual Pelvis Node: Feet
Up
Pitch
Spine
Spine
Spine
Virtual Pelvis Node: Head
Feet
Foot Trajectory
Ground

Figure 4.10: *The pelvis virtual nodes extracted from the morphology of a fictional 6-legged creature.*

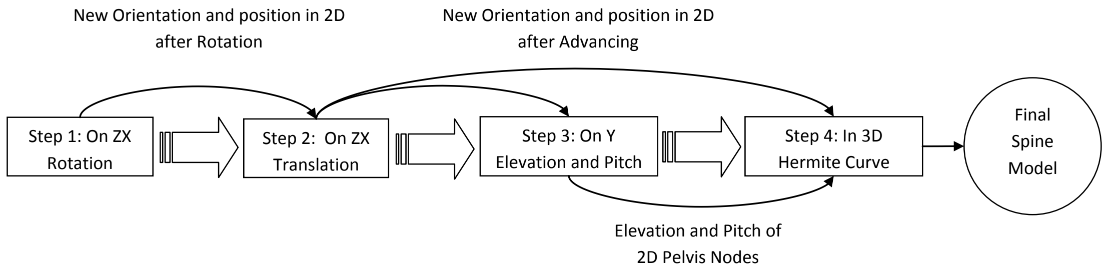

New Orientation and position in 2D after Rotation
New Orientation and position in 2D after Advancing
Step 1: On ZX Rotation
Step 2: On ZX Translation
Step 3: On Y Elevation and Pitch
Step 4: In 3D Hermite Curve
Final Spine Model
Elevation and Pitch of 2D Pelvis Nodes

Figure 4.11: *A workflow explaining the different steps used to calculate the flexible spine model.*

4.2.1 Step 1: Spine Orientation

In Figure 4.12, we explain how the rotation (orientation change) is achieved. The spine nodes are nodes between the virtual pelvis nodes and part of the spine model with no foot attached to them. An exception is the head node which is considered as a virtual pelvis node. We need to calculate the final position and orientation of the spine nodes as they are part of the original multi-legged character spine morphology.

When a new orientation is needed (on the ZX-plane), we propagate this needed orientation on the pelvis nodes from the head node toward the back pelvis node. Each pelvis node will try to satisfy its share based on its relative angular limits, sending the unsatisfied rest in the propagation direction (the joints limits are fixed by the morphology parameters, as described in Section 3.1). When all pelvis nodes are constrained, as in Figure 4.12 - Step 1.3, we rotate the whole spine around one of the pelvis nodes in order to satisfy the needed orientation. In our animation system we always choose the pelvis node just before the head node, a preference that we observed in real-life animal videos. In all previous steps, bones length is always satisfied. So after calculating the position of the pelvis nodes, we place the spine nodes between them based on this bones length constraint.

We must note that a pelvis node can be constrained by joint angular limits and by the feet attached to it. For example, a pelvis node with a fully extended foot in *stance* phase cannot move without breaking the leg bone length limit. These kind of constraints are processed in the final step of our method (Section 4.2.4).

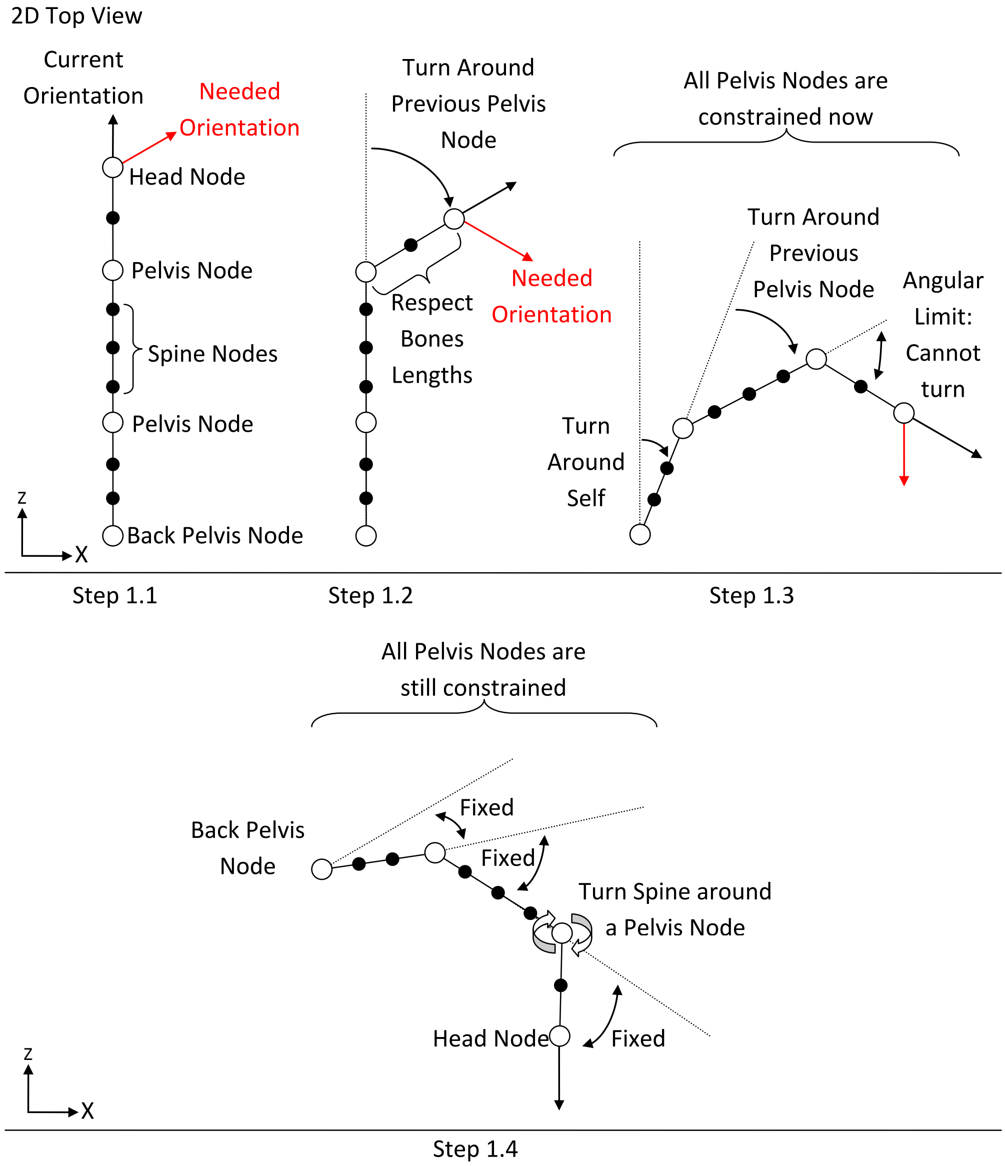

2D Top View
Current Orientation
Needed Orientation
Head Node
Pelvis Node
Spine Nodes
Pelvis Node
Back Pelvis Node
Z
X
Turn Around Previous Pelvis Node
Respect Bones Lengths
Needed Orientation
All Pelvis Nodes are constrained now
Turn Around Previous Pelvis Node
Angular Limit: Cannot turn
Turn Around Self
Step 1.1
Step 1.2
Step 1.3
All Pelvis Nodes are still constrained
Back Pelvis Node
Fixed
Fixed
Turn Spine around a Pelvis Node
Head Node
Fixed
Z
X
Step 1.4

Figure 4.12: *A step by step illustration that shows how the change of spine orientation is computed on the* ZX*-plane.*

4.2.2 Step 2: Spine Advancing

In Figure 4.13, we explain how we translate the spine model. Based on the current orientation and the needed distance, a new head node position is calculated (the target head node in red). The needed distance is calculated using the needed pelvis speed fixed by the user. To maintain a coherent movement of the spine model, we place the virtual pelvis nodes (in blue) on the previous step spine model (in black) in a way that respects the bones length constraint. In this way, the last step spine model is considered as a support model that helps in maintaining the fluidity of the movement. In Figure 4.13(b), we re-calculate the relative orientation between pelvis nodes based on the new positions and the support model. For the back pelvis node, we change the relative orientation in way that relaxes the constraints (in green) in a temporal way. The duration of this relaxation is based on empirical data. At the end of this step, we obtain the position and orientation of the spine model on the ZX-plane and 3D calculations begin.

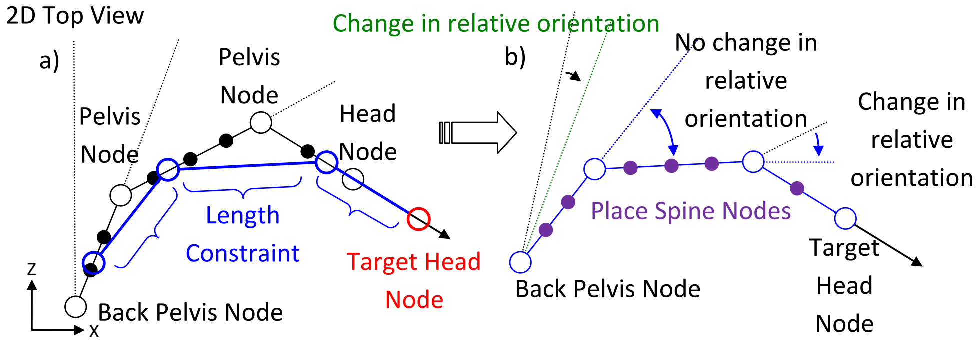

2D Top View
a)
Pelvis Node
Pelvis Node
Head Node
Length Constraint
Target Head Node
Back Pelvis Node
z
x
Change in relative orientation
b)
No change in relative orientation
Change in relative orientation
Place Spine Nodes
Back Pelvis Node
Target Head Node

Figure 4.13: *Illustration on how we translate the pelvis and spine nodes, on the ZX-plane, based on an orientation and needed distance.*

4.2.3 Step 3: Spine Elevation and Pitch

As we decomposed the character pelvis into several virtual nodes, we decompose also the previous height and pitch calculations found in Figure 3.8 into its respective nodes, as shown in Figure 4.14. In this way, we obtain the preferred height for each pelvis node based on its convex hull projection. We integrate the pseudo physics system in each virtual pelvis nodes to add more realism to the spine model. By doing so, the final needed pelvis node elevation is calculated using the pseudo particle-based physics instead of the preferred height only.

For each pelvis node pitch angle, we calculate first the environment-based pitch using the projection of the virtual pelvis nodes on the convex hull, in blue in Figure 4.14. Second, we calculate the pitch of the pelvis node caused by the

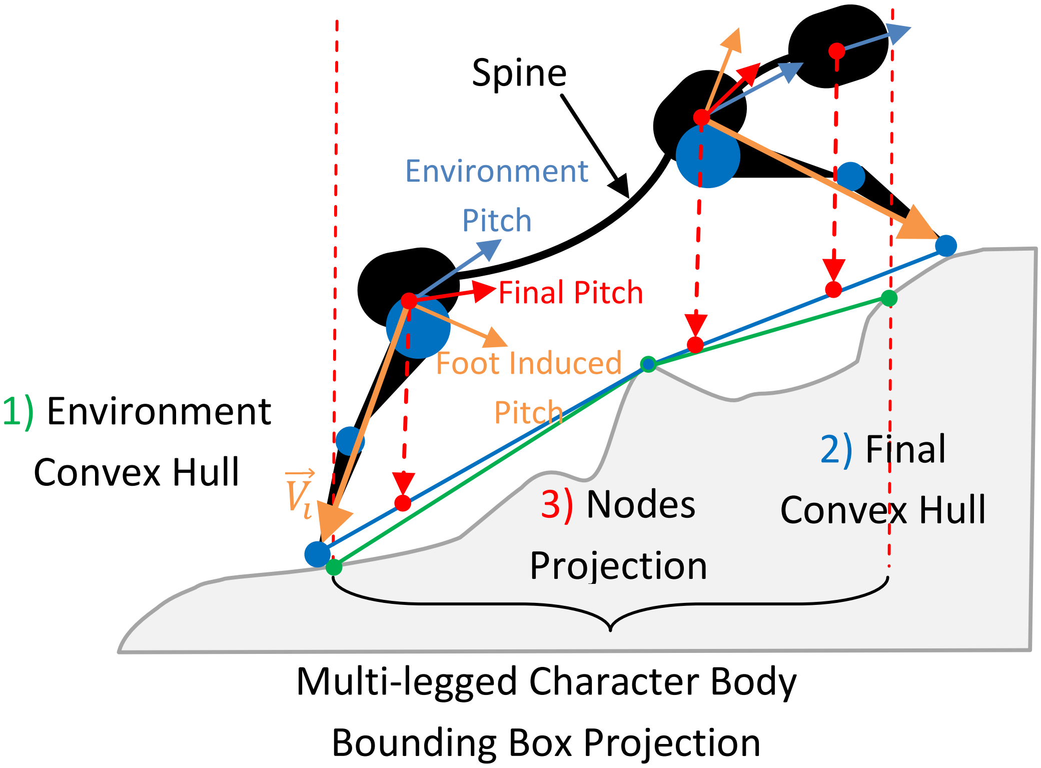

Spine
Environment Pitch
Final Pitch
Foot Induced Pitch
1) Environment Convex Hull
$\vec{V_i}$
3) Nodes Projection
2) Final Convex Hull
Multi-legged Character Body Bounding Box Projection

Figure 4.14: *Computation of each node's height using several convex hulls that covers the environment slice and computation of each virtual pelvis node pitch.*

relative position of each foot. Let $\vec{V_i}$ be the vector that connects the current pelvis node with the foot i (in orange in Figure 4.14). We calculate the perpendicular vector on $\vec{V_i}$ in the direction of the creature progression. This perpendicular vector represents how much this foot affects the pitch angle of its pelvis node. We call this perpendicular vector the foot induced pitch. Final pelvis node pitch is the average of the environment pitch and the feet induced pitches.

4.2.4 Step 4: Final 3D Spine

Using the 2D positions calculated in **Step 1-2** (Section 4.2.1 & 4.2.2) and elevations calculated in **Step 3** (Section 4.2.3), we obtain a preliminarily 3D position for each virtual pelvis node. And using the 2D orientations calculated in **Step 1-2** and pitches calculated in **Step 3**, we obtain a preliminarily tangent direction for each of them. We construct a B-Spline (Hermite) curve between these

3D positions using the previous tangents data (Appendix F). We sample this curve using the bones length constraint in order to calculate the final pelvis and spine nodes position and 3D orientation. By doing so, the visual representation of each pelvis node can have a different 3D position from the one calculated in the previous steps. We consider the 3D positions calculated until **Step 3** as guidelines for the Hermite curve, making the pseudo particle-based physics system independent from the visual system. Sometimes, the final position of the virtual pelvis nodes can not be satisfied by the CCD IK system because of joint constraints. In this case, **Step 4** is repeated based on the closer position that the IK system can ensure from the needed one. In Figure 4.15, we show the final generated flexible spine model on an abstract lizard model. Its spine consists of 3 pelvis nodes and 9 spine nodes.

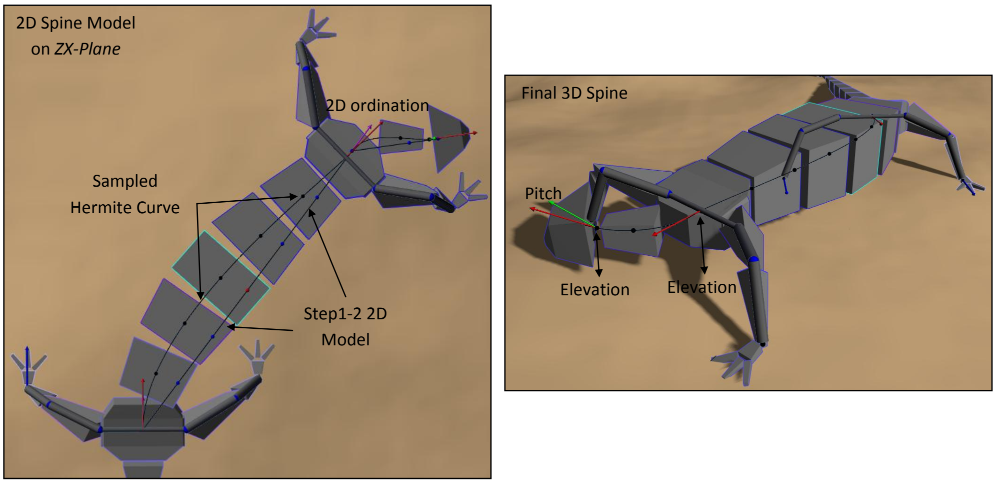


Figure 4.15: *Final generated flexible spine model in an abstract lizard model.*

4.3 Other Visual Effects

To make the final animation more realistic, we add two components: non-linear character progression and gait randomizer. The idea behind adding these components is to make each simulated character locomotion more distinctive and to make the final simulation less repetitive.

Non-Linear Character Progression. Some of the simulated animals (like insects, arachnids, *etc.*) move in a burst-like style. They accelerate (toward a max speed) and decelerated (come to a stall) in a random-like way. They never maintain a constant speed [Ale96, Ale03, Wil67]. We implement this kind of burst-like movement by varying each multi-legged character speed using a sinusoidal-like wave (Figure 4.16). This sinusoidal-like wave amplitude is the max speed imposed by the user. We only vary the wavelength, static period and max speed period randomly throughout the simulation. This effect can be turned off in real-time by the user.

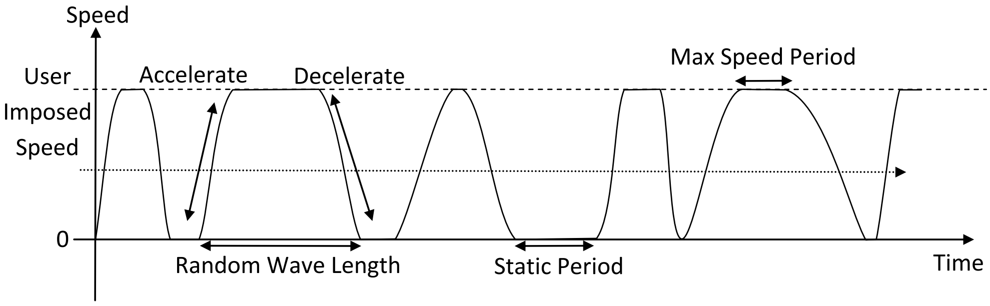


Figure 4.16: *Example of the speed variations during the random non-linear character progression.*

Gait Randomizer. By randomizing the gait of each multi-legged character, the final simulation becomes less repetitive as different characters will not have the same locomotion style at the same time (doing the same feet movement at the same time). When generating these random gaits, shown in Figure 4.17, we make sure that not all feet are in *swing* phase at the same time and that each foot enters its *swing* phase before another random foot enters its *stance* phase. The idea is to cover the whole gait disk, in a logical way. We couple the gait randomizer with the non-linear character progression (if it is active) by affecting a new gait to the multi-legged character each time its speed drops to zero (it comes to a stall). Again these small effects can be turned off in real-time by the user.

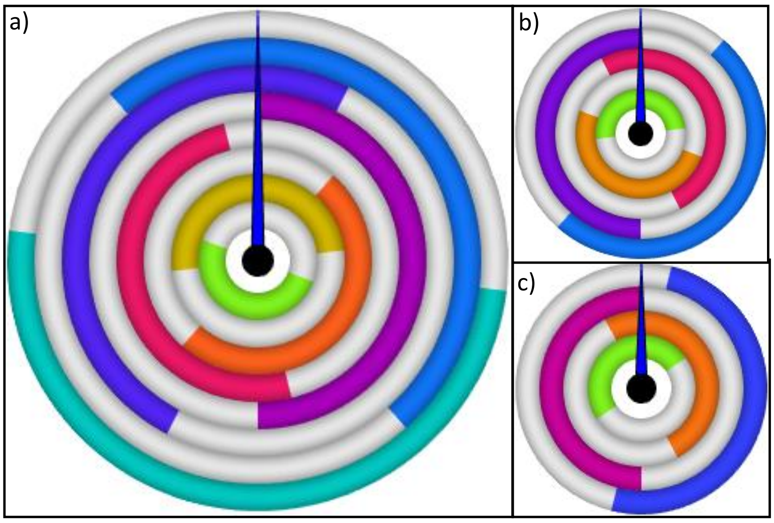


Figure 4.17: *Examples of the resulting random gaits for different morphologies. (a) 8-legged. (b) 5-legged. (c) 4-legged.*

4.4 Performance and Results

In this section we discuss the performance of the locomotion system after adding the pseudo physics and the flexible spine. The non-linear character progression and gait randomizer have little to no-effect on the overall performance of the locomotion system: their contribution is more visual.

In Figures 4.18, we show a frame by frame snapshot of a lizard running upward. By adding a visual yaw effect to the spine model tangents, the animated lizard moves in a believable way compared to a real life lizard. We calculate the yaw value for each foot based on its current position (in white in Figures 4.18) and its rest position (in green). The value of the final visual yaw effect is the average yaw for all feet.

In Figures 4.19, we show a frame by frame snapshot of a wolf running with a gallop like gait. With the integration of the pseudo physics and the flexible spine, the wolf moves in a quite natural and life-like way, relatively similar to Figure 4.3 and 4.8.

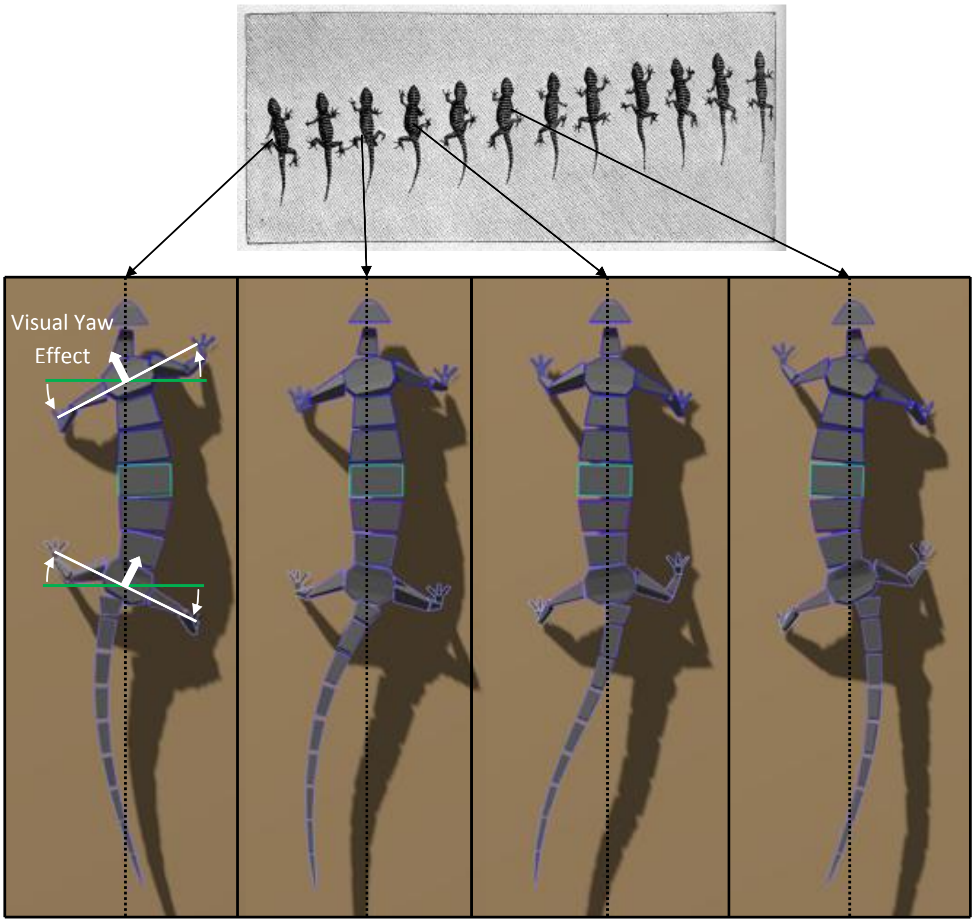


Figure 4.18: *A lizard running upward in a believable way compared to a real life lizard, the top image is courtesy of [Mar93]. The visual yaw effect is calculated using the current position of the feet on the ZX-Plane. Lizard tail is animated using the system presented in next chapter.*

We use the same previously described environment to do the performance tests: heightmap with obstacles (3000 crates, 40% dynamic). The simulation include the LOD aspect with all characters in the camera field of view. We do not add any new LOD techniques on these newly added components. In all tests the final simulation stays real-time (∼30fps).

In Figure 4.20 we show average calculation time, on each simulation step, when adding pseudo physics to 8-legged creatures (spider model). We can observe that calculation time increases by 9%. Meaning that our pseudo physics system is efficient enough to be used without any performance hit.

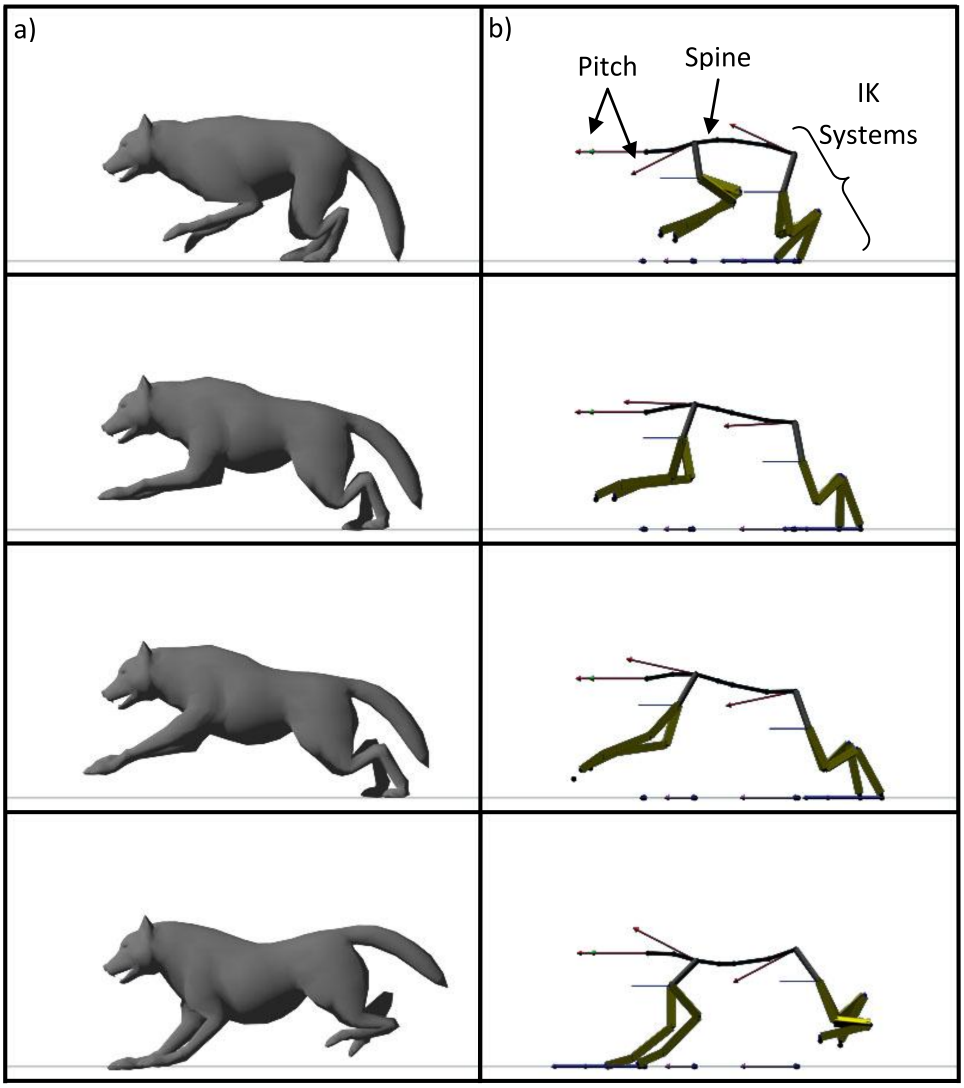


Figure 4.19: *A wolf running (galloping) to the left with its spine model being deformed based on the pseudo physics and the pitch control. a) 3D mesh. b) spine model and IK systems*

In Figure 4.21 we show average calculation time, on each simulation step, when adding the spine model coupled with the pseudo physics to 4-legged creatures (wolf and lizard model). Every quadruped has 3 virtual pelvis nodes and an average of 7 spine nodes. We can observe that calculation time increases, when adding a spine model, for about 30-40%. This increase is caused by several things: triple pelvis elevation calculations, sampling of the Hermite curve, rotation and translation propagation, *etc.* As for coupling the flexible spine with pseudo physics, we can observe the same calculation time increase (9%) shown before in Figure 4.20. This performance hit, by using a spine model, is justified as quadrupeds now move in a more natural and realistic way.

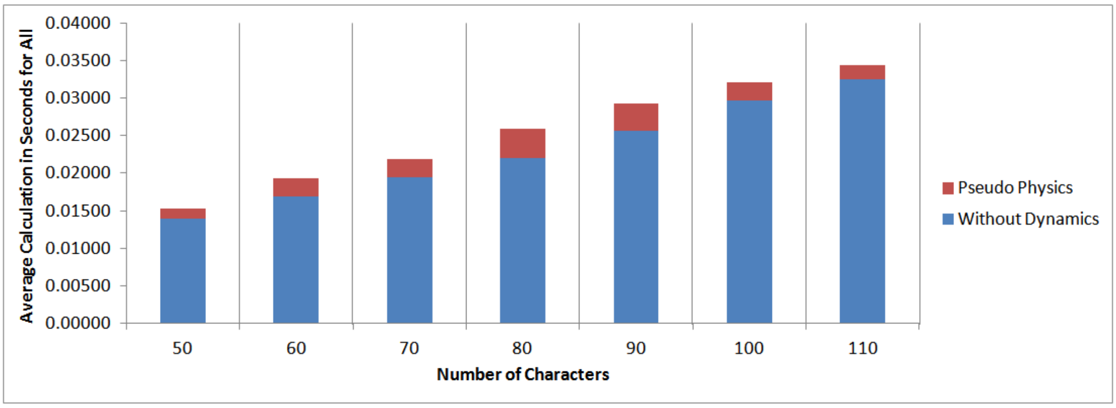


Figure 4.20: *Average calculation time when adding pseudo physics to the locomotion system.*

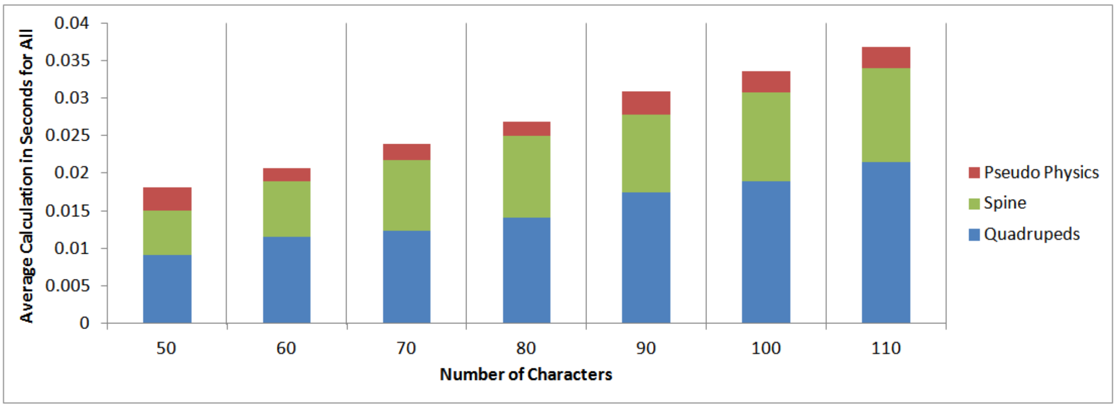


Figure 4.21: *Average calculation time when adding our spine model coupled with pseudo physics to the locomotion system. In second.*

4.5 Conclusion

We presented several components in this chapter that adds naturality and realism to the real-time locomotion system. We implemented a pseudo physics system capable of adding more realistic movements to the pelvis. Making the animated creatures more believable as they move on the sagittal plan based on the user assigned gait, while keeping the calculations simple as we use minimalist dynamics modeling from particle-based physics. We added also a flexible spine model that gives the simulated quadrupeds more DOF, allowing them to move in a life-like way. The proposed model uses simple geometric calculations and satisfies joints limits. By implementing other visual components like non-linear character progression and gait randomizer, the final simulation looks less repetitive and the multi-legged characters behave in a more organic-way. Our system even maintains its efficiency, as adding these extra blocks does not affect the total performance in a drastic way.

The generic spine model and pseudo-physics system [AMG+12] were accepted in the 9th Workshop on Virtual Reality Interaction and Physical Simulation (VRIPHYS) 2012 and to be presented in the conference next December held in Darmstadt, Germany.

Chapter 5

Controlled Oscillation Effects

Contents

5.1 Pendulums System Overview 104

5.2 Time Based Spring Dampers Control 105

5.3 Pendulums Strategies 107

5.3.1 Father Pursuit Strategy 108

5.3.2 Son Pursuit Strategy 109

5.3.3 Final workflow 110

5.4 Advantages 111

5.5 Implementation in Animation Systems 113

5.5.1 Adding Physical Reaction Effects to any Skeleton-Based Bodies 113

5.5.2 Cloth Simulation 115

5.5.3 Secondary Motion in Locomotion System: Results and Performance 117

5.6 Conclusion 119

An articulated body is a chain of rigid bodies, with many Degrees of Freedom (DOF), length constraints, angular limits, *etc.* The physics that govern its movement is computationally expensive, numerically imprecise, and often difficult to predict and to control. Many surveys (Chapter 2) like the one by van Welbergen

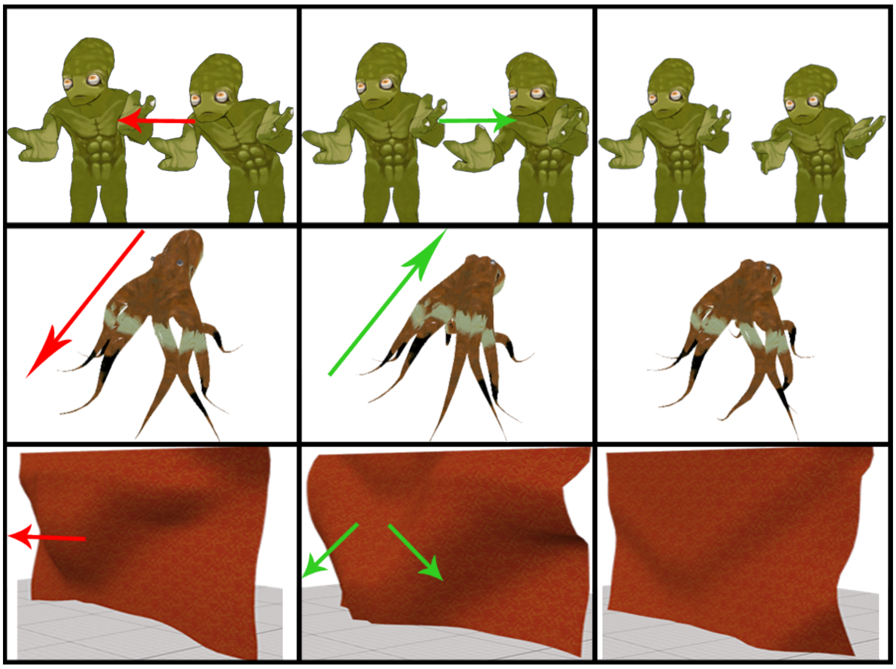


Figure 5.1: *Some example uses of our system. Red arrows represent external perturbations. Green arrows represent the system's response.*

et al. [vWvBE+09] gives a good overview of the different methods and paradigms used to simulate these articulated bodies, and shows the tradeoffs between animation control and motion naturalness. In this context, there is a crucial demand for real-time methods providing physically plausible with controllable effects [BHW96].

In this chapter, we propose an original system that adds physical like reaction effects to any skeleton-based object, in real-time with a full user control using a proposed 3D pendulums paradigm. The effects we seek to obtain are based on damped oscillatory motions that propagate through an articulated chain. The effect may be visually plausible like a rope moving in the wind, or a body reacting to external forces. In our system each bone of the articulated body is animated by a 3D pendulum. Each pendulum is guided by a spring damper that pulls it toward a user-definable target direction. Our approach has two objectives. Firstly, we ensure body length constraint (*Hard Constraint*) between any two joints by only working on the angle between the bodies. Secondly, we make a predictable real-time system in which we can control the reaction time to reach a user-defined direction and also the regime (critical or underdamped) of the oscillations around this direction. Our pendulums have three degrees of control: reaction time, damping and target direction. This concept of 3D pendulums may be applicable in a multitude of scenarios with some of them illustrated in Figure 5.1. Plus these 3D pendulums can be used in the previously presented locomotion system in order to add secondary motions, introducing oscillation/vibration effects to the final simulation. We must emphasize that our system is better suited for acyclic bodies.

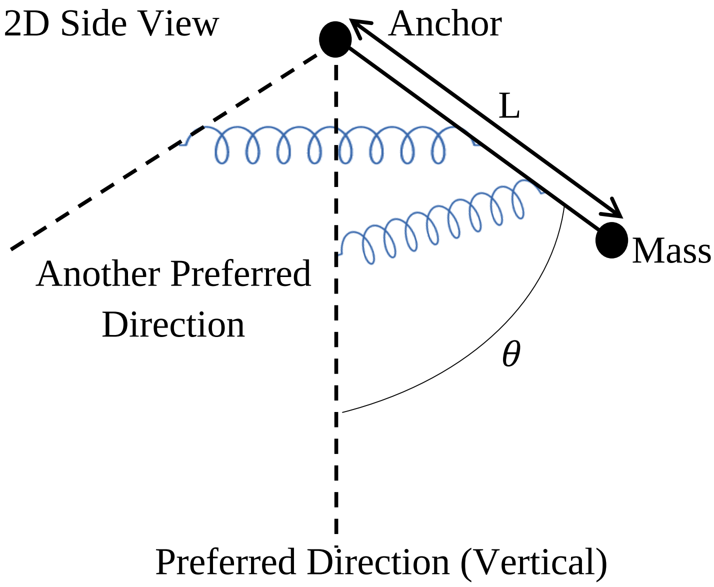


Figure 5.2: *Simplified representation of the pendulums that we propose, with the spring and the preferred target direction.*

Our system is really easy to implement, as we will see in the following sections, with no need for a full physics simulation, or any kind of complex calculations (like in a fully *physics-based* simulation). 3D pendulums (described in Section 5.1) are the building blocks of our system, with each bone of an articulated body animated with a pendulum. A pendulum is an anchored bar, with a fixed length L, attracted to its target direction by a spring (see Figure 5.2). This spring pulls the pendulum toward this direction. This idea allows the system to easily add plausible oscillations to any animation with a temporal control (explained in Section 5.2). In Section 5.3 we present the linear algorithms that deals with a tree of pendulums or a skeleton and how the motion of a single pendulum is propagated to its father and sons. In Section 5.4 we explain more our contribution in contrast of existing techniques: PD controller and the Model Reference Adaptive Control (MRAC) controller. Performance and uses are discussed in more details in Section 5.5.

We concentrate only on deforming the skeleton of the articulated body, without any specific treatment to the 3D mesh (like in [CGC+02]). Our mesh is animated by the classical linear blend skinning. In Section 5.5.2, we show a rigid tissue represented by a tree of bones animated by our approach with a large time-step and controllable computation, with no intention to compete against realistic fabric simulation, like the one presented by Volino *et al.* in [VMTF09].

5.1 Pendulums System Overview

Our pendulums oscillate visually like real 3D pendulums by computing their movements with a mass-spring approach on the angle. A mass-spring approach is a very simple way to simulate *physics-based* animation [MSJT08], as it offers an intuitive and flexible means of modeling a mechanical system. In Pixar's movie WALL-E [KL08], they used a mass-spring system in a derived fashion to animate large crowds of humans and robots in a believable way. While in [NH04], Nagurka and Huang used a mass-spring damper model to generate a plausible animation of a bouncing ball.

We design a pendulum $\overrightarrow{V}$ as a rotating bar attracted to its preferred direction by two springs: one spring on each 2D plane XY and ZY as illustrated in Figure 5.3. We choose this scheme with two springs instead of one spring to avoid spiral rotation motion around the target direction. The computation of pendulum $\overrightarrow{V}$ motion is done using its projections $\overrightarrow{V_{XY}}$,$\overrightarrow{V_{ZY}}$ independently. During the motion, after calculating the two new spring positions in 2D $\overrightarrow{V_{XY}}$ and $\overrightarrow{V_{ZY}}$, the 3D position $\overrightarrow{V}$ is obtained by combining them and ensuring that $\|\overrightarrow{V}\| = \|\overrightarrow{V_{XY}}\| = \|\overrightarrow{V_{ZY}}\| = L$. In the current implementation we omit the twist component around the axe of the 3D pendulum $\overrightarrow{V}$ which is the third DOF (to be added in future work). It is interesting to notice that by using the target direction $\overrightarrow{-Y}$, we can give the impression of gravity that always pulls the bodies toward the ground.

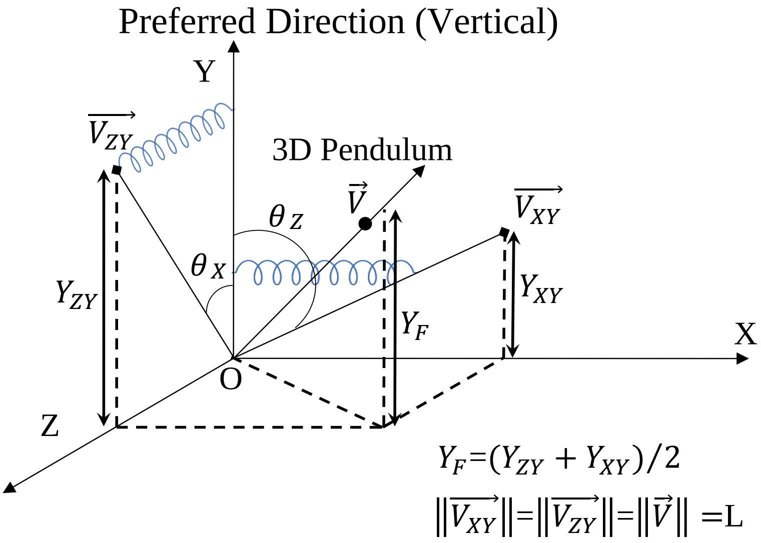

Preferred Direction (Vertical)
Y
$\overrightarrow{V_{ZY}}$
3D Pendulum
$\vec{V}$
θ_Z
$\overrightarrow{V_{XY}}$
θ_X
Y_{ZY}
Y_{XY}
Y_F
X
O
Z
$Y_F=(Y_{ZY}+Y_{XY})/2$
$\|\overrightarrow{V_{XY}}\|=\|\overrightarrow{V_{ZY}}\|=\|\vec{V}\|$ =L

Figure 5.3: *A 3D pendulum composed of two 2D pendulums with Y as their preferred direction.*

5.2 Time Based Spring Dampers Control

Let m be a mass connected to a spring with stiffness constant k. This mass oscillates around a rest position x_0 with a viscous damper that has a damping coefficient c. Based on Newton's second law of physics the acceleration is $\ddot{x} = -(k(x - x_0) + c\dot{x})/m$ where x is the current position of the mass, and $\dot{x}$ is its velocity. We integrate this motion using the Verlet scheme [MSJT08] which was numerically stable during our experiment. Giving a random position x to the mass, it oscillates around the rest value x_0, seeking to minimize the error $(x - x_0)$ until reaching zero. This oscillation depends directly on the constants (k, c, m). In order to achieve temporal control on the spring damper movement, we use the *Settling Time* T_s principle. It is the time required for the mass position x to reach its max amplitude inside a given error interval (see Figure 5.4) and remains inside it. This interval is symmetrical around x_0.

$$T_s = -\frac{ln(tolerancefraction)}{\zeta * w_0} \tag{5.1}$$

Where the *tolerance fraction* is the needed error interval shown in Figure 5.4, w_0 is the natural frequency and ζ is the damping of the ordinary differential equation governing a damped harmonic oscillator:

$$m\ddot{x} + c\dot{x} + k(x - x_0) = 0$$

or

$$\ddot{x} + 2 * \zeta * w_0 * \dot{x} + w_0^2 * (x - x_0) = 0$$

with

$$\zeta = \frac{c}{2mw_0}, w_0 = \sqrt{\frac{k}{m}} \tag{5.2}$$

We fix the tolerance fraction to 5% based on our experiments in equation (5.1). The user only needs to provide the *settling time* T_s (reaction duration) and damping value ζ (critically damped or underdamped). Using this *tolerance fraction* constant, *settling time* and damping, the natural frequency w_0 is automatically calculated from equation (5.1) and the spring damper constants k and c are calculated from equation (5.2). With that we achieve a total control over the curve of the spring damper based on the user needs while maintaining its dynamic aspect.

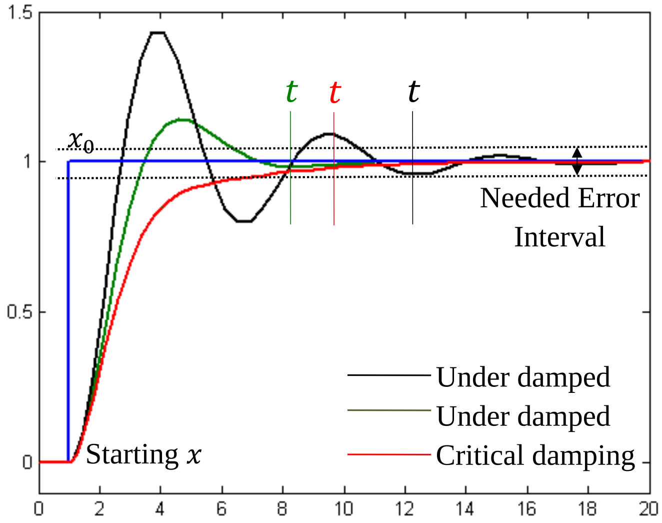


Figure 5.4: *Spring oscillation under different damping values.*

Figure 5.4 illustrates springs oscillating under different damping values. They oscillates around their x_0 until full stop, with their respective settling time. The third spring damper is a critical spring damper which converges toward x_0 faster than the others, and without oscillation.

5.3 Pendulums Strategies

The skeleton of an articulated body is a tree of connected joints (articulations). By connecting several 3D pendulums and by defining the target direction for each one of them, the final result is a tree of pendulums that map the articulated structure, as shown in Figure 5.5. Some of the 3D pendulums act as a father node for several others. When they move, the anchor points of their children move.

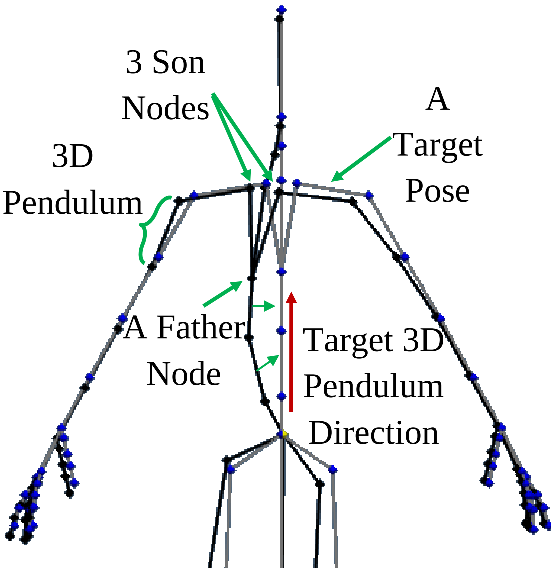


Figure 5.5: *3D pendulums tree in black with target direction in grey.*

These 3D pendulums need to interact with each other in order to have a believable reaction like when the son nodes move according to their father movement. We define two strategies used in conjunction to achieve this goal: Father Pursuit strategy (Section 5.3.1) and Son Pursuit strategy (Section 5.3.2). For simplicity, these strategies will be described in a 2D plane.

5.3.1 Father Pursuit Strategy

The objective of this strategy is to propagate the motion of the father 3D pendulum toward its children, thus they need to incorporate this movement in their own motion. Figure 5.6 illustrates two connected pendulums P_A and P_B. A and B are the positions of each mass. L_A,L_B are the lengths of the bars. θ_A,θ_B are the errors that each pendulum seeks to minimize. In this example the preferred direction of the pendulums are identical (the dashed $\overrightarrow{-Y}$).

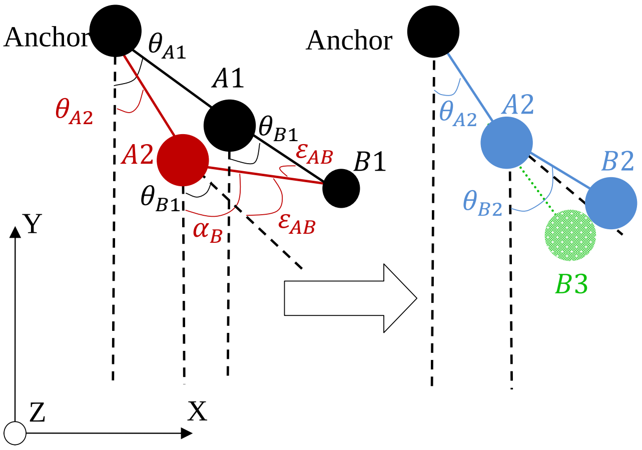


Figure 5.6: *Father Pursuit Strategy.*

The update system is a top-down system scheme, starting from the anchor toward the leaf. First, on time t_1 (in black) the error that we try to minimize is θ_{A1} in P_A and θ_{B1} in P_B. Now, on time t_2 (in red):

1. P_A moves. Its spring damper tries to minimize the error, and has a new position $A2$.
2. P_B: the angle ε_{AB} between the two vectors $\overrightarrow{B1A1}$ and $\overrightarrow{B1A2}$ is added to its own error, $\alpha_B = \theta_{B1} + \varepsilon_{AB}$.
3. P_B: letting the spring damper integrate its equations, we obtain a new angle value θ_{B2} which contains the new pursuit error.
4. P_B: based on L_B the new position $B2$ (in blue) is calculated.

Without this process, the new position of P_B would have been $B3$ (in green), which is not correct and would have produced a non-logical disconnected motion. This Father Pursuit process is extended to every pendulum in the chain. A third pendulum P_C follows the motion of its father P_B, and so on. By extending this process in 3D we have a totally plausible physical chain of 3D pendulums. Each one reacting to its father's movement while oscillating around its target direction.

5.3.2 Son Pursuit Strategy

The objective of this strategy is to reflect the perturbation that can occur on the son level, to reflect it on its father. It occurs when the mass of P_B takes a perturbation as a result of being pulled or collided with another object, as seen in green in Figure 5.7. The perturbation is regarded as a change in the position, as if we only take the final position resulted of an impulse applied to a rigid body.

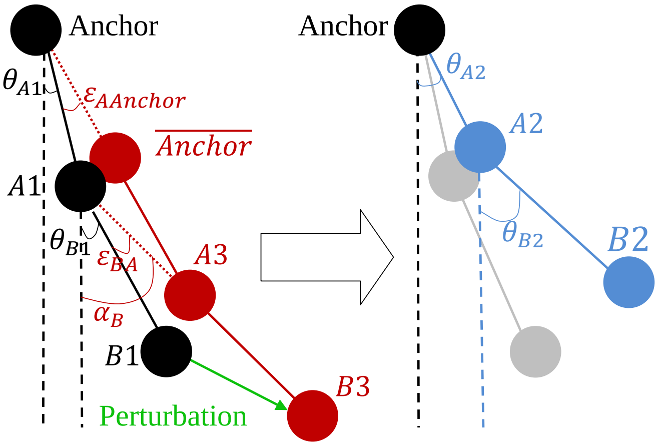


Figure 5.7: *Son Pursuit Strategy.*

1. The perturbation induces its full impact as if the mass P_B was not attached (in red).

2. P_B's mass has two positions: $B1$ (the old one) and $B3$ (the new one). Inverting the previously detailed computation of the father induced error ε_{AB}, we calculate the child error ε_{BA}, the angle between $\overrightarrow{A1B3}$ and $\overrightarrow{A1B1}$ and adding it to θ_{B1}, we obtain $\alpha_B = \theta_{B1} + \varepsilon_{BA}$.

3. The mass of P_A should follow, as it is being pulled by its son now. The new position $A3$ is calculated easily by choosing on the line $A1B3$ the point $A3$ where $\|A1B1\| = \|A3B3\|$.

4. This process propagates toward the anchor.

5. The new positions are recalculated based on the fixed anchor position.

With this scheme, all the errors that the spring dampers need to minimize because of a perturbation are calculated in a bottom-up way starting from the son that took the perturbation toward the anchor.

5.3.3 Final workflow

In a tree of pendulums, calculation cycles may occur when two nodes are influencing each others in an endless loop (father influencing its son, then the son influencing its father, and so on.). To avoid these kinds of loops, we use an update system inspired by Featherstone's divide-and-conquer algorithm [Fea99a, Fea99b]. This algorithm eliminates any cyclic calculation problems and breaks the computation into two main linear passes. The first is a bottom-up pass through the articulated body tree, and the second is carried out from the top to the bottom. We adopt this paradigm completely. Only the calculations differ, as listed below:

1. For each 3D pendulum perturbed in the tree: resolve this perturbation by applying it on its mass then calculate the errors ε in a bottom-up iteration toward its ancestors according to the Son Pursuit strategy.

2. For each father 3D pendulum integrate all the children errors (ε_1,ε_2 *etc.*) to its own error θ.

3. Start the standard top-down pass starting from the anchor toward the leaf according to the Father Pursuit strategy.

In the previous step 2, there are many ways to calculate the integration:

- Summing up all the perturbation errors coming from its children: it is the method used to produce all of our results. It is the simplest method, and the one we have chosen after testing.
- Calculating an average: the father node will be perturbed in the same direction as the previous method, but with less amplitude. It is useful when the application decides that the father should be less affected by its children.
- Doing a weighted average based on:
  - The Mass: the heavier son has more influence on its father.
  - The importance of each branch: assigning predefined priorities on the children.

We can imagine many other possibilities based on a specific application's needs. Our system is quite easy to implement and the actual calculations in each strategy require only basic knowledge of 3D vector math. No prior knowledge of physics systems is required; we do not compute the inertia matrix nor do we use the notion of force. At the same time we can use physics principles to enhance the end result like in the case of the father pendulum integrating its children's errors based on the inertia matrix.

5.4 Advantages

In the following section we demonstrate our system with a skeleton based skinned 3D models. Starting from the bind (rest) pose of the skeleton, we **automatically** create a 3D pendulum for each skeletal connection (bone) with the same length and with its preferred rest direction calculated from the bone rest pose orientation. By maintaining the hierarchy of the base skeleton, we have a tree of pendulums that maps this skeleton perfectly. While playing motion data, we modify the target direction of each corresponding pendulum, **exactly** mimicking the base animation. If the 3D pendulums start to react to an external perturbation, each of the 3D pendulums orientation and position is applied to its corresponding bone.

Our method may be related to a PD controller, and also to the MRAC controller that uses the Adaptive Control proposed by Landau in [Lan79] and used by Kokkevis *et al.* in [KMB96]. Like the PD controller, a MRAC controller is used to add reaction effects to articulated bodies, but differs in the way that it adapts its calculated response based on the needed reference model. It provides the user with direct control over the speed and type of response to sudden changes. All of these controllers main goal is like our pendulums: to bring a bone to a preferred direction. But we differ in several essential ways. Firstly, even without any external perturbation, the other methods always try to return to a preferred direction (or position). This introduces a delay (Figure 2.19) in the produced animation between the target pose (keyframe or MoCap data) and the response of the PD controller (as seen in [ZH02]). On the other hand, our system is a superimposed layer over the motion data and only reacts when there is an external perturbation. Our system plays exactly the motion data with no delay, and only adds the reaction effects when needed. Another difference is the controllability; our system is designed to be temporally controlled (control over the reaction time). Temporal control in the case of the PD controller is hard to achieve and demands some hand tuning of the gain constants. In [ACSF07], Allen *et al.* show the ability to temporally control PD controllers (using adaptive calculation of the gain constants), but it involves some heavy calculations of the inertia matrix of each joint on each motion clip, with specific calculations in the case of an external perturbation (calculating the re-entry motion clip). Finally, all the mentioned controllers are always critically damped. On the other hand, our pendulums can be critically damped or underdamped while maintaining the temporal control (Figure 5.4). A MRAC controller is designed to be temporally controlled but it calculates and adapts its gain constants on each frame using forces and torque calculations. Our pendulums do not need any adaptive pass once the user sets the reaction time and damping. Additionally, this reaction time and damping can be modified in real-time.

5.5 Implementation in Animation Systems

In this section, we present several ways to use the 3D pendulums tree: adding physical effects to lifeless models like an octopus, modifying pre-defined motion data with physics reactions, anecdotally a cloth simulation (which is normally a closed-loop problem) and most importantly, integrating these 3D pendulums in our original locomotion system (secondary motion in Section 5.5.3).

In all cases, the pendulum's reaction time, damping, and target direction is totally controllable. We do not manage collisions, but we can easily imagine a system that creates an impulse (change in the position) on each 3D pendulum to counter any penetrations that occur. Section 5.5.1 and 5.5.2 use the following test machine: Intel Core 2 Duo 2GHz, 2 GB RAM, with an ATI X1400, 256 MB.

5.5.1 Adding Physical Reaction Effects to any Skeleton-Based Bodies

In Figure 5.8, we use our system on a lifeless octopus model. By adding some simple procedural animation to its tentacles (pulling only the root node of each tentacle toward the center at random intervals) the rest of the model reacts in a passive way, modifying the animation and adding plausible physics effects. The octopus model performance data can be found on table 5.1.

	Joints Number	Perturbations Number	Only MoCap	With oscillation effects
Octopus	150	8	NaN	0.3 ms
Alien	92	2	0.06 ms	0.46ms

Table 5.1: Average pendulums calculation time per frame on our test machine.

We can also use our system on animated models. In that case on each frame, the motion data takes control of the skeleton changing the preferred direction of each pendulum. With no external perturbations, the 3D pendulums rigorously follow the animation data. When an external perturbation occurs, our system reacts to this while continuously trying to return to the desired target pose. With such a technique, our system adds plausible physical reaction effects to predefined animation data, as a superimposed animation layer. These reactions

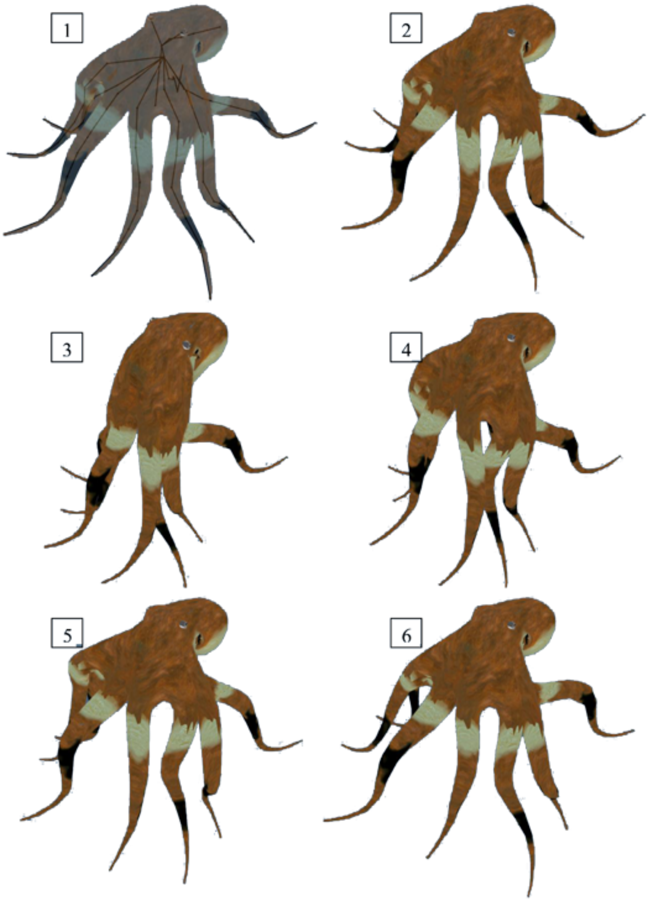


Figure 5.8: *From top left: [1] The model with its 3D pendulums. [2] Rest pose. [3] We pull all the tentacles toward the center. [4] Reacting. [5] Tentacles overshooting (underdamped regime). [6] Return to rest pose.*

can furthermore be customized by making a section of the body more rigid, more flexible, changing the reaction time, or tuning the damping. This gives the user a powerful tool to modulate the reaction of the body in a very easy and intuitive way.

Performance data for the alien model in Figure 5.9 can also be found on table 5.1. As we can observe, the average computation time for each frame rises from 0.06 ms to 0.46 ms, which stays negligible. This added cost is the result of reading the motion capture data in order to change the pendulum's target direction, integrating the perturbation and then performing the main integration (as previously described).

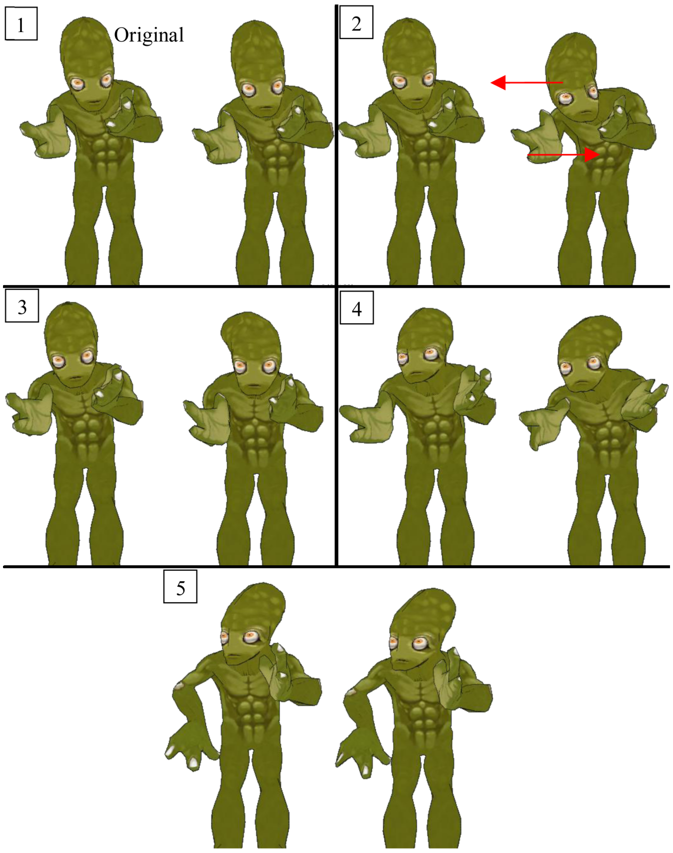


Figure 5.9: *From top Left: [1] Original (on the left) and our simulated articulated body (on the right). [2] Two perturbations. [3] to [5] Reaction and returning to the original MoCap.*

5.5.2 Cloth Simulation

Although cloth is a closed loop problem, we are capable of giving the impression of an animated cloth by simply creating several vertical 3D pendulums that cover the cloth, plus attaching several horizontal 3D pendulums to each vertical one (one vertical is shown in Figure 5.10).

By doing a weighted average between the positions of all horizontal pendulums activated by their vertical father, weighted based on the distance between each horizontal pendulum and its vertical father, we compute the final cloth position. This results in a fully reactive cloth, without any tearing problems, that maintains its horizontal and vertical dimensions, while giving total control over the reaction time. We are not aiming to compete against more general, visually and physically accurate cloth simulators that are, for example, better suited to simulate actual

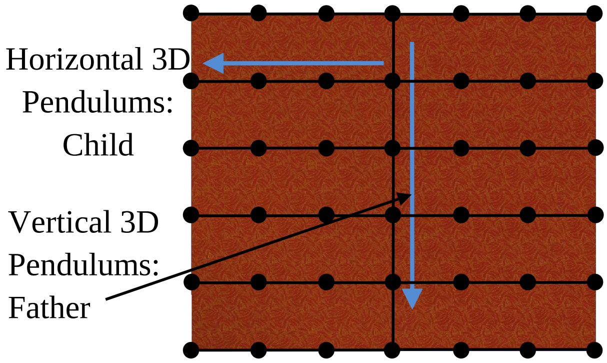


Figure 5.10: *A cloth represented as a tree of pendulums.*

human cloth. We are just proposing a less sophisticated, but stable and relatively fast method that can plausibly simulate the motion of reactive cloth. In Figure 5.11, an external perturbation is applied to the middle three vertical pendulums. In order to optimize calculation time, those three vertical pendulums are the only ones actively being simulated (with the horizontal children of each one of them). The mesh is simulated using approximately 1500 3D pendulums. The average calculation time of these pendulums with the post calculations for the final cloth is around 5 ms.

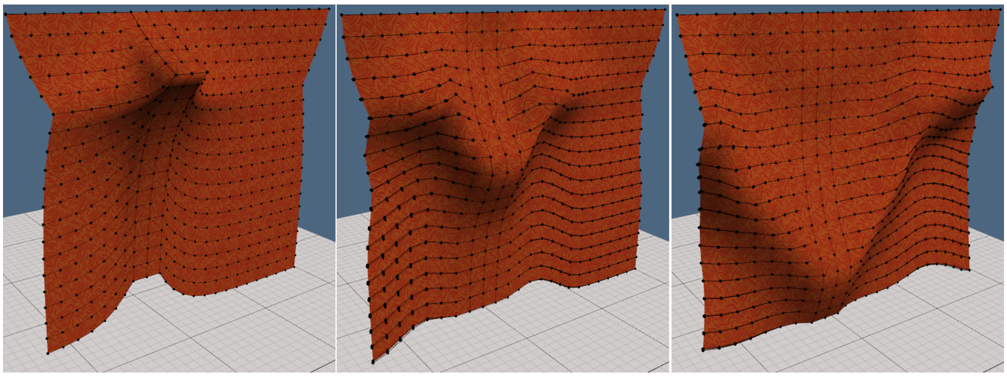


Figure 5.11: *Cloth being pulled in the middle with a visual representation of the 3D pendulums.*

5.5.3 Secondary Motion in Locomotion System: Results and Performance

We integrate the previous 3D pendulums in the main locomotion system presented in Chapter 3. We add these oscillation effects to the ant/spider antennas and wolf/lizard tail, shown in Figure 5.12. We also add these oscillations to the actual ant/spider body, giving them a more organic-feel. These added secondary motions make the final simulation more realistic, as body parts of the simulated creatures react to its actual movement, which gives the multi-legged characters a plausible and a believable organic feeling.

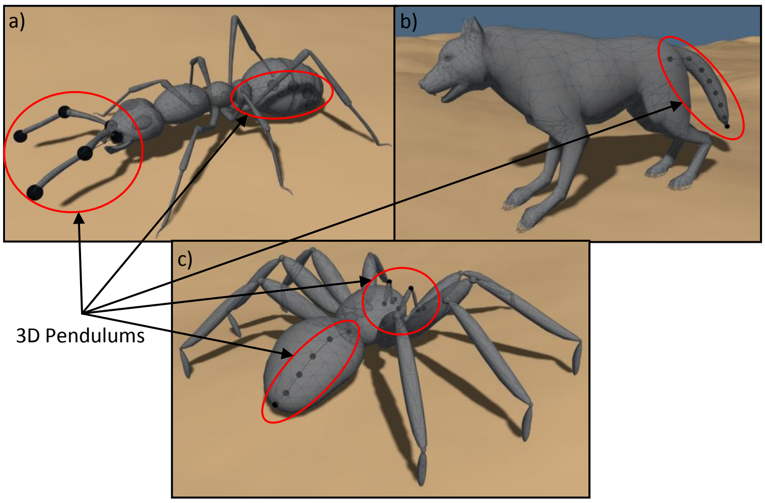


Figure 5.12: *Examples of the integration of the 3D pendulums system in the main locomotion system. a) Ant antennas and body. b) Wolf tail. c) Spider antennas and body.*

In Figure 5.13 we show the resulting wolf tail animation after adding secondary motion to it. The resulting wolf tail is more life-like and reacts to the actual wolf movement. We show also two varieties of the tail responses (a flexible vs. a rigid tail), which is easy to achieve and change in real-time using the presented 3D pendulums system. Lizard tail is shown in Figure 4.18.

In Figure 5.14 we show the average calculation time for the 3D pendulums, on each simulation step, when integrated in our real-time locomotion system. We added 3 pendulums tree to each simulated multi-legged character, with an average of 5 nodes per tree. Environment is simple (flat terrain with no obstacles).

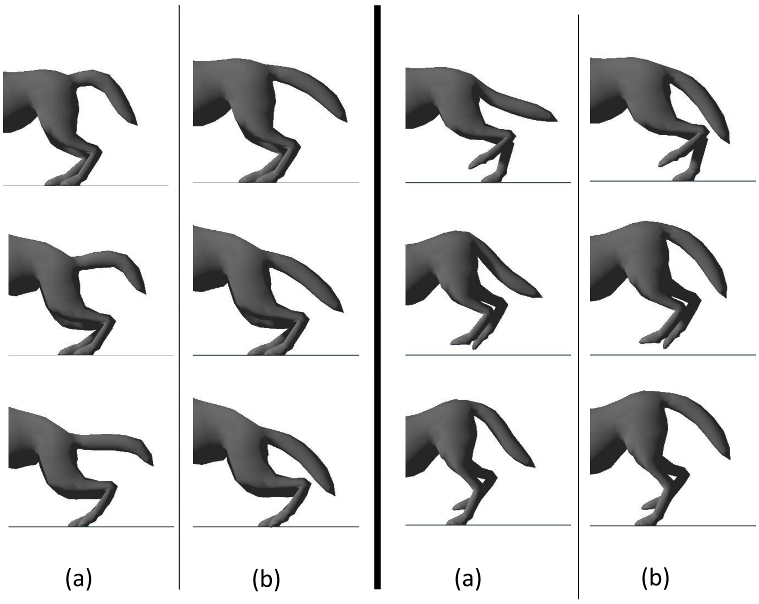


Figure 5.13: *Frame by frame example of the wolf tail animation after adding secondary motion. (a) Simulating a flexible tail. (b) Simulating a more rigid tail.*

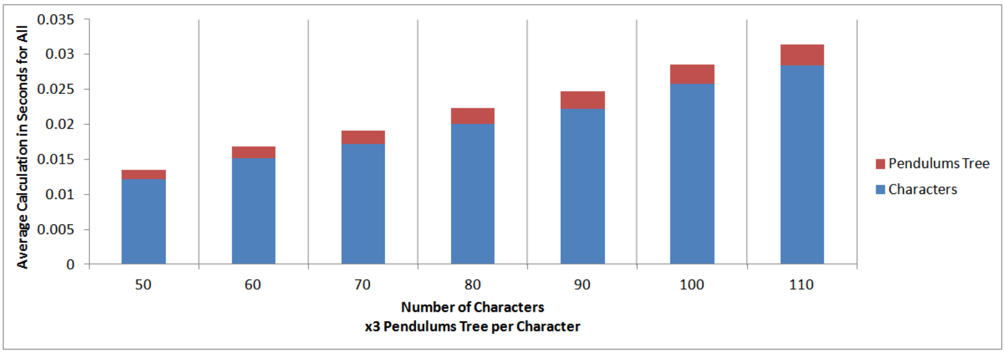


Figure 5.14: *Added Calculation time when using pendulums tree in our locomotion system. In seconds.*

We can observe that our oscillator system is quite efficient: for a 110 characters (330 pendulums tree, 1550 3D pendulums) it takes only ∼3 milliseconds to change the target direction, integrate the root node movement using previous strategies and to do a dynamic step for all spring dampers. Which means an increase of about 10% in the overall locomotion system calculation time. This added cost is quite negligible compared to the visual effects of the added secondary motions that helps generating a better immersive virtual simulation.

5.6 Conclusion

The presented system for secondary motions is linear, straightforward, and based on simple 3D pendulums. It is capable of adding physical like reaction effects to skeleton-based bodies very easily. Additionally it is highly customizable as we can control in real-time the target direction, reaction time and damping of the motion. And the results show a better final animation as adding these secondary motions give an organic feel to the simulated multi-legged characters. Thus, it helps in making the generate locomotion animations more plausible and believable.

This pendulums system [AKGM+11] is published in the Transactions on Edutainment VI journal and was presented in the Computer Animation and Social Agents (CASA) Conference 2011 in China.

The current system does not enforce angular constraints. We need to incorporate them into our future work in order to simulate real-life joint constraint that exists in most skeleton-based bodies.

Chapter 6

General Conclusions

In this thesis we have been interested in locomotion animations, an essential part of virtual worlds simulations. Virtual creatures should have the ability to move freely through these virtual worlds, in order to make these computer based simulations more immersive. The generated animations should be life-like and convincing, and the system should be efficient enough to populate the simulation with the maximum number of characters. These points are what motivated us in developing our locomotion system. Our system is **Procedural-Based** because motion data for the multi-legged characters that we target are either rare or non-existent and because **Procedural-Based** techniques are more generic, controllable and adaptive compared to **Data-Driven** ones. We use *kinematics-based* techniques with a minimal physics approach as fully *physics-based* ones suffer from bad performance problem, difficulty to implement and lack of controllability.

We presented in this thesis a real-time generic locomotion system capable of animating a wide range of multi-legged character morphologies with intuitive user control. Our locomotion system fulfills four main objectives. It is capable of **adapting** the generated animation to a complex dynamic environment and different morphologies. The user has a total **control** over the final locomotion and can design the style needed through our user-friendly interfaces. The system generates **plausible** animations that are believable and life-like. Finally, it is **efficient** enough to simulate dozens of creatures in real-time. The system is *kinematics-based* and uses biomechanics observations in order to generate final animation. Observations like the one found in [INM66, MT55, Muy57, Ale96] that describe how most terrestrial animals move: placing one foot in front of the others in a successive way until reaching the creature point of interest (target).

Our system is composed of four main blocks that interact with each other in order to generate final animation: a character controller, a gait manager, a 3D path constructor and a footprints planner. **The character controller** is the central main structure in our system and relies on the three other blocks to generate final locomotion. It manages the feet movement by using the tempo set by the user in the gait manager. It calls the footprints planner whenever a foot is in *swing* phase in order to calculate this foot 3D trajectory. The character controller manages also the pelvis movement using the assigned 2D orientation, speed and the elevation of the environment underneath the multi-legged character. **The gait manager** is responsible of the locomotion cycle tempo. The user can design the needed gait by setting the *stance* and *swing* phase of each foot using our user-friendly interface. **The 3D path constructor** is capable of efficiently generating 3D trajectories between any two 3D points in the environment using a **Grid-Based** path planner and two discretization maps: *obstacles' map* and *elevations map* . Finally, **the footprints planner** evaluates in real-time all possible 3D trajectories that navigates through the environment toward all possible targets starting from the current foot position. Then, our main algorithm picks the best **couple** in this search space: best 3D trajectory toward best footprint target. This algorithm is efficient enough to be called several times for each creature, which helps in navigating dynamic environments. By using the CCD IK system to calculate the position of the intermediate leg joints that connects the pelvis with the feet, we generate the multi-legged character final locomotion animation. We implement also several LOD techniques that help in increasing the number of simulated characters while always generating plausible animations. Our system can animate dozens of multi-legged characters, in real-tim, in a dynamic and complex environment.

In order to make final animation more life-like we added three main components that help making the generated animation more realistic without affecting the efficiency of our system. First, **pelvis movement using pseudo physics**. We generate the sinusoidal-like ballistic movement of the pelvis (observed in nature), using particle-based physics computations and values of the gait pattern.

This pseudo physics system uses Newton laws and the feet phases in order to automatically calculate the forces that each foot is exerting on the pelvis. Without any extra setting, the user can edit this movement easily through the gait manager interface. Another added component is a **flexible spine model**. We proposed a simple geometry-based calculations and 3D Hermite curves in order to generate a flexible spine model while animating quadrupeds. This spine is an essential part when animating these quadrupeds as it gives them more agility (more DOF). Our model is easy to implement and enforces joint limits in order to have logical results. Finally, we introduced a tree of **3D pendulums** that adds secondary motions to the animated multi-legged characters. These oscillation effects (like in the ants antenna or the wolf tail) help in giving an organic feel to the animated creatures, thus helping in creating a better immersive experience. The developed system uses simple 3D vector-based math without any physics calculations.

Through our user-friendly interfaces, users can control our system parameters to generate the desired locomotion. List of all parameters can be found in Appendix D. We are currently working on an automatic method that calculates the best parameters values based only on one or two user criteria's. Like finding the best gait pattern and feet spacing based only on a fixed speed, to generate a natural looking locomotion.

An interesting aspect in future work is adding balance strategies to the multi-legged characters, like taking small steps, weight shifting, changing support, *etc.* to counter external pushes and perturbations. To achieve that we need to have more morphology specific preferences in order to simulate the way real-life creatures counter these kind of external perturbations. We also need more complex physics and dynamics to simulate the effects of these forces.

Our system is capable of generating locomotion for bipeds as shown in Figure 6.1. But the final locomotion still lacks realism, because in the case of humans the animation should be as close to reality as possible otherwise it will not be convincing. To achieve that we need a better foot model with a better simulation of the metatarsus[1] movement in order to capture the essence of human locomotion. Generating natural human locomotion will be one of our main focus as future work.

---

[1] The metatarsus or metatarsal bones are a group of five long bones in the foot located between the tarsal bones of the hind- and mid-foot and the phalanges of the toes.

By achieving all these perspectives, our system will be a near complete generic animation system. Our ambition that it will be used in the industry, as it can resolve many problems that currently exist. Until now, recent real-time simulations like video games and serious games still suffer from repetition in the characters animation, long development cycle and the need of application specific systems. Our system will give these animated characters the capabilities to adapt and react to the environment in real-time, in a natural and logical way. Plus, it is reusable and generic thus decreasing development time.

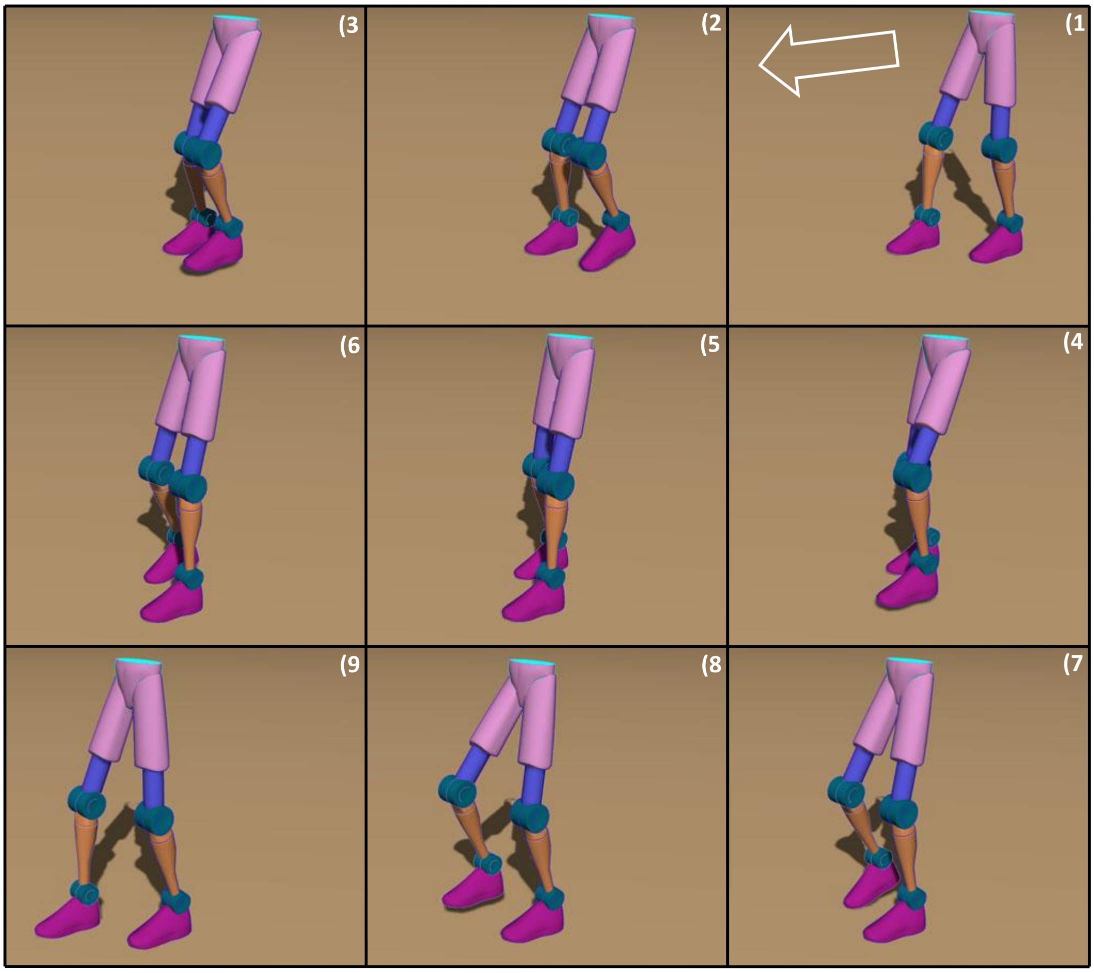


Figure 6.1: *A biped moving forward.*

Glossary

MoCap Motion capture is the process of recording a movement and translating that movement on to a digital model. It refers to the process of recording actions of an actor/creature, and using that information to animate a digital model in 2D or 3D computer animation. x–xii, 3, 8, 10–14, 18, 19, 27–30, 77, 112, 115, 129

Motion graphs are graphs that manage the transition between different motion data clips. 12–14

AI Artificial Intelligence. 5, 14

BFS Breadth-First Search is a search algorithm that begins at the root node and explores all the neighboring nodes. Then for each of those nearest nodes, it explores their unexplored neighbor nodes, and so on, until it finds the goal. 34, 57, 58, 138

CCD Cyclic-Coordinate Descent. xvi, 17–19, 44–46, 66, 71, 74, 93, 122

COM Center Of Mass is a point in space where, for the purpose of various calculations, the entire mass of a body is concentrated. 22–24, 29, 30

Convex hull is the minimal convex set containing a group of points. 48, 49, 61, 91

CPU Central Processing Unit is the portion of a computer system that carries out the instructions of a computer program, to perform the basic arithmetical, logical, and input/output operations of the system. The CPU plays a role somewhat analogous to the brain in the computer. 70

Database is an organized collection of data, managed to some level of quality (measured in terms of accuracy, availability, usability, and resilience). 10, 12, 14

DFS Depth-first search is an algorithm for traversing or searching a tree, tree structure, or graph. One starts at the root (selecting some node as the root in the graph case) and explores as far as possible along each branch before backtracking. 34, 137

DOF Degrees of freedom is the number of independent parameters that define the displacement and deformation of the body. Like the shoulder joint has three rotation DOF while the knee has only one rotation DOF. A car has two translation DOF (normally it does not fly). 16, 17, 19, 20, 22–24, 35, 36, 86, 99, 101, 104, 123

Fps Frames Per Second (fps) is the frequency (rate) at which an imaging device produces unique consecutive images called frames. The term applies equally well to computer graphics, video cameras, film cameras, and motion capture systems. xx, 74–76, 96

FSM Finite-State Machine is a mathematical model used to design computer programs and digital logic circuits. It is conceived as an abstract machine that can be in one of a finite number of states. The machine is in only one state at a time; the state it is in at any given time is called the current state. It can change from one state to another when initiated by a triggering event or condition, this is called a transition. A particular FSM is defined by a list of the possible transition states from each current state, and the triggering condition for each transition. 21, 25, 36, 37

Genetic algorithm is a search heuristic that mimics the process of natural evolution. This heuristic is routinely used to generate useful solutions to optimization and search problems, using techniques inspired by natural evolution, such as inheritance, mutation, selection, and crossover 26, 30

GLSL OpenGL Shading Language is a high-level shading language based on the syntax of the C programming language. It was created by the OpenGL ARB to give developers more direct control of the graphics pipeline without having to use assembly language or hardware-specific languages. 4

IK Inverse Kinematics problem is simply stated as: given the desired position of the skeleton's hand, what must be the angles at all of the skeleton's joints? xvi, 11, 14, 16–19, 29–31, 44–46, 66, 67, 71, 74, 77, 93, 97, 122

IPM Inverted Pendulum Model is a pendulum which has its mass above its pivot point. It is used in dynamic animation systems as an approximation of the whole articulated body when it is supported on one leg. 24–26, 29, 77

Keyframe They are poses in animation, manually generated, that defines the starting and ending points of any smooth transition. They are called "frames" because their position in time is measured in frames. A sequence of keyframes defines which movement the viewer will see, whereas the position of the keyframes in an animation defines the timing of the movement. Because only two or three keyframes over the span of a second do not create the illusion of movement, the remaining frames are filled with in-betweens. xii, 8, 10, 12, 18, 23, 27, 31, 77, 112, 129

LOD Level of details in computer graphics, involves decreasing the complexity of a 3D object representation as it moves away from the viewer or according other metrics such as object importance, eye-space speed or position. This basic concept can be generalized into other domains like animations, physics, *etc.* xix, 43, 70, 71, 73–75, 96, 122

Mesh A polygon mesh or unstructured grid is a collection of vertices, edges and faces that defines the shape of a polyhedral object in 3D computer graphics and solid modeling. The faces usually consist of triangles, quadrilaterals or other simple convex polygons, since this simplifies rendering, but may also be composed of more general concave polygons, or polygons with holes. 7, 8, 18, 45, 46, 97, 104, 116

Motion blending produces new motions (blends) by combining multiple clips according to time-varying weights. xii, 10, 19

Motion warping Motion Warping consists of modifying a single motion in order to fit a new trajectory. xii, 12, 13

MRAC Model Reference Adaptive Control is a closed loop controller with parameters that can be updated to change the response of the system. The output of the system is compared to a desired response from a reference model. The control parameters are update based on this error. The goal is for the parameters to converge to ideal values that cause the plant response to match the response of the reference model. 104, 112

OpenGL OpenGL (Open Graphics Library) is a standard specification defining a cross-language, cross-platform API for writing applications that produce 2D and 3D computer graphics. 4

PD Proportional-Derivative is a generic control loop feedback mechanism (controller) and it is the most commonly used feedback controller. A PD controller calculates an error value as the difference between a measured process variable and a desired setpoint. The controller attempts to minimize the error by adjusting the process control outputs. The response of the controller can be described in terms of the responsiveness of the controller to an error, the degree to which the controller overshoots the setpoint and the degree of system oscillation. 21, 26–28, 30, 104, 112, 133

PPP Probabilistic Path Planning. 35

QP Quadratic programming is a special type of mathematical optimization problem. It is the problem of optimizing (minimizing or maximizing) a quadratic function of several variables subject to linear constraints on these variables. 23, 30, 75

Rasterize is the task of taking an image described in a vector graphics format (shapes) and converting it into a raster image (pixels or dots) for output on a video display or printer, or for storage in a bitmap file format. 59

Retargeting is the process of adapting an existing motion data to a new morphology, the motion data can be captured or generated (MoCap or keyframe respectively.) xii, 10–12, 31

RL Reinforcement learning is a trial and error learning that learns froms directily interacting with an environment. The system learns from the consequences of its actions, rather than from being explicitly taught. It selects its actions based on its past experiences (exploitation) and new choices (exploration). 13, 26, 35, 36

RPP Randomized Path Planning. 35, 36

Secondary motion are passive motions generated in response to the movements of characters and other objects or environmental forces [HOZ97]. Secondary motions are not normally the main focus of an animated scene, yet their absence can distract or disturb the viewer, destroying the illusion of reality created by the scene. 4, 5, 19, 78, 103, 113, 117–119, 123

SIMBICON SIMple BIped CONtrol. 26, 77

Skinning Every vertex in a Polygon Mesh is tied to at least one joint through an influence. An influence stores the vertex, the joint index and a weight which specify how much influence the joint has over the vertex. A vertex can have several influences but only one for each joint. The sum of all the weights in a vertex' influences should always be 1.0. So by animating the joints the vertex follows in a logical way, and the actual mesh is animated. 7, 104

SLIP Spring-Loaded Inverted Pendulum generalizes the IPM by replacing the fixed length leg with a spring, thereby capturing energy storage and release during running. 25, 26

Appendices

Appendix A

PD Controller

Many systems use PD controllers in order to calculate the needed torques to be applied in the joints. A PD controller is a feedback control mechanism that calculates this torque based on the following equation:

$$\tau = K_d(\theta_G - \theta_C) + K_p(\dot{\theta}_G - \dot{\theta}_C) \tag{A.1}$$

(K_d, K_p) are the propotional/derivative constants. $(\theta_G, \dot{\theta}_G)$ goal (needed) angle/angular velocity. $(\theta_C, \dot{\theta}_C)$ current angle/angular velocity. τ is the calculated torque. So on each simulation step, the PD controller calculates the error between the current and goal values, then it outputs a torque based on this error and the constants (K_d, K_p) , shown in Figure A.1. So the controller adapts itself based on the feedback of the simulation. The final calculated torque depends heavily on the constants (K_d, K_p) that are normally manually tweaked: the PD can be *High Gain* (rigid, respond faster to error but less stable [1]) or *Low Gain* (soft, respond slower to error but more stable).

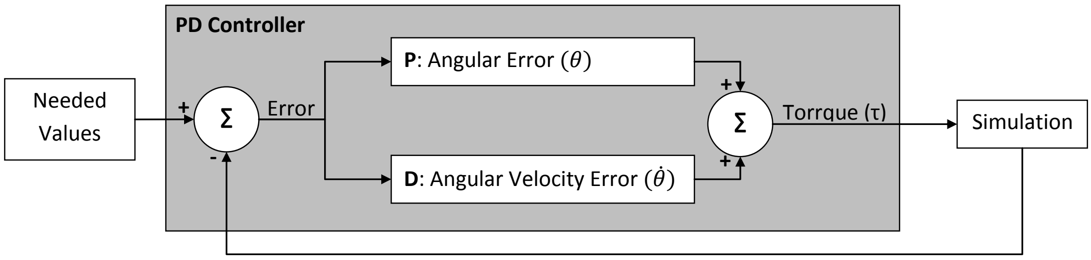


Figure A.1: *PD controller loop.*

---

[1] [TGTL11] propose a stable high gain PD controller

Appendix B

Physics Engines Responses

Figure B.1 show some of the results we obtained from experimenting with State-of-The-Art commercial and open-source physics engines. Even when simulating real world physics, we can notice a huge disparity between the generated results after first collision of identical spheres and cubes with identical starting status. Even a completely different behavior of a chain of rigid bodies connected to a fixed point in space. Which one is the ground truth?

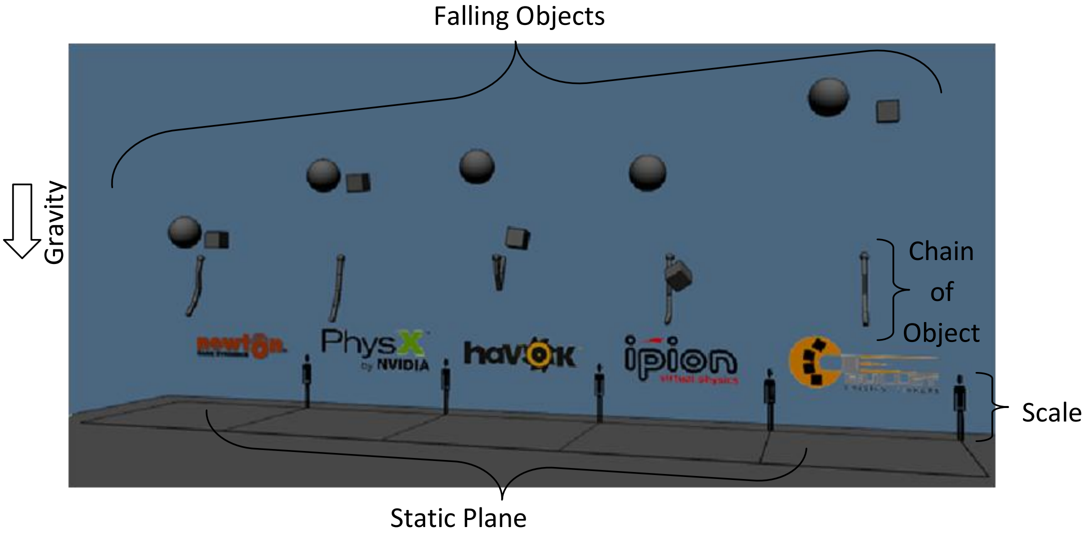

Falling Objects
Gravity
Chain of Object
Scale
Static Plane
newton
PhysX by NVIDIA
havok
ipion
virtual physics

Figure B.1: *Example of physics engines responses (commercial and open-source). Left to right: Newton Dynamics, PhysX by NVidia, Havok, IPION and Bullet Physics. A snapshot of the simulation after first collision of identical spheres and cubes with identical starting status. And final position of a chain of rigid bodies connected to a fixed point in space after several simulation steps.*

Appendix C

Depth-First vs Breadth-First Search

Depth-First Search (DFS) is an algorithm for naively traversing or searching a tree, tree structure, or graph. Starting from the root, this algorithm explores as far as possible along each branch before backtracking, as shown in Figure C.1.

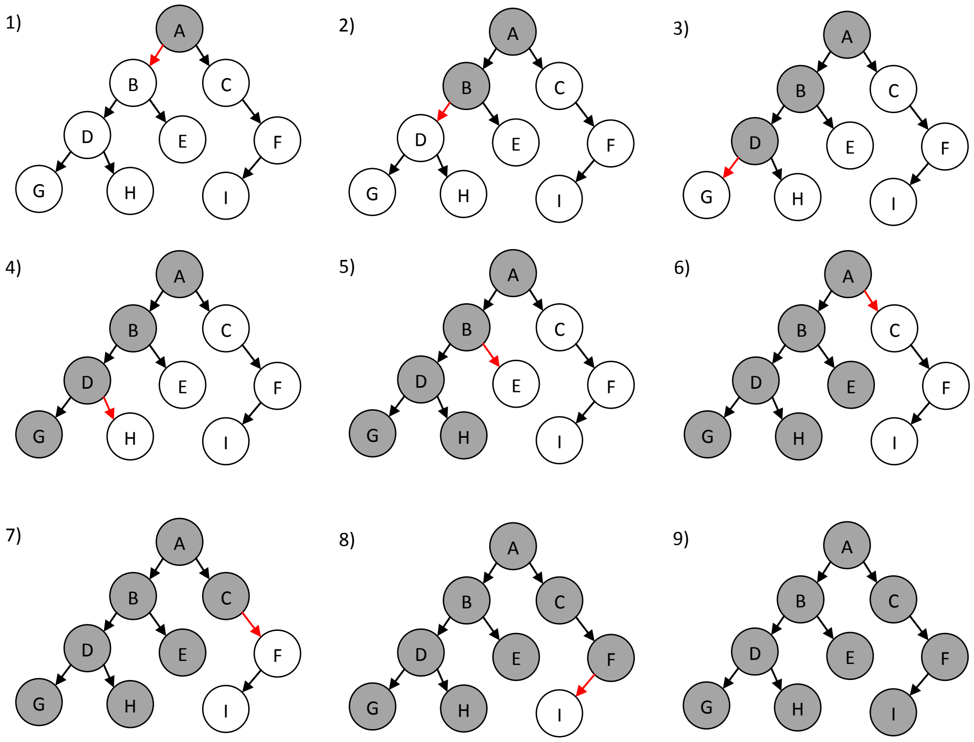


Figure C.1: *Depth-First Search.*

While Breadth-First Search (BFS) begins at the root node and explores all the neighboring nodes. Then for each of those nearest nodes, it explores their unexplored neighbor nodes, and so on, until it finds the goal, as shown in Figure C.2.

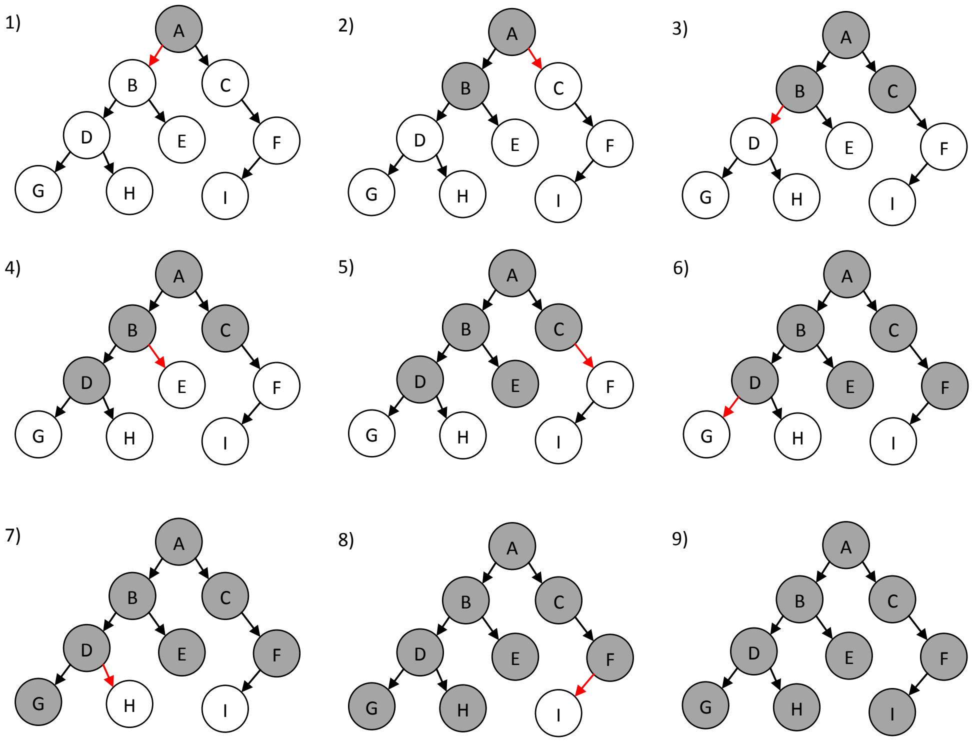


Figure C.2: *Breadth-First Search.*

Appendix D

Locomotion System Parameters

The presented system in Chapter 3 has the following parameters that the user can edit and set in real-time:

	Parameter Name	Unit	Count	Type	Description
	Pelvis Speed	m/s	1	**M**	The needed pelvis speed.
	Step Height	m	1	**O**	The needed step height.
	Orientation	$radians$	1	**O**	The needed orientation.
Per Foot	Gait	%	2	**M**	The start of the foot *swing* phase and its duration, represented as a percentage of the whole cycle.
	Relative Position	$Vec2D$	2	**M**	The needed relative position of the foot, represented as a 2D vector on the horizontal plane (Figure 3.7).

Table D.1: A detailed description of the locomotion system parameters. **M** designate mandatory parameters. **O** designate optional parameters.

Based on Table D.1, a quadruped has $1 + 4 \times 2 + 4 \times 2 = 17$ parameters that are essential to describe its locomotion animation style. By fixing one parameter like the pelvis speed, other parameters can be optimized in order to find the best gait and feet spacing to accommodate for this needed speed. And the feet forces described in Section 4.1 can be used as a fitness function.

Appendix E

Main Algorithm

The goal of the algorithm is to find the best **couple** (target-trajectory) in the search space. The search space contains all the possible trajectories that go from the starting point toward all the possible targets. The algorithm uses partial scores to limit the number of explored solutions.

Algorithm:

```
$C$ is an abbreviation for Current
$B$ is an abbreviation for Best
$S$ is an abbreviation for Score
$H$ is an abbreviation for Height
$StartH$ = Starting Position elevation
- Declare: $BPath3D$, $BPath3DS$, $BTarget$, $BTargetS$
for all Sorted Targets do
  - Get $CurTarget$ = Target that we want to test
  if $CurTargetS < BTargetS \times BPath3DS$ then
    Stop The loop, best pair (Target,3D Trajectory) is already found
  end if
  $TargetH = CurTarget$ Elevation
  for all Heights between $StartH$ & $TargetH$ by slices of $10cm$ do
    - Add all the obstacles with elevation higher than the needed height as
    forbidden cell in the obstacles' map then
    - 2D path planning toward this target
    - $ScoreC2D = CurTargetS \times CurPath2DS$
    - $ScoreB = BTargetS \times BPath3DS$
    if $ScoreC2D > ScoreB$ then
      - 3D path planning toward this target
      - $ScoreC3D = CurTargetS \times CurPath3DS$
      if $ScoreC3D > ScoreB$ then
        - Set all the $B$ variables: $BPath3D$, $BPath3DS$, $BTarget$, $BTargetS$ from
        the current values
      end if
    end if
  end for
end for
```

Appendix F

Hermite Curve

Hermite Curve is a third-degree spline with each polynomial of the spline in Hermite form. The Hermite form consists of two control points and two control tangents for each polynomial. Hermite Curves, Figure F.1, are used to smoothly interpolate data between key-points (like object movement in keyframe animation or camera control), and in our case to smoothly connect 3D waypoints producing our 3D trajectories. To calculate a hermite curve, the following vectors are needed:

- $\overrightarrow{P1}$: the start waypoint of the curve.
- $\overrightarrow{T1}$: the tangent (*e.g.* direction and speed) how the curve lefts the start waypoints.
- $\overrightarrow{P2}$: the end waypoint of the curve.
- $\overrightarrow{T2}$: the tangent (*e.g.* direction and speed) how the curves enters the end waypoint.

To calculate each point on the curve P, the following equation is used:

$$P = S \times H \times C \tag{F.1}$$

S is the Step Vector and it is given as follow:

$$S = \begin{bmatrix} s^3 \\ s^2 \\ s^1 \\ 1 \end{bmatrix} \; where\ s = [0..1] \tag{F.2}$$

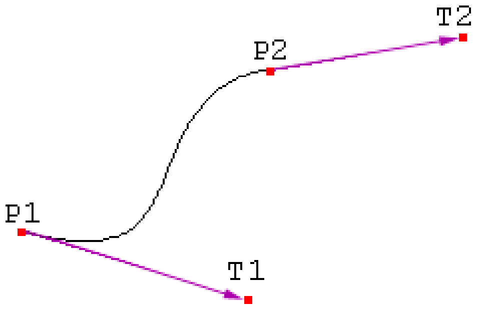


Figure F.1: *Example of a Hermite curve.*

C is our Control Points Vector:

$$C = \begin{bmatrix} \overrightarrow{P1} \\ \overrightarrow{P2} \\ \overrightarrow{T1} \\ \overrightarrow{T2} \end{bmatrix} \tag{F.3}$$

And finally H represents the hermite polynomials, it is a matrix of constants that defines the Hermite Curve behavior:

$$H = \begin{bmatrix} +2 & -2 & +1 & +1 \\ -3 & +3 & -2 & -1 \\ +0 & +0 & +1 & +0 \\ +1 & +0 & +0 & +0 \end{bmatrix} \tag{F.4}$$

To automatically calculate the previous tangents ($\overrightarrow{T1}$, $\overrightarrow{T2}$), a subset of Hermite Curves called Cardinal splines can be used.

$$\overrightarrow{T_i} = a \times (\overrightarrow{P_{i+1}} - \overrightarrow{P_{i-1}}) \ where \ a = [0..1] \tag{F.5}$$

References

[ACSF07] Brian Allen, Derek Chu, Ari Shapiro, and Petros Faloutsos. On the beat!: timing and tension for dynamic characters. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*, SCA '07, pages 239–247, Aire-la-Ville, Switzerland, Switzerland, 2007. Eurographics Association. 112

[AdSP07] Yeuhi Abe, Marco da Silva, and Jovan Popović. Multiobjective control with frictional contacts. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*, SCA '07, pages 249–258, Aire-la-Ville, Switzerland, Switzerland, 2007. Eurographics Association. 23

[AF02] Okan Arikan and D. A. Forsyth. Interactive motion generation from examples. *ACM Trans. Graph.*, 21:483–490, July 2002. 12

[AFO05] Okan Arikan, David A. Forsyth, and James F. O'Brien. Pushing people around. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, SCA '05, pages 59–66, New York, NY, USA, 2005. ACM. 13, 29

[AKGM$^+$11] Ahmad Abdul Karim, Thibaut Gaudin, Alexandre Meyer, Axel Buendia, and Saida Bouakaz. Transactions on edutainment vi. chapter Adding physical like reaction effects to skeleton-based animations using controllable pendulums, pages 111–121. Springer-Verlag, Berlin, Heidelberg, 2011. 28, 119

[AKGM$^+$12] Ahmad Abdul Karim, Thibaut Gaudin, Alexandre Meyer, Axel Buendia, and Saida Bouakaz. Computer animation and virtual worlds. chapter Procedural Locomotion of Multi-Legged Characters in Dynamic Environments. 2012. 78

[Ale96] R.M.N. Alexander. *Optima for animals*. Princeton paperbacks. Princeton University Press, 1996. 57, 62, 83, 86, 94, 121

[Ale03] R.M.N. Alexander. *Principles of animal locomotion*. Princeton University Press, 2003. 57, 62, 83, 86, 94

[ALP04] Yeuhi Abe, C. Karen Liu, and Zoran Popović. Momentum-based parameterization of dynamic character motion. In *Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, SCA '04, pages 173–182, Aire-la-Ville, Switzerland, Switzerland, 2004. Eurographics Association. 29

[AMG$^+$12] Ahmad Abdul Karim, Alexandre Meyer, Thibaut Gaudin, Axel Buendia, and Saida Bouakaz. Generic Spine Model with Simple Physics for Life-Like Quadrupeds and Reptiles. In *VRIPHYS 2012: 9th Workshop on Virtual Reality Interaction and Physical Simulation*, December 2012. 99

[AP06] Yeuhi Abe and Jovan Popović. Interactive animation of dynamic manipulation. In *Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation*, SCA '06, pages 195–204, Aire-la-Ville, Switzerland, Switzerland, 2006. Eurographics Association. 29

[BA79] Donna C. Boone and Stanley P. Azen. Normal range of motion of joints in male subjects. *The Journal of Bone and Joint Surgery*, 61:756–759, 1979. 19

[Bar96] David Baraff. Linear-time dynamics using lagrange multipliers. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, SIGGRAPH '96, pages 137–146, New York, NY, USA, 1996. ACM. 20

[Bar97] Ronen Barzel. Faking dynamics of ropes and springs. *IEEE Comput. Graph. Appl.*, 17:31–39, May 1997. 39

[Ber09] Alain Berthoz. *La simplexité*. Odile Jacob, 2009. 30, 50

[BHW96] Ronen Barzel, John F. Hughes, and Daniel N. Wood. Plausible motion simulation for computer graphics animation. In *Proceedings of the Eurographics workshop on Computer animation and simulation '96*, pages 183–197, New York, NY, USA, 1996. Springer-Verlag New York, Inc. xiii, 39, 102

[BK96] Norman I. Badler and Hyeongseok Ko. Animating human locomotion with inverse dynamics. *IEEE Comput. Graph. Appl.*, 16:50–59, March 1996. 29

[Bla05] R.W. Blake. *Efficiency And Economy in Animal Physiology*. Cambridge environmental chemistry series. Cambridge University Press, 2005. 72

[BMJC01] Vincent Bonnafous, Eric Menou, Jean-Pierre Jessel, and René Caubet. Co-operative and concurrent blending motion generators. In *WSCG (Short Papers)*, pages 130–, 2001. xii, 10

[BMtT90] Ronan Boulic, Nadia Magnenat-thalmann, and Daniel Thalmann. A global human walking model with real-time kinematic personification. *The Visual Computer*, 6:344–358, 1990. 15

[BPP07] Philippe Beaudoin, Michiel Van De Panne, and Pierre Poulin. Automatic construction of compact motion graphs, 2007. 13

[BR01] John E. A. Bertram and Andy Ruina. Multiple walking speed frequency relations are predicted by constrained optimization. *Journal of Theoretical Biology*, 2001. 15

[Bre65] J. E. Bresenham. Algorithm for computer control of a digital plotter. *IBM Syst. J.*, 4:25–30, March 1965. 55, 59

[BSG$^+$07] Reinhard Blickhan, Andre Seyfarth, Hartmut Geyer, Sten Grimmer, Heiko Wagner, and Michael Günther. Intelligence by mechanics. *Hilosophical Transactions of the Royal Society - Series A: Mathematical, Physical and Engineering Sciences*, 365, 2007. 25

[BUT04] Ronan Boulic, Branislav Ulicny, and Daniel Thalmann. Versatile walk engine. *Journal of Game Development*, 1:2004, 2004. xiii, 14

[CBvdP09] Stelian Coros, Philippe Beaudoin, and Michiel van de Panne. Robust task-based control policies for physics-based characters. In *ACM SIGGRAPH Asia 2009 papers*, SIGGRAPH Asia '09, pages 170:1–170:9, New York, NY, USA, 2009. ACM. 26, 77

[CBvdP10] Stelian Coros, Philippe Beaudoin, and Michiel van de Panne. Generalized biped walking control. In *ACM SIGGRAPH 2010 papers*, SIGGRAPH, pages 130:1–130:9, New York, NY, USA, 2010. ACM. xiii, 26, 77

[CBYvdP08] Stelian Coros, Philippe Beaudoin, Kang Kang Yin, and Michiel van de Pann. Synthesis of constrained walking skills. *ACM Trans. Graph.*, 27:113:1–113:9, December 2008. 26

[CC10] N. Courty and A. Cuzol. Conditional stochastic simulation for character animation. *Comput. Animat. Virtual Worlds*, 21:443–452, May 2010. xii, 11

[CGC$^+$02] Steve Capell, Seth Green, Brian Curless, Tom Duchamp, and Zoran Popović. Interactive skeleton-driven dynamic deformations. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '02, pages 586–593, New York, NY, USA, 2002. ACM. 104

[CHP89] J. E. Chadwick, D. R. Haumann, and R. E. Parent. Layered construction for deformable animated characters. In *Proceedings of the 16th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '89, pages 243–252, New York, NY, USA, 1989. ACM. 7

[CKHL11] Myung Geol Choi, Manmyung Kim, Kyunglyul Hyun, and Jehee Lee. Deformable motion: Squeezing into cluttered environments. *Comput. Graph. Forum*, 30(2):445–453, 2011. xii, 12

[CKJ$^+$11] Stelian Coros, Andrej Karpathy, Ben Jones, Lionel Reveret, and Michiel van de Panne. Locomotion skills for simulated quadrupeds. In *ACM SIGGRAPH 2011 papers*, SIGGRAPH '11, pages 59:1–59:12, New York, NY, USA, 2011. ACM. 14, 15, 30, 31, 32, 75, 78, 87

[CLC$^+$05] Joel Chestnutt, Manfred Lau, German Cheung, James Kuffner, Jessica Hodgins, and Takeo Kanade. Footstep planning for the honda asimo humanoid. In *in Proceedings of the IEEE International Conference on Robotics and Automation*, 2005. 35

[CLS03] Min Gyu Choi, Jehee Lee, and Sung Yong Shin. Planning biped locomotion using motion capture data and probabilistic roadmaps. *ACM Trans. Graph.*, 22:182–203, April 2003. 35

[CR06] Grégori Clauzel and Lionel Revéret. Animation 3d en temps-réel de quadrupèdes par simulation physique rapport. Master's thesis, INRIA Rhône-Alpes, 2006. 30, 32

[DAJ11] Damien Djaouti, Julian Alvarez, and Jean-Pierre Jessel. Classifying Serious Games: The G/P/S Model. In Patrick Felicia, editor, *Handbook of Research on Improving Learning and Motivation through Educational Games: Multidisciplinary Approaches*, chapter 6, pages 118–136. IGI Global, http://www.igi-global.com, 2011. x, 1

[Dij59] E. W. Dijkstra. A note on two problems in connexion with graphs. *NUMERISCHE MATHEMATIK*, 1(1):269–271, 1959. 34

[dL11] Martin de Lasa. *Feature-Based Control Of Physics-Based Character Animation*. PhD thesis, 2011. xiii, 23

[dLH09] Martin de Lasa and Aaron Hertzmann. Prioritized optimization for task-space control. In *Proceedings of the 2009 IEEE/RSJ international conference on Intelligent robots and systems*, IROS, Piscataway, NJ, USA, 2009. IEEE Press. 23

[dLMH10] Martin de Lasa, Igor Mordatch, and Aaron Hertzmann. Feature-based locomotion controllers. *ACM Trans. Graph.*, 29(4):131:1–131:10, July 2010. xiii, 23, 38, 75

[dSAP08] Marco da Silva, Yeuhi Abe, and Jovan Popović. Interactive simulation of stylized human locomotion. In *ACM SIGGRAPH 2008 papers*, SIGGRAPH '08, pages 82:1–82:10, New York, NY, USA, 2008. ACM. 29

[DSP06] Kevin G. Der, Robert W. Sumner, and Jovan Popović. Inverse kinematics for reduced deformable models. In *ACM SIGGRAPH 2006 Papers*, SIGGRAPH '06, pages 1174–1179, New York, NY, USA, 2006. ACM. 18

[Fau99] Francois Faure. Fast refinable equation solution for articulated solid dynamics. *IEEE TRANSACTIONS ON VISUALISATION AND COMPUTER GRAPHICS, VOLUME 5, NUMBER*, 3:268–276, 1999. 20

[FDCM97] François Faure, Gilles Debunne, Marie-Paule Cani, and Franck Multon. Dynamic analysis of human walking. In *8th Eurographics Workshop on Computer Animation and Simulation*, Sep 1997. 22

[Fea87] Roy Featherstone. *Robot Dynamics Algorithm*. Kluwer Academic Publishers, Norwell, MA, USA, 1987. Manufactured By-Publishers, Kluwer Academic. 20

[Fea99a] Roy Featherstone. A divide-and-conquer articulated body algorithm for parallel o(log(n)) calculation of rigid body dynamics. part 1: Basic algorithm. 1999. 20, 110

[Fea99b] Roy Featherstone. A divide-and-conquer articulated-body algorithm for parallel o(log(n)) calculation of rigid-body dynamics. part 2: Trees, loops,& accuracy. 1999. 20, 110

[FK99] R.J. Full and D.E. Koditschek. Templates and anchors neuromechanical hypotheses of legged locomotion on land. *The Journal of Experimental Biology*, 1999. 25

[FP03] Anthony C. Fang and Nancy S. Pollard. Efficient synthesis of physically valid human motion. *ACM Transactions on Graphics (SIGGRAPH 2003)*, 22(3):417–426, July 2003. 23

[FRDC04] Laurent Favreau, Lionel Reveret, Christine Depraz, and Marie-Paule Cani. Animal gaits from video. In *ACM SIGGRAPH / Eurographics Symposium on Computer Animation, SCA*, Grenoble, France, 2004. 30

[GB08] James Gain and Dominique Bechmann. A survey of spatial deformation from a user-centered perspective. *ACM Trans. Graph.*, 27:107:1–107:21, November 2008. 7

[Gir87] Michael Girard. Interactive design of 3-d computer-animated legged animal motion. In *Proceedings of the 1986 workshop on Interactive 3D graphics*, I3D '86, pages 131–150, New York, NY, USA, 1987. ACM. xiii, 15, 17, 30, 31, 81

[GL98] Michael Gleicher and Peter Litwinowicz. Constraint-based motion adaptation, 1998. 11

[Gle97] Michael Gleicher. Motion editing with spacetime constraints, apr 1997. xii, 11

[Gle98] Michael Gleicher. Retargetting motion to new characters. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '98, pages 33–42, New York, NY, USA, 1998. ACM. 11

[Gle01] Michael Gleicher. Comparing constraint-based motion editing methods, mar 2001. xii, 12, 13

[GM85] Michael Girard and Anthony A. Maciejewski. Computational modeling for the computer animation of legged figures. In *Proceedings of the 12th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '85, pages 263–270, New York, NY, USA, 1985. ACM. xiii, 15, 17, 30, 31, 81

[GMHP04] Keith Grochow, Steven L. Martin, Aaron Hertzmann, and Zoran Popović. Style-based inverse kinematics. *ACM Trans. Graph.*, 23:522–531, August 2004. 18

[GRVT06] Alejandra García Rojas, Frédéric Vexo, and Daniel Thalmann. Individualized reaction movements for virtual humans. In *Proceedings of the 4th international conference on Computer graphics and interactive techniques in Australasia and Southeast Asia*, GRAPHITE '06, pages 79–85, New York, NY, USA, 2006. ACM. 28

[GSKJ03] Michael Gleicher, Hyun Joon Shin, Lucas Kovar, and Andrew Jepsen. Snap-together motion: assembling run-time animations. *ACM Trans. Graph.*, 22:702–702, July 2003. 12, 13

[HdSP07] Eugene Hsu, Marco da Silva, and Jovan Popović. Guided time warping for motion editing. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*, SCA '07, pages 45–52, Aire-la-Ville, Switzerland, Switzerland, 2007. Eurographics Association. 11

[HG07] Rachel Heck and Michael Gleicher. Parametric motion graphs. In *Proceedings of the 2007 symposium on Interactive 3D graphics and games*, I3D '07, pages 129–136, New York, NY, USA, 2007. ACM. 12

[HKG06] Rachel Heck, Lucas Kovar, and Micheal Gleicher. Splicing upper-body actions with locomotion. In *Eurographics*, 2006. xii, 10

[HKT10] Edmond S. L. Ho, Taku Komura, and Chiew-Lan Tai. Spatial relationship preserving character motion adaptation. *ACM Trans. Graph.*, 29(4):33:1–33:8, July 2010. 11

[HNR68] Peter Hart, Nils Nilsson, and Bertram Raphael. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, February 1968. 34

[Hod91] Jessica k. Hodgins. Biped gait transitions. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 1991. 15

[HOZ97] Jessica K. Hodgins, James F. O'Brien, and Victor Brian Zordan. Combining active and passive simulations for secondary motion. In *ACM SIGGRAPH 97 Visual Proceedings: The art and interdisciplinary programs of SIGGRAPH '97*, SIGGRAPH '97, pages 168–, New York, NY, USA, 1997. ACM. xxvi, 5, 129

[HRE+08] Chris Hecker, Bernd Raabe, Ryan W. Enslow, John DeWeese, Jordan Maynard, and Kees van Prooijen. Real-time motion retargeting to highly varied user-created morphologies. *ACM Trans. Graph.*, 27:27:1–27:11, August 2008. 30, 31, 77

[HWBO95] Jessica K. Hodgins, Wayne L. Wooten, David C. Brogan, and James F. O'Brien. Animating human athletics. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, SIGGRAPH '95, pages 71–78, New York, NY, USA, 1995. ACM. 21

[HWC00] Donald F. Hoyt, Steven J. Wickler, and Edward A. Cogger. Time of contact and step length the effect of limb length, running speed, load carrying and incline. *The Journal of Experimental Biology*, 2000. 15

[IAF05] Leslie Ikemoto, Okan Arikan, and David Forsyth. Learning to move autonomously in a hostile world. In *ACM SIGGRAPH 2005 Sketches*, SIGGRAPH '05, New York, NY, USA, 2005. ACM. 37

[IAF06] Leslie Ikemoto, Okan Arikan, and David Forsyth. Knowing when to put your foot down. In *Proceedings of the 2006 symposium on Interactive 3D graphics and games*, I3D '06, pages 49–53, New York, NY, USA, 2006. ACM. 14

[INM66] VERNE T. INMAN. Human locomotion. *Canadian Medical Association*, 1966. xiv, xxi, 15, 42, 80, 81, 121

[Joh09] Rune Skovbo Johansen. *Dynamic Walking With Semi-Procedural Animation*. PhD thesis, 2009. 77

[JYL09] Sumit Jain, Yuting Ye, and C. Karen Liu. Optimization-based interactive motion synthesis. *ACM Trans. Graph.*, 28:10:1–10:12, February 2009. 23, 76

[Kal08] Marcelo Kallmann. Analytical inverse kinematics with body posture control. *Comput. Animat. Virtual Worlds*, 19:79–91, May 2008. 18

[KDR05] Arthur D. Kuo, J. Maxwell Donelan, and Andy Ruina. Energetic consequences of walking like an inverted pendulum: Step-to-step transitions. *American College of Sports Medicine*, 33, April 2005. 25

[KG04] Lucas Kovar and Michael Gleicher. Automated extraction and parameterization of motions in large data sets. In *ACM SIGGRAPH*

2004 Papers, SIGGRAPH '04, pages 559–568, New York, NY, USA, 2004. ACM. xii, 11, 18, 19

[KGP02] Lucas Kovar, Michael Gleicher, and Frédéric Pighin. Motion graphs. *ACM Trans. Graph.*, 21:473–482, July 2002. 12, 13

[KH10] Taesoo Kwon and Jessica Hodgins. Control systems for human running using an inverted pendulum model and a reference motion capture sequence. In *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '10, pages 129–138, Aire-la-Ville, Switzerland, Switzerland, 2010. Eurographics Association. 29

[Kha86] O Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *Int. J. Rob. Res.*, 5:90–98, April 1986. xviii, 34, 55, 57

[KHKL09] Manmyung Kim, Kyunglyul Hyun, Jongmin Kim, and Jehee Lee. Synchronized multi-character motion editing. In *ACM SIGGRAPH 2009 papers*, SIGGRAPH '09, pages 79:1–79:9, New York, NY, USA, 2009. ACM. xii, 12

[KKI02] Shunsuke Kudoh, Taku Komura, and Katsushi Ikeuchi. The dynamic postural adjustment with the quadratic programming method. In *In International Conference on Intelligent Robots and Systems*, pages 2563–2568, 2002. xiii, 23, 38

[KKI06] Shunsuke Kudoh, Taku Komura, and Katsushi Ikeuchi. Stepping motion for a human-like character to maintain balance against large perturbations. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA 2006)*, pages 2661–2666. IEEE, 2006. 23, 24, 38

[KKK$^+$01a] S Kajita, F Kanehiro, K Kaneko, K Yokoi, and H Hirukawa. The 3d linear inverted pendulum model: a simple modeling for a biped walking pattern generation. *Proceedings 2001 IEEERSJ International Conference on Intelligent Robots and Systems Expanding the Societal Role of Robotics in the the Next Millennium Cat No01CH37180*, 1(4):239–246, 2001. 24

[KKK$^+$01b] James Kuffner, Jr. Koichi, Nishiwaki Satoshi Kagami, Masayuki Inaba, and Hirochika Inoue. Footstep planning among obstacles for biped robots. In *Proc. of 2001 IEEE Intl. Conf. on Intelligent Robots and Systems (IROS*, pages 500–505, 2001. 35

[KKK$^+$02] Shuuji Kajita, Fumio Kanehiro, Kenji Kaneko, Kiyoshi Fujiwara, Kazuhito Yokoi, and Hirohisa Hirukawa. A realtime pattern generator for biped walking. In *ICRA*, pages 31–37, 2002. xiii, 24

[KKK$^+$03] Shuuji Kajita, Fumio Kanehiro, Kenji Kaneko, Kiyoshi Fujiwara, and Kensuke Harada Kazuhito Yokoi. Biped walking pattern generation by using preview control of zero-moment point. In *in Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1620–1626, 2003. 24

[KKN$^+$03] James Kuffner, Satoshi Kagami, Koichi Nishiwaki, Masayuki Inaba, and Hirochika Inoue. Online footstep planning for humanoid robots. In *in Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA*, pages 932–937, 2003. 35

[KL08] Paul Kanyuk and Chris Lawrence. Brain springs: fast physics for large crowds on wall e. In *ACM SIGGRAPH 2008 talks*, SIGGRAPH, pages 26:1–26:1, New York, NY, USA, 2008. ACM. 104

[KMA05] R. Kulpa, F. Multon, and B. Arnaldi. Morphology-independent representation of motions for interactive human-like animation. *Computer Graphics Forum, Eurographics 2005 special issue*, 24:343–352, 2005. 12

[KMB96] Evangelos Kokkevis, Dimitri Metaxas, and Norman I. Badler. User-controlled physics-based animation for articulated figures. In *Proceedings of the Computer Animation*, CA '96, pages 16–, Washington, DC, USA, 1996. IEEE Computer Society. 112

[Kok04] E. Kokkevis. Practical physics for articulated characters. In *Proceedings of Game Developers Conference*, 2004. 20

[KRFC09] Paul Kry, Lionel Revéret, François Faure, and Marie-Paule Cani. Modal locomotion: animating virtual characters with natural vibrations. *Comput. Graph. Forum*, 28(2):289–298, 2009. Special Issue: Eurographics 2009. xiii, 27

[KSG02] Lucas Kovar, John Schreiner, and Michael Gleicher. Footskate cleanup for motion capture editing, 2002. 14

[Kuo01] Arthur D. Kuo. A simple model of bipedal walking predicts the preferred speed-step length relationship. *Journal of Biomechanical Engineering*, pages 264–269, 2001. 25

[Kuo02] Arthur D. Kuo. Energetics of actively powered locomotion using the simplest walking model. *Journal of Biomechanical Engineering*, 2002. 25

[KvdP00] Maciej Kalisiak and Michiel van de Panne. A grasp-based motion planning algorithm for character animation. In *Eurographics Workshop on Computer Animation and Simulation*, pages 43–58, 2000. 36

[Lam09] Fabrice Lamarche. TopoPlan: a topological path planner for real time human navigation under floor and ceiling constraints. *Computer Graphics Forum*, 28(2), March 2009. 33

[Lan79] Yoan D. Landau. *Adaptive control : the model reference approach*. Dekker, New York :, 1979. 112

[LF98] Cynthia R. Lee and Claire T. Farley. Determinants of the center of mass trajectory in human walking and running. *The Journal of Experimental Biology*, 1998. 22

[LK06] Manfred Lau and James J. Kuffner. Precomputed search trees: planning for interactive goal-driven animation. In *Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation*, SCA '06, pages 299–308, Aire-la-Ville, Switzerland, Switzerland, 2006. Eurographics Association. 37

[LKL10] Yoonsang Lee, Sungeun Kim, and Jehee Lee. Data-driven biped control. *ACM Trans. Graph.*, 29:129:1–129:8, July 2010. 13, 27

[LLKP11] Sergey Levine, Yongjoon Lee, Vladlen Koltun, and Zoran Popović. Space-time planning with parameterized locomotion controllers. *ACM Trans. Graph.*, 30:23:1–23:11, May 2011. 37

[LLL11] Thomas Lopez, Fabrice Lamarche, and Tsai-Yen Li. Space-time planning in dynamic environments with unknown evolution. In *Proceedings of the 4th international conference on Motion in Games*, MIG'11, pages 316–327, Berlin, Heidelberg, 2011. Springer-Verlag. 37

[LLP09] Yongjoon Lee, Seong Jae Lee, and Zoran Popović. Compact character controllers. In *ACM SIGGRAPH Asia 2009 papers*, SIGGRAPH Asia '09, pages 169:1–169:8, New York, NY, USA, 2009. ACM. 13, 14

[LP02] C. Karen Liu and Zoran Popović. Synthesis of complex dynamic character motion from simple animations. *ACM Trans. Graph.*, 21:408–416, July 2002. xiii, 23

[LS99] Jehee Lee and Sung Yong Shin. A hierarchical approach to interactive motion editing for human-like figures. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '99, pages 39–48, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co. 11

[Lue84] David G. Luenberger. *Linear and Nonlinear Programming*. Addison-Wesley, 1984. 18

[LvdPE96] Joseph Laszlo, Michiel van de Panne, and Fiu Eugene. Limit cycle control and its application to the animation of balancing and walking. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, SIGGRAPH '96, pages 155–162, New York, NY, USA, 1996. ACM. 21

[LvdPF97] Joseph F. Laszlo, Michiel van de Panne, and Eugene Fiume. Control of physically-based simulated walking. In *IMAGINA*, 1997. 21

[LZ08] Wan-Yen Lo and Matthias Zwicker. Real-time planning for parameterized human motion. In *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '08, pages 29–38, Aire-la-Ville, Switzerland, Switzerland, 2008. Eurographics Association. 37

[Mar93] Etienne-Jules Marey. Locomotion comparée chez les différents animaux, nouvelles applications de la chronophotographie. *La Nature*, pages 215–218, 1893. 96

[McC10] Luke McCann. What is motion capture? `http://lukemccann.wordpress.com/motion-capture/`, September 2010. 8

[MdLH10] Igor Mordatch, Martin de Lasa, and Aaron Hertzmann. Robust physics-based locomotion using low-dimensional planning. *ACM Trans. Graph.*, 29(4):71:1–71:8, July 2010. xiii, 24, 25, 38, 76

[MFCD99] Franck Multon, Laure France, Marie-Paule Cani, and Gilles Debunne. Computer animation of human walking: a survey. *Journal of Visualization and Computer Animation (JVCA)*, 10:39–54, 1999. Published under the name Marie-Paule Cani-Gascuel. xii, 7, 38

[MKHK08] Franck Multon, Richard Kulpa, Ludovic Hoyet, and Taku Komura. Motion in games. chapter From Motion Capture to Real-Time Character Animation, pages 72–81. Springer-Verlag, Berlin, Heidelberg, 2008. 11, 12

[MLPP09] Uldarico Muico, Yongjoon Lee, Jovan Popović, and Zoran Popović. Contact-aware nonlinear control of dynamic characters. In *ACM SIGGRAPH 2009 papers*, SIGGRAPH '09, pages 81:1–81:9, New York, NY, USA, 2009. ACM. 27

[MPP11] Uldarico Muico, Jovan Popović, and Zoran Popović. Composite control of physically simulated characters. *ACM Trans. Graph.*, 30:16:1–16:11, May 2011. 29

[MSJT08] Matthias Müller, Jos Stam, Doug James, and Nils Thürey. Real time physics: class notes. In *ACM SIGGRAPH 2008 classes*, SIGGRAPH, New York, NY, USA, 2008. ACM. 104, 105

[MT55] E. Muybridge and R. Taft. *The human figure in motion*. Dover pictorial archive series. Dover Publications, 1955. 15, 121

[MT98] Walter Maurel and Daniel Thalmann. Human shoulder modeling including scapulo-thoracic constraint and joint sinus cones. *Computers and Graphics*, 24:203–218, 1998. 19

[Muy57] E. Muybridge. *Animals in Motion*. Dover Publications, Inc., 1957. xxi, 15, 80, 81, 83, 86, 87, 121

[MZ90] Michael McKenna and David Zeltzer. Dynamic simulation of autonomous legged locomotion. In *Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '90, pages 29–38, New York, NY, USA, 1990. ACM. 20, 30

[MZH$^+$08] Ronald Metoyer, Victor Zordan, Benjamin Hermens, Chun-Chi Wu, and Marc Soriano. Psychologically inspired anticipation and dynamic response for impacts to the head and upper body. *IEEE Transactions on Visualization and Computer Graphics*, 14:173–185, January 2008. 28

[MZS09] Adriano Macchietto, Victor Zordan, and Christian R. Shelton. Momentum control for balance. *ACM Trans. Graph.*, 28:80:1–80:8, July 2009. 29

[Nat11] P. Nattharith. *Mobile Robot Navigation Using a Behavioural Control Strategy*. University of Newcastle upon Tyne, 2011. With University of Newcastle upon Tyne. School of Mechanical and Systems Engineering. 57

[NH04] Mark Nagurka and Shuguang Huang. A mass-spring-damper model of a bouncing ball. *American Control Conference, 2004. Proceedings of the 2004*, pages 499 – 504, June 2004. 104

[NKZ12] Rubens F. Nunes, Paul G. Kry, and Victor B. Zordan. Using natural vibrations to guide control for locomotion. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, I3D '12, pages 87–94, New York, NY, USA, 2012. ACM. xiii, 27

[NWB+10] Nam Nguyen, Nkenge Wheatland, David Brown, Brian Parise, C. Karen Liu, and Victor Zordan. Performance capture with physical interaction. In *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '10, pages 189–195, Aire-la-Ville, Switzerland, Switzerland, 2010. Eurographics Association. 28

[OM01] Masaki Oshita and Akifumi Makinouchi. A dynamic motion control technique for human-like articulated figures. *Comput. Graph. Forum*, 20(3), 2001. 11

[OM11] Masaki Oshita and Naoki Masaoka. Generating avoidance motion using motion graph. In *MIG'11*, pages 120–131, 2011. 13

[PH05] Marko B Popovic and Hugh Herr. Global motion control and support base planning. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2005. xiii, 23

[PHH04] Marko Popovic, Andreas Hofmann, and Herr Hugh. Angular momentum regulation during human walking: biomechanics and control. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2405–2411, 2004. 22

[PLB95] T. Pozzo, Y. Levik, and A. Berthoz. Head and trunk movements in the frontal plane during complex dynamic equilibrium tasks in humans. *Exp Brain Res*, 106(2):327–38, 1995. 22

[PP97] Y C Pai and J Patton. Center of mass velocity-position predictions for balance control. *Journal of Biomechanics*, 30(4):347–354, 1997. 22

[PSL02] Julien Pettre, Thierry Simeon, and Jean-Paul Laumond. Planning human walk in virtual environments. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and System*, volume 3, pages 3048–3053, lausanne, Switzerland, 2002. 36

[PT06] Jerry E. Pratt and Russ Tedrake. Velocity-based stability margins for fast bipedal walking. In *Fast Motions in Biomechanics and Robotics*, pages 299–324. Springer, 2006. xiii, 24

[PW99] Zoran Popović and Andrew Witkin. Physically based motion transformation. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '99, pages 11–20, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co. xii, 11, 12

[PZLM10] Jia Pan, Liangjun Zhang, Ming C. Lin, and Dinesh Manocha. A hybrid approach for simulating human motion in constrained environments. *Comput. Animat. Virtual Worlds*, 21:137–149, May 2010. 36

[Rai86] Marc H. Raibert. *Legged robots that balance*. Massachusetts Institute of Technology, Cambridge, MA, USA, 1986. 9

[Rai90] Marc H. Raibert. Trotting, pacing and bounding by a quadruped robot. *Journal of Biomechanics*, 23, 1990. 15

[Rat05] Ori Ratner. *The Inverted Pendulum Model In Multi-Agent Simulations*. PhD thesis, 2005. 24

[RBNP08] Marc Raibert, Kevin Blankespoor, Gabriel Nelson, and Rob Playter. Bigdog, the rough-terrain quadruped robot. *The 17th World Congress The International Federation of Automatic Control*, 2008. 31

[RCP07] John Rebula, Fabián Cañas, and Jerry Pratt. Learning capture points for humanoid push recovery. In *7th IEEE-RAS International Conference on Humanoid Robots*, 2007. 24

[RGL05] Stephane Redon, Nico Galoppo, and Ming C. Lin. Adaptive dynamics of articulated bodies. *ACM Transactions on Graphics (SIGGRAPH 2005)*, 24(3), 2005. 20, 71

[RH91] Marc H. Raibert and Jessica K. Hodgins. Animation of dynamic legged locomotion. *SIGGRAPH Comput. Graph.*, 25:349–358, July 1991. 21

[Rob06] Thomas Robert. *Analyse Biomécanique Du Maintien De L'équilibre Debout Suite À Une Accélération Transitoire De La Surface D'appui.* PhD thesis, 2006. 22

[RP07] Paul S. A. Reitsma and Nancy S. Pollard. Evaluating motion graphs for character animation. *ACM Trans. Graph.*, 26, October 2007. 12

[SA09] Benjamin Stephens and Christopher Atkeson. Modeling and control of periodic humanoid balance using the linear biped model. *The 9th IEEE-RAS International Conference on Humanoid Robots*, 2009. 24

[Sim94] Karl Sims. Evolving virtual creatures. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, SIGGRAPH '94, pages 15–22, New York, NY, USA, 1994. ACM. 30, 32

[SK00] WJ Schwind and DE Koditschek. Approximating the stance map of a 2-DOF monoped runner. *Journal of Nonlinear Science*, 10(5):533–568, 2000. 25

[SKF07] Ari Shapiro, Marcelo Kallmann, and Petros Faloutsos. Interactive motion correction and object manipulation. In *Symposium on Interactive 3D Games and Graphics*, I3D'07, 2007. 35

[SKG03] Hyun Joon Shin, Lucas Kovar, and Michael Gleicher. Physical touch-up of human motions. In *Proceedings of the 11th Pacific Conference on Computer Graphics and Applications*, PG '03, pages 194–, Washington, DC, USA, 2003. IEEE Computer Society. 11

[SKG05] Mankyu Sung, Lucas Kovar, and Michael Gleicher. Fast and accurate goal-directed motion synthesis for crowds. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, SCA '05, pages 291–300, New York, NY, USA, 2005. ACM. 13, 35

[SKRF11] Shawn Singh, Mubbasir Kapadia, Glenn Reinman, and Petros Faloutsos. Footstep navigation for dynamic crowds. In *Symposium on Interactive 3D Graphics and Games*, I3D '11, pages 203–203, New York, NY, USA, 2011. ACM. 29

[SLGS01] Hyun Joon Shin, Jehee Lee, Michael Gleicher, and Sung Yong Shin. Computer puppetry: An importance-based approach, apr 2001. 11

[SP05] Adnan Sulejmanpašić and Jovan Popović. Adaptation of performed ballistic motion. *ACM Trans. Graph.*, 24:165–179, January 2005. 29

[SR06] Manoj Srinivasan and Andy Ruina. Computer optimization of a minimal biped model discovers walking and running. *Nature*, 2006. 25

[SRH⁺08] Ljiljana Skrba, Lionel Reveret, Franck Hétroy, Marie-Paule Cani, and Carol O'Sullivan. Quadruped animation. In *Eurographics State-of-the-Art Report*, pages 1–17, Hersonissos, Creete, Greece, 2008. 1, 30, 87

[SRH⁺09] Ljiljana Skrba, Lionel Reveret, Franck Hétroy, Marie-Paule Cani, and Carol O'Sullivan. Animating Quadrupeds: Methods and Applications. *Computer Graphics Forum*, 28, 2009. 1, 30, 87

[SvABV09] H. Martin Schepers, Edwin H. F. van Asseldonk, Jaap H. Buurke, and Peter H. Veltink. Ambulatory estimation of center of mass displacement during walking. *IEEE Transactions On Biomedical Engineering*, 2009. 22

[SZGP05] Robert W. Sumner, Matthias Zwicker, Craig Gotsman, and Jovan Popović. Mesh-based inverse kinematics. *ACM Trans. Graph.*, 24:488–495, July 2005. 18

[TCHL12] Yi-Jheng Huang Ting-Chieh Huang and Wen-Chieh Lin. Real-time horse gait synthesis. 2012. 31

[TGTL11] Jie Tan, Yuting Gu, Greg Turk, and C. Karen Liu. Articulated swimming creatures. In *ACM SIGGRAPH 2011 papers*, SIGGRAPH '11, pages 58:1–58:12, New York, NY, USA, 2011. ACM. 30, 133

[TLC⁺10] Yao-Yang Tsai, Wen-Chieh Lin, Kuangyou B. Cheng, Jehee Lee, and Tong-Yee Lee. Real-time physics-based 3d biped character animation using an inverted pendulum model. *IEEE Transactions on Visualization and Computer Graphics*, 16:325–337, March 2010. 24, 29, 77

[TLP07] Adrien Treuille, Yongjoon Lee, and Zoran Popović. Near-optimal character animation with continuous control. In *ACM SIGGRAPH 2007 papers*, SIGGRAPH '07, New York, NY, USA, 2007. ACM. 13, 77

[Tor97] Nick Torkos. *Footprint Based Quadruped Motion Synthesis*. PhD thesis, 1997. 29

[TSC96] Daniel Thalmann, Jianhua Shen, and Eric Chauvineau. Fast realistic human body deformations for animation and vr applications. In *Proceedings of the 1996 Conference on Computer Graphics International*, CGI '96, pages 166–, Washington, DC, USA, 1996. IEEE Computer Society. 7

[Tur95] Russell Turner. *LEMAN: a system for constructing and animating layered elastic characters*, pages 185–203. Academic Press Ltd., London, UK, UK, 1995. 7

[TvdP98] Nick Torkos and Michiel van de Panne. Footprint-based quadruped motion synthesis. In *Graphics Interface*, 1998. 29

[vBPE10] B. J. H. van Basten, P. W. A. M. Peeters, and A. Egges. The step space: example-based footprint-driven motion synthesis. *Comput. Animat. Virtual Worlds*, 21:433–441, May 2010. 29

[VdPLHF00] Michiel Van de Panne, Joe Laszlo, Pedro Huang, and Petros Faloutsos. Dynamic human simulation: Towards agile animated characters. In *IEEE International Conference on Robotics and Automation*, 2000. 9

[VHB$^+$08] Miomir Vukobratovic, Hugh M. Herr, Branislav Borovac, Mirko Rakovic, Marko B. Popovic, Andreas Hofmann, Milos Jovanovic, and Veljko Potkonjak. Biological principles of control selection for a humanoid robot's dynamic balance preservation. *I. J. Humanoid Robotics*, 5(4):639–678, 2008. 22

[VMTF09] Pascal Volino, Nadia Magnenat Thalmann, and François Faure. A simple approach to nonlinear tensile stiffness for accurate cloth simulation. *ACM Transaction on Graphics*, 2009. 104

[vWvBE$^+$09] H. van Welbergen, B.J.H. van Basten, A. Egges, Z.M. Ruttkay, and M. H. Overmars. Real time character animation: A trade-off between naturalness and control. In *Eurographics*, 2009. 102

[vWvBE$^+$10] Herwin van Welbergen, Ben J. H. van Basten, Arjan Egges, Zsófia Ruttkay, and M. H. Overmars. Real time animation of virtual humans: A trade-off between naturalness and control. *Computer Graphics Forum, Eurographics 2010*, 29, 2010. xii, 1, 7, 16, 38

[WC91] L. C. T. Wang and C. C. Chen. A combined optimization method for solving the inverse kinematics problems of mechanical manipulators. *Robotics and Automation, IEEE Transactions on*, 7(4):489–499, 1991. 18

[Wel94] Chris Welman. *Inverse Kinematics and Geometric Constraints for Articulated Figure Manipulation [microform].* Simon Fraser University, 1994. 17

[WFH09] Jack M. Wang, David J. Fleet, and Aaron Hertzmann. Optimizing walking controllers for uncertain inputs and environments. *ACM Trans. Graph.*, 2009. 26

[WFH10] Jack M. Wang, David J. Fleet, and Aaron Hertzmann. Optimizing walking controllers for uncertain inputs and environments. *ACM Trans. Graph.*, 29(4):73:1–73:8, July 2010. 26

[WG01] Jane Wilhelms and Allen Van Gelder. Fast and easy reach-cone joint limits. *J. Graph. Tools*, 6:27–41, September 2001. 19

[Wil67] Donald M. Wilson. Stepping patterns in tarantula spiders. *The Journal of Experimental Biology*, 1967. 72, 94

[WMZ08] Chun-Chih Wu, Jose Medina, and Victor B. Zordan. Simple steps for simply stepping. In *Proceedings of the 4th International Symposium on Advances in Visual Computing*, ISVC '08, pages 97–106, Berlin, Heidelberg, 2008. Springer-Verlag. 29

[WP03] Jia-chi Wu and Zoran Popović. Realistic modeling of bird flight animations. In *ACM SIGGRAPH 2003 Papers*, SIGGRAPH '03, pages 888–895, New York, NY, USA, 2003. ACM. 30

[WP09] Kevin Wampler and Zoran Popović. Optimal gait and form for animal locomotion. In *ACM SIGGRAPH 2009 papers*, SIGGRAPH '09, pages 60:1–60:8, New York, NY, USA, 2009. ACM. 30, 32

[WTR11] Xiaomao Wu, Maxime Tournier, and Lionel Reveret. Natural character posing from a large motion database. *IEEE Comput. Graph. Appl.*, 31:69–77, May 2011. 18

[WZ10] Chun-Chih Wu and Victor Zordan. Goal-directed stepping with momentum control. In *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '10, pages 113–118, Aire-la-Ville, Switzerland, Switzerland, 2010. Eurographics Association. 24, 76

[YKH04] Katsu Yamane, James J. Kuffner, and Jessica K. Hodgins. Synthesizing animations of human manipulation tasks. *ACM Transactions on Graphics (SIGGRAPH 2004)*, 23(3), August 2004. 36

[YLS04] Po-Feng Yang, Joe Laszlo, and Karan Singh. Layered dynamic control for interactive character swimming. In *Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, SCA '04, pages 39–47, Aire-la-Ville, Switzerland, Switzerland, 2004. Eurographics Association. 21

[YLvdP07] KangKang Yin, Kevin Loken, and Michiel van de Panne. Simbicon: simple biped locomotion control. *ACM Trans. Graph.*, 26, July 2007. xiii, 21, 26, 77

[YP03] KangKang Yin and Dinesh K. Pai. Footsee: an interactive animation system. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, SCA '03, pages 329–338, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association. 29

[YPP05] Kangkang Yin, Dinesh K. Pai, and Michiel Van De Panne. Pacific graphics (2005) abstract data-driven interactive balancing behaviors, 2005. 14

[YPS07] Jesse W. Young, Biren A. Patel, and Nancy J. Stevens. Body mass distribution & gait mechanics in fat-tailed dwarf lemurs (cheirogaleus medius) & patas monkeys (erythrocebus patas). *Journal of Human Evolution*, 2007. 15

[ZH99] Victor B. Zordan and Jessica K. Hodgins. Tracking and modifying upper-body human motion data with dynamic simulation. In *COMPUTER ANIMATION AND SIMULATION*, pages 13–22, 1999. 11

[ZH02] Victor Brian Zordan and Jessica K. Hodgins. Motion capture-driven simulations that hit and react. In *Proceedings of the*

2002 ACM SIGGRAPH/Eurographics symposium on Computer animation, SCA '02, pages 89–96, New York, NY, USA, 2002. ACM. 27, 28, 112

[ZMCF05] Victor Brian Zordan, Anna Majkowska, Bill Chiu, and Matthew Fast. Dynamic response for motion capture animation. *ACM Trans. Graph.*, 24:697–701, July 2005. 27, 28

[ZMM+07] Victor Zordan, Adriano Macchietto, Jose Medin, Marc Soriano, Chun-Chih Wu, Ronald Metoyer, and Robert Rose. Anticipation from example. In *Proceedings of the 2007 ACM symposium on Virtual reality software and technology*, VRST '07, pages 81–84, New York, NY, USA, 2007. ACM. 28, 29

[Zor10] Victor Zordan. Angular momentum control in coordinated behaviors. In *Proceedings of the Third international conference on Motion in games*, MIG, pages 109–120, Berlin, Heidelberg, 2010. Springer-Verlag. 22