A. Egges. Real-time Animation of Interactive Virtual Humans. PhD Thesis, University of Geneva, Geneva, Switzerland. August 2006.

UNIVERSITÉ DE GENÈVE

FACULTÉ DES SCIENCES

Département d'informatique — Professeur José Rolim

FACULTÉ DES SCIENCES ÉCONOMIQUES ET SOCIALES

Département de systèmes d'information — Professeur Nadia Magnenat-Thalmann

Real-time Animation of Interactive Virtual Humans

THÈSE

présentée à la Faculté des Sciences de l'Université de Genève
pour obtenir le grade de Docteur ès sciences, mention informatique

par

Arjan EGGES
de
Winschoten (Pays-Bas)

GENÈVE

2006

Acknowledgments

First of all, I would like to express my deepest gratitude to Professor Nadia Magnenat-Thalmann for allowing me to form a part of the MIRALab team and the challenging and motivating research environment it encompasses. I would like to thank her for her support and her continuous encouragement to explore new research questions and to improve my results.

Also, I would like to thank the members of my jury, Professor José Rolim (University of Geneva), Professor Klaus Scherer (University of Geneva), and Prof. Ipke Wachsmuth (University of Bielefeld), for taking the time to evaluate my thesis work and for their very insightful evaluation reports.

Thanks to all my colleagues in MIRALab for their collaboration over the last five years. Without them, this work would not have been possible. I would like to thank Dr. Laurent Moccozet for his feedback on the thesis and for his assistance in preparing the French summary. I would like to thank Dr. Tom Molet for his collaboration in the starting phase of this research by assisting me during the motion capture sessions, as well as for his feedback and input. Furthermore, my gratitude goes to both Lionel Egger and Nedjma Cadi for their help in designing the 3D models, editing the animations, and preparing the demonstrations. To conclude, I would like to thank all other MIRALab members for the inspiring discussions and their friendship.

On a final note, I would like to express my warm gratitude to all my friends and family, and especially Nancy, for being there and for their love and support.

Abstract

Over the last years, there has been a lot of interest in the area of Interactive Virtual Humans (IVHs). Virtual characters who interact naturally with users in mixed realities have many different applications, such as interactive video games, virtual training and rehabilitation, or virtual heritage. The main purpose of using interactive virtual humans in such applications is to increase the realism of the environment by adding life-like characters. The means by which these characters perceive their environment and how they express themselves greatly influences how convincing these characters are. The work presented in this thesis aims to improve the expressive capabilities of virtual characters, notably the *animation* of IVHs.

Because of the complexity of interaction, a high level of control is required over the face and body motions of the virtual humans. In order to achieve this, current approaches try to generate face and body motions from a high-level description. Although this indeed allows for a precise control over the movement of the virtual human, it is difficult to generate a natural-looking motion from such a high-level description. Another problem that arises when animating IVHs is that motions are not generated all the time. Therefore a flexible animation scheme is required that ensures a natural posture even when no animation is playing. Finally, because of the many different components that an Interactive Virtual Human consists of, the animation model should be as efficient as possible.

In this thesis, we will present a new animation model, based on a combination of motion synthesis from motion capture and a statistical analysis of prerecorded motion clips. As opposed to existing approaches that create new motions with limited flexibility, our model *adapts* existing motions, by automatically adding dependent joint motions. This renders the animation more natural, but since our model does not impose any conditions on the input motion, it can be linked easily with existing gesture synthesis techniques for IVHs.

In order to assure a continuous realistic motion, a basic layer of motions, called idle motions, is always present. These motions are generated by sequencing prerecorded motion segments organised in a graph. The path followed through this graph is controlled by high-level constraints, such as an emotional state. On top of that, small variations in posture are added so that the character is never static. Because we use a linear representation for joint orientations, blending and interpolation is done very efficiently, resulting in an animation engine especially suitable for real-time applications.

Résumé

Beaucoup d'intérêt a été accordé aux Humains Virtuels Interactifs (HVI) au cours des dernières années. Des personnages virtuels qui interagissent naturellement avec des utilisateurs dans des environnements de réalité mixte ont des applications nombreuses et variées, telles que les jeux vidéo, l'entraînement et la rééducation virtuels ou l'héritage virtuel. Les moyens par lesquels ces personnages perçoivent leur environnement et par lesquels ils s'expriment influent grandement sur leur capacité à convaincre. Beaucoup de recherches ont été menées pour améliorer l'interaction entre les humains et les personnages virtuels, par exemple en utilisant des modèles élaborés de reconnaissance de modèles et de génération de réponses ou en utilisant des méthodes basées sur les conversations réelles ou des chats. Il est aussi maintenant possible d'interagir avec un ordinateur en utilisant la parole et le langage naturel, et même les informations visuelles telles que les mouvements de la tête, les expressions faciales et les gestes peuvent être reconnus et interprétés. Bien que l'état de l'art dans le domaine des technologies de la perception ne permette pas encore une interprétation sans faille de toutes ces données, de nombreuses techniques sont d'ores et déjà commercialement exploitées.

Au-delà des capacités en progression des ordinateurs à analyser et à interpréter l'information, ils peuvent aussi mieux *répondre* à ce qu'ils perçoivent. La réponse peut utiliser le texte, la parole ou une sortie visuelle plus élaborée telle que le contrôle d'un personnage 3D. Le domaine qui bénéficie le plus logiquement de ces techniques est celui de l'industrie du jeu par ordinateur. Dans de nombreux jeux, des personnages 3D interactifs sont contrôlés par le joueur ou ils se présentent comme des opposants interactifs faisant partie de l'environnement virtuel. Dans la plupart de ces jeux, ces personnages sont contrôlés à l'aide de système d'animation par script qui séquencent et jouent différents segments d'animation ou de son. Quels que soient les avancées récentes dans ce domaine de recherche, contrôler et animer un personnage virtuel reste encore une tâche pénible qui nécessite beaucoup de travail manuel de design et de travail d'animation.

D'un point de vue conceptuel, les techniques perceptuelles (reconnaissance de la parole, reconnaissance des expressions faciales, etc.) et des techniques de réponses

(synthèse de la parole, animation de personnages virtuels) forment une partie d'un cycle, appelé le **cycle Perception-Action**. Ce cycle décrit le mécanisme de feedback qui définit une interaction quelconque entre une entité et son environnement. En agissant dans un environnement, une entité le modifie. Les changements résultant sont ensuite perçus et une nouvelle action est exécutée à la suite. Cette thèse se focalise sur l'aspect *expressif* du cycle.

Les Humains Virtuels Interactifs peuvent s'exprimer au travers de la parole et de l'animation du visage et du corps. En général, ces composants génèrent automatiquement la parole et les gestes à partir d'une représentation abstraite de la sortie souhaitée. Cette représentation abstraite peut être par exemple un texte ou une structure XML complexe. Un système de dialogue produit une telle représentation abstraite selon ce que l'utilisateur réalise. Comme nous allons le voir, les modèles de personnalités et d'émotions jouent aussi un rôle important. Un simulateur de personnalité et d'émotion influence le résultat du dialogue ainsi que l'animation résultante.

Il existe de nombreuses techniques d'animation d'humains virtuels. La représentation d'une animation dépend de son niveau d'abstraction et de la structure du monde dans laquelle elle est utilisée. Au niveau le plus général, nous considérons une **animation** comme une fonction continue qui associe des instants clés sur des images. La continuité de cette fonction est une propriété importante, car elle permet d'afficher la situation courante de l'objet à n'importe quel instant. Parce qu'en pratique il n'est pas possible de définir une animation comme une fonction continue—ce qui nécessiterait de définir un nombre infini d'images—la plupart des représentations d'animation sont basées sur les key-frames ou images-clés. Dans cette approche, une animation est définie par une séquence discrète d'images-clés associées à des instants clés. Afin de revenir à la fonction continue d'animation d'origine, une technique d'*interpolation* est nécessaire. Le choix de cette technique dépend de la représentation des images-clés.

Pour produire des animations temps réel du corps, une approche commune consiste à travailler sur le squelette sous-jacent plutôt que sur le modèle lui-même. Lorsqu'on utilise une approche géométrique, l'animation du modèle est plus précise, mais elle dépend du modèle. Une approche basée sur le squelette permet de produire des animations indépendantes du modèle et de l'environnement, mais aussi moins de paramètres à manipuler. En considérant qu'un Humanoïd 3D est contrôlé par l'animation de son squelette, il est important de choisir la représentation appropriée pour la transformation des articulations du squelette. La représentation la plus général est une matrice de transformation qui contient une translation et une rotation. Une matrice ne peut représenter une rotation que si elle est orthonormalisée. Cela signifie que pendant l'animation, des opérations supplémentaires sont nécessaires pour assurer l'orthonormalité des matrices, ce qui est cher en temps de calcul. En outre, les matrices de rotation ne sont pas adaptées aux interpolations de rotations. C'est pourquoi d'autres représentations de rotations d'articulations ont été proposées. Quand une rotation est représentée par trois rotations autour des axes du système de coordonnées, on parle de représentation par angles d'Euler. La représentation par les angles d'Euler est assez efficace car elle utilise trois variables (angles) pour définir trois degrés de liberté. Cependant, l'interpolation peut être couteuse en temps de calcul, car elle requiert des intégrations numériques. En outre, les angles d'Euler ont le problème dit du *Gimbal lock* ou de la perte d'un degré de liberté qui apparaît quand une série de rotation à 90 degrés est appliquée. A cause de l'alignement des axes, ces rotations peuvent s'annuler. Une autre représentation est appelée l'axe angle. Cette représentation définit un axe de rotation arbitraire et une rotation autour de cet axe. Bien que cette représentation soit assez efficace, elle nécessite encore beaucoup de charge de calcul pour l'interpolation. Les quaternions sont aussi une représentation usuelle pour les rotations. Les quaternions sont une extension des nombres complexes. Des quatre composants, un est un nombre scalaire réel et les trois autres forment un vecteur dans un espace imaginaire. L'interpolation des quaternions est réalisée par une approche dite *SLERP*. Enfin, les rotations peuvent être représentées par la carte exponentielle. Un avantage décisif de cette approche est qu'elle permet de réaliser des transformations complètement dans le domaine linéaire. Cela solutionne de nombreux problèmes dont souffrent les méthodes précédentes. Les rotations sont représentées par une matrice antisymétrique ou son vecteur 3D équivalent.

Définir des animations sous forme de rotations et de translation d'articulations plutôt que par manipulations géométriques permet de définir un bon socle pour un système d'animation temps réel. Cependant, un niveau de contrôle plus élevé est nécessaire, car animer toutes les articulations à la main requiert beaucoup de manipulations. En outre, il est difficile de produire des animations naturelles en utilisant des méthodes procédurales, à moins d'y consacrer énormément de temps. Une approche intéressante consiste à produire de nouvelles animations à partir de données de mouvement enregistrée. Cette approche est appelée *performance animation* et son objectif principal est de combiner le naturel des mouvements enregistrés avec la flexibilité de la synthèse procédurale de mouvement. Il existe un grand nombre d'algorithmes pour synthétiser des mouvements du corps à partir de données. Toutes ces approches sont très similaires et présentent la même base. Trois étapes sont impliquées :

1. La première étape est la segmentation des données de mouvement capturées en segments disjoints. Différents chercheurs utilisent différentes techniques et critères de segmentation qui sont pratiquées soit manuellement soit automatiquement. En général, le graphe est étendu avec de nouvelles transitions en utilisant des segments de mouvements existants du graphe.

2. Les segments d'animation sont stockés dans une structure de graphe. Là encore, la connectivité et la complexité du graphe résultant dépendent de la méthode qui est utilisée.

3. Finalement, de nouvelles animations sont construites en suivant un chemin à travers le graphe des mouvements.

Une des techniques les plus connues pour la synthèse de mouvements est appelée *Motion Graphs* développée par Kovar er al. La principale contribution de leur technique est la construction automatique d'un graphe à partir d'une collection de segments de mouvement préenregistrés. Le graphe est construit à partir de deux opérations : éclatement d'un segment de mouvement et ajout d'une transition entre deux segments de mouvement. L'emplacement de l'éclatement d'un segment de mouvement dépend de savoir si une transition vers un autre segment de mouvement est favorable ou non selon un critère de distance. Ce travail met en avant un problème important qui apparaît quand on veut comparer différentes animations : quel critère doit être utilisé afin de déterminer la distance entre deux postures ? A critère simplement basé sur les distances entre les angles des articulations n'est pas suffisant à cause de plusieurs raisons :

1. La norme simple des vecteurs ne permet pas de prendre en compte la sémantique des paramètres : dans la représentation de l'angle de l'articulation, certains paramètres ont beaucoup plus d'impact que d'autres sur le personnage (par exemple l'orientation de la hanche par rapport à celle du poignet). En outre, il n'existe pas de façon appropriée d'associer des poids fixes à ces paramètres, car l'effet d'une rotation d'articulation sur la forme du corps dépend de la configuration instantanée du corps.

2. Un mouvement est définit seulement à une transformation de coordonnées 2D près. Ce qui signifie que le mouvement reste fondamentalement inchangé si on le translate à la surface du plan du sol ou si on lui applique une rotation autour de l'axe vertical. Ainsi, comparer deux mouvements nécessite d'identifier des systèmes de coordonnées compatibles.

3. Une combinaison continue nécessite plus d'information qu'on ne peut en obtenir d'images individuelles. Une transition continue doit tenir compte non seulement des différences de postures des corps, mais aussi de la vitesse des articulations, de leur accélération et éventuellement des dérivés de plus haut degré.

Au lieu d'utiliser la différence entre les orientations des articulations, Kovar propose d'utiliser une autre métrique. Il utilise un nuage de points qui représente la forme de la posture. Bien que la technique du graphe de mouvement soit très utile dans de nombreuses applications, la méthode présente un certain nombre de limitations qui en rendent son usage moins adapté pour des personnages interactifs. A cause du choix d'utiliser le nuage de points, le critère de distance est lourd en termes de calculs. En outre, le graphe de mouvement nécessite des contraintes précises. A proximité des points du début et de la fin de l'animation, le chemin lui-même doit être défini. Quand on a besoin de contrôler un personnage à un plus haut niveau, des contraintes aussi précises ne sont pas forcément facilement accessibles.

Il y a plusieurs efforts de recherche focalisés sur le contrôle automatique du mouvement du visage et du corps d'un personnage. Des exemples de tels systèmes sont BEAT, GRETA, REA ou MAX. La plupart de ces systèmes ont une façon plutôt détaillée de spécifier les séquences d'animation du visage et du corps à jouer. Ce haut niveau de contrôle est principalement perceptible dans la façon dont les mouvements des mains et des bras sont construites. Bien qu'un contrôle très précis de ces mouvements est souhaitable afin d'obtenir un lien fort avec par exemple un système de dialogue, les animations résultantes ne sont pas très naturelles car elles ne sont définies que pour quelques articulations. Aussi, parce que ces systèmes ne prennent pas en considération l'ensemble du corps, les personnages ont l'air statique pendant l'interaction. Actuellement, il n'existe pas de système qui puisse gérer des animations détaillées du visage et du corps et qui produise des animations naturelles du personnage.

Dans cette thèse nous nous sommes fixés le but de développer une approche qui présente suffisamment de flexibilité pour être utilisées avec des humains virtuels interactifs mais permette aussi de synthétiser des mouvements qui sont *réalistes*. Afin de créer ce *synthétiseur idéal de mouvements* pour un humain virtuel interactif crédible, il est nécessaire d'avoir un système qui permette à la fois un haut niveau de contrôle et un haut niveau de réalisme. L'objectif global de cette thèse est de développer des techniques qui permettent d'aller dans la direction de ce synthétiseur idéal de mouvements tout en gardant à l'esprit les contraintes du temps réel. Afin d'atteindre cet objectif global, nous proposons plusieurs contributions spécifiques à l'état de l'art dans plusieurs domaines de recherche :

Manipulation temps-réel d'animation Nous proposons une nouvelle représentation d'animation du corps qui permet des manipulations rapides. La représentation est adaptée à des opérations linéaires comme la carte exponentielle, mais avec l'avantage additionnel que seulement un petit sous-ensemble du vecteur de posture est nécessaire pour de nombreuses applications, permettant ainsi d'améliorer grandement l'efficacité de la gestion de l'animation.

Synthèse flexible de mouvements Nous présentons un synthétiseur flexible de mouvements qui adapte complètement automatiquement des segments de mouvements préenregistrés pour créer de nouveaux mouvements réalistes. Le synthétiseur de mouvements est aussi capable d'adapter et de séquencer des mouvements en temps réel en utilisant un critère de distance rapide qui s'appuie sur la représentation de l'animation proposée.

Gestes réalistes automatisés Nous montrons une nouvelle technique qui permet la production de gestes réalistes en temps réel à partir de mouvements générés de façon procédurale définis seulement pour quelques articulations. Nous présentons une méthode pour déterminer automatiquement les mouvements des articulations dépendantes à un coût de calcul bas.

Interaction expressive complète du corps Comme preuve de concept de l'adéquation du gestionnaire d'animation, nous développons un gestionnaire de dialogue capable de contrôler simultanément l'animation du visage et du corps. Une interface simple est proposée pour permettre une définition cohérente des mouvements du visage et du corps, qui seront automatiquement synchronisés avec le signal de la parole, créé à partir d'un synthétiseur vocal. Cela permettra à un humain virtuel interactif de répondre à un utilisateur en utilisant des mouvements du visage et du corps. Etant donné que les expressions émotionnelles du visage et du corps ont besoin d'être contrôlées, nous présentons un simulateur de personnalité et d'émotion efficace adapté pour les animations de personnages interactifs.

Nous allons maintenant décrire chacune de ces contributions en détail.

Manipulation temps-réel d'animation Nous allons maintenant présenter une collection de techniques qui permet de synthétiser de nouveaux mouvements en utilisant des mouvements préalablement enregistrés ou produits à la main par un designer. Etant donné que notre but est de réaliser cette synthèse automatiquement, nous avons besoin d'une méthode adaptée de représentation de l'animation. Aussi, parce que nous connaissons le domaine d'application dans lequel les animations vont être utilisées (mouvements de communication interactive), nous pouvons tirer profit de méthodes qui analysent statistiquement les mouvements enregistrés afin de déterminer les dépendances entre différentes variables. Notre approche pour la représentation des mouvements du corps consiste à utiliser une représentation par les composants principaux. Nous avons choisi cette approche parce qu'elle permet de déterminer les dépendances entre les différentes variables (dans notre cas, les rotations des articulations). Comme nous le montrerons, c'est une propriété puissante qui permet de nombreuses optimisations dans la représentation des données et leur traitement. Parce que la méthode de l'analyse par les composants principaux n'opère que sur des données exprimées dans un espace linéaire, nous devons utiliser une représentation des orientations dans un tel espace, comme avec les cartes exponentielles. Alexa a déjà proposé la carte exponentielle comme une représentation valide des animations et il a aussi montré que des transformations linéaires et d'autres opérations matricielles sont possibles avec une telle représentation. C'est pourquoi nous avons choisi de représenter nos animations dans ce format. Nous avons acquis un grand ensemble de postures à partir d'animation enregistrées avec le système de capture du mouvement VICON. Nous représentons chaque orientation avec ses trois valeurs définissant la matrice antisymétrique de la carte exponentielle. En plus, une rotation globale (racine) est définie. Etant donné que nous utilisons les 25 articulations H-Anim pour chaque posture, notre échantillon de données est de dimension 78. Pour les composants principaux on obtient une matrice de transformation 78×78. Dans les paragraphes suivants, nous allons expliquer comment cette représentation est utilisée pour la synthèse du mouvement ainsi que pour la synthèse automatique des mouvements des articulations dépendantes.

La représentation d'animation à base de carte exponentielle est encapsulée dans notre moteur d'animation appelé MIRAnim. L'architecture de l'application est une approche multipistes, dans laquelle plusieurs flux doivent être mélangés en une animation finale. L'objectif de notre moteur d'animation est de fournir une structure générique qui permette l'implémentation de différentes stratégies de mélanges. Ceci est particulièrement important respectivement aux différentes représentations des animations en fonction de l'application. Par exemple, les mouvements obtenus par synthèse de mouvement sont fusionnés en utilisant la représentation par les composants principaux. Pour des fusions plus génériques, dans lesquels différents flux d'animation doivent se jouer sur différentes parties du corps, une méthode basée sur les articulations est nécessaire. De plus, dans le système final, nous aurons besoin de réaliser des opérations de fusion à la fois sur le visage et le corps qui sont deux formats d'animation complètement différents et qui nécessitent différentes stratégies. L'approche que nous allons présenter est adaptée pour n'importe laquelle des situations présentées précédemment. Un grand ensemble d'outils de fusion, par exemple, distorsion, éclatement et atténuation du temps sont disponibles. L'avantage de cette approche générique est qu'une fois qu'un outil de fusion a été défini, il peut être utilisé pour n'importe quel type d'animation, indépendamment de son type. Pour pouvoir utiliser ces outils de fusion, une seule interface a besoin d'être établi entre la structure de donnée utilisée pour la fusion et la structure d'animation d'origine. La structure de base utilisée dans le moteur de fusion est l'interface dénommée `BlendableObject`. Un objet fusionnable est la représentation d'une animation qui peut être fusionnée avec d'autres animations. La principale fonctionnalité de cet objet est une fonction qui associe des instants clés à des images-clés. Une image-clé dans le moteur de fusion est appelée une `AbstractFrame`. Une image abstraite est constituée d'un ensemble d'éléments appelés les objets `AbstractFrame-Element`. Chacun de ces éléments est une liste de valeurs en virgule flottante. Par exemple, dans le cas d'animations du corps, un `AbstractFrameElement` pourrait être une liste de 3 valeurs représentant une carte exponentielle de rotation, ou une liste de 3 valeurs représentant une translation 3D. Une image abstraite pourrait alors consister en une combinaison d'éléments qui sont soit des rotations soit des translations. Dans le cas d'une animation faciale, les éléments pourrait être une liste de 1 valeur, représentant une valeur de FAP dans le standard MPEG-4. Au travers de l'interface du `BlendableObject` chaque animation est définie comme une fonction $A : t \to K$, où t est un instant clé compris entre t_s et t_e, tel quel $0 \leq t_s < t_e < \infty$ et K est l'image correspondante dans l'animation. Il faut maintenant une structure qui permette de prendre un certain nombre de ces pistes d'animation et de les fusionner ensemble selon certains paramètres. Ces paramètres sont définis au travers de l'interface `BlendingParams`. Le paramètre le plus général est une valeur de poids utiliser pour le mélange. Outre une liste de poids, l'interface `BlendingParams` propose aussi une liste de coefficients de changement d'échelle, des *scales*. L'évolution des poids et des scales au cours du temps est contrôlé par une courbe paramétrable. Différents type de `BlendingParams` peuvent être multiplexés et des objets de paramètres de fusion ajustés peuvent être définis. Par exemple, un objet `BlendingParams` peut être définis spécifiquement pour une animation du corps, qui définit un masque les articulations. Quand il est multiplexé avec un objet `BlendingParams` définissant une courbe de poids, le résultat est un ensemble de courbes définies pour chaque articulation activée dans le masque. N'importe quelle combinaison arbitraire de ces paramètres de fusion est possible, permettant une structure de paramétrisation flexible de la fusion. Ici, l'avantage de l'indépendance de la stratégie de fusion revient. Une fois qu'un paramètre de fusion tel qu'une courbe de fusion est implémenté, il peut être appliqué sur n'importe quelle animation.

Synthèse flexible de mouvements Maintenant que nous avons représenté les postures du corps dans l'espace des composants principaux, nous pouvons commencer à utiliser les puissantes propriétés d'une telle représentation pour la synthèse de mouvement. Afin de créer de nouveaux mouvements à partir d'une collection de mouvements existants, il y a deux opérations principales qui seront souvent appliquées sur les images de ces animations :

1. Calcul de distance entre deux images.

2. Interpolation entre deux images.

La conformité de la première opération est très importante en synthèse du mouvement, car elle définit quand et où des transitions ou des adaptations appropriées de segments de mouvement pourront être réalisées. L'interpolation entre images est utilisée quand un segment de l'animation d'origine est adapté de façon à respecter un ensemble de contraintes. Des exemples de telles contraintes sont une nouvelle posture cible, ou une translation du mouvement d'origine pour une autre position et/ou orientation. Le calcul de distance entre deux images est tout spécialement pratiqué dans les méthodes de synthèse de mouvement. Quand on utilise un critère de distance basé sur la géométrie en combinaison avec une telle approche, le temps nécessaire pour le pré-calcul est important. Notre critère de distance entre images est basé sur la représentation des postures en composants principaux. Comme discuté précédemment, une représentation par les composants principaux regroupe ensemble les mouvements des articulations dépendantes. Ainsi, en définissant la distance entre deux postures P et Q comme la distance Euclidienne pondérée entre les deux vecteurs correspondants des composants principaux p et q, les dépendances des articulations seront prises en compte comme partie intégrante du calcul de distance. Les valeurs des poids sont choisis comme les valeurs propres déterminées durant l'analyse par les composants principaux. L'espace des composants principaux étant linéaire, le calcul de la distance peut être effectué aussi rapidement (plus rapidement) que dans les méthodes basées sur les articulations précédemment évoquées. Cependant, l'utilisation des composants principaux a une autre propriété qui va permettre une augmentation significative du temps de calcul de la distance : la réduction de dimension. Alors que les composants principaux les plus élevés représentent les postures qui apparaissent le moins, elles sont à peu près nulles et donc ne contribue pas significativement au facteur de distance. Ce qui signifie qu'en faisant varier le nombre de composants principaux utilisés, nous pouvons atteindre un compromis raisonnable entre rapidité et précision. En comparant les différentes distances obtenus à partir d'un nombre variable de composants, nous avons déterminé qu'en utilisant un vecteur de composants composé seulement des 10 premières valeurs, dans 80% des cas, aucune erreur de distance ne se produit, alors que dans les 20% restants, l'erreur de distance ne sera pas visible en moyenne. Ainsi, en utilisant l'approche des composants principaux, une accélération substantielle est obtenue tout en maintenant des résultats visuellement équivalents.

Notre système de synthèse de mouvement utilise une représentation par graphe pour relier les différents segments d'animation enregistrés dans la base de données. Comme le système va être utilisé pour simuler un humain virtuel interactif, nous allons utiliser comme base uniquement des mouvements qui seront utiles pour ce type de personnage. La plupart de ce genre de systèmes se concentre sur la création de gestes du haut du corps (buste). L'exception à cette situation est le travail de Cassel et al. Ils montrent que non seulement le buste est important pour la communication, mais qu'en fait c'est tout le corps qui l'est. Ils notent aussi que le *changement d'appui* (balancement d'un pied sur l'autre) est une des fonctions les plus importantes dans la communication corporelle et qu'il indique un changement de sujet. Bien qu'ils aient implémenté de tels mouvements dans REA, leur intérêt était clairement orienté vers l'expressivité communicante du personnage et beaucoup moins dans le réalisme de ses mouvements. Notre but est de synthétiser des mouvements réalistes de tout le corps en nous focalisant sur les mouvements de changement d'appui. Ces mouvements sont générés automatiquement à partir d'une base de données de séquences d'animation préenregistrées. A cause des différentes postures de départ et de fin dans chaque segment, cette base de données ne contient pas toutes les transitions possibles entre les postures de départ et celles de fin. C'est pourquoi nous avons développé une technique qui permet de projeter une animation sur un ensemble de différentes postures clés. Afin que cette technique puisse être aussi réaliste que possible, nous devons déterminer laquelle des animations s'adaptera le mieux à une posture de départ p et une posture de fin q. Afin de le déterminer, le système doit choisir l'animation dont la première et la dernière images sont respectivement les plus proches des postures p et q. Afin de déterminer la distance entre les images, nous utilisons la méthode de calcul déjà décrite précédemment. Afin de sélectionner l'animation qui s'adaptera le mieux entre p et q, on calcule pour chaque animation $a \in A$ le maximum M_a des distances $d_{a_0,p}$ et $d_{a_e,q}$. L'animation que nous utiliserons sera celle qui a la valeur M_a minimale. En interpolant dans l'espace des composants principaux, nous adaptons l'animation sélectionnée de façon à ce qu'elle démarre avec la posture p et finisse avec la posture q. En appliquant cette approche, une nouvelle séquence d'animation est produite en connectant différents segments d'animation avec des postures de départ et de fin similaires.

A part les postures de changement d'appui, les petites variations de postures apportent grandement au réalisme d'une animation. A cause de facteurs tels que la respiration, les petites contractions musculaires... les humains ne peuvent jamais exactement tenir la même posture. Il n'y a pas eu beaucoup de recherche dans ce domaine, à l'exception du travail de Perlin, qui est basé sur l'application de bruit de Perlin sur le coude, le cou et la hanche. Cette méthode produit des animations bruitées assez réalistes alors que le bruit n'est appliqué qu'à quelques articulations. Cependant, les variations de postures affectent l'ensemble des articulations, ce qui ne peut pas être simplement généralisé en appliquant des fonctions de bruits sur toutes les articulations à cause des dépendances entre les articulations. Contrairement à l'approche de Perlin, nous utilisons la représentation par les composants principaux pour chaque image-clé. Etant donné que les variations sont appliquées sur les composants principaux et pas directement sur les articulations, cette méthode produit des variations aléatoires tout en conservant la dépendance entre les articulations. En plus, parce que les composants principaux représentent les dépendances entre les variables dans les données, les composants principaux sont les variables qui ont le *maximum d'indépendance*. En tant que tel, nous pouvons les traiter *séparément* pour générer des variations de postures. Une méthode pour générer directement des variations est d'appliquer une fonction de bruit de Perlin sur un sous-ensemble de composants. Une autre méthode qui produit ces petites variations est basée sur la forme des courbes dans les données de mouvement capturé. Cette méthode applique un modèle statistique pour générer des courbes (aléatoires) similaires. Cette approche permet de maintenir certaines tendances dans les variations qui sont propres à certaines personnes (tels que des mouvements typiques de la tête). Un avantage supplémentaire par rapport au bruit de Perlin est que cette approche est complètement automatique et qu'elle évite d'avoir à définir les fréquences et les amplitudes pour générer le bruit, bien que ces paramètres puissent être automatiquement extraits par analyse du signal. Dans notre méthode, nous analysons les segments d'animation qui ne contiennent pas de changement d'appui ou de mouvements autres que de petites variations. Les mouvements de bruit résultant sont appliqués sur les mouvements synthétisés. Dans notre application, la méthode de synthèse de bruit est appliquée à un sous-ensemble de valeurs de composants. Pour chaque composant, les variations sont générées indépendamment. Cela ne donne pas de résultats irréalistes car la dépendance entre les composants principaux est relativement faible. Le résultat final de la synthèse de variation ne montre pas de répétition grâce au bruit aléatoire et au traitement séparé des composants principaux. Le mouvement final sera utilisé pour contrôler le personnage de façon à ce qu'une couche de base de mouvement, appelés les mouvements d'inactivité (*idle motions*), soit toujours présente.

Gestes réalistes automatisés Les systèmes de synthèse de mouvements du corps produisent souvent des gestes qui sont définis comme des mouvements spécifiques du bras provenant d'une représentation plus conceptuelle du mouvement. Traduire de telles spécifications de haut niveau de gestes en animations résulte souvent en des mouvements qui semblent mécaniques. Ceci est du au fait que les mouvements ne sont spécifiés que pour quelques articulations alors que dans les captures de mouvement, chaque mouvement d'articulation a une influence sur celui d'autres articulations. Par exemple, en bougeant la tête de gauche à droite, des mouvements de la colonne vertébrale et des épaules vont être induits. Cependant, les animations basées sur les captures de mouvements n'offrent pas la flexibilité requise par un système de synthèse de gestes. De tels systèmes bénéficieraient grandement d'une méthode qui puisse automatiquement et en temps réel calculer des mouvements crédibles pour les articulations qui sont dépendantes du geste. Notre méthode utilise les composants principaux pour produire des mouvements paraissant plus naturels. Les composants sont ordonnés de façon à ce que les composants les plus bas indiquent une forte occurrence dans les données alors que les composants les plus élevés indiquent une faible occurrence dans les données. Cela permet par exemple de compresser les animations en ne retenant que les indices des composants les plus bas. Les animations qui sont proches de celles qui sont dans la base de données qui ont été utilisées pour l'analyse par les composants principaux auront des indices de composants élevés proches de zéro. Une animation qui sera très différente de celles qui sont dans la base de données aura plus de bruit dans les indices des composants élevés pour compenser la différence. Si on suppose que la base de données qui est utilisée pour l'analyse par les composants principaux est *représentative* des mouvements généraux qui sont exprimés par des humains pour communiquer, alors les indices de composants élevés représentent la partie de l'animation qui n'est pas naturelle (ou : qui n'apparaît pas fréquemment dans la base d'animations). Quand on retire ces indices de composants élevés ou qu'on leur applique un filtre de changement d'échelle, cela produit une erreur dans l'animation finale. Cependant, étant donné que le filtre retire la partie non naturelle de l'animation, le résultat est un mouvement qui contient précisément les mouvements des articulations dépendantes. En variant les indexes des composants principaux auquel le filtre d'échelle démarre, on peut contrôler la proximité de l'animation résultat par rapport à l'animation par images-clés d'origine. Pour calculer les mouvements des articulations dépendantes, il suffit d'appliquer une fonction de changement d'échelle. Ainsi cette méthode est particulièrement adaptée pour les applications temps réel.

Interaction expressive complète du corps Pour qu'un système d'animation soit adapté pour des humains virtuels interactifs, il est crucial de pouvoir appliquer des contraintes sur le synthétiseur de mouvements. Ces contraintes pourraient être définies par exemple par un gestionnaire de dialogue qui contrôle les sorties et les expressions d'un personnage. Comme cas d'étude, nous allons présenter une extension du synthétiseur de mouvement qui sélectionne différents mouvements en fonction de différents *états émotionnels*. Afin de réaliser cette extension, nous devons tout d'abord décider de la représentation des émotions à utiliser. Il existe plusieurs représentations des émotions, mais elles ne sont pas toutes adaptées pour représenter des postures émotionnellement expressives. Par exemple, la liste d'émotions proposée dans le modèle OCC est trop détaillée par rapport à ce que les gens peuvent effectivement percevoir. Coulson a démontré que seulement quelques émotions sont faciles à distinguer à partir d'une seule posture du corps. Ainsi il semble logique d'utiliser une représentation des émotions qui s'appuie sur cette observation. Nous avons choisi d'utiliser l'espace activation-évaluation comme espace représentatif des émotions pour le contrôle des mouvements du corps, à cause de la similarité avec la précision perceptive telle qu'établie par Coulson. Le disque activation-évaluation est une représentation bidimensionnelle qui définit un disque avec deux axes : un axe activation et un axe évaluation. Un avantage supplémentaire à utiliser cette représentation est que les autres modèles multidimensionnels peuvent être facilement projetés sur le disque, en exprimant les différentes émotions sous forme de paires activation-évaluation.

Avant qu'une nouvelle animation soit créée, les animations qui sont dans la base de données doivent être annotées avec des informations émotionnelles supplémentaires. Pour chaque segment d'animation, nous définissons le changement de contenu émotionnel en spécifiant le début et la fin de l'animation sous forme d'un intervalle bidimensionnel sur le disque activation-évaluation. Etant donné un état émotionnel $[e_e, e_a]$, le synthétiseur de mouvement sélectionne alors automatiquement les animations qui ont un intervalle cible qui contient ce point dans l'espace émotion. Afin de s'assurer qu'il est toujours possible d'effectuer une transition quelque soit le contenu émotionnel, un ensemble de transitions sans contraintes est ajouté aussi, de telle sorte que lorsqu'aucun intervalle cible ne peut être sélectionné, une transition neutre est toujours possible. Sans compter que les transitions elles-mêmes changent en fonction de l'état émotionnel. Une option additionnelle fournie par le synthétiseur est l'adaptation automatique de la durée de la pause entre les transitions selon le niveau d'activation. Des niveaux d'activations plus élevés résulteront en des pauses plus courtes (et ainsi en plus de transitions).

Maintenant qu'un mécanisme de contrôle de haut-niveau a été établi pour l'état émotionnel, l'animation finale peut être construite en combinant différentes animations en utilisant le moteur MIRAnim. Nous avons intégré le moteur d'animation dans un prototype de système de dialogue qui produit des réponses annotées à partir des informations de l'utilisateur. Associés à l'animation faciale produite pour la parole, les annotations dans le texte indiquent quelles actions supplémentaires du visage et du corps devraient être jouées, synchronisées avec la parole. Associé aux mouvements et aux gestes, le synthétiseur tourne continuellement, fournissant ainsi à l'humain virtuel interactif des mouvements d'inactivité. Ces mouvements d'inactivité changent selon l'état émotionnel obtenu à partir du système de dialogue en utilisant la méthode décrite précédemment. Finalement, les mouvements d'inactivité et les gestes sont fusionnés à la volée.

Conclusions Nous avons présenté une nouvelle représentation pour les animations, basée sur l'analyse par les composants principaux de segments de mouvements obtenus par capture. L'analyse par les composants principaux est une technique bien connue qui a déjà été appliquée dans de nombreux contextes. Cependant les applications sur les mouvements du corps ont été rares à cause des contraintes sur la représentation des orientations. Nous avons montré que les composants principaux peuvent être appliqués avec succès sur une représentation des orientations basée sur la carte exponentielle. Nous avons décrit des algorithmes efficients basés sur les composants principaux pour la manipulation rapide d'animations, particulièrement le critère de distance et l'adaptation temps réel des animations. La moteur d'animation MIRAnim est construit autour de cette structure et permet de fusionner et de jouer des animations de tous types. Le gestionnaire d'animations est adapté pour différents types d'applications et il permet de basculer entre des personnages contrôlés par scénario ou des personnages interactifs contrôlés en continu sans interrompre les animations qui sont jouées.

Sur la base des performances du critère de distance et de l'adaptation des animations, nous avons démontré une technique de synthèse de mouvement qui ne nécessite pas de cycle de pré-calcul contrairement aux techniques existantes, notamment celle du Motion Graph. Ce qui offre la possibilité d'adapter dynamiquement le graphe de mouvements. En outre, notre méthode est basée sur un graphe dont les sommets ont une sémantique. Par conséquent, un mécanisme de contrôle a pu être établi qui permet de choisir un chemin dans le graphe, une option qui n'est pas possible dans les méthodes existantes. Nous avons aussi présenté un contrôle de haut-niveau pour les postures émotionnelles en définissant une stratégie qui assigne des contraintes sur les arêtes du graphe. Notre technique est indépendante du type de mouvement, contrairement aux systèmes existants qui se focalisent sur des types de mouvements particuliers, comme la marche.

Nous avons présenté une méthode efficace pour générer automatiquement les mouvements des articulations dépendantes en temps-réel à partir des mouvements de quelques articulations. Le filtre des composants principaux est simple à implémenter et les paramètres peuvent être choisis de façon à adapter la quantité de mouvement à ajouter. Notre approche ne place aucune limitation sur le mouvement en entrée. En outre, grâce à l'intégration avec le synthétiseur automatique de mouvement, notre approche garantit une posture naturelle, même quand aucun mouvement n'est activé. Cela ajoute de la *vie* au personnage qui n'est pas disponible dans les systèmes actuels.

Nous avons développé un système interactif qui génère des réponses appropriées en fonction des entrées de l'utilisateur. Notre approche permet la spécification des mouvements du visage et du corps qui sont joués de façon synchronisée avec le signal de la parole. Là encore, les mouvements résultat sont fusionnés avec les mouvements en cours. Grâce à la gestion indépendante des animations, notre approche peut être facilement intégrée dans des systèmes interactifs existants. En outre, nous avons présenté un système pour la simulation de la personnalité et des émotions. L'état émotionnel est utilisé pour exprimer automatiquement les émotions sur le visage et le corps, ce qui n'était pas encore disponible pour des personnages interactifs.

Enfin, nous avons développé un prototype fonctionnel du système intégré complet. Plusieurs logiciels sont disponibles pour tester les différentes méthodes qui ont été décrites dans cette thèse. Une boîte à outils simple à utiliser à été développée, pour permettre de jouer et d'adapter des animations selon les besoins de l'animateur. En outre, le prototype a été intégré avec un outil d'exécution de scénario afin de permettre au moteur d'animation de jouer un scénario d'animation avec plusieurs personnages dans un environnement 3D. Le service d'interaction ajoute la possibilité d'interagir avec ces personnages à tout moment.

Travail futur Bien que nous ayons proposé plusieurs contributions par rapport à l'état de l'art dans cette thèse, beaucoup de travail reste encore à faire dans le domaine. Comme dans toute recherche, la nôtre à ses limitations. Nous pensons que l'utilisation des technologies que nous avons décrites dans les précédents chapitres représente un pas dans la bonne direction, mais nous avons identifié plusieurs thèmes de recherche dans ce domaine passionnant.

Dans l'implémentation actuelle de notre système, la séquence d'animation est produite à partir d'un texte annoté. Bien qu'une telle approche est suffisante pour évaluer le moteur d'animation qui contrôle les mouvements de l'humain virtuel, elle représente une tâche particulièrement consommatrice de temps si un script complexe doit être produit. Afin de permettre de développer facilement des humains virtuels interactifs, il est nécessaire de développer un système qui le fasse automatiquement. Il y a déjà eu des recherches dans ce domaine, notamment BEAT ou MAX, mais ces systèmes ont besoin de beaucoup de prétraitement et d'annotations. Une direction de recherche intéressante consisterait à annotés automatiquement les données de mouvement capturées pendant les conversations, puis, à partir de ces données, d'essayer de générer automatiquement de nouveaux mouvements synchronisé avec la parole produite par un système de texte-vers-parole. Une telle approche permettrait aussi d'inclure des gestes du corps plus complexes.

Bien que la fusion entre les différents types de mouvements soit simple à réaliser dans notre système, il ne repose pas sur des paramètres physiques. Ainsi, des collisions des pieds ou des glissements peuvent se produire. Nous avons proposé quelques solutions simples approximatives pour ces problèmes, mais le système n'a pas encore un contrôle complet sur le corps. Des actions plus complexes, telles que courir, sauter ou s'asseoir, sont très complexes à gérer à partir d'un texte annoté. Un mécanisme de sélection d'actions plus complexe doit être proposé qui puisse contrôler ce type d'actions tout en restant suffisamment flexible.

Un point très important qui n'est pas traité dans ce travail est les interactions avec des objets dans l'environnement. Fréquemment pendant des interactions, on pointe des objets, on prend des objets ou on les utilise pour réaliser des tâches. Il est aussi indispensable d'ajouter le contrôle du regard de l'humain virtuel, pour lequel l'intégration avec la position de la caméra dans l'environnement est requise. Contrôler les interactions avec les objets dans un environnement virtuel est un thème de recherche complexe qui en est encore à ses prémices. Les méthodes d'animation basées sur les articulations n'ont pas une représentation spatiale immédiate des effecteurs finaux du corps, ce qui rend difficile la définition de mouvements qui interagissent avec des objets tels qu'une porte ou un tiroir. L'interaction avec des objets en combinaison avec un système interactif est rendue encore plus complexe. Afin d'y arriver, il est nécessaire d'intégrer un système de perception virtuel afin que l'humain virtuel obtienne un feedback de ce qui se passe dans son environnement virtuel. Ce feedback perceptif aura un effet sur le comportement de l'humain virtuel.

La plupart des humains virtuels sont des modèles géométriques 3D dont le mouvement est contrôlé par la manipulation du squelette. L' 'intérieur' de ces humains est vide. Un vrai challenge consisterait à modéliser cette partie interne. Nous avons déjà effleuré cet aspect en modélisant la personnalité et les émotions. Cependant de vrais êtres humains adaptent leur comportement selon de nombreux signaux physiologiques. Ces signaux affectent non seulement le comportement des humains, mais aussi leur apparence, par exemple, rougir, transpirer, pleurer... Ces processus physiologiques représentent un aspect clé de nos expressions et de notre système comportemental, mais il n'y a actuellement quasiment aucune recherche pour intégrer ces signaux dans le comportement des humains virtuels. En outre, des motivateurs généraux tels que l'appétit ou la fatigue pourraient être implémentés. Au final, un humain virtuel ne cesserait plus d'exister quand l'application se termine, mais il/elle pourrait aller se coucher ou faire d'autres choses, ce qui en retour inspirerait d'autres conversations avec l'utilisateur lors de nouvelles sessions d'application.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

ELIZA: Hello, How do you do. Please state your problem.
User: I don't want to talk about my problems.
ELIZA: Your problems?
User: Yes.
ELIZA: You seem to be quite positive.
...

This small fragment of text is an example of an interaction between a user and one of the first interactive programs, ELIZA [127], developed in 1966 by Joseph Weizenbaum. Interacting with ELIZA was done by typing text on a keyboard. The program recognized words and patterns from the phrase that was entered, like 'problem' or 'mother' and based on that, a predefined response was given. The development of this program has spawned a big area of research, oriented toward the creation of human-computer interaction that levels human-human interaction. The main idea behind this research is that humans should not adapt to different computer interfaces, but that *computer interfaces should adapt to humans*. Since 1966, a lot of progress has been made. The interaction itself has been improved by using more elaborate pattern recognition and response generation [1], or by using heuristic methods based on real conversations or chats between humans [56]. Also, it is now possible to interact with a machine using speech and natural language, and even visual information like head movements, facial expressions and gestures can be recognized and interpreted. Although the current state-of-the-art of perceptual technologies does not yet allow for a flawless interpretation of all this data, a lot of these techniques are already commercially exploited.

Next to the improved capability of computers to analyse and interpret information, computers can also better *respond* to what is perceived. This can be a response using

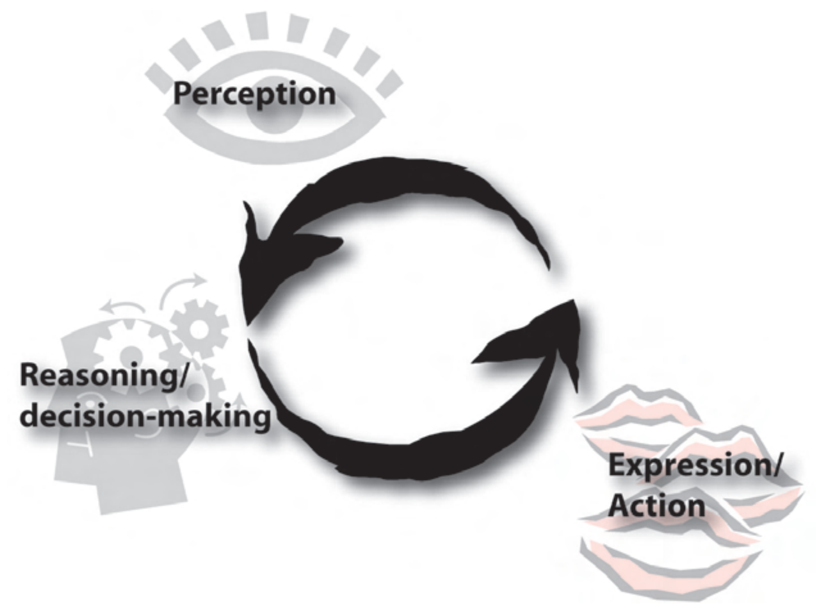


Figure 1.1: The Perception-Action loop.

text, speech, or more elaborate visual output such as controlling a character in 3D. The most evident area that profits from these techniques is the area of computer entertainment industry. In many games, interactive 3D characters are controlled by the player or they form an interactive component as a part of the 3D gaming environment. In most games, such characters are controlled using scripted animation systems that sequence and play different animation and sound clips. Regardless of the recent advancements in this research area, controlling and animation virtual character still remains a tedious task and it requires a lot of manual design and animation work.

On a conceptual level, the earlier mentioned perceptual techniques (speech recognition, facial expression recognition, etc.) and responsive techniques (speech synthesis, virtual character animation) form a part of a loop, called the **Perception-Action loop** [33][1], see Figure 1.1. This loop describes the feedback mechanism that defines any interaction of an entity with its environment. By acting in an environment, an entity changes it. These changes are then perceived and a new action is commenced (or not).

This thesis focuses on the *expressive* part of this loop. The main objective of the work that will be presented in the following chapters is to provide for an efficient method to control both face and body movements of virtual characters, as well as techniques that increase the realism of character motions without impinging the tractability of the character control method. In fact, the research that will be presented constitutes an elaborate *realism-adding filter* that can be placed between an automatically gener-

[1]This is a term originating from the Robotics and Sensor research area.

ated animation and the target character.

In the following chapter, we will start with discussing related research that is relevant to this thesis. We will also explain where our contributions fit in with the existing work. The Chapters 3, 4 and 5 contain the detailed presentation of our approach. Chapter 6 covers some implementation issues and we present our conclusions in Chapter 7.

CHAPTER 2

State of the Art

In this chapter, we will present an overview of relevant research related to the development of Interactive Virtual Humans (IVHs). Virtual Humans can express themselves through speech, facial animation and body animation. Generally, these components automatically generate the speech and the motions from an abstract representation of the desired output. This abstract representation can be for example a text or a complex XML structure. A dialogue system outputs such an abstract representation depending on what the user is doing. As we will show in the coming sections, personality and emotion models also play an important role as a part of IVH. A simulator of personality and emotions will influence the dialogue output, as well as the resulting animations. Figure 2.1 shows an overview of the main components that together form an IVH simulator. Since the work in this thesis focuses mainly on the *expressive* side of IVHs, the diagram shows the different expressive components that are required. The choice of these components is not a random one, but they form a commonly accepted structure of an IVH that is used with minor varieties throughout the literature discussed in this chapter. In this chapter, we will discuss existing work in each of the relevant areas of research. Each section will be concluded with a short discussion about the limitations and open problems.

This chapter is divided in the following sections. First, we will present the background on virtual human animation in general in Section 2.1. We will discuss different representations of animations, how they can be edited and how an animation is played on a 3D model of a virtual human. Then, we will discuss some methods that allow for a higher level control of virtual humans by using motion capture techniques, in Section 2.2. We will discuss how motion captured animations can be adapted and used to create new animations that look realistic. Subsequently, we will present existing IVH

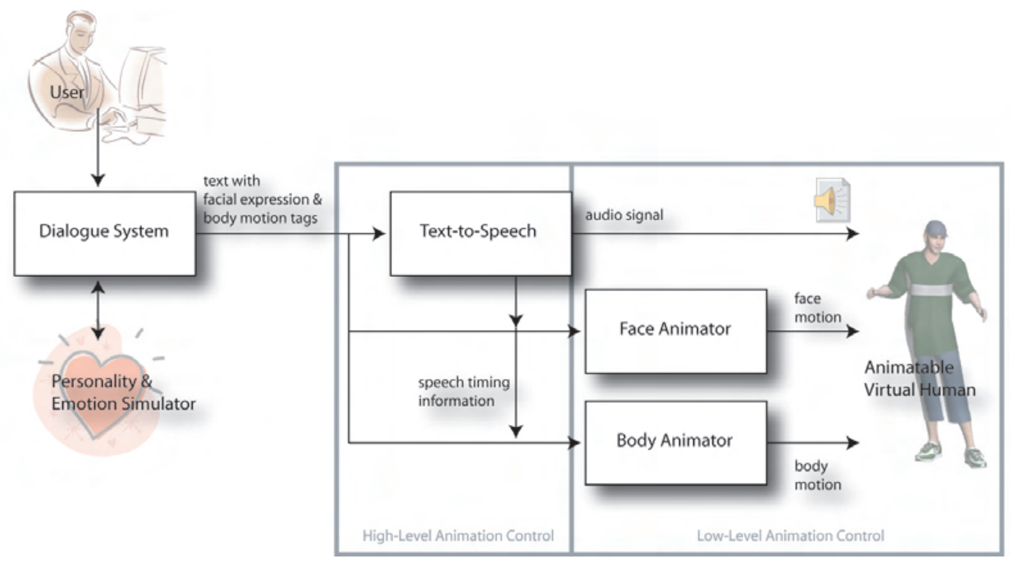


Figure 2.1: Main components of an Interactive Virtual Human simulator.

systems in Section 2.3. We will show how these systems can create an animation from an abstract representation of the desired output. We will also discuss various relevant aspects of body and face animation that need to be taken into account, such as personality, emotion, body posture, facial expressions and gestures. In Section 2.4, we will present the motivations behind the work that is presented in this thesis. Section 2.5 introduces a list of specific objectives, each of them proposing a contribution to a field of research.

2.1 Virtual Human Animation

There exist many different techniques to animate virtual humanoids. In this section, we will give an overview of the different approaches that exist. Both the architecture and the performance of any animation pipeline will greatly be influenced by the approaches that are chosen.

The representation of an animation depends on its level of abstraction and the structure of the world that it is used in. At the most basic level, we consider an **animation** as a continuous function $A : t \mapsto F$, where t is a timekey $\in [t_s, t_e]$ with $0 \leq t_s < t_e < \infty$, and F is the corresponding frame of the animation. The continuity of this function is an important property, because it allows to display the object's current state at any desired time. Because in practice it is not possible to define an animation as a continuous function—one would need to define an infinite amount of frames—most of the animation approaches rely on *key-frames*. In this approach, an animation is defined by a discrete sequence of key-frames, related with a timekey. In order to go back to the original continuous function, an *interpolation* technique is required. The choice of the interpolation approach depends on the representation of key-frames. In the following section, we will talk about different representations of animations, in combination with interpolation techniques. Most of the animation models are restricted to the **spatial domain**, but one could also imagine animations that change the color or texture of the 3D object (in the case of facial animation, think of blushing for example). In principle, any animation can be coded as a direct manipulation of the geometry of the target 3D object. However, there clearly are many disadvantages to this approach:

- it is a lot of work to develop low-level animations; this also makes it difficult to retain a reasonable level of realism;
- animations are not easy to transfer from one model to another as the animations will be dependent on the 3D model;
- editing the animations afterward is a difficult task; common operations on animations (scaling, masking, and so on) are difficult to apply.

Different animation techniques have been developed, that allow describing an animation in more abstract terms, while being easily adaptable to different 3D objects. In the following sections, we will look into such higher-level approaches for virtual humanoid animation. Especially when one wants to develop a (semi-)automatic animation system, these techniques form a crucial part of the animation pipeline.

Another important point to consider when one conceives an animation system, is its **usability** from a designer point-of-view. When we look at the design process for animations, there are two commonly used techniques:

- **Manual approach:** an animation is constructed from a set of key-frames, manually designed by an animator. Many commercial packages are available that allow an animator to create key-frames (or postures in the case one is dealing with virtual characters). Since the animator has a complete control over the resulting animation, this is a very popular approach. However, unless a lot of time is invested, the realism of the resulting animation is rather low.
- **Pre-recorded animations:** an animation is recorded using a motion capture or tracking system (such as Vicon [124] or MotionStar [82]). The MotionStar system uses magnetic tracking and it is much faster than the Vicon tracking system, which uses a multi-camera optical tracking approach. The Vicon however produces much more precise animations. This is why in many commercial applications, such as games or movies, an optical motion capture system is used to drive the character motions. A motion capture system is an excellent time-saver, because a lot less manual work is required to produce animations. A major disadvantage of using such a system, is that the animations need to be adapted so that they look good on different characters. Also, when using recorded motion clips, one must address the problem of unnatural transitions between the different clips.

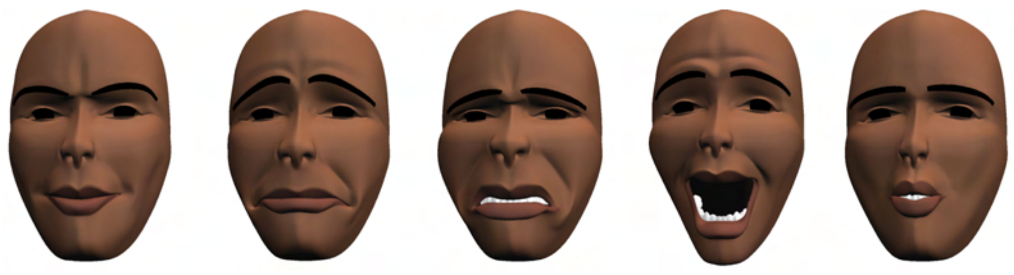


Figure 2.2: Facial expressions generated by a pseudo-muscle spline approach [125].

In Section 2.2, we will discuss a few research efforts that try to combine these two approaches. Although we generally consider a human being as a single object, strangely enough this is not the case in the existing research. Animation systems for virtual humans generally focus either on facial animation or body animation. The reason for this separation is the different type of animation that is required for the face and the body, as will be shown in the following section.

2.1.1 Facial Animation

Facial Animation is a research domain that started around the 1970s with the pioneering work done by Parke [96]. Similar to the body animation techniques discussed in the previous section, facial animation techniques have also been developed that try to abstract from the geometrical model. Apart from basic key-frame animation, models have been developed based on muscle physics as well as a 'pseudo-muscle' approach. We will describe each of these techniques in short.

Muscle-based methods of facial animation try to model the physics of the skin, fat, bone and muscle tissue on the face. This results in a physical simulation of the face that is controlled by applying muscle contractions. There are several researchers who have proposed systems based on this approach, such as the work done by Platt and Badler [103] or Waters [126].

Although such an approach can result in realistic deformations for the face, it is also a computationally heavy method. Several methods have been proposed that try to integrate geometric deformation of a polygon mesh with muscle-based models. These methods are for example based on splines [125] (see also Figure 2.2, Free Form Deformation (FFD) [61] or Radial Basis Functions [91].

Not only the model used to deform the face is of importance. Also a system of *parameterization* is required so that animations can be played on different faces. There have been principally three different parameterization systems. The first one is called Facial Action Coding System (FACS) and was developed by Ekman and Friesen [38]. FACS defines face regions—so-called Action Units—that allow expressing each type of facial movement. Another parameterization system is MPA, which stands for Mini-

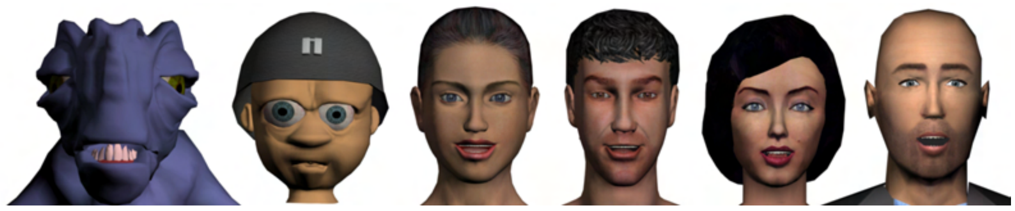


Figure 2.3: Facial expressions based on the MPEG-4 standard applied on different 3D models [44].

mum Perceptible Action [60]. In this approach, each MPA defines a collection of visible characteristics on the face (such as eyebrows or lips) together with the proper interdependent deformations of these characteristics. Finally, the MPEG-4 standard defines a widely used system for standardized face modelling and animation. MPEG-4 defines 84 Face Definition Parameters (FDPs) that are used to shape the face. A subset of these FDPs can be animated, resulting in 68 Facial Animation Parameters (FAPs) of which 2 are used to define high-level expressions. A more detailed overview of the MPEG-4 standard for facial animation can be found in the research done by Garchery [44] and Kshirsagar [72, 71] at MIRALab—University of Geneva. Figure 2.3 shows some examples of MPEG-4 compliant facial expressions applied on different models.

2.1.2 Body Animation

Although the issues of realism play an important role in both face and body animation, the lack of realism is most apparent in body animation systems. This is because of several reasons:

- Since body motions are generally described by joint rotations and translations, there are many degrees of freedom to control, thus creating more room for errors. Facial animation systems generally work with a smaller set of control points, with less degrees of freedom.
- Many rotation representations are not in linear space (as for example rotation matrices or quaternions), resulting in possible Gimbal locks (see page 12) or interpolation problems. Generally, facial animation control points can be manipulated in linear space.
- Certain physical constraints apply to body motions, because they involve an actual movement in space as well as the deformation of a concave object. For example self-collisions or foot sliding artefacts need to be avoided. Facial animation systems only have to take care of such constraints on a very limited basis (for example: self-collision in the lip area).

In this section, we will describe various representations and standards that are used to animate a 3D body. We will show the advantages and disadvantages of the different representations. We will also discuss some techniques to interpolate and mix different postures.

Skeleton-Based Animation

For producing real-time body animations, it is common practice to work on the underlying skeleton instead of the model itself. When using a geometry-based approach, the animation model is more precise, but the animation is dependent on the model. A skeleton-based approach allows for model- and environment-independent animations as well as fewer parameters to manipulate. There are currently two well known standards for body animation: MPEG-4 and VRML H-Anim. We will now shortly discuss each of these standards.

The H-Anim specification of animatable humanoids forms a part of the VRML language [50]. H-Anim defines a hierarchy of joints, with as a root node the so-called `HumanoidRoot`. The latest upgrade of the standard (H-Anim 200x), defines four levels of articulation (0-3). In Figure 2.4, an overview of the joint rotation centres defined in H-Anim is depicted. Since the H-Anim 200x standard is quite new, a lot of recent work is based on the H-Anim 1.1 standard.

MPEG-4 does not only define a standard for face modelling and animation, it also defines one for bodies. Similarly to the facial animation, two sets of parameters are defined for describing and animating the body: the Body Definition Parameter (BDP) set, and the Body Animation Parameter (BAP) set. However, this standard is far less used than the H-Anim standard.

Representation of Skeleton Posture

Given that a 3D Humanoid model is being controlled through a skeleton motion, it is important to choose the right representation for the transformations of the skeleton joints. The most basic representation is to define a 4×4 transformation matrix that contains a translation and a rotation component[1], where the rotation is defined as a 3×3 matrix. Transforming a skeleton joint becomes tricky when looking at the rotation component. A rotation involves three degrees of freedom (DOF), around the x, y and z axis, whereas a rotation matrix defines 9 (as a 3×3 matrix). A matrix can only represent a rotation if it is orthonormalised. This means that during animation, additional operations are required to ensure the orthonormality of the matrix, which is computationally intensive. Furthermore, rotation matrices are not very well suited for rotation interpolation. Therefore, other representations of joint rotations have been proposed. For a more detailed discussion of each of the representations, we refer to Grassia [48].

When a rotation is represented as three rotations around the three coordinate axes, it is called the Euler Angle representation (see Figure 2.5). An Euler angle can be written

[1]For a more detailed description, please see any standard textbook on Computer Graphics, such as Hearn and Baker [55].

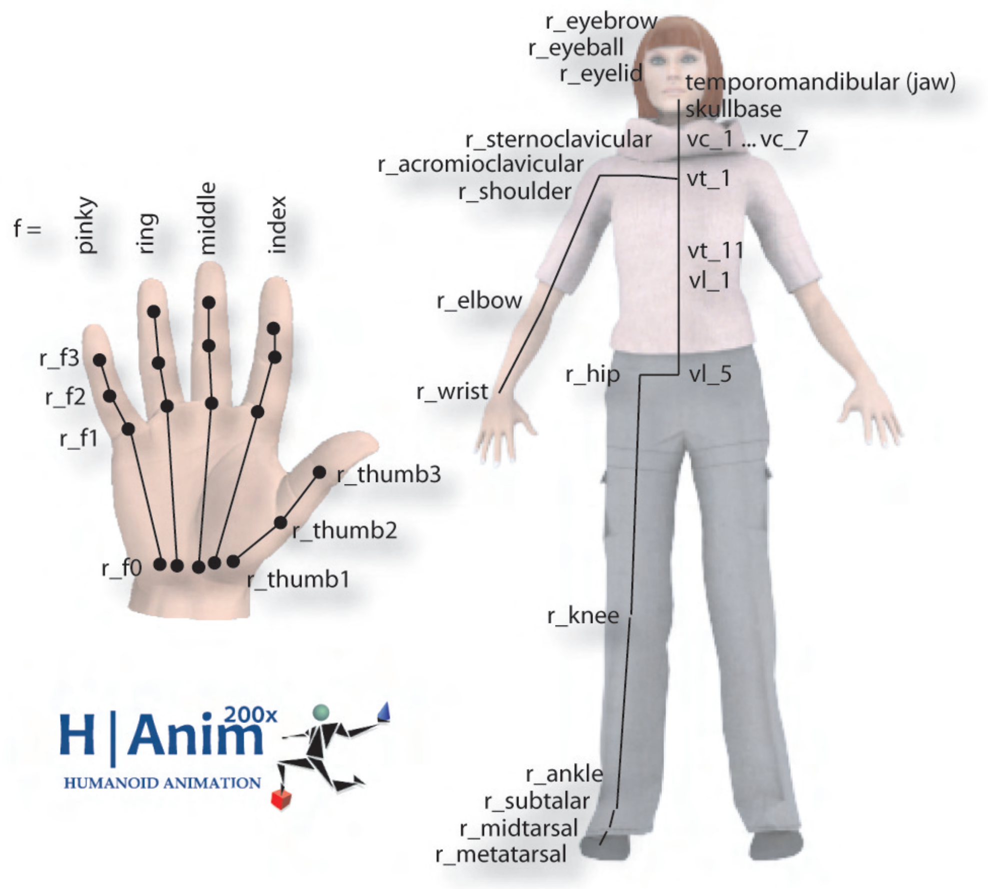


Figure 2.4: The H-Anim standard defines a joint skeleton structure for humanoids.

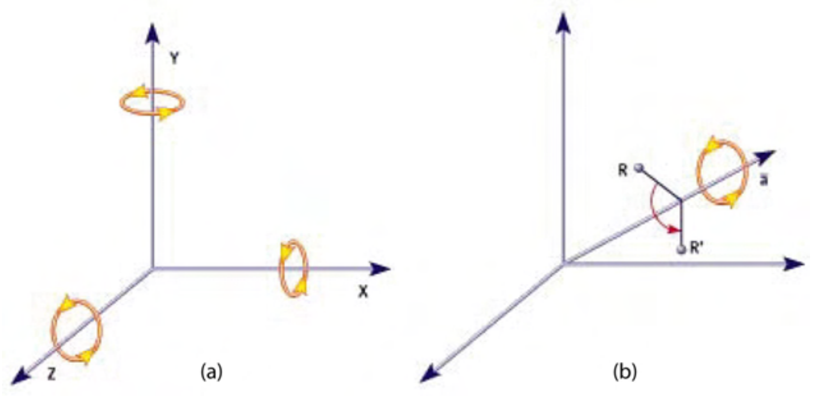


Figure 2.5: Two different angular representations of rotations: (a) Euler Angles (b) Axis Angle.

as (r_x, r_y, r_z), meaning: "Rotate a point counter clockwise around the x axis by r_x degrees, followed by a rotation around the y axis by r_y degrees, followed by a rotation around the z axis by r_z degrees. The Euler angle representation is quite efficient since it uses three variables (angles) to define three degrees of freedom. However, interpolation can be computationally expensive, since it requires numerical integration. Furthermore, Euler Angles have a Gimbal lock problem (or: the loss of one degree of freedom), that occurs when a series of rotations at 90 degrees are performed. Due to the alignment of the axes, these rotations can cancel out each other.

Another representation of rotation is called Axis Angle (see also Figure 2.5). This representation defines an arbitrary axis and a rotation around this axis. Although this representation is quite efficient, it still requires a lot of computational efforts for interpolation.

Quaternions are also a common way to represent rotations. Quaternions are actually an extension to complex numbers. Of the 4 components, one is a real scalar number w, and the other 3 form a vector in imaginary ijk space:

$$q = w + xi + yj + zk \tag{2.1}$$

From now on, we denote the complete quaternion space as $\mathbb{H}$. We define a *unit quaternion* as a quaternion with norm 1:

$$\| q \| = \sqrt{w^2 + x^2 + y^2 + z^2} = 1 \tag{2.2}$$

The unit quaternion space corresponds to the set of vectors that form the 'surface' of a 4D hypersphere of radius 1, and it corresponds exactly to the 3D rotation space. The 4D hypersphere space is generally denoted as S^3, where

$$S^3 = \{q \in \mathbb{H}; \| q \| = 1\} \subseteq \mathbb{H} \tag{2.3}$$

A quaternion can represent a rotation by an angle θ around a unit axis a:

$$q = \cos\frac{\theta}{2} + a_x \sin\frac{\theta}{2} + a_y \sin\frac{\theta}{2} + a_z \sin\frac{\theta}{2} \tag{2.4}$$

also written as:

$$q = \cos\frac{\theta}{2} + a \sin\frac{\theta}{2} \tag{2.5}$$

Given a vector $v \in \mathbb{R}^3$, it can then be projected in the quaternion space $\mathbb{H}$ by defining $v' = (0, v)$. The vector resulting from the rotation defined by a quaternion q applied on v' is then defined as $q \cdot v' \cdot q^{-1}$. There are some well-known interpolation methods for quaternions that will be covered later on. Quaternions provide for a simple way to represent rotations and they are suitable for real-time applications. A disadvantage of quaternions is that they are difficult to visualize.

Finally, rotations can be represented using the exponential map. Alexa [3] has described a method using the exponential map to allow for transformations that are performed completely in the linear domain, thus solving a lot of problems that the previous methods are suffering from. Rotations are represented by a skew-symmetric matrix. For every real skew-symmetric matrix, its exponential map is always a rotation matrix (see for example Chevalley [21]). Conversely, given a rotation matrix R, there exists some skew-symmetric matrix B such that $R = e^B$. The skew-symmetric matrix representation of a rotation is very useful for motion interpolation [95], because it allows to perform linear operations on rotations. A three-dimensional real skew-symmetric matrix has the following form:

$$B = \begin{bmatrix} 0 & -c & b \\ c & 0 & -a \\ -b & a & 0 \end{bmatrix} \tag{2.6}$$

Such an element can also be represented as a vector $r \in \mathbb{R}^3$ where:

$$r = \begin{bmatrix} a \\ b \\ c \end{bmatrix} \tag{2.7}$$

The exponential map representation r represents a rotation of $\theta = \sqrt{a^2 + b^2 + c^2}$ degrees around axis r. The exponential of a matrix B is defined by Rodrigues' formula:

$$e^B = I_3 + \frac{\sin\theta}{\theta} B + \frac{(1 - \cos\theta)}{\theta^2} B^2 \tag{2.8}$$

Similarly, methods exist that define how to calculate a determination of the (multivalued) matrix logarithm. For example, Gallier and Xu [43] present methods to calculate exponentials and logarithms, also for matrices with higher dimensions.

Although the exponential map representation of orientation does allow for easy manipulation and interpolation, there are singularities in the domain. A log map can map each orientation to an infinite number of points, corresponding to rotations of $2n\pi + \theta$ about axis v and $2n\pi - \theta$ about axis $-v$ for any $n \in \mathbb{N}$ [48]. Consequently, measures have to be taken to ensure that these singularities do not interfere with the interpolation (please see the paper by Grassia [48] for a more detailed discussion).

Interpolation

The most basic method to interpolate between two animations is linear interpolation:

$$R(t) = \alpha(t)R_0(t) + (1 - \alpha(t))R_1(t) \tag{2.9}$$

$$\text{where} \qquad \alpha(t) = \frac{t_e - t}{t_e - t_s} \qquad \text{and} \qquad t \in [t_s, t_e]$$

with $R_0(t)$ and $R_1(t)$ the two values to be combined at time t, $R(t)$ the resulting value, $\alpha(t) \in [0,1]$ the interpolation coefficient, and t_s and t_e are respectively the start and end time of the interpolation. Such linear interpolations ensure a C^0 continuity, are easy to use and are computationally cheap. Such linear interpolations are only suitable for representations of rotations in the linear domain, such as the exponential map.

However, for animation data such as rotation or position, a C^0 continuity is not sufficient due to the important impact of the velocity on the visual output. A cubic interpolation method can be of help here. A very well-known cubic interpolation technique is called cubic Hermite interpolation, which is defined as follows:

$$R(t) = h_0(t)R_0(t) + H_0(t)R_0'(t) + h_1(t)R_1(t) + H_1(t)R_1'(t) \tag{2.10}$$

where $R_0'(t)$ and $R_1'(t)$ are time derivatives of $R_0(t)$ and $R_1(t)$, e.g. linear or angular velocities. The Hermite interpolation method uses four *Hermite basis functions*. Example basis functions are:

$$\begin{cases} h_0(t) = 2\alpha^3(t) - 3\alpha^2(t) + 1 \\ H_0(t) = \alpha^3(t) - 2\alpha^2(t) + \alpha(t) \\ h_1(t) = -2\alpha^3(t) + 3\alpha^2(t) \\ H_1(t) = \alpha^3(t) - \alpha^2(t) \end{cases} \tag{2.11}$$

Since quaternions are not linear, an alternative interpolation method, such as SLERP (Spherical Linear intERPolation) [114] is required. When SLERP is applied to unit quaternions, the quaternion path maps to a path through 3D rotations in a standard way. The effect is a rotation with uniform angular velocity around a fixed rotation axis. SLERP gives a straightest and shortest path between its quaternion end points. SLERP uses the following equation to interpolate between q_0 and q_1, which interpolation coefficient t:

$$\text{SLERP}(t, q_0, q_1) = (q_1 \cdot q_0^{-1})^t \cdot q_0 \tag{2.12}$$

As can be observed, for $t = 0$ and $t = 1$, $q = q_o$ or q_1 respectively. Since rotations are represented by unit quaternions (and thus $|q| = 1$), the quaternion power is relatively simple to calculate:

$$q^t = \cos\frac{t\theta}{2} + a\sin\frac{t\theta}{2} \quad \text{for} \quad q = \cos\frac{\theta}{2} + a\sin\frac{\theta}{2} \tag{2.13}$$

When interpolating between two quaternions, it is important to make sure that the shortest possible path on the hypersphere—also called the *geodesic*—is taken. This can be achieved by negating one of the quaternions (q and $-q$ map to the same rotation) if their dot product has a negative sign.

A similar scheme to Hermite cubic interpolation exists for quaternion interpolation. This approach is called Spherical Cubic Interpolation (SQUAD) [14]. The idea behind it is to specify not only the start and end rotations, but also the rotational speeds at these points. This allows one to pass smoothly through the interpolation points by specifying the same leaving speed as your arrival speed. Spherical cubic interpolation is defined within the quadrilateral a, p, q, b as:

$$\text{SQUAD}(t, q_0, p, q, q_1) = \text{SLERP}(2t(1-t), \text{SLERP}(t, q_0, q_1), \text{SLERP}(t, p, q)) \quad (2.14)$$

where q_0 and q_1 are the start and end quaternions, and p and q are quaternions defining the path of the curve (the curve touches the midpoint of the geodesic pq when $t = 0.5$).

Deforming the Body from Skeleton Posture

The final step in obtaining a character motion is the conversion from the skeleton-based animation into a deformation of a 3D mesh, representing a virtual character. This process is generally called *skeleton-driven deformation*, or SDD. In research literature, an early version was presented by Magnenat- Thalmann et al. [85], who introduced the concept of Joint-dependent Local Deformation (JLD) operators to smoothly deform the skin surface. This technique has been given various names such as Sub-Space Deformation (SSD), linear blend skinning, or smooth skinning. Several attempts have been made to overcome the limitation of geometric skin deformation by using examples of varying postures and blending them during animation. Pose space deformation [77] approaches the problem by using artistically sculpted skin surfaces of varying posture and blending them during animation. More recently, Kry et al. [70] proposed an extension of that technique by using Principal Component Analysis (PCA), allowing for optimal reduction of the data and thus faster deformation. Sloan et al. [117] have shown similar results using RBF for blending the arm models. Their contribution lies in that they make use of equivalent of cardinal basis function. Allen et al. [4] present another example-based method for creating realistic skeleton-driven deformation. More recently, Mohr et al. [90] have shown the extension of the SDD by introducing pseudo joints. Finally, Magnenat-Thalmann et al. [84] propose an extension of the SDD that overcomes the undesirable effect of vertex collapsing (see Figure 2.6).

2.2 Performance Animation

Defining animations as joint rotations and translations as opposed to direct geometry manipulation, already gives a good basis for a real-time animation system. However, a higher level control is needed, since animating all the joints by hand still is a lot of work. Furthermore, it is difficult to create natural motions using only procedural methods,

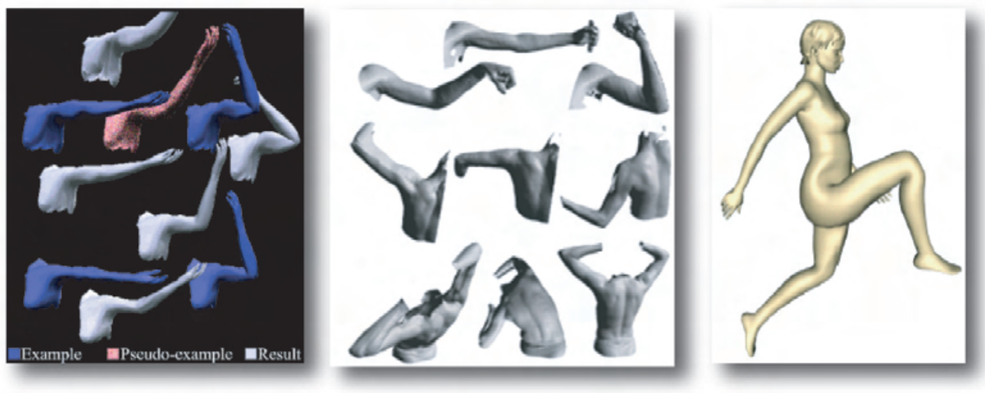


Figure 2.6: Results of skinning algorithms by Sloan et al. [117], Allen et al. [4] and Magnenat-Thalmann et al. [84].

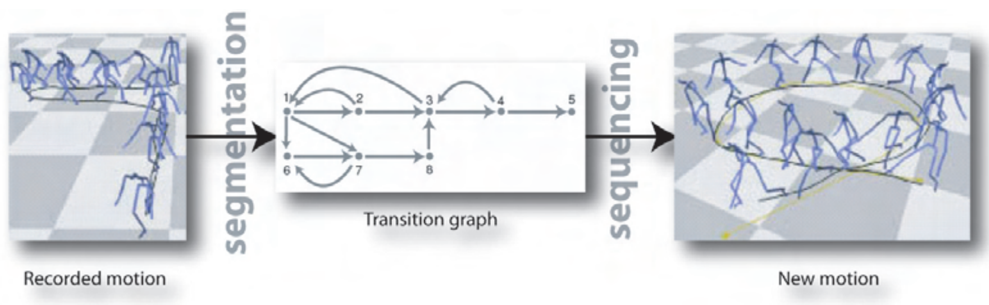


Figure 2.7: Generic pipeline in a motion synthesis system that uses motion captured segments. Motion captured data is segmented and represented in a graph like structure. New motions are then synthesized by traversing a path in the graph.

unless a lot of time is invested. An interesting approach is to create new motions using recorded motion data. This approach is called **performance animation** and its main goal is to try to combine the naturalness of recorded motions with the flexibility of procedural motion synthesis. Although recorded motions are regularly used for facial animation [72], the performance animation technique is the most popular for controlling the body. We will now present some different performance animation techniques for the body.

There is a broad number of algorithms for body motion synthesis from motion data (see Figure 2.7). All of these approaches are very similar at the basis. Mainly three steps are involved:

1. The first step is the segmentation of motion captured data into separate motion segments. Different researchers use different methods and criteria of segmentation, and it is performed either manually or automatically. Generally, the graph is extended with new transitions, using existing motion segments in the graph.

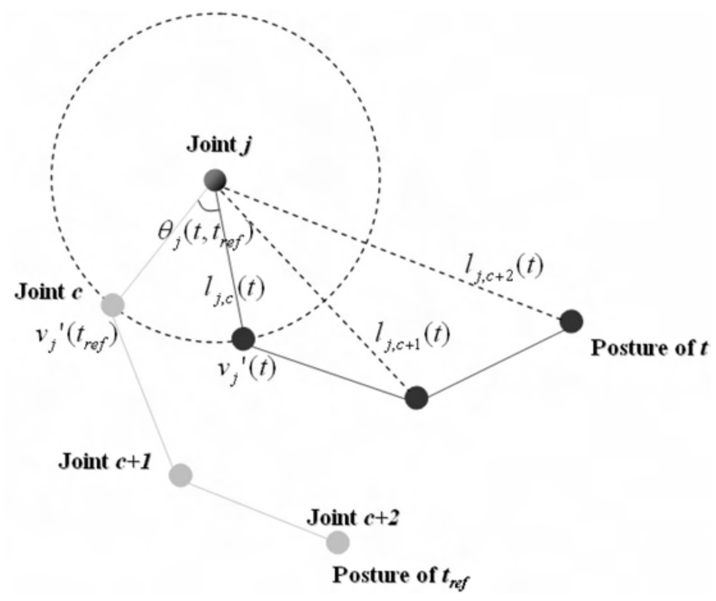


Figure 2.8: Measurement values by Ahn et al. [2].

2. The animation segments are stored in a graph-like structure. Again the connectivity and complexity of the resulting graph depends on the method that is used.

3. Finally, new animations are constructed by traversing a path through the motion graph.

We will now discuss a few research efforts in particular. Kovar et al. [66] proposed a technique called Motion Graphs for generating animations and transitions based on a motion database. The main contribution in their technique is the automatic construction of a motion graph from a collection of pre-recorded motion segments. The graph is constructed using two operations: splitting a motion segment, and adding a transition between two motion segments. Where a segment is split depends on whether or not a transition to another motion segment is favourable according to a distance criterion. This work therefore touches an important problem that arises when one wants to compare different animations: what criterion should be used to define the distance between two body postures?

Before talking about the methods proposed by Kovar et al., we will first give an overview of more straightforward methods of frame distance calculation. Several methods have been proposed to compute distances between frames different motion error metrics have been investigated. The approach proposed by Ahn et al. [2] aims to construct clusters of frames that are close to each other, based on a cost function that is defined as follows:

$$E_j(t, t_{ref}) = E_{pos,j}(t, t_{ref}) + \alpha \cdot E_{ori,j}(t, t_{ref}) \tag{2.15}$$

The cost is the sum of the position and the orientation difference. The time t is the current frame and t_{ref} the estimated key frame. The constant α is the weighting value of the orientation difference. Position and orientation difference are formulated by measurement values as shown in Figure 2.8. In addition to these measures, a weighting factor $r_j(t)$ depending on the amount of joint children of the joint, is defined as follows:

$$r_j(t) = \sum_{c}^{leaf} l_{j,c}(t) \tag{2.16}$$

The angle difference between two frames is calculated as follows:

$$\theta_j(t, t_{ref}) = \arccos \frac{v'_j(t) \cdot v'_j(t_{ref})}{\parallel l_{j,c}(t) \parallel^2} \tag{2.17}$$

where $l_{j,c}$ is the segment length between joint j and its child joint c, and v'_j the trajectory of joint j. The position error between two frames t and t_{ref} is then given as follows:

$$E_{pos,j}(t, t_{ref}) = \parallel r_j(t) \cdot \theta_j(t, t_{ref}) \parallel \tag{2.18}$$

and the orientation error is defined as:

$$E_{ori,j}(t, t_{ref}) = 2 \parallel \log(q_j(t)^{-1} \cdot q_j(t_{ref})) \parallel \tag{2.19}$$

Per cluster, they then determine the key posture by solving a lowest-cost optimization problem formulated as follows (m is the range of the optimal cluster):

$$\min_{t_{ref}} (\sum_{t}^{m} E_j(t, t_{ref})) \tag{2.20}$$

Lee et al. [75] propose a two-layer representation for motions. The higher layer is a statistical model that provides support for the user interfaces by clustering the data to capture similarities among character states. The lower layer is a Markov process that creates new motion sequences by selecting transitions between motion frames based on the high-level directions of the user. This is very similar to the motion graph approach where new motions are generated by following a path through the graph, based on high-level constraints.

The Markov process is represented as a matrix of probabilities with the elements P_{ij} describing the probability of transitioning from frame i to frame j. The probabilities are estimated from a measure of similarity between frames using a exponential function:

$$P_{ij} = e^{\frac{D_{i,j-1}}{\sigma}} \tag{2.21}$$

The distance between frame i and frame j is defined as follows:

$$D_{ij} = d(p_i, p_j) + \nu d(v_i, v_j) \tag{2.22}$$

The first term $d(p_i, p_j)$ describes the weighted differences of joint angles, and the second term $d(v_i, v_j)$ represents the difference of joint velocities. Parameter ν is a weight value. The position distance is computed as follows:

$$d(p_i, p_j) = \parallel p_{i,0} - p_{j,0} \parallel^2 + \sum_{k=1}^{m} w_k \parallel \log(q_{j,k}^{-1}, q_{i,k}) \parallel^2 \tag{2.23}$$

where $p_{i,0} \in \mathbb{R}^3$ is the root position at frame i and $q_{i,k} \in S^3$ is the orientation of joint k with respect to its parent in frame i. The joint rotations are summed over m joint orientations, with manually defined weights w_k for each joint. Both of these methods use weighted joint angle differences as a basis for the distance criterion, which on the first impression does seem like a logical choice. However, Kovar et al. [66] notice several problems with such joint-based approaches:

1. Simple vector norms fail to account for the meanings of the parameters. Specifically, in the joint angle representation some parameters have a much greater overall effect on the character than others (e.g., hip orientation vs. wrist orientation). Moreover, there is no meaningful way to assign fixed weights to these parameters, as the effect of a joint rotation on the shape of the body depends on the current configuration of the body.

2. A motion is defined only up to a rigid 2D coordinate transformation. That is, the motion is fundamentally unchanged if we translate it along the floor plane or rotate it about the vertical axis. Hence comparing two motions requires identifying compatible coordinate systems.

3. Smooth blends require more information than can be obtained at individual frames. A seamless transition must account not only for differences in body posture, but also in joint velocities, accelerations, and possibly higher-order derivatives.

Instead of using the joint orientation differences to estimate the pose, Kovar et al. propose another distance metric. They use a point cloud that assumes the shape of the body posture. Additionally, in order to address the second point in the distance criterion selection, they propose the calculation of the distance as an minimal cost problem:

$$\min_{\theta, x_0, z_0} \sum_i w_i \parallel p_i - T_{\theta, x_0, z_0} p_i' \parallel^2 \tag{2.24}$$

using the variables x_0, z_0 and θ (position and orientation in the frontal-lateral plane). Fortunately, there is a closed form solution to this optimization problem:

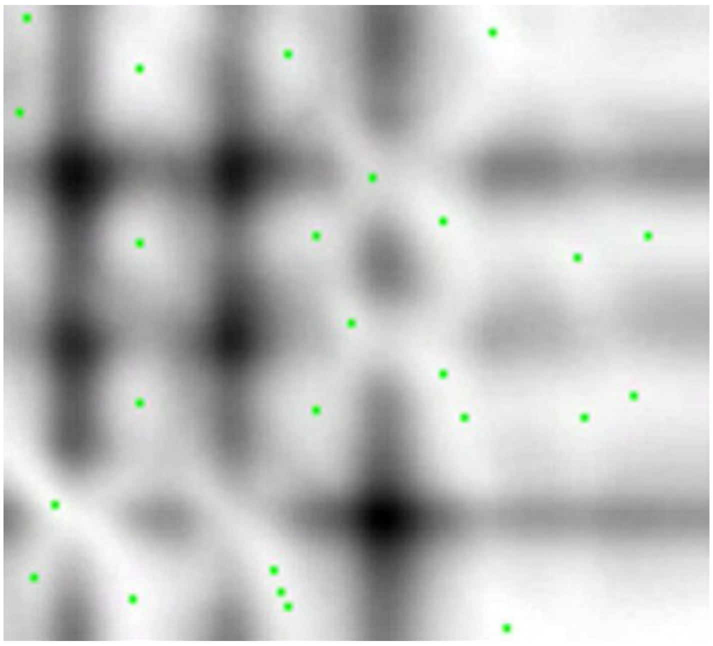


Figure 2.9: Example error function output for a distance comparison between two animations. The points indicate error minima and they correspond to frame combinations that could be used to form a transition [66].

$$\theta = \arctan \frac{\sum_i w_i(x_i z'_i - x'_i z_i) - \frac{1}{\sum_i w_i}(\overline{x}\overline{z'} - \overline{x'}\overline{z})}{\sum_i w_i(x_i x'_i + z'_i z_i) - \frac{1}{\sum_i w_i}(\overline{x}\overline{x'} + \overline{z}\overline{z'})} \tag{2.25}$$

$$x_0 = \frac{1}{\sum_i w_i}(\overline{x} - \overline{x'}\cos\theta - \overline{z'}\sin\theta) \tag{2.26}$$

$$z_0 = \frac{1}{\sum_i w_i}(\overline{z} + \overline{x'}\sin\theta - \overline{z'}\cos\theta) \tag{2.27}$$

where $\overline{x} = \sum_i w_i x_i$ and the other barred terms are defined similarly. Using this distance criterion, it is now possible to define an error function that displays the distance between each pair of frames for two animations. A segmentation and transition is then chosen according to the minima of this error function (the green dots in Figure 2.9), in combination with a threshold error value.

Similar to the point clouds of Kovar et al., Loy *et al.* [80] compute distance matrices with shape matching between each pair of frames of an animation. Both families of approaches have advantages and drawbacks. Although joint-based methods are faster than geometry-based methods, the latter one provides a more precise distance. However, geometry-based methods are computationally expensive.

Next to the previously mentioned approaches, there is some similar work that tries to segment pre-recorded motion clips in order to automatically create new motions. Li et al.[78] propose a method that ‘learns’ a motion texture from sample data. Using the motion texture, they then generate new motions. This method only synthesizes motions that are similar to the original motion. The system is mostly useful for rhythmic motions with repeating segments, such as dance. Similarly, Kim et al.[63] propose a method of control for motions, by synthesizing motions based on rhythm. Pullen and Bregler [106] proposed to help the process of building key-frame animation by an automatic generation of the overall motion of the character based on a subset of joints animated by the user. There the motion capture data is used to synthesize motion for joints not animated by the user and to extract texture (similar to noise) that can be applied on the user controlled joints. Finally, relevant work has been done by Arikan et al. [9, 10] that defines motion graphs based on an annotated motion database. They also present an automatic annotation method that can be used to segment motion capture data automatically. However, since motion captured clips often contain unwanted joint movements, clear constraints have to be defined during the annotation process.

Discussion and Open Problems

Although the motion graph technique can be very useful in some applications, there are some limitations to the method that make it less suitable for interactive characters:

- Due to the choice of using point clouds, the distance criterion is a computationally expensive one. This is not necessarily a problem, because the distances and transitions only need to be calculated once.
- When traversing the motion graph, a set of constraints is required that defines what path should be taken. The choice of such constraints will have a big influence on the real-time performance.
- The vertices and edges do not have any meaning, since they are generated directly from the error function. This will make it more difficult to control the animation.
- The motion graph approach needs precise constraints. Next to the begin and end points of the animation, the path itself should be defined. When we need to control a character from a higher level, such precise constraints may very well not be available.

As we will show in the following section, the control of the character is very important for IVH systems. This means that some crucial problems need to be overcome in order for motion synthesis techniques to be useful for our intended application.

2.3 Interactive Virtual Human Systems

In this section, we will discuss systems that simulate interactive virtual humans. We will first start by giving an introduction to dialogue management systems. These systems are responsible for selecting an appropriate response, given an event. Then, we will present an overview of the research in personality and emotion simulation. Section 2.3.3 will provide an overview of systems that control face and body movements of virtual characters.

2.3.1 Interaction

Dialogue Management Systems

One of the first attempts to create a system that interacts with a human through natural language dialogue, was ELIZA [127]. It used pattern-matching techniques to extract information from a sentence. ELIZA had no sophisticated dialogue model; it just had a simple trigger-reaction mechanism. Although such a system is very limited and dialogue systems have evolved a lot since then, this approach is still popular, because of the ease of implementation and use. A well-known extension of the ELIZA system is Alice [1]. Alice uses a pattern-response mechanism coded in AIML (Artificial Intelligence Modelling Language), an XML-based language. However, in many dialogue systems, a simple question-answer structure is not sufficient. As a result, there are many different systems developed that used finite state machines [25]. Also, dialogue systems are being integrated with more generic models of agents, such as the BDI model [15, 107]. BDI stands for belief, desire, intention, which are the three possible kinds of statements that can occur in an agent's state. A lot of BDI systems use some kind of extension of logic to express beliefs, desires and intentions. This is called a BDI logic (for an example of this, see the text book by Wooldridge [130]). Examples of applications that use the BDI model to develop a conversational agent are the work done by Sadek et al. [110] and Ardissono et al. [7]. Finally, there is a more or less black box approach to dialogue management, that uses statistical data obtained from a corpus of existing conversations to select the best answer given an input from the user, see for example Jabberwacky [56].

Discussion and Open Problems

Although many different approaches to dialogue management exist, there are only a few systems that try to incorporate the control of virtual characters. Also, emotions and personality are generally not included in dialogue systems, although there are some systems that propose a so-called 'BDI-E' model, which is an extension of the BDI model with emotions. Recent work by Och et al. [92] shows an expressive face, that is linked with such an emotion model. However, this system only works for the face and there is not yet any control over the body.

In the next section, we will discuss the theory of personality and emotion and how that can be used as a part of IVHs. Then we will present various systems for simulating the visual front-end (both body and face) for IVHs.

2.3.2 Personality and Emotion Simulation

Recent research in the area of interaction has acknowledged the importance of emotions as a crucial part of IVHs [100]. In this section, we will give a short overview of the research related to personality and emotions.

When discussing theory and simulation of personality and emotion, we have to address work done in both Computer Science and Psychology. Psychology tries to discover the *nature* of emotions and personality. It describes the structure of each of these notions, how it can be attributed and what their effect is on human behaviour. Computer Science tries to simulate the effects that personality and emotion has on human beings and use it to make Human Computer Interaction (HCI) more natural. The research that is a part of HCI and that tries to use emotion—and in a lesser way personality—to increase naturalness of HCI, is called Affective Computing [100].

Personality

The study of personality investigates how individuals differ from each other. The most commonly accepted form of personality study is based on trait theory. Personality traits are relatively stable dispositions that give rise to characteristic patterns of behaviour [51]. Although trait-based personality models have been criticized in the past [87], they are still one of the most popular representations of personality. Among personality researchers, there is still debate on how many traits are needed to provide a comprehensive description of personality. For example, Eysenck proposes three traits of personality: extraversion, neuroticism and psychoticism [41, 42], whereas Cattell et al. advocates 16 traits [20]. However, the most widely accepted theory is a structure consisting of five factors (also called the Big Five)[2]. One of the advantages of the Big Five framework is that it can assimilate other structures. An empirical demonstration of this property is given by Goldberg and Rosolack [47] by integrating Eysenck's three-factor system into the Big Five. Each of the five dimensions is defined by its desirable and undesirable qualities. Table 2.1 summarizes the Big Five model and the desirable and undesirable qualities of each trait.

Not all personality researchers believe that personality is a static set of traits that can be described independently of behaviour and situations. In fact, Mischel and Shoda [88, 89] have developed a personality theory, that accounts for variability patterns across different situations. Although the trait-based approach to modelling personality does

[2]See for example Digman [32], Goldberg [46] and Costa and McCrae [28].

Trait	Desirable	Undesirable
Extraversion	outgoing, sociable, assertive	introverted, reserved, passive
Agreeableness	kind, trusting, warm	hostile, selfish, cold
Conscientiousness	organised, thorough, tidy	careless, unreliable, sloppy
Emotional stability	calm, even-tempered, imperturbable	moody, temperamental, nervous
Intellect or Openness	imaginative, intelligent, creative	shallow, unsophisticated, imperceptive

Table 2.1: The Big Five personality traits and their desirable and undesirable qualities (from Hampson [51]).

have some disadvantages (for a more detailed discussion of this see Hampson [51]), it is still quite useful for IVH development, because the concept of personality traits can be easily translated into a computational model (for an example, see Johns and Silverman [58]). Furthermore, a lot of resources (such as five factor personality tests) are available, so that any research performed can be easily evaluated and adapted.

The effect of personality on human motion has been researched as well. Kozlowski and Cutting [31, 67] have shown in their work on the perception of human walking that one can recognise the gender and sometimes even the person from a walking human. They use point-light displays to remove any familiarity cue, so that their experiments only concern the movement itself. Animating 3D humanoids with motion captured data has a similar effect.

Emotion

The concept of emotion has been widely researched in the psychology field. Many approaches and theories exist, but according to Cornelius [27], they can be broadly organised in four different theoretical perspectives: the Darwinian, Jamesian, cognitive, and social constructivist perspectives.

According to the **Darwinian perspective**, emotions are based on important survival functions that have been selected because they have solved certain problems we have faced as a species. As such, we should see the same emotions, more or less, in all humans. Many recent research efforts are following Darwin's approach in trying to understand emotions from an evolutionary perspective. For example, the work by Ekman [37] or LeDoux [74] can be named. Notably, Ekman has mostly been working on demonstrating the universality of certain human facial expressions of emotion.

Theory and research from the **Jamesian perspective** follows the perspective that it is impossible to have emotions without bodily changes and bodily changes always come first. Like Darwin, James considered emotions to be more or less automatic responses to events in an organism's environment that helped it to survive. Studies by Levenson et al. [76], Laird et al. [73], and Strack et al. [120], have shown that emotions indeed follow facial expressions. Stepper and Strack [118] have shown that postural feedback may drive emotional experience as well.

The **cognitive approach** to the study of emotions starts with the work of Arnold [11]. The central assumption of the cognitive perspective is that thought and emotion are inseparable. More specifically, all emotions are seen within this perspective as being dependent on *appraisal*, the process by which events in the environment are judged as good or bad for us. The most well-known attempt to construct a computational model of appraisal is the work by Ortony, Clore and Collins [93]. Their model of appraisal is called the OCC model. This model specifies how events, agents and objects from the universe are used to elicit an emotional response depending on a set of parameters: the goals, standards and attitudes of the subject. The appraisal approach shows the importance of the **dynamics** of emotions. The work by Scherer [112] discusses the actual processes that form the appraisal of an event or a situation, and how that affects the emotional state.

Finally, the **social constructivist** approach follows the idea that emotions are cultural products that owe their meaning and coherence to learned social rules. Culture, for social constructivists, plays a central role in the organization of emotions at a variety of levels. Most importantly, culture provides the content of the *appraisals* that generate emotions. The most relevant studies based on this approach are the work by Averill [12] and Harré [52].

In each of these perspectives, it is generally assumed that an emotional state can be viewed as a set of dimensions. The number of these dimensions varies among different researchers. Ekman [37] has identified six common expressions of emotion: fear, disgust, anger, sadness, surprise and joy, Plutchik [104] proposes eight (see Figure 2.10), and in the OCC appraisal model [93], there are 22 emotions. We would also like to mention a 2-dimensional emotion representation called the **activation-evaluation space** [113], which defines emotions along two axes on a circle (see Figure 2.11), where the distance from the centre defines the power of the emotion. The advantage of this approach is that different discrete emotions can be placed on the disc [30], which provides for a possibility to link the activation-evaluation model with different multidimensional emotion models, such as the OCC emotions or Ekman's expressions.

Discussion and Open Problems

There is a lot of ongoing research that tries to include emotion/personality models as a part of the decision process [101]. For example, the OCC model has been implemented as a part of a affective reasoning system in the PhD work done by Clark D. Elliott [40]. On the level of implementation, there are various approaches to how emotions are implemented as a part of an interactive system, whether it concerns a general

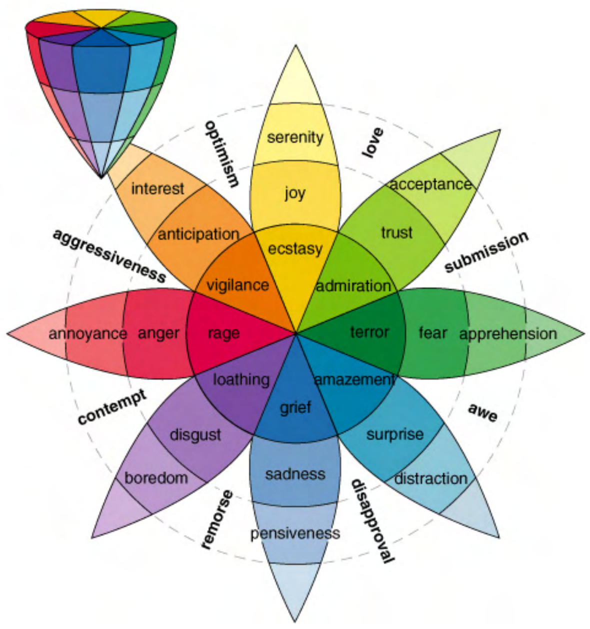


Figure 2.10: Plutchik [104] defines eight basic emotions. He states that emotions are like colors. Every color of the spectrum can be produced by mixing the primary colors.

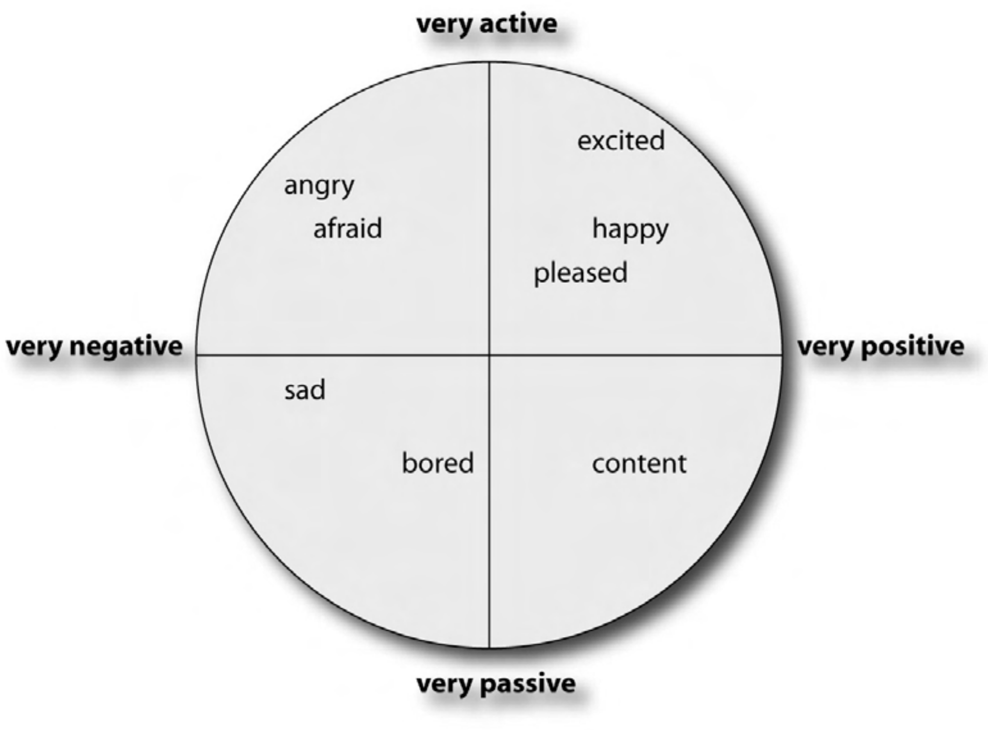


Figure 2.11: Activation-evaluation emotion disc.

influence on behaviour [86], or a more traditional planning-based method [58]. Also, rule-based models [5], probabilistic models [13, 23] and fuzzy logic systems [39] have been developed.

In the next section, we will present and discuss the existing research in the creation of Interactive Virtual Humans. As will be shown, the expression of emotions is mainly done by synthesizing facial expressions as a part of the interaction [72, 71]. The work by Mark Coulson [29] however shows that people are indeed capable of recognising emotions from **body postures**. Currently, Interactive Virtual Human systems do not yet include the expression of emotions through different body postures. Some work has been done by Sloan et al. [116] to synthesize happy or sad walking sequences. Also Unuma et al. [122] have defined an emotion-based character animation system. Both these approaches focus mainly on happy/sad or brisk/tired walking sequences, which is still very far from a complete emotional body animation system. Another important point is that there is no consensus within the IVH research community on which emotion model to use, which makes it difficult to compare the different systems and to define a 'best practice'. Finally, most systems do not include a personality model, and there are not many systems that implement an integrated personality/emotion approach to face and body animation.

2.3.3 Expressive Agents

In this section, we will discuss different approaches to the expression of behaviour by virtual agents. First, we will describe methods for automatically synthesizing expressive body behaviour. Then we will describe methods for synthesizing expressive face behaviour.

Posture and Gesture Synthesis

While engaged in a conversation, a human can produce gestures **consciously** or **unconsciously**. Conscious gestures are composed of two groups [8]: *emblematic gestures* and *propositional gestures*. Emblematic gestures can have a different meaning depending on the culture. Examples of western emblematic gestures are the thumb-and-index-finger ring gesture that signals *okay* or *good* or the thumbs-up gesture. Propositional gestures relate to size, shape, direction or relation, for example: pointing at an object with the index finger, or using the hands to show the size of an object.

The gestures that are produced unconsciously, are of four types [16]:

Iconic gestures the form of the gesture describes a feature, action or event; for example turning the wrist while the fingers are bent to depict a 'pouring' action;

Metaphoric gestures similar to iconic gestures, only the concept that they represent has no physical form; for example a rolling motion of the hand while saying 'the meeting went on and on';

Deictic gestures these gestures visualize different aspects of the discourse in physical space in front of the narrator; for example pointing to a fictional position to the left, while saying '...so when the train arrived in Geneva...';

Beat gestures small baton like movements that mainly serve a pragmatic function as an orienting or evaluative comment on the discourse; for example saying 'she talked first, I mean second' accompanied by a hand flicking down and then up.

There are several research endeavours towards the development of a gesture synthesizer. The BEAT project [19] allows animators to input typed text that they wish to be spoken by an animated human figure, and to obtain as output speech and behavioural characteristics. Another project at MIT that concerns gesture synthesis is REA [17]. REA (or: the Real Estate Agent) is based on similar principles as the BEAT system. The MAX system, developed by Kopp and Wachsmuth [64], automatically generates face and gesture animations based on an XML specification of the output. This system is also integrated with a dialogue system, allowing users to interact with MAX in a construction scenario. Finally, we would like to mention the work of Hartmann et al. [53, 54], which provides for a system—called GRETA—to automatically generate gestures from conversation transcripts using predefined key-frames. Examples of the visual results of these systems are shown in Figure 2.12.

These systems produce gesture procedurally. As such, the animator has a lot of control on the global motion of the gesture, but the resulting motions tend to look mechanic. This is the effect of an interface problem that exists between high-level gesture specifications and low-level animation. When humans move their arm joints, this also has an influence on other joints in the body. Methods that can calculate dependent joint motions already exist, see for example the research done by Pullen and Bregler [106]. In their work, they adapt key-framed motions with motion captured data, depending on the specification of which degrees of freedom are to be used as the basis for comparison with motion capture data. However the method they propose is not fully automatic: the animator still needs to define the degrees of freedom of the dependent joints.

Recent work from Stone et al. [119] describes a system that uses motion capture data to produce new gesture animations. The system is based on communicative units that combine both speech and gestures. The existing combinations in the motion capture database are used to construct new animations from new utterances. This method does result in natural-looking animations, but it is only capable of sequencing pre-recorded speech and gesture motions. Many IVH systems use a text-to-speech engine, for which this approach is not suitable. Also, the style and shape of the motions are not directly controllable, contrary to procedural animation methods.

Especially when one desires to generate motions that reflect a certain style, emotion or individuality, a highly flexible animation engine is required that allows for a precise definition of how the movement should take place, while still retaining the motion realism that can be obtained using motion capture techniques. The EMOTE model [22]

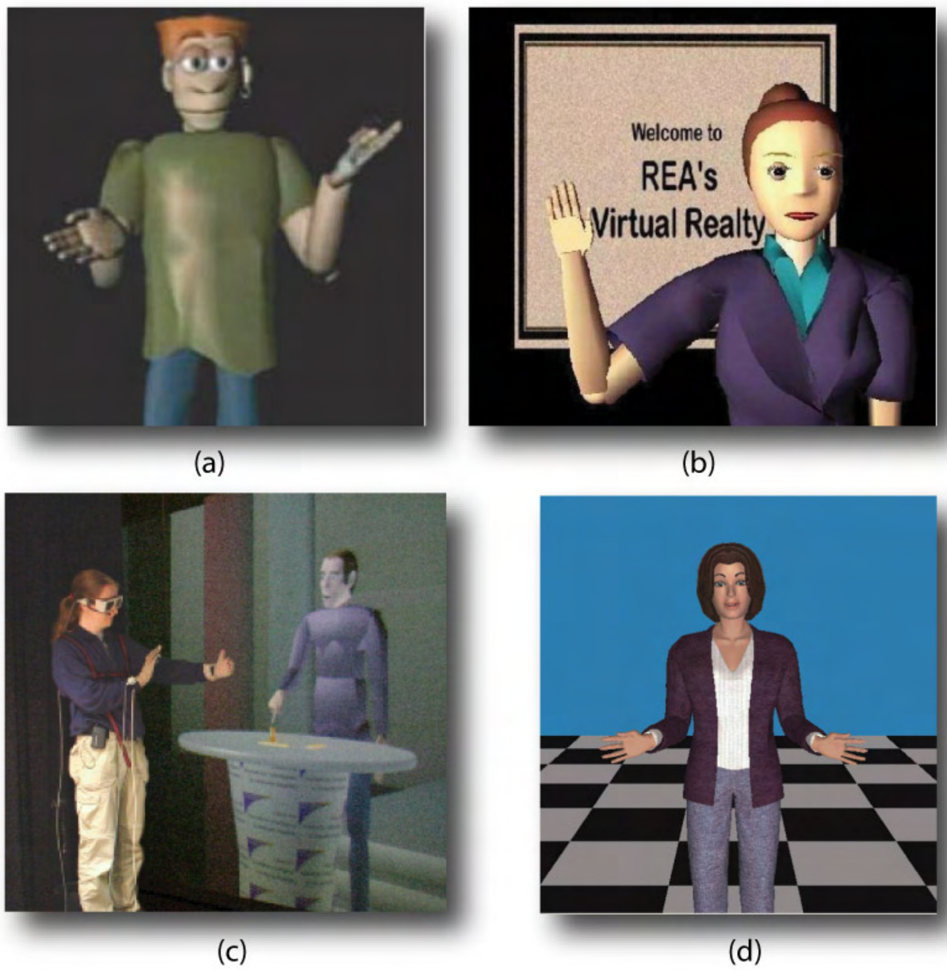


Figure 2.12: Example image results of different gesture synthesis systems: (a) the BEAT system [19] (b) the REA system [17] (c) the MAX system [64] (d) the GRETA system [54].

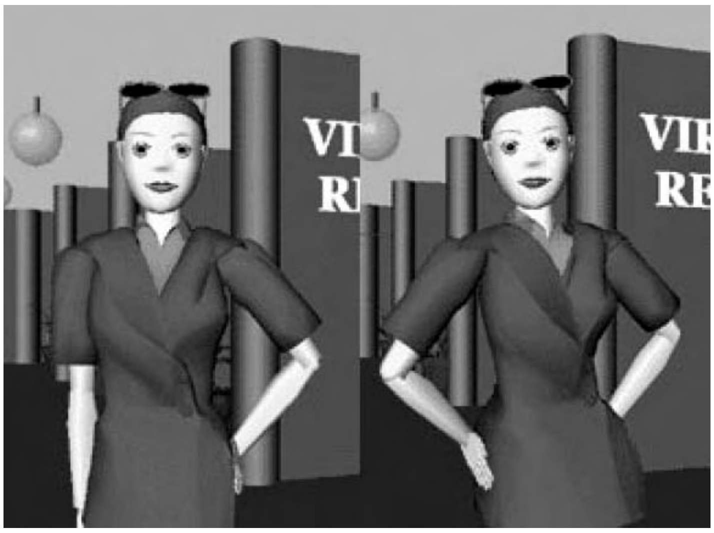


Figure 2.13: Some example postures of the Real Estate Agent [17, 18].

aims to control gesture motions using effort and shape parameters. As a result, gestures can be adapted to express emotional content or to stress a part of what is communicated. Currently no method exists that allows such expressive gestures, while still having a natural-looking final animation.

Most of the previously mentioned systems only focus on gestures performed by the hand and arms. However, there are many indications that the whole body is used during communications. For example, Kendon [62] shows a hierarchy in the organization of movements such that the smaller limbs such as the fingers and hands engage in more frequent movements, while the trunk and lower limbs change relatively rarely. More specifically, posture shifts and other general body movements appear to mark the points of change between one major unit of communicative activity and another [111]. Also, posture shifts are used as a means to communicate *turn-taking*[3] during dialogues [49]. Cassell et al. [18] describes experiments to determine more precisely when posture shift should take place during communication and they applied there technique to the REA Embodied Conversational Agent [17], resulting in a virtual character being capable of performing posture shifts (of limited animation quality, however). Figure 2.13 shows some example images of postures in the REA system.

Facial Expression and Speech Synthesis

In Section 2.1.1, we have discussed different approaches to simulate facial movements on a 3D face model. Also, we have shown some methods to parameterize these approaches so that the control mechanism of the face (for example MPEG-4) is independent of the 3D model. However, a higher level control is required when one wants to

[3]They are called *situational shifts* by Blom and Gumperz [49].

control a face as a part of an IVH. One of the main output formats of an IVH is *speech*. Therefore a mechanism is required that can convert this speech into an accurate facial animation signal. Additionally, emotional expressions need to be blended with a speech signal, as well as other face motions, such as head movements, gaze and eye blinking. More specifically, Ekman [37] defines several different groups of facial expressions during speech: emblems, emotion emblems, conversational signals, punctuators, regulars, manipulators and affect displays. These groups serve either as communicative signals or as movements corresponding to a biological need (eye blinking, wetting the lips, and so on).

For a more detailed description of facial animation systems and their integration with speech, we refer to the work done by Kalra and Garchery [60] and Kshirsagar [72, 71] at MIRALab—University of Geneva.

Virtual Human Control Mechanisms

In order to control IVHs, a language is necessary that describes what the IVH should do exactly. The level on which this language should specify what the agent should do, depends on the design and requirements of the intended application. For example, some systems produce gestures procedurally using very detailed tags in the utterance specification, whereas other systems require simply a high-level indication of which animation file to load. Also, some languages might depend on the technology that is behind them. Because of all these variables, a lot of different mark-up languages for IVHs exist and they are all on different levels of abstraction and restrained to a certain domain of applications. For a recent overview of such languages, we refer to the paper by Arafa et al. [6].

Since our work focuses on the application of animation techniques in Interactive Virtual Humans, a lot of these mark-up languages are not suitable since they are either on a too conceptual level or they do not allow for the specification of multimodal discourse elements. The two mark-up languages of particular interest are RRL [102] and MURML [68]. MURML is the mark-up language that is used to control the MAX multimodal agent [64]. This language provides for a specification of a discourse utterance (text to be spoken by an IVH), in synchrony with gesture motions. The gesture motions are defined in the XML structure and are generated afterwards by a gesture synthesizer (see Figure 2.14). RRL defines the multimodal control on different levels. An example of the input to an animation system is given in Figure 2.15. The GRETA and BEAT systems define a very similar kind of language.

Discussion and Open Problems

Most of the IVH systems discussed in this section have a rather detailed way of specifying the body and face animation sequence to be played. This high level of control

```
<utterance>
  <specification>
     And now take <time id="t1"/> this bar. <time id="t2"/>
  </specification>
  <behaviorspec id="gesture_1">
    <gesture>
      <affiliate onset="t1" end="t2" focus="this"/>
      <function name="refer_to_loc">
        <param name="refloc" value="Loc-Bar_1"/>
      </function>
    </gesture>
  </behaviorspec>
</utterance>
```

Figure 2.14: Example of a multimodal sequence of text to be spoken in synchrony with gestures using MURML.

```
<animationSpec>
   <seq>
      <par>
         <gesture begin="0" dur="400"
            id="g3" identifier="f_smileopen"/>
         <par>
            <gesture begin="0" dur="655"
               id="g4" identifier="g_positive_greeting"/>
            <audio dur="1502" src="hello.mp3"/>
         </par>
      </par>
   </seq>
</animationSpec>
```

Figure 2.15: (Simplified) example of an audio signal to be played in synchrony with gestures using RRL.

is mostly visible in the way that the arm and hand motions are being constructed. Although a precise control of such motions is desirable in order to provide a strong link with for example a dialogue system, the resulting animations do not look very natural since they are only defined for a few joints. Also, because these systems do not take the whole body into account (except for REA in a limited way), the 3D characters are rather static during the interactions. Currently, there is no IVH system that can handle detailed face and body gesture definitions and that will produce a natural looking animation of the character.

2.4 Motivation

In the previous sections, we have discussed many different aspects of research related to the simulation of Interactive Virtual Humans. Since IVHs are an integration of many different research efforts, it is difficult to define a coherent goal by only looking at one research area. On the other hand, when looking at the virtual humans themselves in such systems, it is clear that many problems still need to be overcome. We have seen in the previous sections, that current IVH systems can generate detailed upper body motions, but that those motions are generally only defined for a few joints. Additionally, these motions are often generated procedurally, and thus lack the precision and naturalness of real human motions. As a result, the generated motions generally do not look convincing. We have presented several candidate techniques that try to use motion captured clips to produce new motions, according to some constraints. We believe that such techniques are very promising for a wide variety of uses, including interactive virtual humans. However, many of such systems have strong limitations to the level of control over the character, as we have discussed in Section 2.2. The problem with IVHs is exactly that they require a *very high* level of control over their motions.

In this thesis, we set out the goal to develop a technique that has sufficient flexibility in order to be used for IVHs, but that allows for the synthesis of motions that are *realistic* [35, 34]. Such a goal is particularly challenging because generally IVH systems are already computationally expensive, since they will often include speech processing and synthesis software, a dialogue manager, a personality and emotion simulator, and many more components. Each of these components is inspired from different fields of research and has its own complex architecture. If one does not pay extreme attention, placing many of such components together in one system will result in a software with astronomical computational requirements. Indeed, if we intend to develop an animation system for real-time interactive virtual humans, then it is indispensable that such a system is as fast as possible and that the architecture behind the system is logical and transparent. If we want to adapt and sequence pre-recorded motion clips in real-time, we can't simply use traditional methods for motion synthesis as described before, because these methods are not fast enough. Not only do we need to adapt motion clips, but our animation system will also need to handle different animation tracks playing at the same time, synchronization between animation and speech, and certainly many other operations that allow for the level of control required by IVHs. The only way to achieve such complex animation management in real-time, is to come up with an animation representation that is extremely fast, but flexible enough to be suitable for the job.

In MIRALab—University of Geneva, the research done by Kshirsagar [72, 71] and Kalra and Garchery [60, 44] provided for a real-time facial communication system, that was integrated with a dialogue manager and a speech synthesizer. At the same time, VHD++ [105] was being developed by both MIRALab and VRLab—EPFL. VHD++ is a real-time scenario simulation tool for multiple virtual humans in a virtual environment. For each virtual human, a scenario can be defined that plays facial animations, body animations, and audio files at the same time. This system was quite successfully used in the LIFEPLUS project[4], where a scenario and environment were created to simulate the life in the ancient city of Pompeii [94]. However, there was no integration between the facial communication system developed by Kshirsagar and the VHD++ system. This was mainly because Kshirsagar's system was designed from an interactive point-of-view, where motions and actions need to be shown on-the-fly, whereas playing predefined scenarios is a very different type of approach. Since European research projects are a very important driving force behind the research done in MIRALab, it was clear that any full-body and face interactive system would need to be integrated in the VHD++ framework, so that it could be used in demonstrations of such projects. However, neither the animation manager inside VHD++ nor the system responsable for creating the expressive content was suited for controlling both the body and face of an interactive human. In this thesis, we will need to propose a more generic animation manager that can handle both the scenario-based animations as well as the interactive virtual human animations. Additionally, if we want the system to be usable as a part of the various project demonstrations, it is necessary that such a system can dynamically switch between the different animation types without having to interrupt the animation cycle.

2.5 Specific Objectives

Given the previously mentioned considerations, we will now define the objectives of this thesis more specifically. As discussed before, performance animation methods provide for natural looking motions, but they provide only a limited level of control over the final animation. On the other hand, current IVH systems generate detailed upper body motions, which are not very realistic because of the procedural approach.

[4]LIFEPLUS is a European Union funded project (contract number IST-2001-34545). Project homepage: http://lifeplus.miralab.unige.ch.

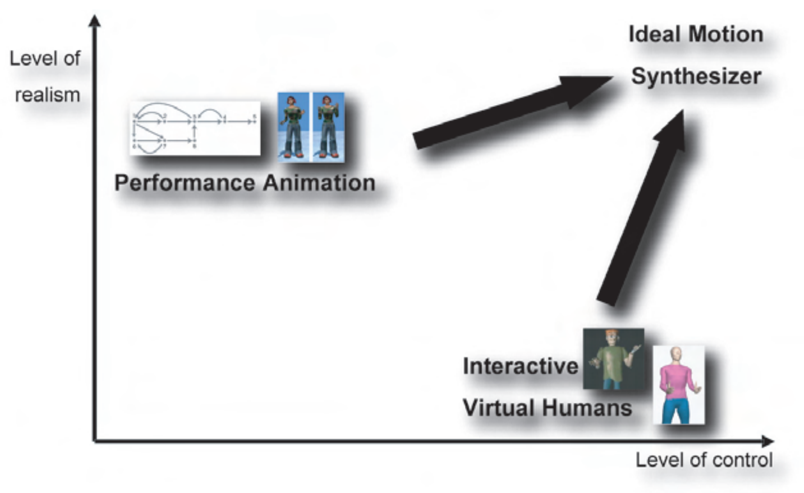


Figure 2.16: Performance animation methods and IVH simulation system, in a graph depicting control versus realism of animations.

In order to create the *ideal motion synthesizer* for a believable IVH, one would need a system that allows for both a high level of control and a high level of realism (see Figure 2.16). The global goal of this thesis is to develop techniques that allow us to move more into the direction of this ideal motion synthesizer, while keeping in mind the real-time constraint. In order to achieve this global goal, we will propose several specific contributions to the state of the art in various research areas:

Real-time animation manipulation We will propose a novel body animation representation that allows for fast and reliable real-time manipulation. The representation will be suitable for linear operations, just like the exponential map (see Section 2.1.2), but with the additional advantage that only a small subset of the posture vector is required for many applications, thus greatly increasing the efficiency of animation handling. This animation representation will be encapsulated in a real-time animation manager that will be able to handle any kind of motion, whether it is generated on-the-fly using existing motions, generated automatically by a gesture synthesis system, or keyframe animations. The manager will be usable for different types of animation control and it allows to dynamically switch between different motion control techniques without interrupting the performance of the system during run-time, as opposed to existing techniques that focus on either scenario-based animation or interactive character animation.

Flexible motion synthesis We will present a flexible motion synthesizer that fully automatically adapts recorded motion clips to create new, realistic motions. The motion synthesizer will be able to adapt and sequence motions in real-time by using a fast frame distance criterion that profits from the animation representation that we propose. As an advantage over previously discussed distance criteria [2, 66, 80], our approach will be both fast and taking into account dependencies between joints. In contrast to the existing approaches [66], our motion synthesizer will use a graph with meaningful vertices. This results in a more precise control over the target motion. Additionally, no pre-computation is required: the motions are adapted and generated during run-time, due to the efficient distance criterion and rapid motion fitting algorithm.

Automatic realistic gestures We will show a new technique that allows for creating realistic gesture motions in real-time from procedurally generated motions defined only for a few joints. We will present a method for automatically calculating the motions of dependent joints, at an extremely low computational cost. Existing methods for generating realistic gestures, such as the one proposed by Stone et al. [119] are only able to generate gestures from a pre-recorded database of motions. Our approach will not place any limitations on the input motion, making the technique suitable to be used with any type of gesture synthesizer.

Full-body expressive interaction As a proof of concept for the adequacy of the animation manager, we will develop a dialogue manager that is capable of controlling both face and body motions. A simple interface will be provided that allows for a coherent definition of face and body motions, which will be automatically synchronized with a speech signal, created using a text-to-speech synthesizer. This will allow an IVH to respond to a user, by employing full face and body motions. Since emotional expressions on both the face and the body need to be governed, we will present an efficient emotion and personality simulator that is suitable to be used for interactive virtual characters. Because of the generic approach, the emotional state can be used to drive both face and body animation.

The only way to demonstrate that our approach is valid, is by developing a prototype that shows the various features of the system. We will build a prototype IVH that includes speech recognition, speech synthesis, emotion and personality simulation, dialogue management, and fully automatic realistic face and body motion synthesis. All these system parts will be integrated inside VHD++ [105] and they will run simultaneously in real-time.

2.6 Organization

In this chapter, we have presented the state of the art in Interactive Virtual Human research. We have also presented the motivations and objectives of this thesis. In Chapter 3, we will present our animation representation and motion synthesis technique.

The animation representation is based on a statistical analysis of human motion. As we will show in this chapter, this representation allows for a fast adaptation of existing motions, as well as the synthesis of new motions. Chapter 4 will show various ways of controlling the motion of a character. We will show how high-level face and body motion control from an emotional state is possible. In this chapter we will also present the main architecture of MIRAnim, our multi-purpose animation engine. We will show a simple means to correct motions that contain collisions, and finally we will show how to automatically calculate dependent joint motions. In Chapter 5, we will present the dialogue manager that is capable of controlling the 3D character, corresponding to the personality and emotional state of the character. We will also show how the dialogue system output is translated into speech and character motion. Chapter 6 discusses various implementation issues. It will also present the IVH prototype that we built. Finally, we will conclude in Chapter 7 with some potential applications of the research and an overview of where to go next.

CHAPTER 3

Motion Synthesis

In this chapter, we will present a collection of techniques that allows for the synthesis of new motions, using a set of previously recorded motions or motions designed manually by an animator. As seen in Chapter 2, such techniques can be very useful for the synthesis of realistic motions, while saving a lot of time. After a short overview of how motion capture works, we will propose a methodology based on a statistical analysis of recorded motion segments, called Principal Component Analysis (PCA). Then within this framework, we will address the various problems that exist in motion synthesis systems and we will show how the PCA approach allows solving these problems in a fast and efficient way. One of the most computationally expensive parts of any motion synthesis system is the calculation of distances between frames. We will present our approach, which significantly reduces the computational cost for these operations, while taking into account the conditions that are required for a meaningful distance calculation. Next, we will show how our approach is used to synthesize new motions from pre-recorded segments, using the distance criterion and a fast animation fitting algorithm. Together, the proposed methods will allow for a so-called **dynamic motion graph**: a motion database that allows for the dynamic addition and removal of animation segments, with a low computational impact. Finally, we will present a novel method for producing noise on dependent joint sets. Adding noise to motions solves the problem of characters that remain static between different motion clips and it will increase the realism and smoothness of the resulting animations. Contrary to existing methods, our approach takes into account the dependencies between joints, allowing for a more prominent noise component, while retaining a realistic body motion.

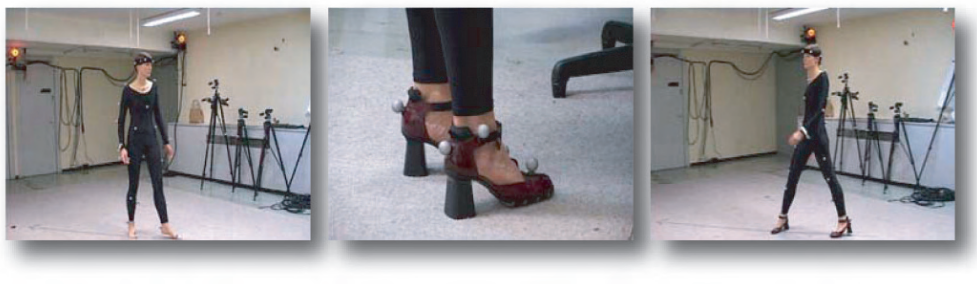


Figure 3.1: The Vicon motion capture system [124].

3.1 Getting the Data

In order to perform motion synthesis, a database of realistic motions is required in a format that is usable. The easiest way to get such motions, is by using a motion capture system. We will first give a short overview of how motion capture works. Then we will show how animations are represented in our system.

3.1.1 Motion Capture

Motion capture is a process that allows recording the movements of a person's body and translating them into a representation that can later on be used to animate a 3D model. There are different types of systems, with different ranges of applications. The most well-known techniques used for motion capturing is optical motion capturing and magnetic motion capturing. Magnetic motion capture systems, such as MotionStar [82], are very well suited for real-time applications, however they are less precise than optical motion capturing systems, such as Vicon [124]. As a basis for our research, we have used the latter system (see Figure 3.1).

In order to record a person's motions with the Vicon system, a set of markers need to be placed on the body. The markers are tracked by multiple cameras—in our system, there are 8—and the marker coordinates are triangulated from the different camera images. The marker positions are converted into an H-Anim skeleton animation represented by joint angle rotations and a global (root) translation. Section 2.1.2 presents a more detailed overview of skeleton-based animation. The advantage of representing animations using the underlying skeleton is that they are independent of the 3D model geometry. However, other problems occur when the animation is played on different models, such as collisions and foot skating. We will show later on how these problems can be addressed (semi-)automatically.

Clips recorded by motion capture are usually bound to a specific coordinate frame of the tracking system and/or start each at different global position and orientation. In order to remove those irrelevant initial conditions, we apply a data alignment procedure on the root joint. Depending on the clip, we align position and orientation (2 + 3 DOF[1]), position and turn (2 + 1 DOF), or only position (2 DOF). In all of these procedures the vertical displacement is kept unchanged because it is valuable information. The first solution is the most severe since it resets the initial global posture back to the world's origin. This is the best choice solution as long as no relevant initial orientation must be preserved. However, in cases where the clips start with meaningful tilt, roll orientation, such as in a laying down posture, the second correction—align position and turn—is preferred. These two alignment procedures cover most of the situations because the initial heading posture is rarely relevant unless being part of the design of the motion clip, in which circumstance we can either align only the position or use no correction at all.

3.1.2 Motions in Linear Space

Since our goal is to perform **motion synthesis** using these pre-recorded animation segments, we need a smart way to represent the animations. Also, because we know the domain of the animations that we will use (interactive communicative motions), we can profit from methods that statistically analyse recorded motions in order to find dependencies between different variables.

For the interpolation and adaptation of motions, we have seen in Chapter 2 that fast techniques exist for various representations of orientations. In many systems, quaternions are used since they do not have any Gimbal lock problems, and fast methods such as SLERP and SQUAD are available for interpolating between different joint angles. However, many statistical methods only operate on data that is represented in linear space. To that end, we need to use a representation of orientations in linear space, such as the **exponential map**. Alexa [3] already has proposed the exponential map as a valid representation for animations, and he has also shown that linear transformations and other matrix operations are possible on such a representation. Therefore, we choose to represent our animations in the exponential map format. Fast methods for the translation between quaternion/rotation matrix representation and exponential map are available, as discussed in Section 2.1.2.

In the next section, we will present a useful statistical analysis tool, the Principal Component Analysis. We have chosen this approach, because it allows for the detection of dependencies between different variables (in our case joint rotations). As will be shown in the following sections, this is a very powerful property, since it allows for many optimalisations in the data representation and treatment. We will first discuss how this analysis works, and then how the results that it yields can be useful for body animation processing.

[1]Degrees Of Freedom.

3.2 Principal Component Analysis

A Principal Component Analysis (or: PCA) is a statistical analysis method that results in an equivalent linear space for a collection of data vectors. If a data vector is transformed into this alternative linear space, the new variables are called the Principal Components (PCs). These variables have a minimized correlation and are ordered so that the first few PCs retain most of the variations present in the original dataset. In order to explain how this analysis works, we need to look at the two mathematical paradigms that it combines: statistics and matrix algebra. For a more detailed explanation of the Principal Component Analysis, as well as the mathematical proofs of its underlying theorems, we refer to the book by Jollife [59].

3.2.1 Statistics

The main idea of statistics is that given a large set of data, we would like to say something meaningful about the relationship between the individual points in the dataset. We will now briefly look at a few existing measures for that from the statistics domain.

Consider the following set of (one-dimensional) data:

$$X = \langle X_1, X_2, \ldots, X_i, , \ldots X_{n-1}, X_n \rangle \tag{3.1}$$

The mean value of this sample is calculated as follows:

$$\overline{X} = \frac{\sum_{i=1}^{n} X_i}{n} \tag{3.2}$$

and the standard deviation is defined as follows:

$$s = \sqrt{\frac{\sum_{i=1}^{n} (X_i - \overline{X})^2}{n-1}} \tag{3.3}$$

Variance is another measure of the spread of data in a data set. It is defined as the standard deviation squared s^2.

The previously mentioned measures operate in one dimension only. In order to treat multidimensional data, the *covariance* measure is useful. Covariance is always measured between two dimensions. The covariance expresses how much two dimensions vary from the mean with respect to each other. The covariance between two statistical variables X and Y is expressed as follows:

$$cov(X, Y) = \frac{\sum_{i=1}^{n} (X_i - \overline{X})(Y_i - \overline{Y})}{n-1} \tag{3.4}$$

In the case of data sets of more than two dimensions, the covariance can be calculated between each combination of dimensions, resulting in a so-called *covariance* *matrix*. This matrix is the basis of the Principal Component Analysis, as we will show in the next section.

3.2.2 Matrix Algebra

Given an $n \times n$ matrix A, then its n eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_{n-1}, \lambda_n$ are the real and complex roots of the characteristic polynomial $p(\lambda) = \det(A - \lambda I)$. Since we are dealing with real data, we are mainly interested in the solutions where $\forall_{i=1}^{n} \lambda_i \in \mathbb{R}^3$. If λ is an eigenvalue of A and the nonzero vector V has the following property:

$$AV = \lambda V \tag{3.5}$$

then V is called an eigenvector of A corresponding to the eigenvalue λ.

There are some interesting properties of the eigenvectors that are used later on in the PCA. First of all, eigenvectors can only be found for square matrices. Not all square matrices do have eigenvectors, but if they do, an $n \times n$ matrix will have n eigenvectors (and corresponding eigenvalues). Because of Equation 3.5, a scaled eigenvector is still an eigenvector and its scaled version multiplied with the base matrix will still give the same result. Therefore, an eigenvector is generally represented as a *unit eigenvector*, with a length of 1. A final and important property of the eigenvectors of a given matrix, is that they are *orthogonal*. This means that the data can be expressed using the perpendicular eigenvectors, instead of the original axes.

3.2.3 Performing the Principal Component Analysis

The first step of the PCA is to calculate the covariance matrix for a given dataset. Since this covariance matrix is square, we can then try to find the eigenvalues and eigenvectors of the matrix. The strength of the PCA is that in fact, the eigenvectors of the covariance matrix represent an equivalent linear space, where the correlation between the new variables is minimalized. Additionally, the eigenvector with the highest eigenvalue is the principal component of the data set. This means that it represents the most significant relationship between the data dimensions.

Once the eigenvalues and eigenvectors are found, the next step is to order them, from highest to lowest. This results in the components in order of significance. In order to reduce the dimension of the data, it is possible to remove the least significant eigenvectors/values. This results in a loss of data, but depending on the strength of the relationship between the different variables, a reasonable size of the PC vector can be chosen to accurately represent the data.

The original data can be rigidly transformed to the Principal Component representation of the data by multiplying each data vector with a transformation matrix, which columns are the eigenvectors, ordered by decreasing eigenvalue. This transformation

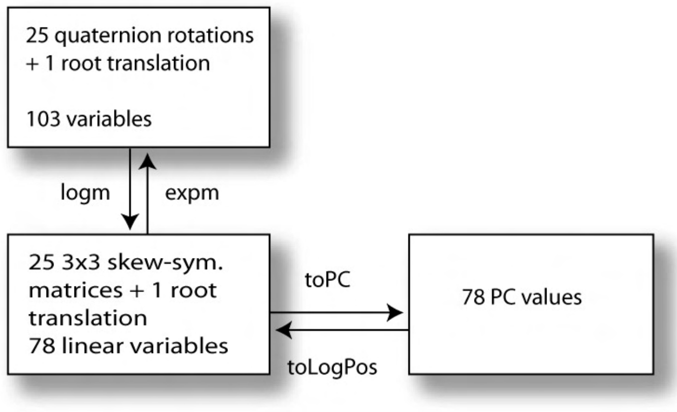


Figure 3.2: Conversion between different representations for postures.

matrix P additionally has the following property:

$$P^{-1} = P^t \tag{3.6}$$

Therefore, the original data vectors are obtained again by multiplying the PC vectors with the transposed transformation matrix.

3.2.4 Principal Components of Body Postures

In the case of body animation, we have obtained a large set of body postures from the recorded animations. In our case, we represent each orientation by the 3 values defining the skew-symmetric matrix of the exponential map. Additionally, a global (root) translation is defined. Since we use 25 H-Anim joints for each posture, our dataset has a dimension of 78. The PCA therefore results in a 78×78 transformation matrix. As we will show in the following sections, this PC representation of the full body posture is quite powerful and we will show its use for different applications. However, since the PCA is a statistical analysis tool that does not take into account the physical properties of body motions, it is also quite useful to have a PC representation only for the upper or lower body. In some cases, using these representations separately avoids foot skating problems or it allows for a more precise adaptation of motion, as will be shown later on. We have performed these PCAs separately on the dataset. Conversion from and to different representations (exponential map, global PCs, upper and lower PCs) is quite fast since they only involve a single matrix multiplication (see Figure 3.2).

3.3 Creating New Motions

Now that we have done the work of representing body postures in PC space, we can start using the powerful features of such a representation for motion synthesis. In order to create new motions from a collection of existing motions, there are two main operations that will often be performed on the frames of these animations:

- calculating the distance between two frames
- interpolating between two frames

The accuracy of the first operation is very important in motion synthesis systems, since it will define when and where proper transitions or adaptations of motion segments can be made. We have also seen in Chapter 2 that there are various ways to define the distance between two frames. The interpolation between frames is used when an original animation segment is adapted so that it follows a set of constraints. Examples of such constraints are a new target posture, or a translation of the original motion to another position and/or orientation.

Especially the calculation of the distance between two frames is very often performed in motion synthesis methods. In the case of Kovar et al. [66] for example, the distance is calculated for every pair of frames for every combination of animation segments. When using a geometry-based distance criterion in combination with such an approach, the time required for pre-processing is extensive. In the next section, we will propose a novel criterion to be used for distance calculations with similar properties as geometry-based methods, but that can be performed at a fraction of the computational cost. After that, we will show how interpolation techniques can benefit from the PC representation and how we use that to generate new animations from existing motion clips.

3.3.1 The Distance Criterion

Our frame distance criterion is based on the PC representation of postures. As discussed previously, a PC representation groups together dependent joint motions. Therefore, by defining the distance between two postures P and Q as the (weighted) Euclidean distance between the two corresponding PC vectors p and q, joint dependencies will be taken into account as a part of the distance calculation:

$$d_{p,q} = \sqrt{\sum_{i=1}^{N} w_i \cdot (p_i - q_i)^2} \tag{3.7}$$

The weight values w_i are chosen as the eigenvalues found during the PCA. Because the PC space is linear, calculating this distance can be done as fast (or faster) as the previously mentioned joint-based methods. However, the use of the PC space has another

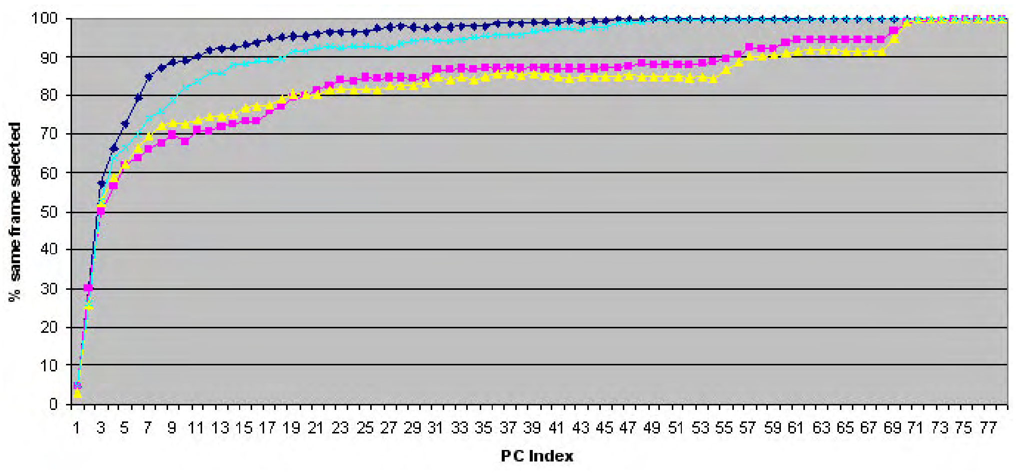


Figure 3.3: Success percentages of same selected closest frame for 4 different posture sets, ranging from using only the first PC value, the first and the second PC value, until using the full PC vector (and thus 100% success).

property that will allow for a significant speedup of the distance calculation: the dimension reduction. Since higher PCs represent lesser occurring body postures, they are mostly 0 and therefore they do not contribute significantly to the distance factor. This means that by varying the amount of PCs used, we can look for a reasonable trade-off between speedup and precision.

In order to quantify this result, we have analysed various distances between postures using different PC vector sizes. We have performed this analysis on 4 sets of posture frames coming from various motion captured animations, each set containing around 600-1000 different frames. For each frame, we have calculated the closest frame within the same set using the PC distance criterion. We have done this calculation for all possible PC vector sizes (in our case from using only the first PC value until using all 78 PC values). For each PC vector size, we then compare the result with the result obtained using the full PC vector size. For each frame, we can verify if the same closest frame was selected as when using the full PC vector size (success). The number of successful frames can be expressed as a percentage. Figure 3.3 shows these percentages for all possible PC vector sizes. For each set, the success percentage curve is shown.

As can be seen from the graph, when using only the first 10 PC values, the same closest frame will be selected in on average 80% of the cases. Although this is quite high, the first impression is that it is not precise enough to use as a replacement for the full PC vector distance calculation. However, the selection of another closest frame in 20% of the cases is not necessarily 'bad'. We need to investigate which closest frame is selected in these cases and how far off it is from the originally selected closest frame. By calculating the full PC vector distance between these two frames, we obtain a distance error. We have calculated the average and maximum distance errors for all different PC

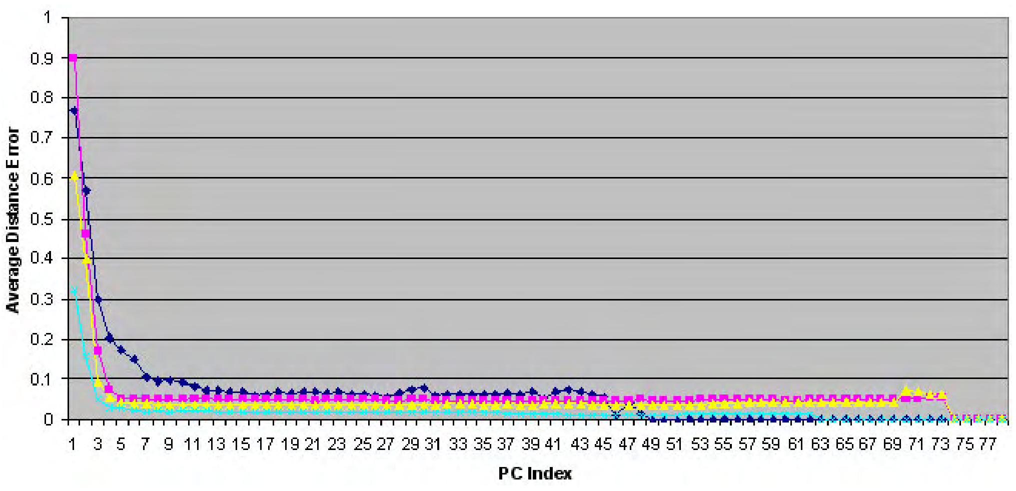


Figure 3.4: The average distance error when another closest frame was selected for 4 different posture sets, ranging from using only the first PC value, until using the full PC vector.

vector sizes. They are plotted in Figure 3.4 and Figure 3.5 respectively.

When looking at these figures, we see that the average distance error for vectors of size 10 or bigger drops below 0.1 and the maximum distance error is below 0.4. The final step is to find out what they values mean **visually**. Figure 3.6 shows various posture pairs with different distance error values. As can be seen from this image, a distance error of 0.1 is almost invisible. A distance error of 0.4 is visible, but it represents a minor difference in posture. For higher distance errors, the postures become significantly different.

From these results we conclude that by using a PC vector consisting of only the first 10 values, in 80% of cases, no distance error will occur, whereas in the remaining 20% of the cases, the distance error will not be visible on average and even in the worst case it will be minor. Thus, using the PC approach, a huge speedup can be obtained, while the results will be visually equivalent.

3.3.2 Synthesizing New Motions

Our motion synthesis system uses a graph-like representation to link the various recorded animation segments in the database. Because the system will be used for simulating an interactive virtual human, as a basis we will use motions that are useful for such a character. In Chapter 2, we have seen that most of the interactive virtual human systems concentrate on the creation of upper body gesture motions. The exception in this case is the work done by Cassell et al. [18]. They show that not only the upper body is important for communication, but that in fact the whole body is involved. They also note that the **balance shift** is one of the most important full body communicative functions,

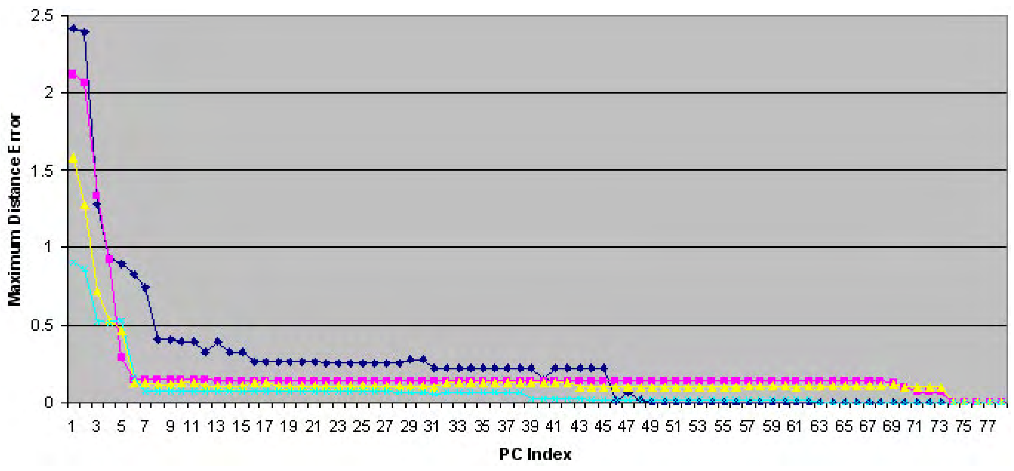


Figure 3.5: The maximum distance error when another closest frame was selected for 4 different posture sets, ranging from using only the first PC value, until using the full PC vector.

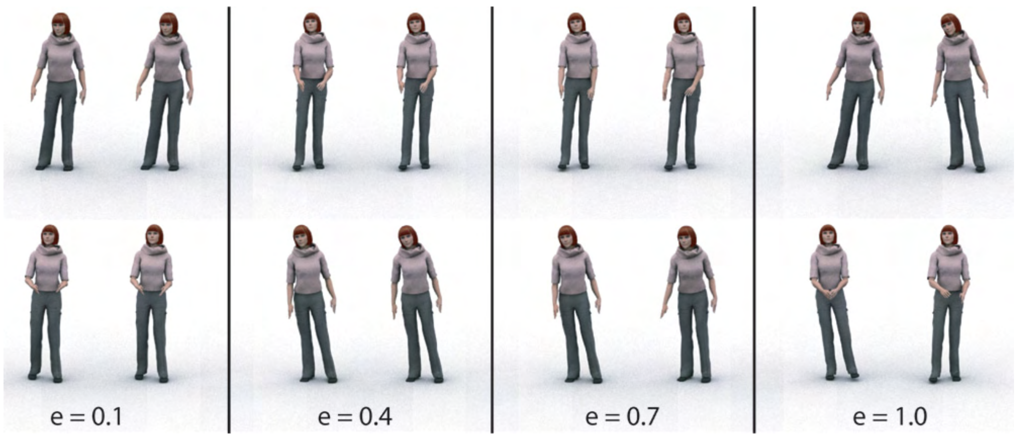


Figure 3.6: A collection of posture pairs with varying posture distances $e = 0.1$, $e = 0.4$, $e = 0.7$ and $e = 1.0$.

and it indicates a change of topic. Although they have implemented such motions as a part of REA [17], their focus clearly was on the communicative expressivity of the character, and not so much on the realism of its motions.

In the following section, we will show how motion synthesis is used to generate *realistic* balance shifting animations. We will also present some other uses for our motion synthesis system, such as the automatic generation of *idle motions*.

The Basic Animation Structure

We can identify different categories of balance postures, such as: balance on the left foot, balance on the right foot or rest on both feet. As such, given a recording of someone standing, we can extract the animation segments that form the transitions between each of these categories. These animation segments together form a **database** that is used to synthesize balancing animations. In order for the database to be usable, at least one animation is needed for every possible category transition. However, more than one animation for each transition is better, since this creates more variation in the motions later on. In order to generate new animations, recorded clips from the database are blended and modified to ensure a smooth transition.

Removing the Translation Offset

Since we will later on use the animations from the database to create balance-shifting sequences, it is necessary to apply a normalisation step so that the relative translations between the animations are minimised. Therefore we estimate the **translation offset** for each of the animation sequences to the origin $(0, 0, 0)$ by the mean value of the first and last frames of the animation (see Figure 3.7). In order to remove this offset from the animation, for each frame we subtract the translation offset from the root translation. There are other possibilities to estimate the translation offset, for example by searching the most central key-posture in the animation, or by using the timing information to determine the middle of the posture shift.

Transition between Categories

We denote the set of categories of resting postures as C. As each animation depicts a transition from one category to another, we can assign the first and last frame of each $a \in A$ to the category in C that they belong to. Therefore a category $c \in C$ corresponds to a set of postures $Q_c = q_1, \ldots, q_m$. Apart from these postures, we add stand-alone postures (that are not part of any animation in the database) in order to increase the variety of resting postures in the category. However, these postures should be within a limited translation interval to avoid unnatural shifts during the transition. If necessary, such postures are normalised by setting the translation to the mean translation of the other postures that are already in the posture set for the corresponding category.

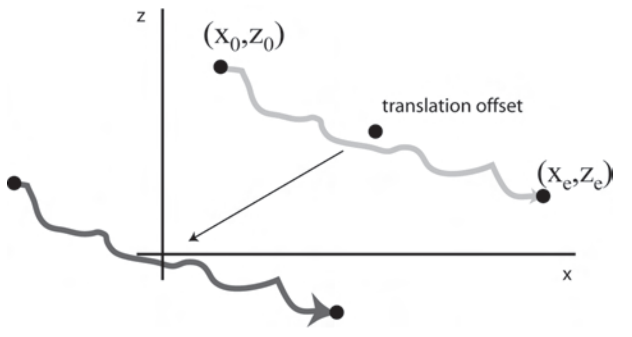


Figure 3.7: This figure illustrates what the estimated translation offset is in the (x, z)-plane (frontal-lateral plane) for an animation sequence.

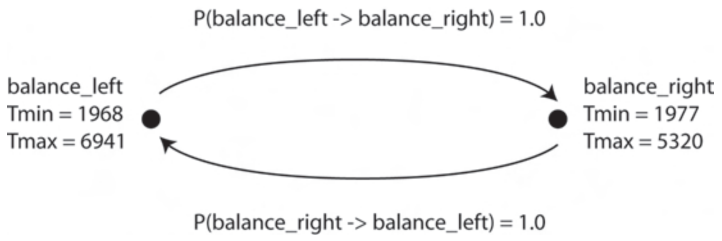

P(balance_left -> balance_right) = 1.0
balance_left
Tmin = 1968
Tmax = 6941
balance_right
Tmin = 1977
Tmax = 5320
P(balance_right -> balance_left) = 1.0

Figure 3.8: A simple example of a category transition graph. In this case, there are two categories: balance on the left foot and balance on the right foot. For both the categories, a minimum and maximum time is determined from the data (written in milliseconds).

From the original animation data, we also determine what the probability is that a transition occurs from one category to another by counting the occurrences in the data. The probability for a transition between category c_1 and c_2 is denoted by $P(c_1 \Rightarrow c_2)$. Obviously, the sum of the probabilities of one category to others should be 1.

Furthermore, for each category c, we extract from the data the minimum and maximum amount of time (denoted by $T_{c,min}$ and $T_{c,max}$) that a person stays in one of the categories, before performing a transition to another category. Figure 3.8 shows a simple example of a probability distribution for two categories and their $T_{c,min}$ and $T_{c,max}$.

Synthesis of Balance Changes

Transition Selection As a first step, we choose a random category c and then a random posture $p \in Q_c$. Furthermore, we choose a random time value t, where $T_{c,min} \leq t \leq T_{c,max}$. This already allows to create an animation consisting of two key-frames containing posture p at time indices 0 and t.

After the posture at time index t, we need to construct a transition to another category. In order to define the next category, we again choose it randomly, but according

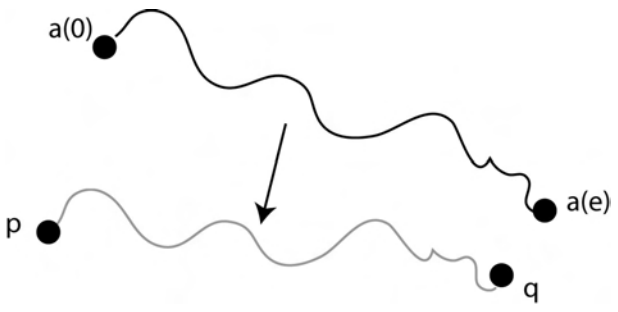


Figure 3.9: An example of fitting an animation from postures (a_0, a_e) to postures (p, q).

to the probability distribution as defined in the category transition graph. This gives the successor category s. Within this category, we choose randomly a posture $q \in Q_s$. Now we have to retrieve the animation from posture p to posture q and add the key-frames of the animation to the animation that we are constructing.

Animation Fitting The earlier constructed database of animations A may not contain all possible animations for all the postures. Therefore, we have developed a technique that allows to map an animation on a set of key-postures. In order for this technique to be as realistic as possible, we have to determine which animation in the database would fit best with the start posture p and end posture q. In order to determine this, the system must choose the animation that has its first frame and last frame as close as possible to respectively p and q. In order to determine the distances between frames, we use the approach described in Section 3.3.1 for calculating the distances between frames.

In order to select the best fitting animation between p and q, we calculate for each animation $a \in A$ the maximum M_a of the distances $d_{a_0,p}$ and $d_{a_e,q}$. The animation that we will use, is the one that has a minimal M_a.

The animation a (=$(a_0, a_1, \ldots, a_e)$) starts at frame 0 and ends at frame e. If we want to fit the animation so that it starts at posture p and ends at posture q, we have to transform a so that $a_0 = p$ and $a_e = q$ (see Figure 3.9). This problem can be solved as a specific case of a more general problem:

> Given an animation a and a list L of tuples (f, p) where f is a frame number, p is a posture represented by a vector of PC values, and L_i = (f_i, p_i) is the i-th element in the list L. Furthermore, $\forall L_i \in L : f_i < f_{i+1}$ (i.e. the elements in the list are ordered according to increasing frame number). Finally, $\forall L_i \in L : 0 <= f_i <= e$. How can we modify the animation a so that is passes through all the postures in the list L at their specified frame numbers?

The animation fitting can be done by calculating the offset of the postures $(f_i, p_i) \in L$ with respect to a_{f_i} and then interpolating between the offsets of all pairs L_i and L_{i+1}.

The algorithm that calculates the modified animation a passing through the key-posture list L. The main function uses a subroutine `fitAnim_part` that fits an animation segment between two frames:

```
fitAnim($a$,$L$)
    $begin = 0, \ \ end = 0$

    for ($i$=0; $i$ <size($L$); i++)
        $begin = end$
        $end = f_i$
        if ($end$!=0)
            fitAnim_part($begin$,$end$,$a_{f_i}$,$p_i$,$a$)

    if ($f_i$ != $e$)
        fitAnim_part($f_i$,$e$,$p_i$,$a_e$,$a$)

fitAnim_part($f_i$,$f_j$,$p_i$,$p_j$,$a$)
    $o_i \ = \ p_i \ - \ a_i$
    $o_j \ = \ a_j \ - \ p_j$
    for ($f = f_i$; $f \leq \ f_j$; $f$++)
        $a_f \ = \ a_f \ + \ o_i \cdot \frac{f_j - f}{f_j - f_i} \ + \ o_j \cdot \frac{f_i - f}{f_i - f_j}$
```

In order to change the best-fitting animation a so that it starts at posture p and end at posture q, we can simply use the function `fitAnim_part`(a_0,a_e,p,q,a). After this, the key-frames of the updated animation segment a' is appended to the sequence we are constructing. Next, a resting posture key-frame is added according to the method explained before. Then again a balance shift is added, and so on. Fitting an animation in this way only works if the difference in translation between p and q and the original starting- and end-point is small. As we are specifically dealing with balance changes that involve slight translation, this approach works very well. However, when one wants to add more general categories of motions (such as walking to an object, sitting down on a chair), more complex methods have to be used, such as retargeting [108, 129].

Synthesizing Other Motions

Although the PCA was performed on a database of standing people, the technique that we have presented is not limited to only synthesizing standing humans. A more extensive database can be constructed that contains not only motions of standing people, but also animations of sitting people that for example change their sitting posture or lying posture. Again, a Category Transition Graph can be constructed that contains animation sequences between different postures and the animations between categories can be refitted just like the animations between standing postures. A problem that arises is the fact that such kinds of postures are relative to one or more objects (a chair or a bed for example). However, these motions can still be played relative to the humans position in space, after performing the normalisation process as described previously.

Idle Motions

The balance shifting sequences can also be used to increase character realism during the playing of scenarios. In nature there exists no motionless character, while in computer animation we often encounter cases where no planned actions, such as waiting for another actor finishing his/her part, is implemented as a stop/frozen animation. We identify many situations where a flexible idle motion generator can help: from synchronisation of speech/body animation duration, to dynamic creation of stand still variations in between two active plays. In Chapter 6, we will show how our implementation of the motion system can indeed successfully be used to automatically generate such idle motions.

Personality and Motion

The source database used for motion synthesis is constructed from multiple recorded animations of a person. In Section 2.3.2, we have discussed the work of Kozlowski and Cutting [31, 67], which indicates that people move in different ways. According to their research, it even seems to be the case that people can recognize relatives and friends from recorded motions where the familiarity cues are removed.

In order to test whether or not this notion of individuality is retained during balance shifts, and if the motions synthesized using our approach have captured (at least a part of) this notion, we have created a motion database for several different people. We then performed a user evaluation (15 subjects) of the motion synthesizer. The test subjects were shown different movies of the virtual character playing either a recorded motion sequence or a motion sequence that was synthesized using our technique. The movies contained recorded and generated motions of 8 different people. The subjects were then asked three questions:

1. what is the gender of the person that performs the animation?
2. who is the person that performs the animation?
3. is the animation recorded using motion capture or automatically generated?

The evaluation results are shown in Figure 3.10. The gender recognition is a bit better for the recorded sequences (about 7% higher than the recognition percentage of the automatically generated motions). Recognising the person from the animation is quite difficult—in about one-third of the cases success—but it doesn't make any difference whether or not the motions are recorded or synthesized. Since this percentage is significantly higher than a random attribution of persons to motion sequences (12.5%), this

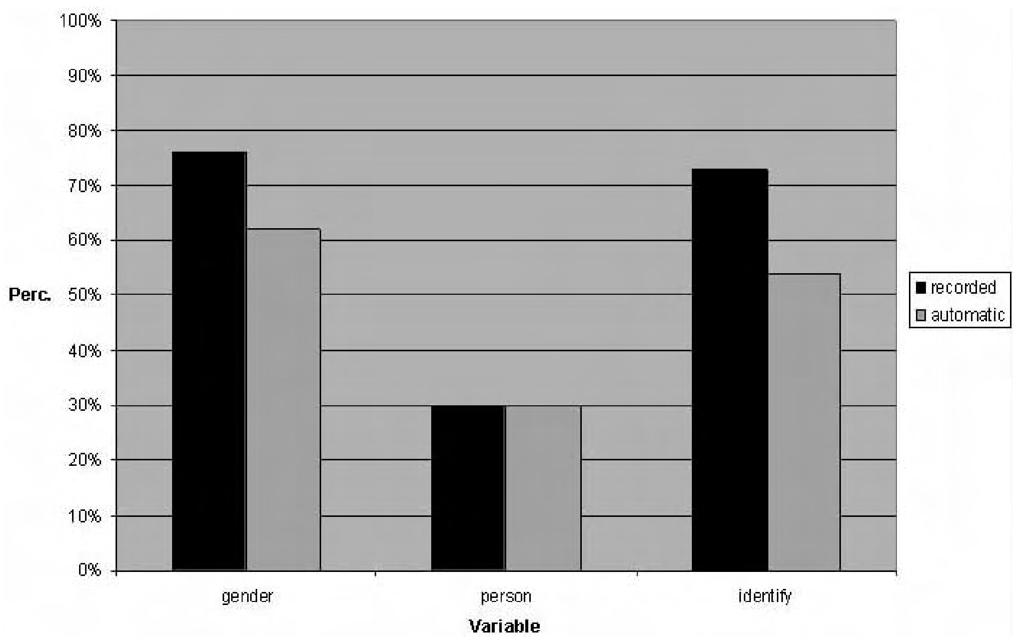


Figure 3.10: Overview of the user evaluation results.

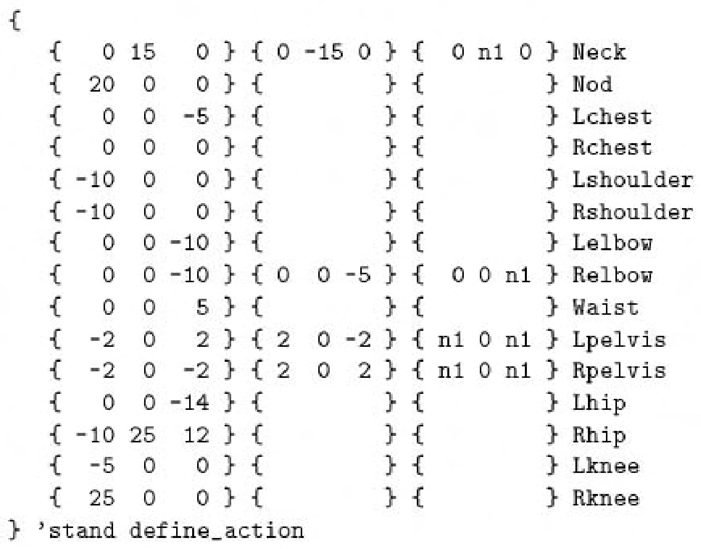
leads us to believe that our technique successfully captures (a part of) the individuality in the motions. Finally, the recorded motions were correctly identified as recordings in 73% of the cases, whereas 54% of the automatically generated motions were correctly identified as fake. From this we conclude that the synthesized animations are indistinguishable from recorded animations in almost 50% of the cases[2].

3.4 Adding Noise

Apart from the balance shifting postures that were discussed in the previous section, small posture variations also greatly improve the realism of animation. Due to factors such as breathing, small muscle contractions etc., humans can never maintain the exact same posture. There has not been a lot of research in this area, except for the work done by Perlin [99], which is based on the application of Perlin-noise [98] on elbow, neck and pelvis joints (see also Figure 3.11). This method generates quite realistic animation noise, while the noise is applied onto a few joints. However, real human posture variations affect all joints, which cannot easily be solved by using noise functions on all joints, because of dependencies between joints.

Contrary to Perlin's approach, we use the Principal Component representation for each key-frame. Since the variations apply to the Principal Components and not directly

[2]This percentage is probably higher, since more than 25% of the recorded motions were incorrectly classified as being automatically generated.



```
{
   {   0 15   0 } { 0 -15 0 } {  0 n1 0 } Neck
   {  20  0   0 } {         } {         } Nod
   {   0  0  -5 } {         } {         } Lchest
   {   0  0   0 } {         } {         } Rchest
   { -10  0   0 } {         } {         } Lshoulder
   { -10  0   0 } {         } {         } Rshoulder
   {   0  0 -10 } {         } {         } Lelbow
   {   0  0 -10 } { 0  0 -5 } {  0 0 n1 } Relbow
   {   0  0   5 } {         } {         } Waist
   {  -2  0   2 } { 2  0 -2 } { n1 0 n1 } Lpelvis
   {  -2  0  -2 } { 2  0  2 } { n1 0 n1 } Rpelvis
   {   0  0 -14 } {         } {         } Lhip
   { -10 25  12 } {         } {         } Rhip
   {  -5  0   0 } {         } {         } Lknee
   {  25  0   0 } {         } {         } Rknee
} 'stand define_action
```

Figure 3.11: Definition of an animation in Perlin's [99] animation system. This script defines a noise function on the elbow, neck and pelvis joints, in order to avoid static, unrealistic standing of virtual characters.

to the joint parameters, this method generates randomised variations that still take into account the dependencies between joints. Additionally, because the PCs represent dependencies between variables in the data, the PCs are variables that have *maximum independency*. As such, we can treat them *separately* for generating posture variations.

One method to generate variations, is to directly apply a Perlin Noise function [98] on a subset of Principal Components. We will present another method that generates these small variations based on the shape of the curves in the motion capture data. This method applies a statistical model to generate similar (randomised) curves. This approach also allows keeping certain existing tendencies in the variations that are specific to certain persons (such as typical head movements, etc.). An additional advantage over using Perlin noise, is that this approach is fully automatic and it avoids the need to define frequencies and amplitudes to generate noise, although these parameters could eventually be determined automatically by analysing the signal. In our method, we analyse animation segments that do not contain any balance shifts or motions other than the small variations. The resulting noise motions will be placed on top of the synthesized motions.

3.4.1 Normalisation of the Data

The selected animation segments need to be normalised, so that only the variations persist and the base postures are removed. This is necessary, since these variations will be synthesized on *top* of the balance shifting postures. In order to normalise an animation

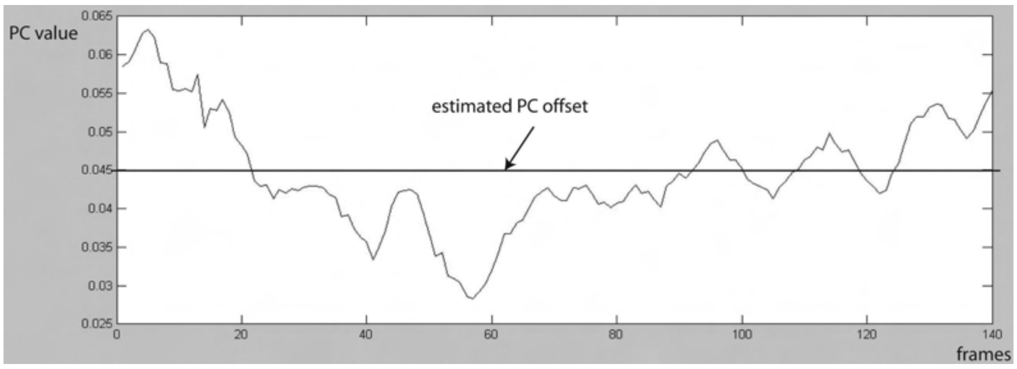


Figure 3.12: This example shows the values of a Principal Component for an animation segment. The offset is estimated by the mean value (which is around 0.045).

segment, we calculate the mean value of each PC in the segment and subtract it from the PC values in each key-frame, therefore removing the base posture and keeping only the variations (see Figure 3.12 for an example). Clearly, this approach only works when dealing with animation sequences where the base posture does not change significantly during the sequence.

3.4.2 Variation Prediction

In this subsection, we describe the method we use to predict the variations according to motion capture data. For illustration of our method, we use the PC values of the animation segment in Figure 3.12. In order to reduce the number of points in the PC curve while retaining its global shape, we convert the PC values in a set of maxima and minima. Then, we will show a technique to predict the next maximum/minimum given its predecessor point.

For estimating the maximum and minimum, a simple algorithm can walk through the points and—given an error value ϵ—determine if a point is a maximum or a minimum. In order to remove the absolute timing from the data, we specify each maximum/minimum by defining its distance (in milliseconds) from the previous maximum/minimum and the corresponding PC value (see Figure 3.13 for an illustration).

The next step is to construct a system that can, given a $(distance, value)$ pair, predict the next $(distance, value)$ pair. The generated $(distance, value)$ pairs then form a new animation sequence. In order to add some randomness to the system, we do not use the points directly, but we apply a point-clustering algorithm. In our application we have applied a k-means clustering algorithm, but other clustering algorithms can also be used (for an overview of different clustering algorithms, see Jain et al. [57]). From the motion data that we have, we can set up a probabilistic model that defines the probability of a transition from one cluster of points to another. The result is a Probability Transition Matrix that defines the probability for any transition between the

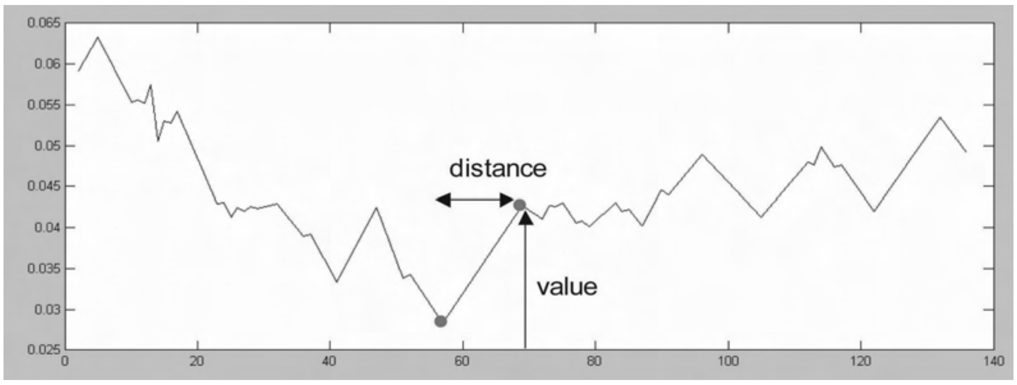


Figure 3.13: This figure shows a sequence of maxima and minima for a Principal Component. Each maximum and minimum is specified by defining the distance to its predecessor and its PC value.

clusters. This model can be used to generate new animations, according to the following algorithm:

1. add the neutral position at time $t = 0$;
2. select randomly a cluster of points;
3. choose randomly a point ($(distance, value)$ pair) within the cluster;
4. add a key-frame to the animation at time $t + distance$ with the specified PC value;
5. based on the last selected cluster, choose the next cluster according to the Probability Transition Matrix;
6. choose randomly a point ($(distance, value)$ pair) within the cluster;
7. go to step 4 until desired length of animation is reached.

In our application, the noise synthesis method is applied on a subset of PC values. For each Principal Component, the variations are generated independently. This does not give unrealistic results, since the dependency between Principal Components is relatively small. The final result of the variation synthesis shows little to no repetition, because of the random choice of points in each cluster, and because of the separate treatment of PCs.

After the PC noise has been generated, it is mixed with the balance shifting motions. The noise is only blended with the animation when a waiting state between transitions takes place. This results in a minimal change in the original animation segments, while ensuring that the character is never completely static.

3.5 Summary

In this chapter, we have addressed the problem of synthesizing new motions from a database of existing motion clips. We have presented a novel distance criterion based on a Principal Component representation of body postures. Our distance criterion allows for a very fast computation of posture distance, while taking into account the dependencies that exist between joints. By using this distance criterion, we have presented a fast motion synthesis system, which allows for the real-time synthesis of motions by automatically adapting a set of previously recorded motion clips. By using different motion databases, we can generate different styles of motions, while a part of the individuality of the original motion is preserved in the new motion clips. Finally, we have presented a simple but effective method to generate posture noise that takes into account joint dependencies. In the next chapter, we will show how such a motion synthesis system can be controlled by defining constraints. We will also present our animation engine, which is built around the PC representation of postures.

CHAPTER 4

Motion Control

In the previous chapter, we have presented a novel animation method based on a PC representation of postures. We have shown how such a representation can be successfully used to synthesize motions from pre-recorded motion segments in real-time. An important next step in the development of such a system is the *control mechanism*. For an animation system to be suitable for Interactive Virtual Humans, it is crucial that constraints can be applied onto the motion synthesizer. These constraints could be defined by for example a Dialogue Manager that controls the outputs and expressions of a character. In this chapter, we will show that it is indeed possible to define such constraints on the motion synthesizer. To illustrate our case, we will present an extension of the motion synthesizer that selects different motions according to different *emotional states*. However, placing only constraints on the motion synthesizer is not enough. Most of the IVH simulation systems also need to play other types of motions at the same time, such as body gestures. This means that a flexible animation engine is required, that can take multiple animation streams and mix them in a meaningful way. We will present an animation engine in this chapter that can blend different body animations; including motions that do not have a specified length and that are generated on the fly, such as idle motions and/or balance shifting motions. We have adopted a generic approach for this animation engine, so that it can be used for any type of animation. As we will show in the following Chapters, the same animation engine also handles the *facial animation*. This makes is a lot easier for IVH systems to control the body and the face, since they are handled by the same interface. Finally, we will show the complete animation pipeline, including a technique that is capable of removing model-dependent collisions from animations on-the-fly.

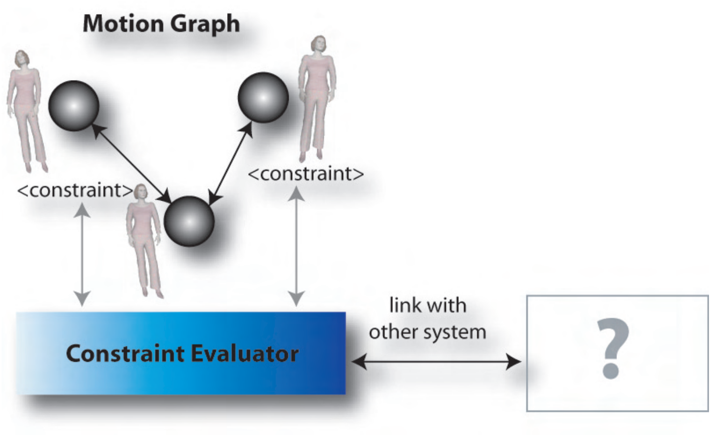


Figure 4.1: A motion graph with additional constraints on its transitions.

4.1 Emotional Motion Synthesis

In this section, we will describe how the motion synthesizer presented in Chapter 3 can be controlled by defining constraints on the transitions in the motion graph. Because in our representation, each transition generally represents a meaningful motion segment (such as a balance shift), we can directly define a condition on each transition. The motion synthesizer then only selects transitions for which the condition evaluates to `true`. The constraints are evaluated by a Constraint Evaluator, which can be linked with other systems that define the type of high-level control (see Figure 4.1 for an overview of this approach).

We will illustrate this approach by looking at how motions can be generated according to a different *emotional state*. In order to do this, we first need to decide what representation for emotions is to be used. There are many different representations of emotions (as shown in Chapter 2), but not all of them are suitable for representing emotional expressive body postures. For example, the emotion list used in the OCC model [93] is too detailed with respect to what people can actually perceive. Coulson [29] shows that only a few emotions are easy to distinguish from only a body posture, so it seems logical to choose an emotion representation that follows this observation. Figure 4.2 shows the distribution of perceived emotions from body postures as found by Coulson, next to the activation-evaluation emotion disc. It can be clearly seen that the latter representation corresponds very closely to the confusion matrix determined by Coulson. We have therefore chosen to use the activation-evaluation space as a representative emotion space for controlling body motions. An additional advantage of using this approach is that other multi-dimensional emotion models can be easily mapped onto the activation-evaluation disc.

In the evaluation-activation space, an emotional state e is defined as a 2-dimensional

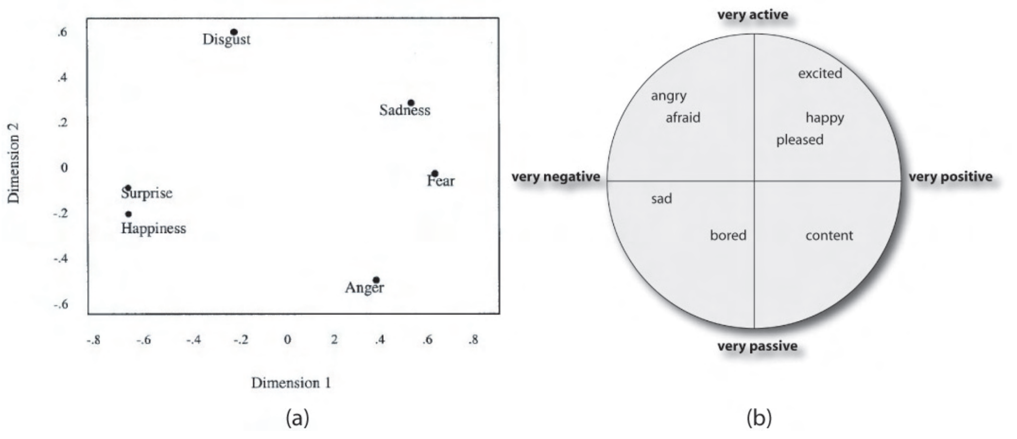


Figure 4.2: The confusion matrix from Coulson [29] compared to the activation-evaluation emotion disc. Apart from the swapped axes, the representation is the same.

vector:

$$[e_e, e_a], \text{ where } -1 \leq e_e, e_a \leq 1 \text{ and } \sqrt{e_e^2 + e_a^2} \leq 1 \tag{4.1}$$

When using a discrete list of n different emotion dimensions (for example based on OCC), a mapping function $f : \mathbb{R}^n \rightarrow \mathbb{R}^2$ that respects the conditions in Equation 4.1 has to be defined. By using this mapping function, an emotional state that is for example defined as a part of an IVH (see Chapter 5) will be directly translated into a format that the motion synthesizer can use to evaluate the transition constraints.

4.1.1 Motion Segment Selection

Before new animations can be created, the animations that are in the animation database need to be annotated with additional emotional information. For each animation segment, we define the change of emotional content (if any) by specifying for both the start and end points of the animation a 2-dimensional interval on the activation-evaluation circle. Figure 4.3 shows some examples of possible intervals and related postures on the activation-evaluation circle. Please note that this system is not limited to these specific intervals, but that we can define any amount and distribution of intervals.

Given an emotional state $[e_e, e_a]$, the motion synthesizer automatically selects animations that have a target interval including this point in the emotion space. In order to make sure that it is always possible to make a transition regardless of the emotional content, a set of transitions with no constraint is added as well, so that when no suitable target interval can be selected, a 'neutral' transition is still possible.

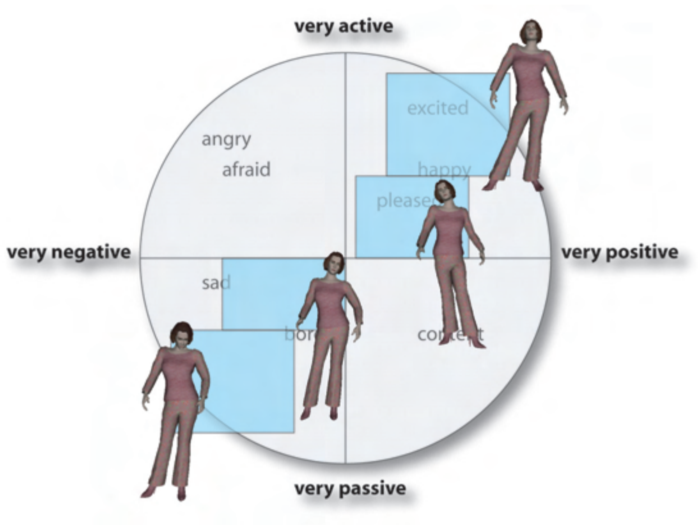


Figure 4.3: Different intervals of emotional states together with example postures.

4.1.2 Adapting the Pause Length

Not only do the transitions themselves change according to the emotional state. An additional option that the motion synthesizer provides is the automatic adaptation of pause length between transitions according to the activation level[1]. Higher activation level will result in shorter pauses (and thus more transitions). In our system the length of a pause is determined using a minimum length p_m and a maximum offset p_o. A random value between p_m and $p_m + p_o$ is then chosen as the final pause length. In order to adapt this value according to the emotional state, the value of p_o is replaced by $p_o\prime$, which is calculated as follows:

$$p_o\prime = (\alpha - e_a) \cdot \beta \cdot p_o, \text{ and } \alpha \geq 1, \beta \geq 0 \tag{4.2}$$

where α and β are parameters that define how the offset length adaptation should be applied. In our current application these values are set by default to $\alpha = 1$ and $\beta = 1$. The application of the length adaptation can be dynamically switched on and off, so that there is no interference when pauses are required to have specific lengths (for example during speech).

4.2 The MIRAnim Animation Engine

In this section, we will present our animation engine, called MIRAnim. The main architecture of the animation engine is a multi-track approach, where several animation

[1]Although not currently implemented, a similar adaptation function can be applied according to the evaluation level.

streams need to be blended into a final animation. There has been quite some research in motion blending. Perlin [99] was one of the first to describe a full animation system with blending capabilities based on procedurally generated motions. There are several researchers who have used weight-based general blending to create new animations [128, 108]. There have also been several efforts to apply motion blending not directly on the joint orientation domain. For example, Unuma et al. [122] perform the motion blending in the Fourier domain and Rose et al. [109] used space-time optimization to create transitions that minimize joint torque. Kovar et al. [65] use registration curves to perform blending. Their approach automatically determines relationships involving the timing, local coordinate frame, and constraints of the input motions. Blend-based transitions have been incorporated into various systems for graph-based motion synthesis [108, 66].

The goal of our animation engine is to provide for a generic structure that allows for the implementation of different blending strategies. This is especially important, since our animations use different representations, depending on the application. For example, the synthesized balance shifting motions and the small posture variations are blended using the PC representation. For more generic blending, where different animation stream should play on different body parts, a joint-based method is required. Additionally, in the final system, we will need to perform blending operations on both body and face animations, which are two completely different animation formats that require different blending strategies. The approach that we will present in this Section is suitable for any of the previously discussed blending approaches. A large set of blending tools, for example time warping, splitting, fading, and so on are available. The advantage of using this generic approach is that once a blending tool has been defined, it can be used for any type of animation, regardless of its type. In order to be able to use these blending tools, only an interface needs to be provided between the data structure used for blending, and the original animation structure. An overview of the blending engine is provided in Figure 4.4.

4.2.1 Blendable Object

The basic structure used in the blending engine is the so-called `BlendableObject` interface. A blendable object is the representation of an animation that can be blended with other animations. The main functionality of this object is a function that maps timekeys onto frames. A frame in the blending engine is called an `AbstractFrame`. An abstract frame consists of a number of elements, called `AbstractFrameElement` objects. Each of these elements is a list of floating point values. For example, in the case of body animations, an `AbstractFrameElement` could be a list of 3 floating points, representing an exponential map rotation, or a list of 3 floating points, representing a 3D translation. An abstract frame could then consist of a combination of abstract frame elements that are either translations or rotations. In the case of facial animation, the

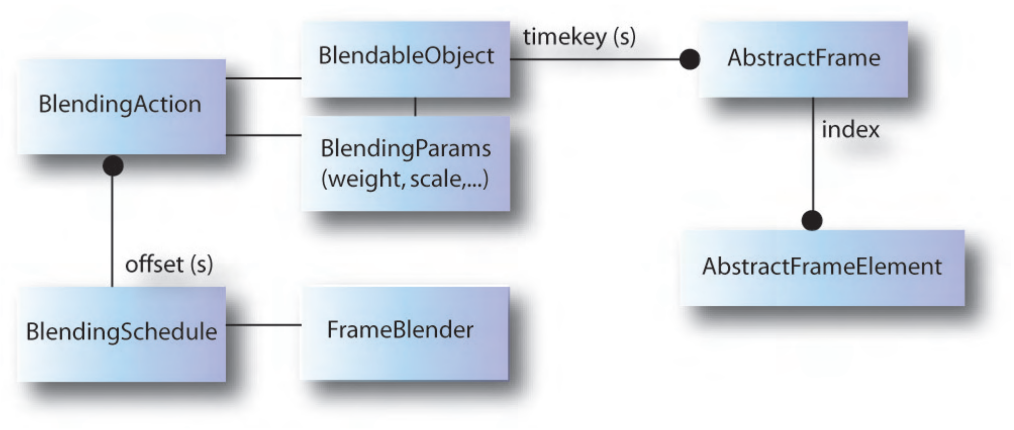


Figure 4.4: Overview of the blending engine data structure.

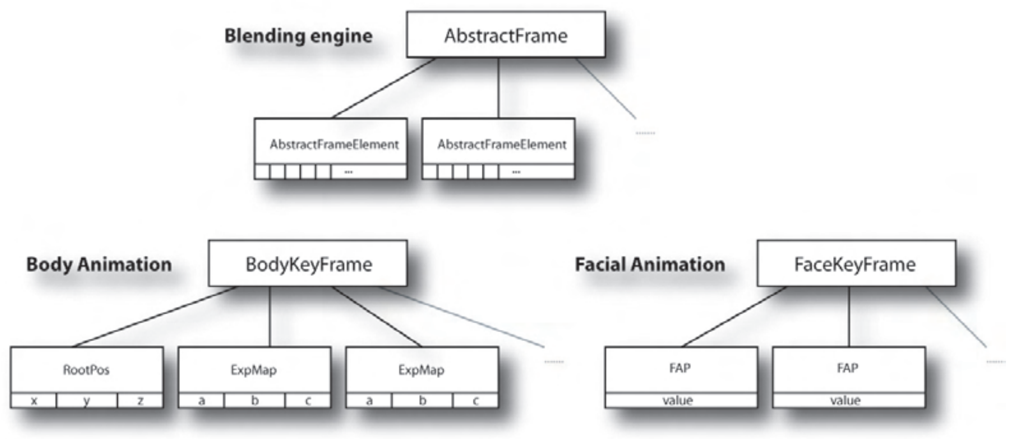


Figure 4.5: The abstract frame implementation for both body and facial animation.

abstract frame element could be a list of 1 floating point, representing a FAP value in the MPEG-4 standard [44, 83]. Figure 4.5 depicts how these structures look like.

In order to provide for a higher flexibility, the blending engine accepts blendable objects with or without a fixed duration. The latter type is especially practical in the case of playing an animation controlled by a motion synthesizer. Since these animations are generated on-the-fly, the duration of the animation may not be known at run-time. In the case of idle motions, the duration is actually ∞, since idle motions should always be generated.

4.2.2 Blending Schedule

Through the `BlendableObject` interface, each animation is defined as a function $A : t \to K$, where t is a timekey $\in [t_s, t_e]$ with $0 \leq t_s < t_e < \infty$, and K is the

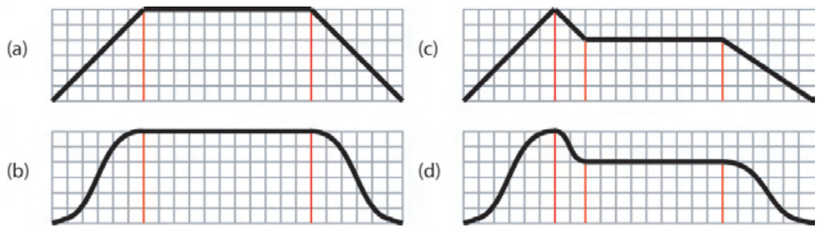


Figure 4.6: Various basic curve structures are available, such as (a) linear fading (b) cubic fading, (c) linear attack-decay-sustain-release, or (d) cubic attack-decay-sustain-release.

corresponding key-frame of the animation. A structure is now required that can take a number of such 'animation tracks' and blend them according to different parameters. The parameters are defined through the `BlendingParams` interface. The most basic parameter used for blending is a weight value to be used for blending. In addition to a list of weights, the `BlendingParams` interface also provides for a list of scales. The evolution of the weights and scales over time is governed by a parametrizable curve. Figure 4.6 shows some examples of curves that can be chosen. Different types of `BlendingParams` objects can be multiplexed, and custom blending parameter objects can be defined. For example, a custom `BlendingParams` object can be created for body animations, that defines a joint mask. When multiplexed with a weight curve `BlendingParams` object, this results in a set of curves defined for each joint in the mask. Any arbitrary combination of such blending parameters is possible, allowing for a flexible blending parameterization structure. Here, also the advantage of the independency of the blending strategy comes forward. Once a blending parameter feature such as curve-based blending is implemented, it can be used for any type of animation.

A `BlendingAction` object is defined as the blendable object, together with the blending parameters. This object has a flag indicating if it should be rendered outside the timekey domain of the blendable object source. This flag is useful when linking the system with a motion synthesizer, where frames are created during run-time, independent of the current length of the animation.

The final step in obtaining a mix of different `BlendingAction` objects requires a structure that allows for activating and deactivating different animations according to the blending parameters. This structure is called a `BlendingSchedule`. A blending schedule consists of a list of `BlendingAction` objects. Each blending action is associated with a timekey, which defines the time that the blending action should start.

The actual blending itself happens in the `FrameBlender` object. This object is by default a linear blender, but it can be replaced by a more complex blender, that allows for example blending of other—non-linear—data structures, such as quaternions. This blender can also be replaced if there are different types of frame elements in the same frame, like for example translations (linear) and rotations (generally non-linear).

The `BlendingSchedule` is again a blendable object. This allows for performing animation blending on different levels, with local blending parameters. When key-frames are obtained from the blending schedule, they are rendered in real-time as a result of the different activated actions and their blending parameters. So, blending actions can be added and removed from the blending schedule during runtime, resulting in a flexible animation blender, adaptable in real-time.

4.2.3 Additional Blending Tools

In addition to the basic data structures and tools used for blending animations, the blending engine also provides for a few extensions that allow to further parameterize the animation and blending process. For example, modifiers can be defined which act as a wrapper around blendable objects. Examples of such modifiers are time stretching, flipping, or looping of animations. Again, custom modifiers can be defined for different animation types. To give an example in the case of body animations: a modifier is available that performs a global transformation on the whole animation. Any sequence of modifiers can be used, since modifiers are again blendable objects.

Additionally, a blendable object is defined that uses a single frame as a source, and that applies a custom scaling curve (chosen from the basic curves in Figure 4.6) over time. This is a useful extension for both facial and body animation, because it allows for a designer to create a single key-frame, such as raised eyebrows or a simple arm posture, and the frame will be automatically animated and blended in using the curve that was defined.

4.3 Real-Time Animation Adaptation

Especially when working with motion captured data in combination with several different characters, retargeting motions to different characters will become necessary. A major problem during the process is that when an animation is played on characters of different sizes, collisions may occur because the shape of the characters is different. During the blending process in the previous section, such problems are not taken into account, because it would significantly increase the computational cost. In our system, the motion adaptation takes place at the end of the animation pipeline. This is also more logical, because in that way, most of the animation and blending process stays character-independent. However, that means that the motion retargeting filter that should be applied has satisfy the *real-time* constraint. There has already been quite some research in motion retargeting. Gleicher [45] presents a motion retargeting method that uses space-time constraints to minimize an objective function $g(x)$ subject to the constrains of the form $f(x) = c$. The constraints can represent the ranges of parameters, or various kinds of spatial-temporal relationship among the body segments and the environment. The objective function is the time integral of the signal displacement between the source and destination motion:

$$g(x) = \int (m^{src} - m^{dest})^2 dt \tag{4.3}$$

Since the whole interval has to be integrated to find the optimal solution, this method cannot be used in real-time. The main obstacle that needs to be overcome for real-time motion retargeting is finding a cheap way of including the physical properties. Normally, an Inverse Kinematics (IK) solver will try to find a solution through an optimization approach, which is computationally expensive. Choi et al. [24] present a method for motion retargeting that is suitable for real-time applications. Their technique is based on inverse rate control, which computes the changes in joint angles corresponding to the changes in end-effector position. Although this method could be easily linked with our animation engine, our goal is slightly different: we wish to *correct* a motion so that it can be played on a specific 3D model, without assuming that the original motion is optimized for a given 3D model.

4.3.1 PC Ranges

In this section, we will present a simple collision removal tool that is based on the PC representation. Although this method is not as precise as previously mentioned work, it is very fast and very simple to calculate, which makes it especially suitable for our application. The main idea behind our approach is limiting the values of the Principal Components. Since each PC represents an independent movement, we only allow PC values within a certain range. Because most of the collisions occur due to upper body movements, and because we do not want to change anything in the lower body motions to avoid foot skating, we consider the PC representation for the upper body. Since most of the relevant motion takes place in the first few PCs, it is not necessary to set these ranges on all the PCs.

The range of values for each PC that needs to be defined, depends on the 3D geometry of the character that the animation will be played on. As such, every character will need to have a range specification. Given this range specification and the motion to be played, the corrected motion can be calculated (see Section 4.3.2). For each PC, we need to determine the range where no collision takes place for the character.

Determining physical boundaries of PCs Before we can determine the range for every PC index, we need to know what the domain of physically possible body motions is for each PC. In order to obtain this information, motion captured data can be used to analyse a wide range of motions. The maximum and minimum values for every PC can be determined from this data, and a safety margin should be applied to be sure that all possible motions are included.

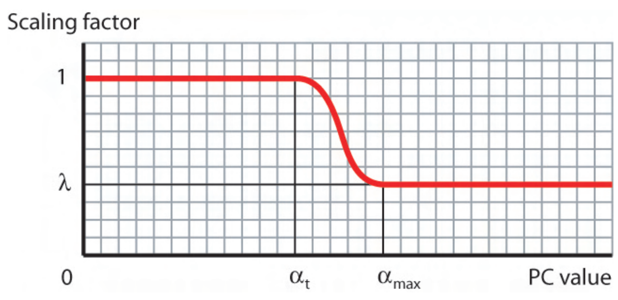


Figure 4.7: The scaling function $u(x)$ defined between α_t and α_{max}.

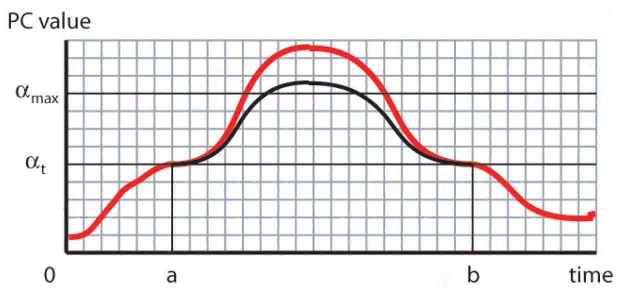


Figure 4.8: Result of applying the scaling function $u(x)$ onto a PC signal.

Determining the range for every PC index Over the previously determined domain, we run an algorithm that—at a given sample rate—calculates the posture corresponding to the PC value. Then, we see if a collision is detected on the character for which we are establishing the range. By performing this operation over the whole domain for each PC, we get the range of PC values of physically possible motions, where no collisions occur. We then devise a damping function, that takes an existing motion and that edits each PC signal so that the signal stays within the previously determined range. The detection of ranges only has to be done once for each character.

4.3.2 Motion Damping

Now that the range for every PC value has been established, a damping function must be applied on each PC signal to ensure that the signal stays within the designated range. Generally we do not know the exact shape of the curve, because the key-frames are generated dynamically. We therefore have devised a damping function that treats a point in time separately by applying a scaling function and then a weighted blending function between the current PC value and the range maximum or minimum. This damping function is only applied to the PC value when it reaches a predefined threshold. To illustrate our approach, we will give an example of a PC value exceeding the maximum. A similar approach is taken for a PC value approaching a minimum boundary.

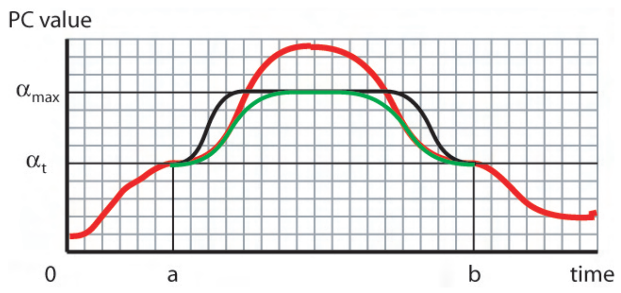


Figure 4.9: Result of applying the damping function $D(x)$ onto a PC signal (in green).

Value-based scaling As a first step, a scaling function $u(x)$ is applied on the PC value, depending on its value. This scaling function is only defined between a threshold α_t and a maximum α_{max}, with a maximum scaling factor of λ (see Figure 4.7). The following function defines this curve:

$$u(x) = 1 + (1 - \lambda)\cos(\frac{x - \alpha_t}{\alpha_{max} - \alpha_t}\pi) \tag{4.4}$$

where $\alpha_t < x < \alpha_{max}$. The result of the application of this scaling function is shown in Figure 4.8.

Signal damping A damping function $D(x)$ is now applied on the scaled PC signal. The damping function uses a weight function $w(x)$ that determines the weight of the maximum value depending on the PC value as follows:

$$w(x) = \frac{1 + \cos(\frac{x-\alpha_t}{\alpha_{max}-\alpha_t}\pi)}{2} \tag{4.5}$$

where again $\alpha_t < x < \alpha_{max}$. The complete damping function $D(x)$ applied onto PC values then looks like this:

$$D(x) = \begin{cases} x & \text{for } x \leq \alpha_t \\ (1 - w(x))u(x)x + w(x)\alpha_{max} & \text{for } \alpha_t < x < \alpha_{max} \\ \alpha_{max} & \text{for } x \geq \alpha_{max} \end{cases} \tag{4.6}$$

Figure 4.9 shows the application of this damping function onto a PC signal that exceeds the maximum α_{max}. A similar damping function has been defined for managing PC values reaching the lower boundary. The parameters that need to be chosen are λ and α_t. The former defines how much 'flattened' the PC signal should be as soon as it crosses the threshold. The threshold α_t defines how soon the algorithm should start flattening the curve.

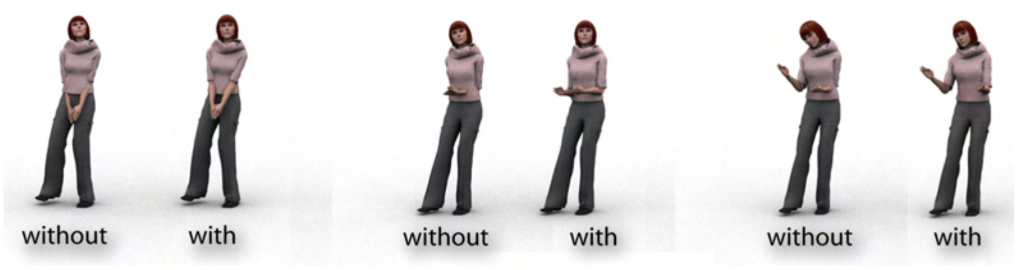


Figure 4.10: Animation frames with and without the damping function applied on them.

4.3.3 Results

We have calculated the PC ranges and have applied the previously mentioned damping function on several animations. The method that we presented can be applied in real-time because of the simplicity of the operations. Figure 4.10 shows some frames from an original animation (with collisions) and the frames after application of the damping function. The system works in most of the cases, but it is not foolproof. This is mainly because the absence of collisions in the separate PC values does not necessarily guarantee no collisions in the final frame, because of minor dependencies between the PC variables. Also, since our method is based on the PC representations, the results will become less good for motions that are very different from the intended application. However, for applications where the types of motions are known, like for example interactive virtual humans, the method works quite well and it is very fast. Generally, we only need to apply the damping function on the first 10 PCs in order to remove most of the collisions. A final limitation of this method is that the damping function affects not only the joints where the collisions occur. This disadvantage can be dealt with by applying the adapted motion only on the joints that are involved in the collision.

4.4 Automatic Dependent Joint Motion Synthesis

As discussed in Section 2.3.3, body gesture synthesis systems often generate gestures that are defined as specific arm movements coming from a more conceptual representation of gesture. Examples are: "raise left arm", "point at an object", and so on. Translating such higher level specifications of gestures into animations often results in motions that look mechanic, since the motions are only defined for a few joints, whereas in motion captured animations, each joint motion also has an influence on other joints. For example, by moving the head from left to right, some shoulder and spine movements normally occur as well. However, motion captured animations generally do not provide for the flexibility that is required by gesture synthesis systems.

Such systems would greatly benefit from a method that can automatically and in real-time calculate believable movements for the joints that are dependent on the ges-

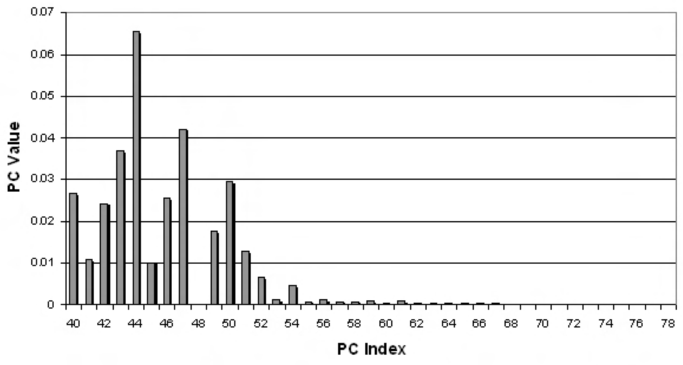


Figure 4.11: (Absolute) PC values of a posture extracted from a motion captured animation sequence.

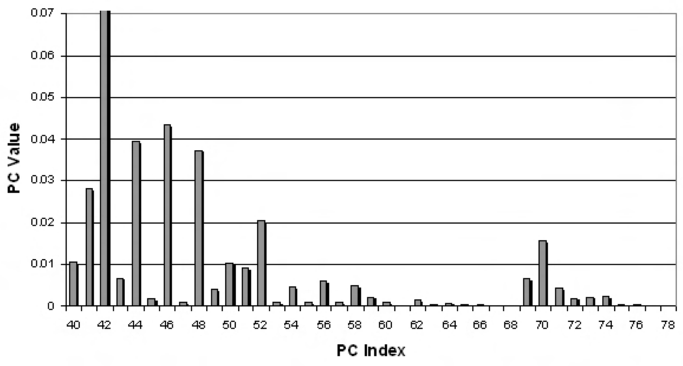


Figure 4.12: (Absolute) PC values of a posture modeled by hand for a few joints.

ture. Methods that can calculate dependent joint motions already exist, see for example the research done by Pullen and Bregler [106]. In their work, they adapt key-framed motions with motion captured data, depending on the specification of which degrees of freedom are to be used as the basis for comparison with motion capture data. Also, recent work by Liu et al.[79] presents a physics-based approach combined with analysis of motion capture data to automatically adapt motions according to different motion styles.

We will present a novel method that uses the PCs to create more natural looking motions. Our method is not as general as the previously discussed work, but it works very well within the upper body gesture domain. Furthermore, it is a very simple method and therefore suited for real-time applications.

The PCs are ordered in such a way that lower PC indices indicate high occurrence in the data and higher PC indices indicate low occurrence in the data. This allows for example to compress animations by only retaining the lower PC indices. Animations

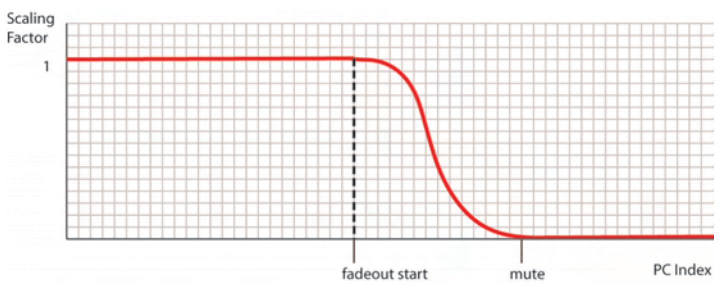


Figure 4.13: An example of a scaling filter that is applied on the PC vector representation of a posture.

that are close to the ones that are in the database that was used for the PCA, will have higher PC indices that are mostly zero (see Figure 4.11) for an example. An animation that is very different from what is in the database, will have more noise in the higher PC indices to compensate for the difference (see Figure 4.12). If one assumes that the database that is used for the PCA is *representative* for general motions that are expressed by humans during communication, then the higher PC indices represent the part of the animation that is 'unnatural' (or: not frequently occurring in the animation database). When we remove these higher PC indices or apply a scaling filter (such as the one displayed in Figure 4.13), this generates an error in the final animation. However, since the scaling filter removes the 'unnatural' part of the animation, the result is a motion that actually contains the movements of dependent joints. By varying the PC index where the scaling filter starts, one can define how close the resulting animation should be to the original key-framed animation.

To calculate the motions of dependent joints, only a scaling function has to be applied. Therefore this method is very well suited for real-time applications. A disadvantage is that when applying the scaling function onto the global PC vectors, translation problems can occur. In order to eliminate these translation artefacts, we have also performed a PCA on the upper body joints only (which does not contain the root joint translation). The scaling filter is then only applied on the upper body PC vector. This solution works very well since in our case, the dependent joint movements are calculated for upper body gestures only, whereas the rest of the body is animated using the idle motion engine. Figure 4.14 shows some examples of original frames versus frames where the PC scaling filter was applied.

4.5 Summary

In this chapter, we have presented various methods for a higher-level control of character motion. We have shown how motion graphs can be extended so that they support the definition of constraints. As a result, it is now possible to create different motions according to different emotional states. Secondly, we have presented our animation

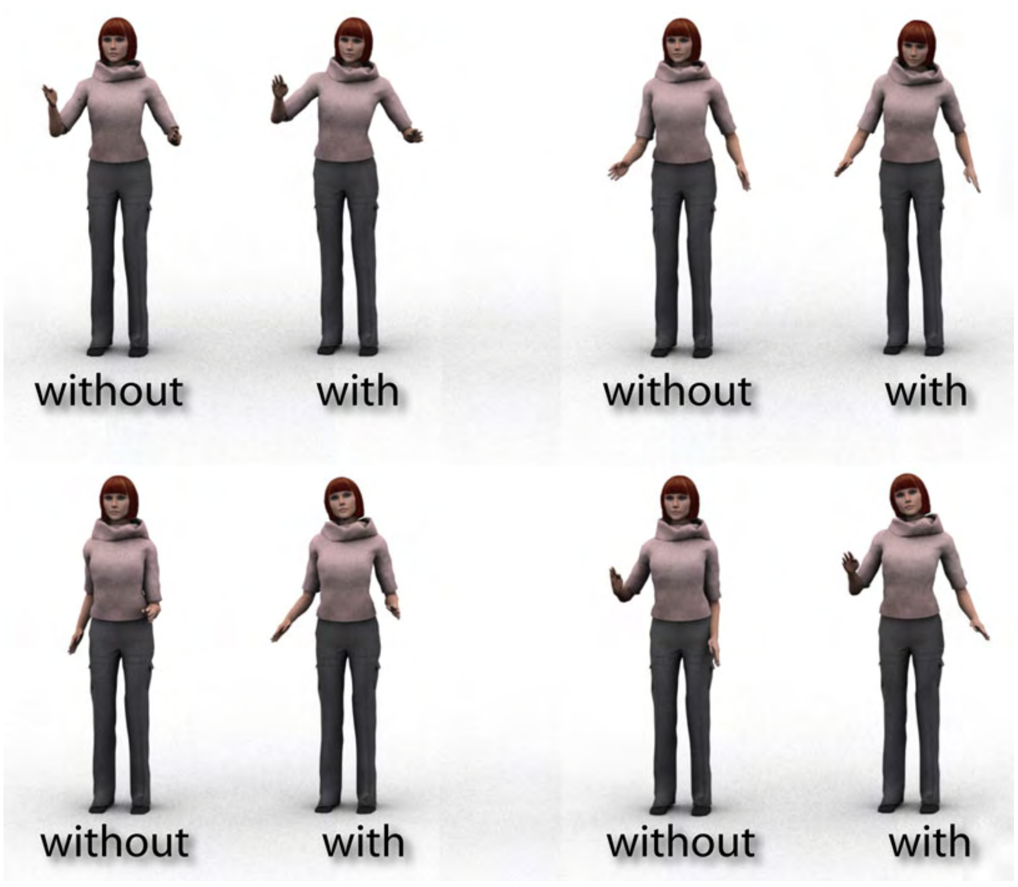


Figure 4.14: Some examples of key frame postures designed for a few joints and the same postures after application of the PC scaling filter.

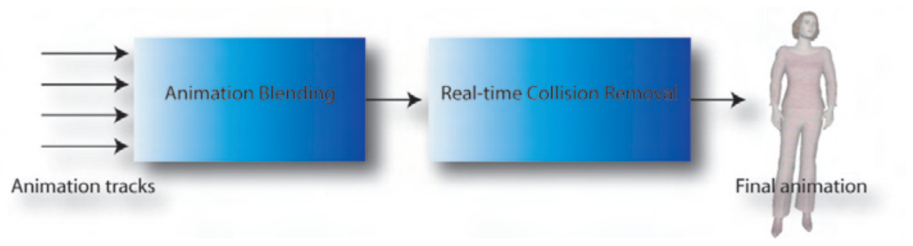


Figure 4.15: Animation pipeline with blending and real-time collision removal.

blending engine that can smoothly blend any type of animation, with a large set of tools available for designers. A collision-removal tool was presented, that can remove collisions from animations in real-time. Finally, we presented a method for real-time dependent joint motion synthesis. Figure 4.15 shows the full animation pipeline.

Now that a character can be controlled on a higher level, it becomes possible to use this system as a visualization component for an Interactive Virtual Human. The next chapter shows how this is done.

CHAPTER 5

Interactive Virtual Humans

A high level motion control mechanism is a crucial part of an Interactive Virtual Human. In the previous chapters, we have presented several useful techniques that allow to automatically generate animations that are realistic and that are highly flexible. In this chapter, we will show how our animation engine can be successfully linked with a system for simulating the behaviour and emotions of Virtual Humans. In the previous chapter, we have already shown how the emotional state can be used to change parameters of the body animation. In this chapter, we will propose a generic approach to emotion/personality modelling, which we use to automatically choose different body motions and facial expressions. We will present a dialogue management system, with integrated expressive information such as body and face motions. In Chapter 2 we have seen different representation languages for such kind of content. Using the motion synthesis and control techniques presented in the previous chapters, we can create a realistic animation from such a high level representation of movement.

5.1 Dialogue Management

As exhibited in Section 2.3, several approaches exist to automatically generate a response from an input given by a user. In order to test the flexibility of the animation system presented in the previous chapters, we need a simple approach that allows controlling several dialogue scenarios, with a capability to add expressive information. In order to add such expressive information, a representation of an emotional state is required. Many existing dialogue managers do not take into account the emotional state. Some systems have proposed emotions as an extension to the BDI model (this is called the ‘BDIE’ model [97]). However, such an approach would not suit our needs, because the BDI model requires a (generally complex) logical representation of the environment, which would complicate the facility to generate different test scenarios. The Jabberwacky [56] or ALICE [1] chatbot approach is a suitable candidate for easy scripting of dialogues, but it does not allow for extensions that govern different emotional states. In this section, we will present a system using a similar approach as ALICE, but with a possibility to add extensions such as an emotional state or other systems. Our system is based on the principle of **finite state machines** (FSMs). This approach allows us to perform a wide range of dialogue script tests, while retaining the flexibility to adapt the dialogue during run-time according to parameters such as an emotional state. In the next section, we will give a short overview of FSM theory, after which we will introduce how a simple dialogue system can be constructed that allows for different behaviour following an emotional state.

5.1.1 Finite State Machines

A **deterministic** finite state machine (or deterministic finite automaton, DFA) is a quintuple $(Q, \Sigma, \delta, q_0, F)$ where:

- Q is a finite set of states;
- Σ a finite set called the alphabet;
- $q_0 \in Q$ the start state;
- F a subset of Q called the final or accepting states, and
- δ is the transition function, a total function from $Q \times \Sigma$ to Q.

A **nondeterministic** FSM is the same as a deterministic FSM, except for the transition function. A nondeterministic FSM has a transition function δ that is a total function from $Q \times \Sigma$ to $\mathcal{P}(Q)$, thus there can be more than one possible transition with the same input[1].

An extension to this FSM theory can be given by allowing FSMs to generate output. This can be achieved in two ways: either output is associated with states (Moore machines) or the output is associated with transitions (Mealy machines). A common extension to the standard nondeterministic FSMs is to allow for the possibility to make a transition without input being processed. Such a transition is called a λ-transition. As a result, the transition function δ is a total function from $Q \times (\Sigma \cup \lambda)$ to $\mathcal{P}(Q)$.

[1]For a good introductory textbook on formal language theory, which encapsulates finite automata, see for example Sudkamp [121].

5.1.2 FSMs and Dialogue

We have chosen to adopt an FSM-based approach for creating dialogue systems. The main reason for this choice is the fact that FSMs are very easy to define and they allow for a wide variety of behaviour. In our system, we employ Mealy nondeterministic finite state machines that allow for λ-transitions.

Each transition in the FSM is linked with a set of conditions and a set of actions. A transition is only possible if its attached conditions are satisfied. When a transition is made, the attached actions are executed. Because the conditions and actions can be linked with modules that perform a task or that can check a statement, this framework allows for different behaviours depending on what types of conditions and actions are available.

The most basic dialogue system follows a pattern-response methodology: a user says/types something and the system responds accordingly. This approach is implemented in well-known systems, such as ALICE [1]. The same behaviour can be achieved using FSMs by defining a pattern matching condition and an output action, for example:

```
<condition type="input_match">
hello my name is *
</condition>

<action type="say">
hello how are you doing?
</action>
```

A small dialogue system can now be constructed with three states q_0, q_1, q_2 where q_0 is the start state and q_2 is the end state. There would be two transitions, one from q_0 to q_1 (linked with the matching condition) and one from q_1 to q_2 (linked with the action that generates output). Figure 5.1 shows a graphic representation of this small dialogue system. The action is linked with a module that provides the interface between FSM and user in the form of an output buffer. The condition is linked with a pattern matcher module and input buffer.

In order to achieve the same functionality as existing pattern-response dialogue systems, our system consists of a collection of these small FSMs, that are running concurrently. Each of these FSMs is called a *dialogue unit*. Every unit handles a specific dialogue between the user and the system. The full dialogue system consists of a kernel of these FSMs that are running and a set of modules that can each perform specific tasks and that have an interface with the FSM kernel. For an overview, see Figure 5.2. The separate modules can be easily added to our system, together with XML readers for module specific conditions and actions, such as reading variables, check if a logical statement is valid, and so on.

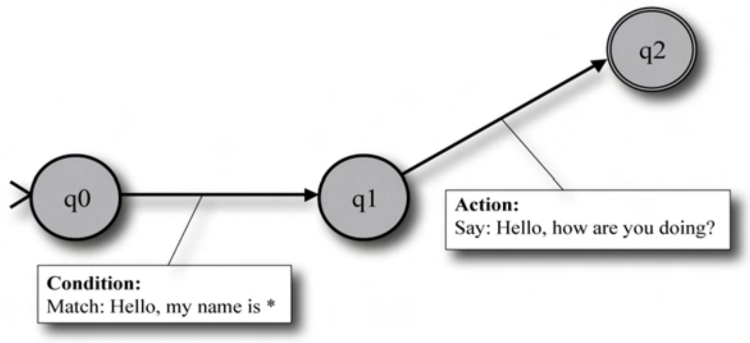


Figure 5.1: An example of a simple dialogue FSM.

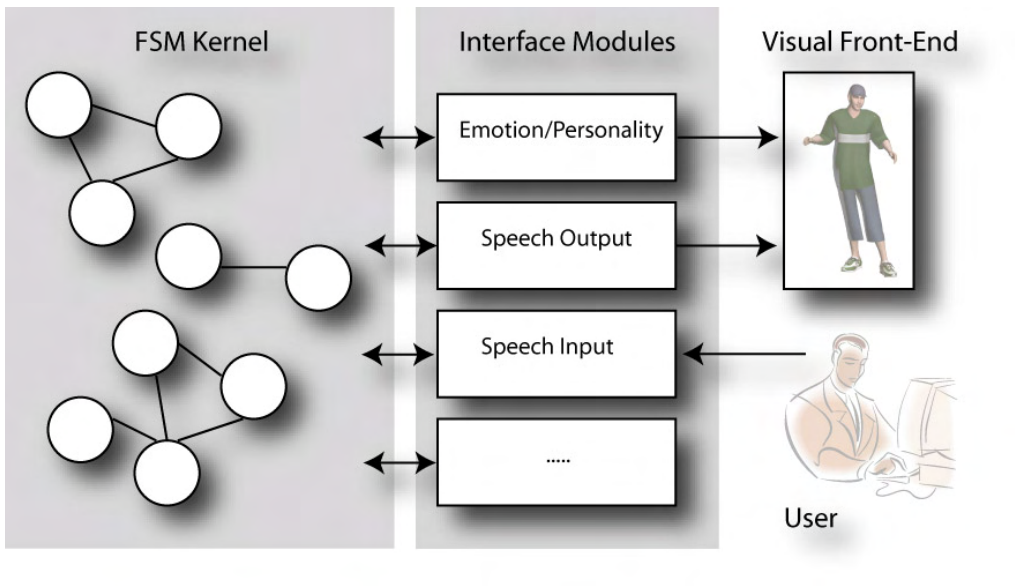


Figure 5.2: An overview of the dialogue system architecture.

5.1.3 The Parallel FSM Algorithm

As our FSMs are concurrently running in one system, we need to extend the definition of an FSM in order to add extra features that influence how the kernel chooses the FSM and the transition that has to be made. From now on, we will use $\mathcal{F}$ to indicate a Mealy nondeterministic finite state machine with lambda transitions. We will now **extend** the definition of $\mathcal{F}$ by defining an additional set of parameters (p, c, s) for it. Parameter p is an integer value in the interval $[0, \rightarrow\rangle$, that defines the *priority* of this FSM in the total system. FSMs with higher priority will be selected for performing a transition first. The second parameter, c, specifies if an FSM is *cyclic* or not. An FSM is called cyclic if it automatically goes to the start state when it reaches an accepting state (especially useful in dialogues for repeating conversation parts). The final parameter, s, specifies whether or not an FSM is *strong*. When an FSM is strong, and it is selected to perform a transition, it obtains a temporary priority of ∞, until it can no longer make a transition. This is a feature especially useful for dialogue. In practice it means that when user and computer are involved in a certain conversation (modelled by one FSM), then the computer will always interpret a user reply as part of the current conversation, if possible.

Two of the three parameters have a direct effect on how FSMs are selected during run-time: parameters s and p. In the following, we assume that we have a set of currently running FSMs V. Also there is a currently selected FSM F_s. Finally, the algorithm uses a set W to store the tentatively selected FSMs in.

```
$W = \emptyset$

for all $F_i \in V$
  if $F_i$ can make a transition
    if $W = \emptyset$
      $W = W \cup \{F_i\}$
    if $\forall k \cdot F_k \in W : \ p_i \geq p_k$
      $\forall k \cdot F_k \in W : \ p_k < p_i$:
          $W = W \backslash \{F_k\}$
      $W = W \cup \{F_i\}$

if $F_s$ can make a transition and $F_s$ is strong
  $W = \{F_s\}$

if $W \neq \emptyset$
   select a random $F_r \in W$
   $F_s = F_r$
```

If an FSM can be selected by this algorithm, then this FSM is allowed to do a transition. When the system is started, all FSMs are set into their start state q_0. Then we start a thread that checks if an FSM can be selected for transition with a certain time interval.

5.1.4 Dialogue Modules

Now that the basic architecture for the dialogue system has been established, a number of modules are available that provide for extra flexibility of the system.

Input Module This module compares an input string to a predefined pattern. It is related to a condition that evolves to *true* or *false* defining whether or not a string follows the pattern.

Output Module This module consists of an action that produces text output. The text can be formatted in different way and includes support for XML tags.

Emotion Module The emotion module handles the relation between the emotional state and the running dialogue. From the dialogue, the emotional state can be updated. Also, conditions are defined that indicate if a certain emotion is above or below a given threshold, thus allowing for different behaviour according to different emotional states (see also Section 5.2).

User Profile Module The goal of this module is to maintain information about the user, such as name, age, and so on. Also, it can serve as an interface between and automatic facial expression and emotion tracker [36].

5.2 Simulating Emotions and Individuality

A dialogue system or intelligent agent will require concrete representations of concepts such as personality and the emotional state so that it can decide what behaviour it will portray. As such, we need to define exactly what we mean by personality, emotion and other related concepts in order to describe how they can be simulated and used by other systems.

5.2.1 Definitions

When we speak of an *individual*, we always refer to it relative to a time t. The moment that the individual starts existing is defined by $t = 0$. The abstract entity that represents the individual at a time t we will call I_t from now on. In the simple case, an individual has a *personality* and an *emotional state*. Generally, an individual's personality is considered to be constant and initialized with a set of values on $t = 0$. The emotional state is dynamic and it is initialized to **0** at $t = 0$ (we will go in more detail about this later on). Thus we define I_t as a tuple (p, e_t), where p represents the personality and e_t represents the emotional state at time t.

As discussed in Section 2.3.2, there are many personality models that consist of a set of dimensions, where every dimension represents a specific property of the personality. Generalizing from these theories, we assume that a personality has n dimensions. In most personality models, each dimension can be denoted by a value in the interval $[0, 1]$. A value of 0 then corresponds to an absence of the dimension in the personality; a value of 1 corresponds to a maximum presence of the dimension in the personality. The personality p of an individual can thus be represented by the following vector:

$$p = \begin{bmatrix} \alpha_1 \\ \vdots \\ \alpha_n \end{bmatrix}, \forall i \in [1, n] : \alpha_i \in [0, 1] \tag{5.1}$$

As an example, we can specify a Five Factor personality (thus $n = 5$) that is very open, very extravert but not very conscientious, quite agreeable and not very neurotic:

$$p = \begin{bmatrix} 0.8 \\ 0.1 \\ 0.7 \\ 0.5 \\ 0.05 \end{bmatrix} \tag{5.2}$$

Emotional state has a similar structure as personality. The emotional state is a set of emotions with a certain intensity. The size of this set depends on the theory that is used. For example, in the OCC model, 22 emotions are defined, while Ekman [37] defines 6 that are used as a basis for facial expression classification. The emotional state is something that changes over time (for example due to various appraisal processes or simply a decay factor). Therefore, when we speak about an emotional state, we speak of it relative to a time t. We define the emotional state e_t as an m-dimensional vector. Again, most of the emotion models conform with the m emotion intensities in the interval $[0, 1]$. A value of 0 corresponds to an absence of the emotion; a value of 1 corresponds to a maximum intensity of the emotion. Such an emotional state vector is given as follows:

$$e_t = \begin{cases} \begin{bmatrix} \beta_1 \\ \vdots \\ \beta_m \end{bmatrix}, \forall i \in [1, m] : \beta_i \in [0, 1] & \text{if } t > 0 \\ \mathbf{0} & \text{if } t = 0 \end{cases} \tag{5.3}$$

When using the evaluation-activation disc as discussed in Section 2.3.2, the emotional state is represented by a two dimensional vector, with both the intensities in the interval $[-1, 1]$ (see also Equation 4.1):

$$e_t = \begin{bmatrix} e_e \\ e_a \end{bmatrix}, \text{ where } -1 \leq e_e, e_a \leq 1 \text{ and } \sqrt{e_e^2 + e_a^2} \leq 1 \tag{5.4}$$

Furthermore, we define an emotional state history ω_t that contains all emotional states until e_t, thus:

$$\omega_t = \langle e_0, e_1, \ldots, e_t \rangle \tag{5.5}$$

5.2.2 Updating the Emotional State

From a dialogue system, intelligent agent or any other A.I. system, the emotion framework will obtain emotional information. This information can, for example, be constructed based on the event appraisal model as described in OCC [93] or by the appraisal model as defined by Scherer [112]. We define the emotional information as a *desired change in emotion intensity* for each emotion. Then, the final emotion intensity can be calculated based on different parameters in the emotion framework. If we represent the desired change of emotion intensity by a value in the interval $[0, 1]$, then the emotion information a (or *emotion influence*) can be described by a vector that contains a desired change of intensity for each of the m emotions:

$$a = \begin{bmatrix} \delta_1 \\ \vdots \\ \delta_m \end{bmatrix}, \forall i \in [1, m] : \delta_i \in [0, 1] \tag{5.6}$$

The emotional state can then be updated using a function $\Psi_e(p, \omega_t, a)$. This function calculates, based on the personality p, the current emotional state history ω_t and the emotion influence a, the change of the emotional state as a result of the emotion influence. A second part of the emotion update is done by another function, $\Omega_e(p, \omega_t)$ that represents internal change (such as a decay of the emotional state). Given these two components, the new emotional state e_{t+1} can be calculated as follows:

$$e_{t+1} = e_t + \Psi_e(p, \omega_t, a) + \Omega_e(p, \omega_t) \tag{5.7}$$

Given the basic update model, little is said about *how* the emotional state is updated over time when an emotion influence has to be processed. There are a lot of different implementations possible, and more elaborate research should be performed to determine the ideal update process. In this section we only give a simple linear implementation of the model that does not take into account the emotion history. The linear implementation is given by the definition of the function $\Psi_e(p, \omega_t, a)$ and the function $\Omega_e(p, \omega_t)$. First, we will give a possible definition for Ψ_e. As we have a vector p of length n and a vector e_t of length m, we define an $m \times n$ matrix P_0 that we will call the *Personality-Emotion Influence Matrix.* This matrix indicates how each personality factor influences each emotion. P_0 has to be defined once, depending on which personality model and emotion model is used. Then, assuming that the personality p will not change, we can calculate the product of P_0 and p, which will give us a vector u indicating the importance of each emotion depending on the personality dimensions:

$$u = P_0 \cdot p = \begin{bmatrix} \epsilon_1 \\ \vdots \\ \epsilon_m \end{bmatrix} \tag{5.8}$$

We use this vector to construct the diagonal matrix P:

$$P = \begin{bmatrix} \epsilon_1 & 0 & \cdots & 0 \\ 0 & \epsilon_2 & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \cdots & 0 & \epsilon_m \end{bmatrix} \tag{5.9}$$

For each emotion, matrix P contains a value that defines how strong an emotion can be given the personality p. For example, a very agreeable and extravert personality will have a highly positive value for the 'joy' emotion and a very small value for the 'distress' emotion. Given the matrix P, $P \cdot a$ calculates the change in the emotional state given the emotion influence a. Thus, the function Ψ_e can be defined as follows:

$$\Psi_e(p, \omega_t, a) = P \cdot a \tag{5.10}$$

As can be noted, this implementation makes no use of the emotional state history. For the Ω_e function, we will only show the implementation of a simple, personality-independent decay. The Ω_e function is defined as an m-dimensional vector (there are m emotions):

$$\Omega_e(p, \omega_t) = \begin{bmatrix} -C_e \\ \vdots \\ -C_e \end{bmatrix} \tag{5.11}$$

C_e defines the amount of decay for each emotion.

5.2.3 Integration with Dialogue Manager

Based on the previously linear implementation of the emotion update process, we defined an OCEAN personality (5 dimensions), as well as an emotional state based on the OCC model. In addition to the 22 emotions defined in the OCC model, we have added *surprise* and *disgust*, following the proposition by Kshirsagar [71]. The complete list of emotions is given in Table 5.1. The personality and emotion module is interfaced with

Admiration	Gratification	Pride
Anger	Gratitude	Relief
Disappointment	Happy-for	Reproach
Disgust	Hate	Remorse
Distress	Hope	Resentment
Fear	Joy	Satisfaction
Fear-confirmed	Love	Shame
Gloating	Pity	Surprise

Table 5.1: The emotions used in the dialogue system, defined by the OCC emotions [93] and surprise+disgust.

the dialogue system through conditions and actions that can be specified in XML. The FSMs from the dialogue system can generate an emotional influence by specifying the following action:

```
<action type="emotional_influence">
   <emotion>hate</emotion>
   <value>0.6</value>
</action>
```

Also, an FSM can check if an emotion is below or above a certain threshold by specifying the following condition:

```
<condition type="above_threshold">
<emotion>fear</emotion>
<value>0.3</value>
</condition>
```

By using these conditions and actions in the dialogue script, different emotional states can trigger different dialogue responses.

5.3 From Dialogue to Animation

The dialogue manager generates responses that are tagged using XML. These tags indicate where a gesture should start and end. As we have already discussed in Section 2.3.3, there are many different representation languages for multimodal content, for example the Rich Representation Language (RRL)[69] or the Virtual Human Markup Language (VHML)[123]. In this section, we will give an example of how such a representation language can be used to control gesture sequences in our system. For testing purposes, we have defined a simple tag structure that allows for the synchronized playback of speech and non-verbal behaviour. An example of a tagged sentence looks like this:

```
<begin_gesture id="g1" anim="shake_head"/>Unfortunately, I have
<begin_gesture id="g2" anim="raise_shoulders"/>no idea
<end_gesture id="g2"/> what you are talking about.<end_gesture
id="g1"/>
```

Within each gesture tag, an animation ID is provided. When the gesture animation is created, these animations are loaded from a database of gestures—also called a Gesticon [69]—and they are blended using the previously described blending engine. The timing information is obtained from the text-to-speech system. Although this is a very rudimentary system, we believe that this way of generating gestures can easily be replaced with another, more elaborate gesture synthesizer, since the animation system is completely independent of what happens on the gesture construction level. The animation system only activates actions at given times with specified animation lengths and blending parameters. Although our current testing system only generates gestures in synchrony with speech, this is not a limitation of the animation system. The animation system is capable of handling any set of actions at any time, even without speech.

5.3.1 From Dialogue to Facial Animation

In this section, we will shortly explain the techniques used to create the facial animation from the output text and speech. The output text is first converted into a speech signal by the text-to-speech engine. At the basic level, speech consists of different phonemes. These phonemes can be used to generate the accompanying face motions, since every phoneme corresponds to a different lip position. The lip positions related to the phonemes are called *visemes*. There are not as many visemes as phonemes, because some phonemes revert to the same mouth position. For example, the Microsoft Speech SDK defines 49 phonemes, but only 21 different visemes. For a more detailed overview of the different phonemes and visemes, we refer to Appendix D.

For each viseme, the mouth position is designed using the MPEG-4 Face Animation Parameters (FAPs). Constructing the facial motion is achieved by sequencing the different mouth position, taking into account the speech timing obtained from the TTS engine. An important issue to take into consideration when creating facial speech is *coarticulation*, or: the overlapping of phonemes/visemes. Generally, coarticulation is handled by defining a dominance function for each viseme. For example, Cohen and Massaro [26] use this technique and they define an exponential dominance function. Similarly, we use the following base function to construct the coarticulation curve:

$$f(x) = e^{-\alpha x} - x \cdot e^{-\alpha} \tag{5.12}$$

where $0 < \alpha < \infty$. The parameter α governs the shape of the curve. Figure 5.3 shows the curve for different values of α. Using this base function, the final coarticulation dominance function is defined as follows:

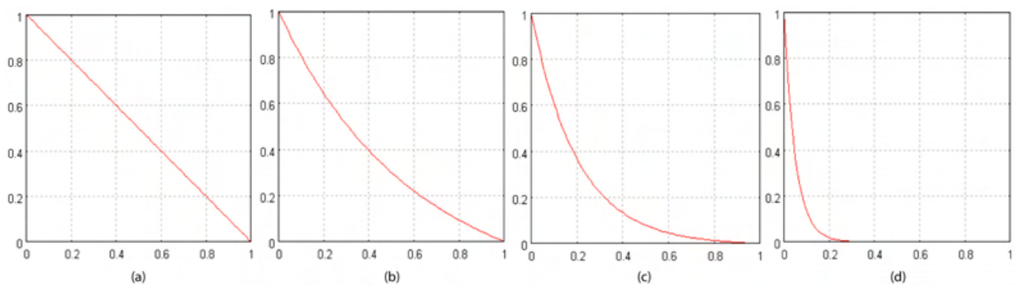


Figure 5.3: Coarticulation base function with different α values: (a) $\alpha = 0$ (b) $\alpha = 2$ (c) $\alpha = 5$ and (d) $\alpha = 10$.

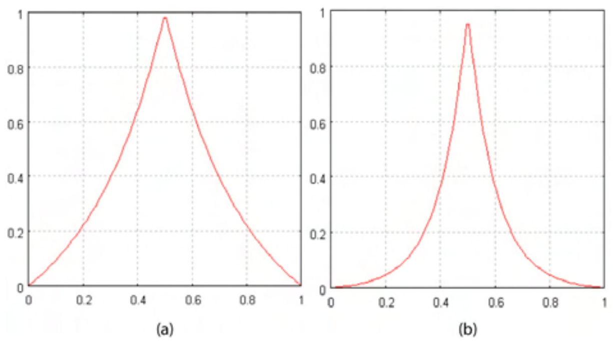


Figure 5.4: Example of complete coarticulation functions with different α parameters: (a) $\alpha = 2$ and (b) $\alpha = 5$.

$$C_\alpha(x) = e^{-\alpha 2|x-0.5|} - 2|x-0.5| \cdot e^{-\alpha} \tag{5.13}$$

Two different examples of the $C_\alpha(x)$ dominance function are given in Figure 5.4. For each viseme, the value of the α parameter can be chosen. Also, the weight of each function, as well as its spread (overlap) can be defined for each viseme. Appendix D shows an example definition of the curves used for the different visemes.

Because of the generic structure of the MIRAnim engine (see Section 4.2), it is a simple task to create the facial animation from the viseme timing information. We define a blending action for each viseme, where the dominance function acts as the weight curve. The blending schedule containing the different blending actions will then automatically perform the viseme blending. Because we use direct FAP blending, our approach also handles tongue movements (see Figure 5.5, as opposed to earlier work by Kshirsagar [72], who used a Principal Component representation of recorded face motions that didn't include tongue movements.

Next to the facial speech motion, also any facial gestures need to be added, defined as tags in the text. An example of an eyebrow raising facial gesture could be defined as follows:

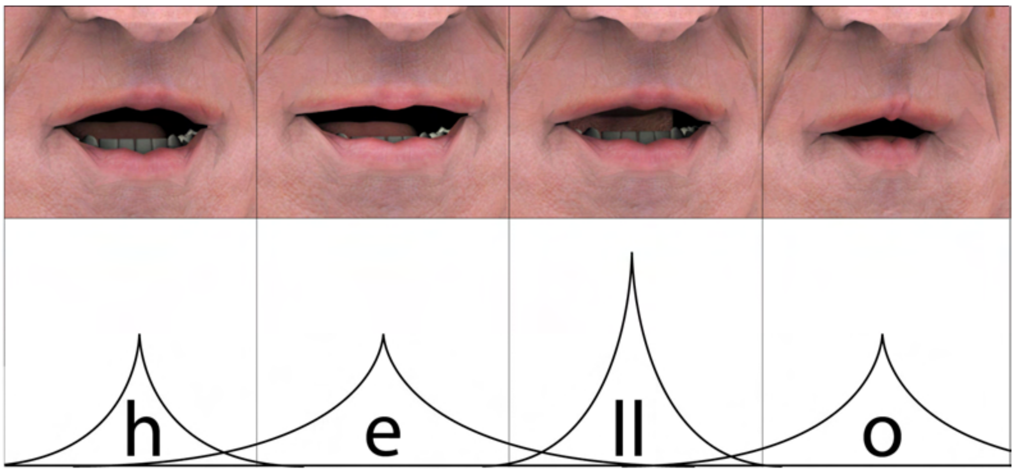


Figure 5.5: Facial animation for *hello* that takes tongue movement into account.

Facial Expression	**Emotion**
Joy	Happy-for, Gloating, Joy, Pride, Admiration, Love, Hope, Satisfaction, Relief, Gratification, Gratitude
Sadness	Resentment, Pity, Distress, Shame, Remorse, Disappointment
Anger	Anger, Reproach, Hate
Surprise	Surprise
Fear	Fear, Fear-confirmed
Disgust	Disgust

Table 5.2: Mapping of OCC emotional on facial expressions [71].

```
Are you <begin_gesture id="g1" anim="raise_eyebrows"/>really
sure about that?<end_gesture id="g1"/>
```

In addition to the facial gestures derived from the tagged text, the emotional state is also shown on the face by mapping the expression onto one of the six facial expression defined by Ekman [37, 71]. The mapping we use is given in Table 5.2. The intensity of the emotion is included by applying a corresponding scaling factor in the blending parameters. Finally, a face blinking generator is added for increased realism. Each face animation track has different weights for the FAPs. The speech animation has a higher weight on the mouth area, whereas the face expression weights are higher on the eye and eyebrow area. By blending the different facial animation tracks, the final animation is obtained.

5.3.2 From Dialogue to Body Animation

Very similar to facial animation synthesis, the body animation is also partly generated from the tagged text. The same definition is employed for the body animation tags. An example of a body animation in combination with speech is given as follows:

```
<begin_gesture id="g1" anim="hello"/>Hello there,
<end_gesture id="g1"/> how are you doing today?
```

Similar to the facial animation synthesis, the TTS timing information is used to plan the length of the gesture motions. A blending fade-in and fade-out is applied on the gesture motions in order to avoid unnatural transitions. Also, the automatic dependent joint motion filter explained in Section 4.4 is applied on the gesture signal. The filter is applied on the upper body and starts fading out at PC index 20 with a mute for PCs > 30 (of a total of 48).

Next to the gesture motions, the idle motion engine is running continuously, therefore providing the IVH with continuous idle motions. These idle motions are automatically changed according to the emotional state obtained from the dialogue system, using the method explained in Section 4.1. Just like for generating the facial expressions, a mapping is required to translate the OCC emotions into a position on the emotion activation-evaluation disc. We have assigned to each OCC emotion, as well as the *surprise* and *disgust* emotion a position on the activation-evaluation disc. Therefore a correspondence matrix D needs to be defined, that specifies for each emotion i its position $(\mu_{a,i}, \mu_{e,i})$, as follows:

$$D = \begin{bmatrix} \mu_{a,1} & \mu_{a,2} & \cdots & \mu_{a,m} \\ \mu_{e,1} & \mu_{e,2} & \cdots & \mu_{e,m} \end{bmatrix} \tag{5.14}$$

A possible distribution of these emotions is given in Figure 5.6. The two-dimensional activation-evaluation emotional state b_t that corresponds to the OCC emotional state e_t (see Equation 5.3) is then calculated as follows:

$$b_t = D \cdot e_t \tag{5.15}$$

In order to obtain the final animation, the resulting emotional idle motions and the gesture motions with dependent joint movements are blended on-the-fly by the MIRAnim engine (see Figure 5.7). Figure 5.8 shows some examples of full body postures obtained using our approach.

5.4 Summary

In this chapter, we have presented a testing dialogue manager that includes emotional information. We have shown how such a dialogue system can be used to control the

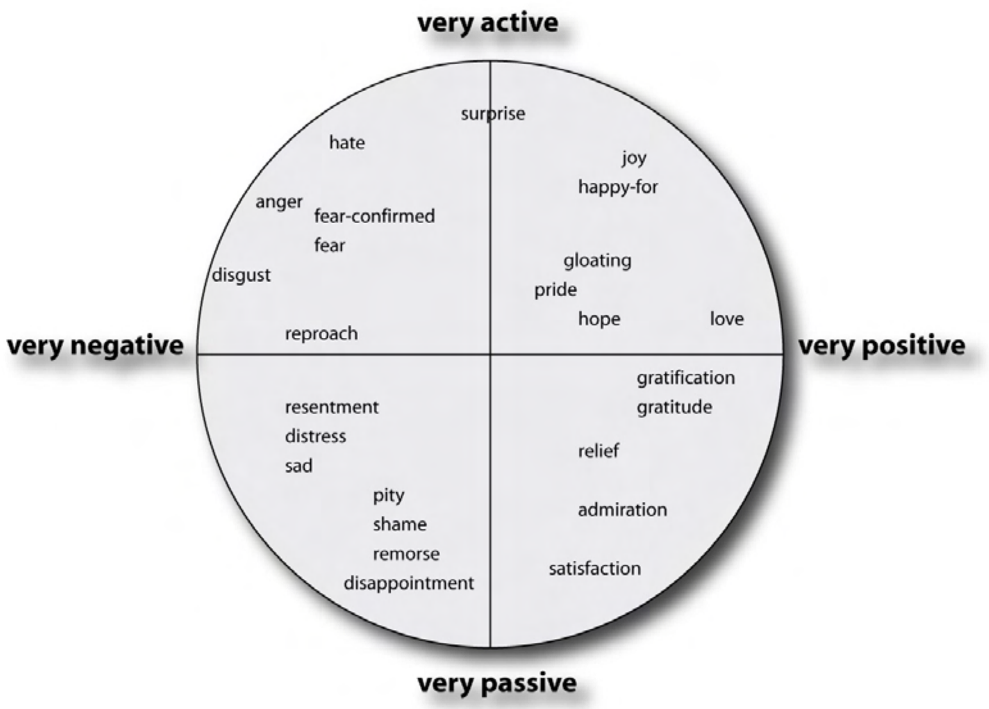


Figure 5.6: Possible mapping of OCC emotions and surprise+disgust on the activation-evaluation disc.

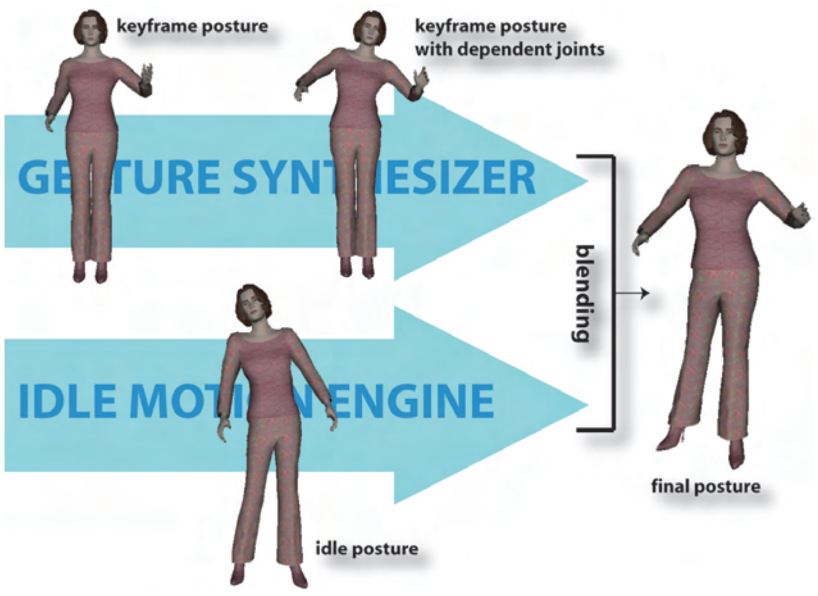


Figure 5.7: Integration of gesture animations, dependent joint motion synthesis and idle motion synthesis.

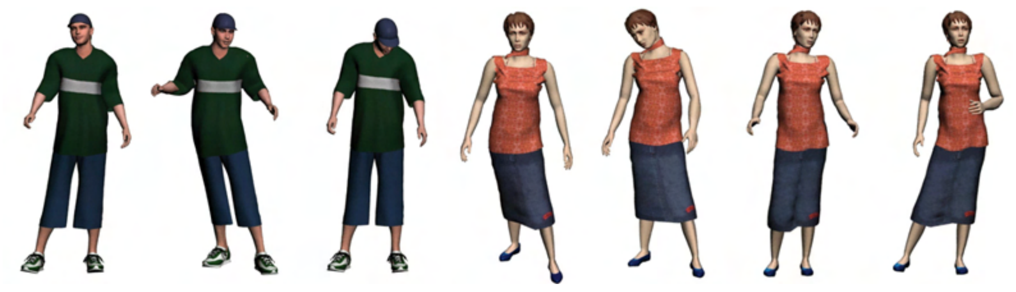


Figure 5.8: Some example postures that show emotional idle motions mixed with gestures with automatic dependent joint motions.

motions of an interactive character. Our approach overcomes the limitations of existing gesture synthesis systems, which do not produce realistic motions. We have shown how motion synthesis and control techniques can be used *in combination* with gesture synthesis systems, in order to produce motions that are both flexible and realistic. Furthermore, our approach provides a fully integrated face and body control system that allows a parameterization depending on the emotional state of the characters. The face and body motions are generated on-the-fly from a high-level representation and they are smoothly blended in with other motions, using the animation engine. In the next chapter, we will discuss some implementation issues of the system. We will also show how our approach can be used to add interactive virtual characters to a scripted scenario, where one can switch dynamically from scripted character control to interactive character control, without any glitch in the animation sequence.

CHAPTER 6

Implementation

In this chapter, we will show in short how the previously described techniques are implemented. We will first give an overview of the framework in which our approach has been integrated. Then we will show the two main components that form the implementation: the animation service and the interaction service.

6.1 The VHD++ Real-Time Animation Framework

VHD++ is a real-time virtual environment framework developed at MIRALab and VR-Lab [105] (see Figure 6.1). This framework allows quick prototyping of VR-AR applications featuring integrated real-time virtual character simulation technologies. The main advantage of using VHD++ is that it is a component-based framework that allows the plug-and-play of different human simulation technologies such as: real-time character rendering in AR (supporting real-virtual occlusions), real-time camera tracking, facial animation and speech, body animation with skinning, 3D sound, cloth simulation, scripting of actions, and interaction. The different components may be grouped into the two following main categories:

- System kernel components responsible for the interactive real-time simulation initialization and execution.
- Interaction components driving external VR devices and providing various GUIs allowing for interactive scenario authoring, triggering and control.

The content to be created and used by the system can be anything ranging from models of the 3D scenes, virtual humans, objects, animations, behaviours, speech, sounds, python scripts, and so on.

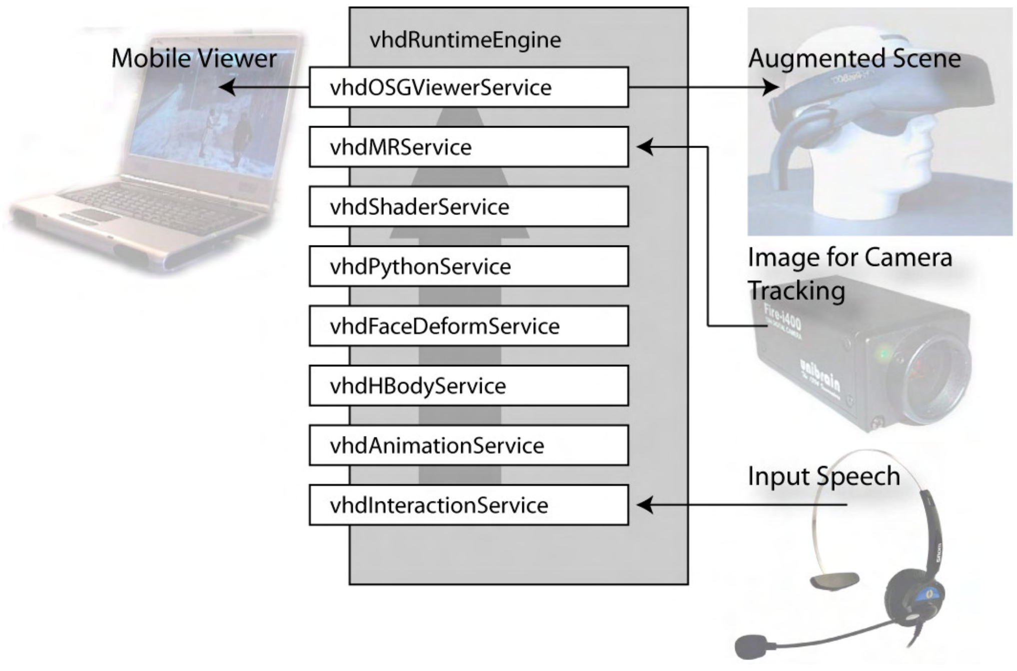
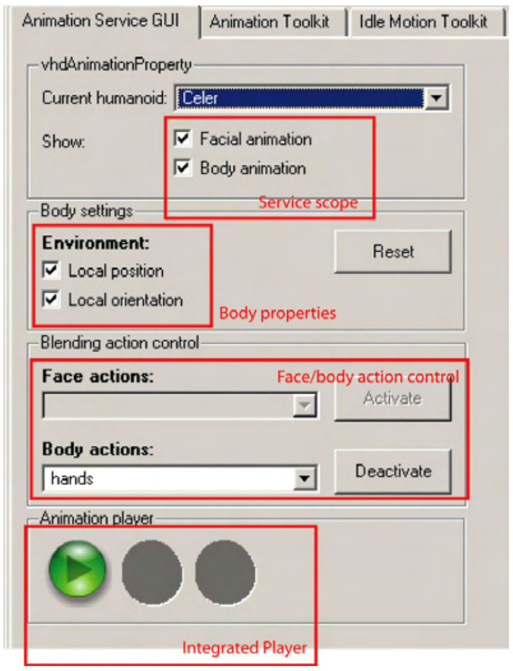


Figure 6.1: VHD++ real-time virtual environment framework.

The software architecture is composed of multiple software components called services, and their responsibilities are clearly defined. They have to take care of rendering of 3D simulation scenes and sound, processing inputs from the external VR devices, animation of the 3D models and in particular complex animation of virtual human models including skeleton animation and respective skin and cloth deformation. They are also responsible for maintenance of the consistent simulation and interactive scenario state that can be modified with python scripts at run-time. Additionally, a set of very useful features exist:

- XML interface for reading and initializing system data. This allows the dynamic loading of scenes, models and animations, or any other type of data required by a service.
- A Python control interface. This allows users to run scripts that directly access features of the services, which is very useful for running AR/VR scenarios.
- All services and GUIs are DLLs. This means that it is very easy to integrate and test various system components.
- CVS based development and availability. CVS is a tool for collaborative software development. Adding features and libraries to VHD++ is a very simple task.

The advantages of using such a framework are numerous. Not only does it allow for an easy test bed of the software that is development, it also ensures an easy use of



Current humanoid: Celer
Show: ✓ Facial animation ✓ Body animation
Environment: ✓ Local position ✓ Local orientation
Face actions:
Body actions: hands

Figure 6.2: The animation service GUI. The GUI includes facilities to easily activate and deactivate actions, as well as player control.

the software by others. Furthermore, because everyone is developing within the same framework, it is much easier to establish a 'best practice' for software development within the group of developers.

All of the software related to the work presented in this thesis has been integrated into the VHD++ environment. We will present a short overview of the software that is concretely available as a part of VHD++. The software is centred around two VHD++ services. The *vhdAnimationService* contains all the animation related components: motion synthesis, blending, loading and saving animations and so on. The *vhdInteractionService* uses the animation service to control a virtual character. The interaction service includes speech recognition and synthesis, a dialogue manager and emotion/personality simulation. In the next sections, we will give an overview of each of these services.

6.2 The Animation Service

The animation service that has been developed consists of the service itself, as well as two GUIs to control the service. For each virtual human, a *vhdAnimationProperty* is defined, that contains a blending schedule, and some settings for each human, such as an option to choose whether or not facial and/or body animation should be played or if the translation/orientation of the virtual human was defined on the global (world) coordinate system or local coordinate system. The service also includes an integrated

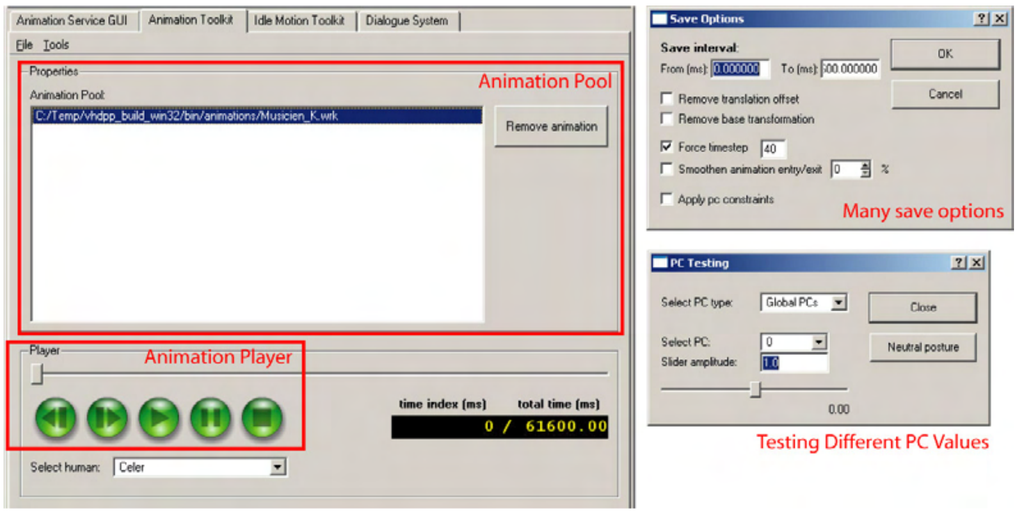


Figure 6.3: The animation toolkit.

player, that plays and blends scheduled animation in a separate thread. Figure 6.2 shows the GUI that is used to control the animation service.

A second useful GUI that has been developed is the Animation Toolkit (see Figure 6.3). This is a simple animation loader, player and editor, that is very practical for testing purposes. Several animations can be loaded in a pool, animations can be saved in several formats, and many other options available. For example, a single frame can be saved, a part of an animation can be saved, or global orientation and translation can be removed if required. Finally, a simple tool is available to test various values of Principal Components.

The GUIs we have described in this section are not the only means available to control the characters. In VHD, a Python interpreter is available that allows users to play scenarios by sending directly commands to the different services. This interface has also been implemented for the animation service. As such, a script can now be used to control the characters. In the following example, two characters are controlled by a script. The script activates pre-recorded actions (such as 'creon_wrk') and actions linked with the motion synthesizer (such as 'creon_idle'). These different actions are blended on the fly and played on the characters. Both sound and body motions are played in synchrony.

```
# global variables
Antigone="Antigone_vhd_occ_rec"
Creon="Creon_vhd_occ_rec"
cam01Creon_crowd="root.Cam01Creon_crowd"

# start the sound
soundService.sndmpPlayMedia(cam01Creon_crowd)
```

```
# no facial animation
animationService.initPlayerScope_face(False)

# start the player
animationService.start_player()
animationService.activateAction_body(Antigone,"antigone_idle")

# creon monologue
animationService.activateAction_body(Creon,"creon_wrk")
voiceService.activateAction(Creon,
  "CreonSpeech.Cam01CreonP1",1.0)
animationService.waitUntilActionHasFinished_body(Creon,
  "creon_wrk",-4.0)
animationService.activateAction_body(Creon,"creon_idle")

# Antigone answers to Creon
animationService.cutOffAction_body(Antigone,
  "antigone_idle",3.0)
animationService.activateAction_body(Antigone,"antigone_wrk")
voiceService.activateAction(Antigone,
  "AntigoneSpeech.Cam01CreonP3",1.0)
animationService.waitUntilActionHasFinished_body(Antigone,
  "antigone_wrk",-4.0)
animationService.activateAction_body(Antigone,"antigone_idle")

soundService.sndmpStopMedia(cam01Creon_crowd)
```

Figure 6.4 shows some examples of the resulting animations obtained by playing various scenarios, including the one presented in this section.

6.3 The Interaction Service

The interaction service includes a dialogue manager, a speech recognition interface and a speech synthesizer. The dialogue scripts are read from an XML file, and they can be dynamically loaded in the GUI. Recording and interpreting speech can be done with any available microphone and speech is recorded between clicking and releasing a button. The interaction service is directly linked with the animation services, and it creates and activates new face and body actions. Because the same service is used for both scripted and interactive animation, it is possible to first play a script as shown in the previous section, and then dynamically switch to animation controlled by the interaction service. Since the animation playing itself is handled by the animation service, the character animation is not disrupted when this switch is made. Figure 6.5 shows the main interface for interaction with a virtual human. Several options are available,

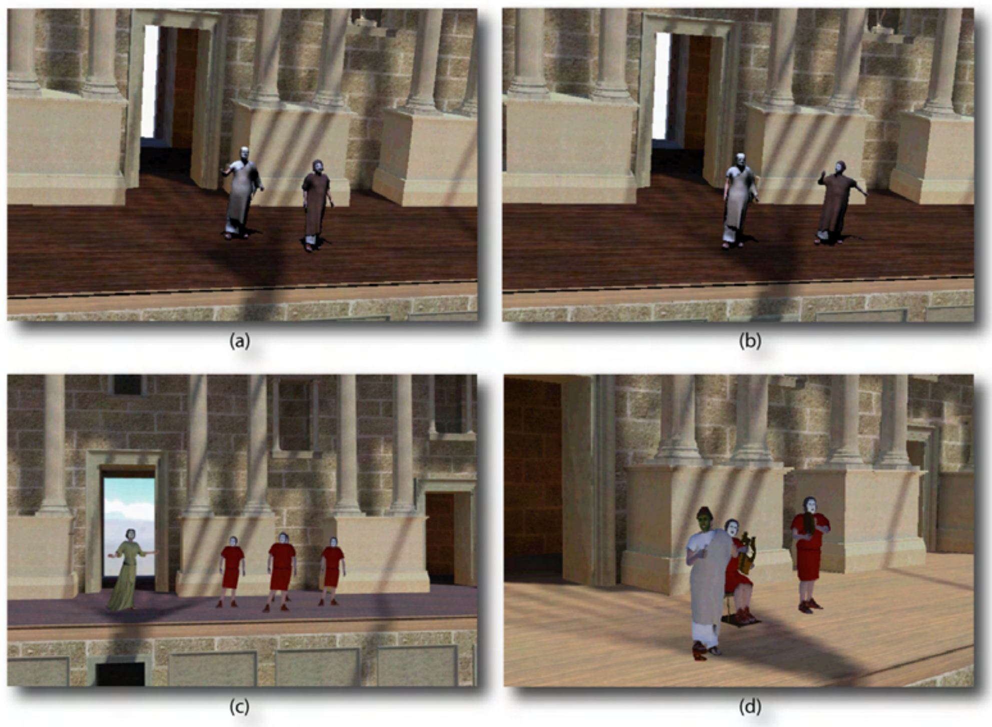
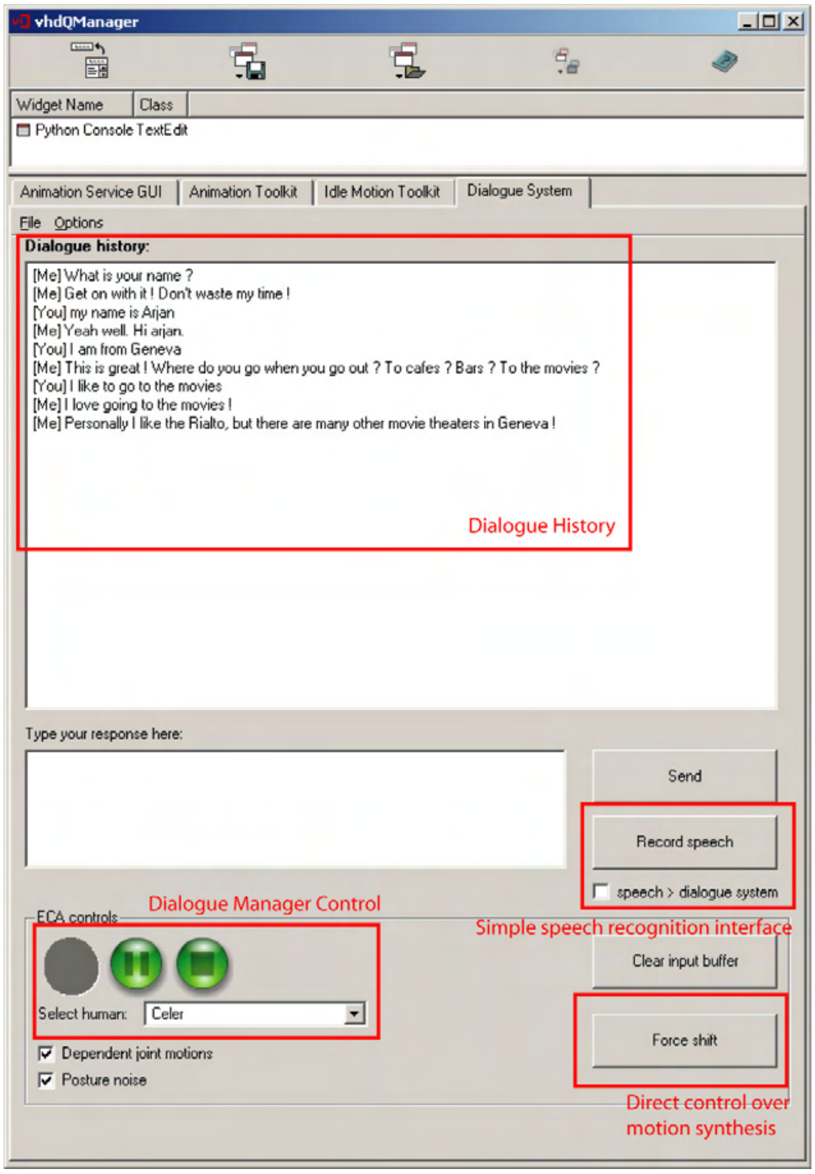


Figure 6.4: Example animations played using the animation engine. Images (a) and (b) are taken from the example scenario.

including the control over which character to select for interaction, the voice to be used for the text-to-speech, and the type of PC filter to use for the dependent upper body joint motion synthesis. Additionally, a direct control is available on the motion synthesizer, in order to directly instruct it to perform a transition.

6.4 Summary

In this chapter, we have presented the implementation of the techniques shown in the previous chapters. Since we have chosen to use VHD++ as the main supporting framework for our software, a high accessibility for other users is ensured. All tools and software that was developed related to this work has been used and tested by various designers in MIRALab. We have presented an overview of the *animation service*, which manages animation of both body and face for all virtual humans in the scene. Next, we have presented the *interaction service*, which encloses the dialogue system and emotion simulator. Because the animation service handles the animation continuously—even during interaction, a dynamic switch between key-frame scenario playing and real-time interaction is now possible without interruption of the running animation. The animation service is currently a crucial part of project demonstrations where scenarios are played on different virtual humans.



[Me] What is your name ?
[Me] Get on with it ! Don't waste my time !
[You] my name is Arjan
[Me] Yeah well. Hi arjan.
[You] I am from Geneva
[Me] This is great ! Where do you go when you go out ? To cafes ? Bars ? To the movies ?
[You] I like to go to the movies
[Me] I love going to the movies !
[Me] Personally I like the Rialto, but there are many other movie theaters in Geneva !

Figure 6.5: The Interactive Virtual Human interface.

CHAPTER 7

Conclusion

In the previous chapters, we have discussed our research in Interactive Virtual Humans. Since the field of Interactive Virtual Human research is vast, we have addressed many different topics and we have proposed various contributions to each of these fields. In this chapter, we will give an overview of our contributions in the light of the goals that we set out at the end of Chapter 2. Then, we will discuss limitations of our approach, as well as interesting avenues for future research.

7.1 Contributions

In this section, we will look in more detail to each of the objectives that we have specified in Section 2.5.

Real-time animation manipulation We have presented a novel animation representation, based on a Principal Component Analysis of motion clips obtained using motion capture techniques. PCA is a well-known technique that has been applied before in many different contexts. However up until now, PCA on body motions has been rare, because of the constraints on the representation of orientations. We have shown that a PCA can be successfully applied on an exponential map representation of orientations. We have shown very effective PC-based algorithms for fast animation manipulation, notably the powerful distance criterion (Section 3.3.1) and the real-time animation fitting algorithm (Section 3.3.2). The MIRAnim animation engine is built around this structure and provides for fast blending and playing of animations of any type. The animation manager is suitable for various types of applications and it allows switching between scenario-based character control and interaction-based character control during run-time, without interrupting the animations that are being played.

Flexible motion synthesis Drawing on the efficiency of the distance criterion and the animation fitting algorithm, we have demonstrated a motion synthesis technique that does not need a pre-computation cycle, as opposed to existing methods, such as the Motion Graph technique [66]. This allows for a possibility to dynamically adapt the motion graph. Additionally, our method is based on a graph with vertices that have a meaning (see Section 3.3.2). As a result, a useful control mechanism is established that allows to choose a desired path through the graph, a feature which is not available in existing methods. We have also presented a high-level emotional body posture control method by employing a strategy that places constraints on edges in the graph (see Section 4.1). Our technique is independent of the motion type, as opposed to existing systems that focus on a specific motion type, such as walking [116].

Automatic realistic gestures We have shown an effective method to automatically generate motions of dependent joints in real-time, given the motion for only a few joints (Section 4.4). The PC filter is very easy to implement and parameters can be chosen to adapt the amount of motion to be added. Our approach does not place any limitations on the input motion. Also, because of the integration with the automatic motion synthesis, our approach ensures a natural posture even when no gesture animation is being placed. The noise function placed on the PC values guarantees that the character is never completely static. This adds a liveliness to the character that was not available in previous systems.

Full-body expressive interaction We have developed an interactive system that generates proper responses given a user input. Our approach allows for the specification of both face and body motions that are played in synchrony with a speech signal. Again, the resulting motions are smoothly blended with already running motions. Because of the independent animation management, our approach can be easily integrated with existing interactive systems. Additionally, we have presented a system for the simulation of personality and emotions. The emotional state is used to automatically show emotions on both the body and face, which was not yet available for interactive characters.

As shown in Chapter 6, we have developed a working prototype of the fully integrated system as a part of VHD++. Several softwares are available to test the various methods that were developed during this thesis work. An easy-to-use toolkit has been developed, so that motions can be played and adapted according to the animator's needs. Additionally, the prototype has been integrated with a scenario-playing tool, so that the animation engine can be used for playing an animation scenario with multiple characters in a 3D environment. The interaction service adds the possibility to interact with these characters when desired.

Interactive Virtual Humans have many different applications. Already in many games, we see the need for such characters, and the level of control that is demanded over such characters keep increasing. If in earlier games many characters where still static, or only moving according to a very limited set of motion clips, nowadays we see complex human motions in games such as The Sims [115]. Not only the entertainment industry benefits from realistic interactive characters. 3D Virtual heritage applications define detail scenarios of virtual people in an historic environment. Giving these virtual people the possibility to interact with for example visitors in a museum will start a new kind of historic experience, where borders between the virtual and real environment will become more transparent. Finally, realistic human behaviour in a virtual environment can aid people to cure social phobia, or it can be used for various training exercises. The technologies explained in this thesis help to increase the realism of IVHs, bringing us closer to a truly natural interactive experience with a virtual human.

7.2 Limitations and Future Work

Although we have proposed several contributions to the state of the art in this thesis, a lot of work still remains to be done in the area. Just like any other research, ours has its shortcomings. We do believe that the use of the technologies that we have discussed in the previous chapters is a step in the right direction, but we identify several interesting avenues of research in this exciting field.

Automatically selecting gestures from generated output text In the current implementation of our system, the output animation sequence is generated from tagged text. Although such a method is sufficient for testing the animation engine that control the virtual human motion, it is a rather time-consuming job if a complex interactive script needs to be developed. In order to allow for IVHs which are easy to develop, a system is required to do this automatically. There has already been some research in this area, notably BEAT [19] or MAX [64], but these systems need a lot of pre-processing and annotation work. An interesting research direction would be to automatically annotate motion capture data of people during conversations, and then based on that data, to try to automatically generate new gesture motions in synchrony with speech generated by a TTS system. Such an approach would also allow including more complex full-body gestures.

Integration with more complex motion control Although blending between different types of motions in our system is very easy to perform, the system does not rely on fixed physical parameters. As such, some foot skating or collisions can occur. We have proposed a few simple approximate solutions for these problems, but the interactive system does not yet have a complete control over the body. More complex actions, such as running, ducking, jumping or sitting down, are very difficult to control from tagged text. A more generic action selection mechanism needs to be developed that can control such types of motions, but that is still flexible enough.

Interaction with objects in the environment A very important point that is not addressed in this thesis work is the interaction with objects in the environment. Often during interactions, one is pointing at other objects, picking up things or using objects to perform a certain task. Also it is desirable to add look-at behaviour to the virtual human, for which the integration with the camera position in the environment is required. The controlled interaction with objects in a virtual environment is a very difficult research problem, which is still in its early stages. Joint-based animation methods do not have a direct spatial representation of the end-effectors in the body, making it difficult to define motions that interact with objects such as a door or a drawer. More difficult even is the interaction with objects in combination with an interactive system. In order to achieve this, a model of virtual perception needs to be included, so that the virtual human has a feedback of what is happening in the environment. This perceptive feedback again will have an effect on the virtual human behaviour.

Modelling the inside of the Virtual Human Most virtual humans are 3D geometrical models, whose motion is controlled by manipulating a skeleton. The 'inside' of these virtual humans is empty. A real challenge would be to model this 'inside' part of a virtual human. We have already addressed this by trying to model emotions and personality. However, true human beings adapt their behaviour according to many different physiological signals. These signals do not only affect the behaviour of humans, but it also affects our appearance, for example blushing, sweating, crying, and so on. These physiological processes form a key part of our expressions and behavioural system, but there is almost no research addressed to modelling these signals as an integral part of virtual human behaviour. Additionally, more basic motivators such as hunger or sleepiness could be implemented. As a final result, a virtual human will not cease to exist when an application is terminated, but he/she will go to sleep or do other things, which in turn would inspire new conversations with the user when the application is launched again.

Bibliography

[1] Alice webpage. http://www.alicebot.org. Accessed May 2006.

[2] J. Ahn and K. Wohn. Motion level-of-detail: A simplification method on crowd scene. In *Proceedings Computer Animation and Social Agents (CASA) 2004*, pages 129–137, 2004.

[3] M. Alexa. Linear combination of transformations. In *Proceedings SIGGRAPH 2002*, pages 380–387. ACM Press, 2002.

[4] B. Allen, B. Curless, and Z. Popovic. Articulated body deformation from range scan data. In *Proceedings SIGGRAPH 2002*, pages 612—619. ACM Press, 2002.

[5] E. André, M. Klesen, P. Gebhard, S. Allen, and T. Rist. Integrating models of personality and emotions into lifelike characters. In A. Paiva and C. Martinho, editors, *Proceedings International Workshop on Affect in Interactions: Towards a New Generation of Interfaces*, pages 136–149. Springer, October 1999.

[6] Y. Arafa, K. Kamyab, and E. Mamdani. Character animation scripting languages: A comparison. In *Proceedings International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS) 2003*, pages 920–921, Melbourne, Australia, 2003. ACM Press.

[7] L. Ardissono and G. Boella. An agent model for nl dialog interfaces. In *Lecture Notes in Artificial Intelligence*, volume 1480, pages 14–27. Springer Verlag, Berlin, Germany, 1998.

[8] M. Argyle. Innate and cultural aspects of human non-verbal communication. In C. Blakemore and S. Greenfield, editors, *Mindwaves: thoughts on intelligence, identity and conciousness*. Oxford: Basil Blackwell, 1987.

[9] O. Arikan and D. A. Forsyth. Interactive motion generation from examples. In *Proceedings SIGGRAPH 2002*, pages 483–490. ACM Press, 2002.

[10] O. Arikan, D. A. Forsyth, and J. F. O'Brien. Motion synthesis from annotations. *ACM Transactions on Graphics*, 22(3):392–401, 2003.

[11] M. B. Arnold. *Emotion and personality*. Columbia University Press, New York, 1960.

[12] J. R. Averill. A constructivist view of emotion. In R. Plutchik and H. Kellerman, editors, *Emotion: Theory, research and experience*, volume 1, pages 305–339. Academic Press, New York, 1980.

[13] G. Ball and J. Breese. Emotion and personality in a conversational agent. In J. Cassell, J. Sullivan, S. Prevost, and E. Churchill, editors, *Embodied conversational agents*, pages 189–219. MIT Press, Cambridge, MA, USA, 2000.

[14] W. Boehm. On cubics: a survey. *Computer Graphics and Image Processing*, 19:201—226, 1982.

[15] M. E. Bratman. *Intentions, Plans, and Practical Reason*. Harvard University Press, 1987.

[16] J. Cassell. A framework for gesture generation and interpretation. In R. Cipolla and A. Pentland, editors, *Computer Vision in Human-Machine Interaction*, pages 191–215. Cambridge University Press, New York, 1998.

[17] J. Cassell, T. Bickmore, M. Billinghurst, L. Campbell, K. Chang, Vilhjálmsson, H., and H. Yan. Embodiment in conversational interfaces: Rea. In *Proceedings of the CHI'99 Conference*, pages 520–527, 1999.

[18] J. Cassell, Y. I. Nakano, T. W. Bickmore, C. L. Sidner, and C. Rich. Non-verbal cues for discourse structure. In *Proceedings ACL Annual Conference*, pages 106–115, Morristown, NJ, USA, July 2001. Association for Computational Linguistics.

[19] J. Cassell, H. Vilhjálmsson, and T. Bickmore. BEAT: the behavior expression animation toolkit. In *Proceedings SIGGRAPH 2001*, pages 477–486. ACM Press, 2001.

[20] R. B. Cattell, H. Eber, and M. M. Tatsuoka. *Handbook for the Sixteen Personality Factor Questionnaire (16PF)*. Institute for Personality and Ability Testing, Champaign, IL, 1970.

[21] C. Chevalley. *Theory of Lie Groups*. Princeton University Press, New York, 1946.

[22] D. Chi, M. Costa, L. Zhao, and N. Badler. The EMOTE model for effort and shape. In *Proceedings SIGGRAPH 2000*, pages 173–182. ACM Press, July 2000.

[23] L. Chittaro and M. Serra. Behavioural programming of autonomous characters based on probabilistic automata and personality. *Computer Animation and Virtual Worlds*, 15(3–4):319–326, 2004.

[24] K.-J. Choi and H.-S. Ko. On-line motion retargetting. *The Journal of Visualization and Computer Animation*, 11(5):223–235, 2000.

[25] G. E. Churcher, E. S. Atwell, and C. Souter. Dialogue management systems: a survey and overview. Technical Report 97.6, School of Computer Studies, University of Leeds, February 1997.

[26] M. M. Cohen and D. W. Massaro. Modeling coarticulation in synthetic visual speech. In N. Magnenat-Thalmann and D. Thalmann, editors, *Models and Techniques in Computer Animation*, pages 139—156. Springer Verlag, Berlin, Germany, 1993.

[27] R. R. Cornelius. *The science of emotion. Research and tradition in the psychology of emotion.* Prentice-Hall, Upper Saddle River (NJ), 1996.

[28] P. T. Costa and R. R. McCrae. Normal personality assessment in clinical practice: The NEO personality inventory. *Psychological Assessment*, 4:5–13, 1992.

[29] M. Coulson. Attributing emotion to static body postures: Recognition accuracy, confusions, and viewpoint dependence. *Journal of Nonverbal Behavior*, 28(2):117–139, 2004.

[30] R. Cowie, E. Douglas-Cowie, S. Savvidou, E. McMahon, M. Sawey, and M. Schröder. Feeltrace: An instrument for recording perceived emotion in real time. In *ISCA Workshop on Speech and Emotion*, pages 19–24, Northern Ireland, 2000. Online proceedings at http://www.qub.ac.uk/en/isca/proceedings/, accessed May 2006.

[31] J. E. Cutting and L. T. Kozlowski. Recognizing friends by their walk: Gait perception without familiarity cues. *Bulletin of the Psychonomic Society*, 9:353–356, 1976.

[32] J. M. Digman. Personality structure: emergence of the five-factor model. *Annual Review of Psychology*, 41:417–446, 1990.

[33] G. Dudek and M. Jenkin. *Computational Principles of Mobile Robotics*. Cambridge University Press, 2000.

[34] A. Egges and N. Magnenat-Thalmann. Emotional communicative body animation for multiple characters. In *First International Workshop on Crowd Simulation (V-Crowds)*, pages 31–40. IEEE Computer Society, 2005.

[35] A. Egges, T. Molet, and N. Magnenat-Thalmann. Personalised real-time idle motion synthesis. In *Pacific Graphics 2004*, pages 121–130, 2004.

[36] A. Egges, X. Zhang, S. Kshirsagar, and N. Magnenat-Thalmann. Emotional communication with virtual humans. In *Multimedia Modelling*, pages 243–263, Taiwan, 2003.

[37] P. Ekman. *Emotion in the human face*. Cambridge University Press, New York, 1982.

[38] P. Ekman and W. V. Friesen. *Facial action coding system: A technique for the measurement of facial movement*. Consulting Psychologists Press, Palo Alto, CA, USA, 1978.

[39] M. El-Nasr, T. Ioerger, and J. Yen. PETEEI: a pet with evolving emotional intelligence. In *Proceedings International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS) 1999*, pages 9–15. ACM Press, 1999.

[40] C. D. Elliott. Using the affective reasoner to support social simulations. In *Proceedings of the Thirteenth Annual Joint Conference on Artificial Intelligence*, pages 194–200, Chambery, France, August 1993. Morgan Kaufmann.

[41] H. J. Eysenck. Biological dimensions of personality. In L. A. Pervin, editor, *Handbook of personality: Theory and research*, pages 244–276. New York: Guilford, 1990.

[42] H. J. Eysenck. Dimensions of personality: 16, 5 or 3?—criteria for a taxonomic paradigm. *Personality and Individual Differences*, 12:773–790, 1991.

[43] J. Gallier and D. Xu. Computing exponentials of skew-symmetric matrices and logarithms of orthogonal matrices. *International Journal of Robotics and Automation*, 18(1):10–20, 2003.

[44] S. Garchery, R. Boulic, T. Capin., and P. Kalra. Standards for virtual humans. In N. Magnenat-Thalmann and D. Thalmann, editors, *Handbook of Virtual Humans*, chapter 16, pages 373–391. John Wiley and Sons, Hoboken, NJ, USA, August 2004.

[45] Michael Gleicher. Retargeting motion to new characters. In *Proceedings SIGGRAPH 1998*, pages 33–42. ACM Press, 1998.

[46] L. R. Goldberg. The structure of phenotypic personality traits. *American Psychologist*, 48(1):26–34, January 1993.

[47] L. R. Goldberg and T. K. Rosolack. The big five factor structure as an integrative framework: An empirical comparison with eysenck's PEN model. In C. F. Halverson Jr., G. A. Kohnstamm, and R. P. Martin, editors, *The Developing Structure of Temperament and Personality from Infancy to Adulthood*, pages 7–35. Lawrence Erlbaum, New York, 1994.

[48] F. Sebastian Grassia. Practical parameterization of rotations using the exponential map. *Journal of Graphics Tools*, 3(3):29–48, 1998.

[49] J. J. Gumperz and J.-P. Blom. Social meaning in linguistic structure: Code-switching in norway. In J. J. Gumperz and D. H. Hymes, editors, *Directions in Sociolinguistics*, pages 407–434. New York: Holt, 1972.

[50] H-Anim specification for a standard humanoid. http://www.h-anim.org/. Accessed May 2006.

[51] S. Hampson. State of the art: Personality. *The Psychologist*, 12(6):284–290, June 1999. British Psychological Society.

[52] R. Harré, editor. *The social construction of emotions*. Basil Blackwell, Oxford, 1986.

[53] B. Hartmann, M. Mancini, and C. Pelachaud. Formational parameters and adaptive prototype instantiation for mpeg-4 compliant gesture synthesis. In *Computer Animation 2002*, pages 111–119, 2002. Computer Graphics Society.

[54] B. Hartmann, M. Mancini, and C. Pelachaud. Implementing expressive gesture synthesis for embodied conversational agents. In *Gesture in Human-Computer Interaction and Simulation: 6th International Gesture Workshop*, Lecture Notes in Computer Science, pages 188–199. Springer Verlag, Berlin, Germany, May 2005.

[55] D. Hearn and M. P. Baker. *Computer Graphics - C version, second edition*. Prentice Hall, New Jersey, 1996.

[56] Jabberwacky chatbot. http://www.jabberwacky.com. Accessed May 2006.

[57] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: a review. *ACM Computing Surveys*, 31(3):264–323, 1999.

[58] M. Johns and B. G. Silverman. How emotions and personality effect the utility of alternative decisions: a terrorist target selection case study. In *Tenth Conference On Computer Generated Forces and Behavioral Representation*, Orlando, FL, USA, May 2001.

[59] T. Jollife. *Principal Component Analysis*. Springer Verlag, New York, 1986.

[60] P. Kalra, S. Garchery, and S. Kshirsagar. Facial deformation models. In N. Magnenat-Thalmann and D. Thalmann, editors, *Handbook of Virtual Humans*, chapter 6, pages 119–139. John Wiley and Sons, Hoboken, NJ, USA, August 2004.

[61] P. Kalra and N. Magnenat-Thalmann. Modeling of vascular expressions in facial animation. In *Computer Animation*, pages 50–58, 1994.

[62] A. Kendon. Some relationships between body motion and speech: an analysis of one example. In A. W. Siegman and B. Pope, editors, *Studies in Dyadic Communication*, pages 177–210. New York: Pergamon, 1972.

[63] T. H. Kim, S. I. Park, and S. Y. Shin. Rhythmic-motion synthesis based on motion-beat analysis. *ACM Transactions on Graphics*, 22(3):392–401, 2003.

[64] S. Kopp and I. Wachsmuth. Synthesizing multimodal utterances for conversational agents. *Computer Animation and Virtual Worlds*, 15(1):39–52, 2004.

[65] L. Kovar and M. Gleicher. Flexible automatic motion blending with registration curves. In *Proceedings Symposium on Computer Animation (SCA) 2003*, pages 214–224. Eurographics Association, 2003.

[66] L. Kovar, M. Gleicher, and F. Pighin. Motion graphs. In *Proceedings SIGGRAPH 2002*, pages 473–482. ACM Press, 2002.

[67] L. T. Kozlowski and J. E. Cutting. Recognizing the sex of a walker from a dynamic point-light display. *Perception and Psychophysics*, 21:575–580, 1977.

[68] A. Kransted, S. Kopp, and I. Wachsmuth. MURML: A multimodal utterance representation markup language for conversational agents. In *AAMAS Workshop on Embodied Conversational Agents*, Italy, 2002.

[69] B. Krenn and H. Pirker. Defining the gesticon: Language and gesture coordination for interacting embodied agents. In *Proceedings of the AISB-2004 Symposium on Language, Speech and Gesture for Expressive Characters*, pages 107–115, University of Leeds, UK, 2004.

[70] P. G. Kry, D. L. James, and D. K. Pai. Eigenskin: Real time large deformation character skinning in graphics hardware. In *Proceedings Symposium on Computer Animation (SCA) 2002*, pages 153–159. Eurographics Association, 2002.

[71] S. Kshirsagar and N. Magnenat-Thalmann. A multilayer personality model. In *Proceedings of the second International Symposium on Smart Graphics*, pages 107–115. ACM Press, June 2002.

[72] S. Kshirsagar, T. Molet, and N. Magnenat-Thalmann. Principal components of expressive speech animation. In *Computer Graphics International 2001*, pages 38–44. IEEE Computer Society, February 2001.

[73] J. D. Laird. The real role of facial response in the experience of emotion: A reply to tourangeau and ellsworth and others. *Journal of Personality and Social Psychology*, 47:909–917, 1984. American Psychological Association.

[74] J. LeDoux. *The emotional brain*. Simon & Shuster, New York, 1996.

[75] J. Lee, J. Chai, P. Reitsma, J. K. Hodgins, and N. Pollard. Interactive control of avatars animated with human motion data. In *Proceedings SIGGRAPH 2002*, pages 491–500. ACM Press, July 2002.

[76] R. W. Levenson, P. Ekman, and W. V. Friesen. Voluntary facial action generates emotion-specific autonomic nervous system activity. *Psychophysiology*, 27:363–384, 1990.

[77] J. P. Lewis, M. Cordner, and N. Fong. Pose space deformations: A unified approach to shape interpolation and skeleton-driven deformation. In *Proceedings SIGGRAPH 2000*, pages 165–172. ACM Press, 2000.

[78] Y. Li, T. Wang, and H. Y. Shum. Motion texture: A two-level statistical model for character motion synthesis. In *Proceedings SIGGRAPH 2002*, pages 465–472. ACM Press, 2002.

[79] C. K. Liu, A. Hertzmann, and Z. Popovic. Learning physics-based motion style with nonlinear inverse optimization. *ACM Transactions on Graphics*, 24(3):1071–1081, 2005.

[80] G. Loy, J. Sullivan, and S. Carlsson. Pose-based clustering in action sequences. In *Proceedings Workshop on Higher-Level Knowledge in 3D Modeling and Motion Analysis 2003*, pages 66–73. ACM Press, 2003.

[81] Microsoft Speech SDK version 5.1 (SAPI5.1). http://www.microsoft.com/speech/download/sdk51/. Accessed May 2006.

[82] Motionstar. http://www.ascension-tech.com/. Accessed May 2006.

[83] MPEG Motion Picture Expert Group. http://www.chiariglione.org/mpeg/. Accessed May 2006.

[84] N. Magnenat-Thalmann, F. Cordier, H. Seo, and G. Papagiannakis. Modeling of bodies and clothes for virtual environments. In *Proceedings of the International Conference on Cyberworlds (CW) 2004*, pages 201–208. IEEE Computer Society, July 2004.

[85] N. Magnenat-Thalmann, R. Laperrière, and D. Thalmann. Joint-dependent local deformations for hand animation and object grasping. In *Proceedings Graphics Interface*, pages 26–33. Canadian Information Processing Society, 1988.

[86] S. Marsella and J. Gratch. A step towards irrationality: Using emotion to change belief. In *Proceedings International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS) 2002*, Bologna, Italy, July 2002. ACM Press.

[87] W. Mischel. *Personality and Assessment*. Wiley, New York, 1968.

[88] W. Mischel and Y. Shoda. A cognitive-affective system theory of personality: reconceptualising situations, dispositions, dynamics and invariance in personality structure. *Psychological Review*, 102:246–268, 1995.

[89] W. Mischel and Y. Shoda. Reconciling processing dynamics and personality dispositions. *Annual Review of Psychology*, 49:229–258, 1998.

[90] A. Mohr and M. Gleicher. Building efficient, accurate character skins from examples. In *Proceedings SIGGRAPH 2003*, pages 165–172. ACM Press, 2003.

[91] J. Y. Noh, D. Fidaleo, and U. Neumann. Animated deformations with radial basis functions. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology*, pages 166–174. ACM Press, 2000.

[92] M. Ochs, R. Niewiadomski, C. Pelachaud, and D. Sadek. Intelligent expressions of emotions. In *1st International Conference on Affective Computing and Intelligent Interaction ACII*, October 2005.

[93] A. Ortony, G. L. Clore, and A. Collins. *The Cognitive Structure of Emotions*. Cambridge University Press, 1988.

[94] G. Papagiannakis, M. Ponder, T. Molet, S. Kshirsagar, F. Cordierand N. Magnenat-Thalmann, and D. Thalmann. LIFEPLUS: Revival of life in ancient pompeii. In *Virtual Systems and Multimedia (VSMM)*, October 2002.

[95] F. C. Park and B. Ravani. Smooth invariant interpolation of rotations. *ACM Transactions on Graphics*, 16(3):277–295, July 1997.

[96] F. I. Parke. Computer generated animation of faces. In *Proceedings ACM Annual Conference*, pages 451–457. ACM Press, 1972.

[97] H. Van Dyke Parunak, R. Bisson, S. Brueckner, R. Matthews, and J. Sauter. A model of emotions for situated agents. In *Proceedings International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS) 2006*, Hakodate, Japan, May 2006. ACM Press.

[98] K. Perlin. An image synthesizer. In *Proceedings SIGGRAPH 1985*, pages 287–296. ACM Press, 1985.

[99] K. Perlin. Real time responsive animation with personality. *IEEE Transactions on Visualization and Computer Graphics*, 1(1):5–15, 1995.

[100] R. W. Picard. *Affective computing*. MIT Press, Cambridge, MA, USA, 1997.

[101] P. Piwek. An annotated bibliography of affective natural language generation. Technical Report ITRI02-02, University of Brighton, July 2002. http://www.itri.brighton.ac.uk/projects/neca/affect-bib.pdf, accessed May 2006.

[102] P. Piwek, B. Krenn, M. Schröder, M. Grice, S. Baumann, and H. Pirker. RRL: A rich representation language for the description of agent behaviour in NECA. In *AAMAS Workshop on Embodied Conversational Agents*, Italy, 2002.

[103] S. M. Platt and N. Badler. Animating facial expression. *Computer Graphics*, 15(3):245–252, 1981.

[104] R. Plutchik. *Emotion: A psychoevolutionary synthesis*. Harper & Row, New York, 1980.

[105] M. Ponder, G. Papagiannakis, T. Molet, N. Magnenat-Thalmann, and D. Thalmann. VHD++ development framework: Towards extendible, component based VR/AR simulation engine featuring advanced virtual character technologies. In *Proceedings of Computer Graphics International (CGI)*, pages 96–104. IEEE Computer Society, 2003.

[106] K. Pullen and C. Bregler. Motion capture assisted animation: Texturing and synthesis. In *Proceedings SIGGRAPH 2002*, pages 501–508. ACM Press, 2002.

[107] A. S. Rao and M. P. Georgeff. Modeling rational agents within a BDI architecture. In R. Fikes and E. Sandewall, editors, *Proceedings of Knowledge Representation and Reasoning*, pages 473—484. Morgan Kaufmann Publishers, April 1991.

[108] C. Rose, M. Cohen, and B. Bodenheimer. Verbs and adverbs: Multidimensional motion interpolation. *IEEE Computer Graphics and Applications*, 18(5):32–40, September/October 1998.

[109] C. Rose, B. Guenter, B. Bodenheimer, and M. Cohen. Efficient generation of motion transitions using spacetime constraints. In *Proceedings SIGGRAPH 1996*, pages 147—154. ACM Press, August 1996.

[110] M. D. Sadek, P. Bretier, and F. Panaget. ARTIMIS: Natural dialogue meets rational agency. In M. E. Pollack, editor, *Proceedings 15th International Joint*

Conference on Artificial Intelligence, pages 1030–1035. Morgan Kaufmann Publishers, 1997.

[111] A. Scheflen. *Communicational structure*. Bloomington: Indiana University Press, 1973.

[112] K. R. Scherer. Appraisal considered as a process of multi-level sequential checking. In K. R. Scherer, A. Schorr, and T. Johnstone, editors, *Appraisal processes in emotion: Theory, Methods, Research*, pages 92–120. Oxford University Press, 2001.

[113] H. Schlosberg. A scale for judgement of facial expressions. *Journal of Experimental Psychology*, 29:497–510, 1954.

[114] K. Shoemake. Animating rotation with quaternion curves. In *Proceedings SIGGRAPH 1985*, pages 245–254. ACM Press, 1985.

[115] The Sims web page. http://thesims.ea.com/. Accessed May 2006.

[116] P. P. Sloan, C. Rose, and M. Cohen. Shape and animation by example. Technical report, Microsoft Research, July 2000. http://research.microsoft.com/graphics/hfap/shapetr.pdf/, accessed May 2006.

[117] P. P. Sloan, C. Rose, and M. Cohen. Shape by example. In *Symposium on Interactive 3D Graphics*, pages 135–143. ACM Press, March 2001.

[118] S. Stepper and F. Strack. Proprioceptive determinants of emotional and nonemotional feelings. *Journal of Personality and Social Psychology*, 64:211–220, 1993.

[119] M. Stone, D. DeCarlo, I. Oh, C. Rodriguez, A. Stere, A. Lees, and C. Bregler. Speaking with hands: Creating animated conversational characters from recordings of human performance. In *Proceedings SIGGRAPH 2004*, pages 506–513. ACM Press, 2004.

[120] F. Strack, S. Stepper, and L. L. Martin. Inhibiting and facilitating conditions of the human smile: A nonobtrusive test of the facial feedback hypothesis. *Journal of Personality and Social Psychology*, 54:768–777, 1988.

[121] Thomas A. Sudkamp. *Languages and Machines: an introduction to the theory of computer science*. Addison Wesley, 1994.

[122] M. Unuma, K. Anjyo, and T. Tekeuchi. Fourier principles for emotion-based human figure animation. In *Proceedings SIGGRAPH 1995*, pages 91—–96. ACM Press, 1995.

[123] Virtual human markup language (vhml). http://www.vhml.org/. Accessed November 2005.

[124] Vicon motion systems. http://www.vicon.com/. Accessed April 2006.

[125] C. L. Y. Wang. *Multi-Resolution Surface Approximation for Animation*. PhD thesis, University of British Columbia, 1993.

[126] K. Waters. A muscle model for animating three-dimensional facial expression. *Computer Graphics*, 21:17–24, 1987.

[127] J. Weizenbaum. ELIZA—a computer program for the study of natural language communication between man and machine. *Communications of the ACM*, 9(1):36–45, 1966.

[128] D. Wiley and J. Hahn. Interpolation synthesis of articulated figure motion. *IEEE Computer Graphics and Applications*, 17(6):39—45, 1997.

[129] A. Witkin and Z. Popovic. Motion warping. In *Proceedings SIGGRAPH 1995*, pages 105–108. ACM Press, 1995.

[130] M. Wooldridge. *Reasoning about rational agents*. MIT Press, 2000.

APPENDIX A

Publications

Following are the publications related to the research done in this thesis:

A.1 Journal Publications

- A. Egges, S. Kshirsagar, and N. Magnenat-Thalmann. Generic Personality and Emotion Simulation for Conversational Agents, *Journal of Computer Animation and Virtual Worlds*. 15(1):1–13. John Wiley and Sons, 2004.

A.2 Proceedings Publications

- A. Egges, G. Papagiannakis, and N. Magnenat-Thalmann. An Interactive Mixed Reality Framework for Virtual Humans. In *Proceedings International Conference on Cyberworlds 2006*. IEEE Computer Society, November 2006. To appear.
- A. Egges and Nadia Magnenat-Thalmann. Emotional Communicative Body Animation for Multiple Characters. *V-Crowds'05*, Lausanne, Switzerland, pages 31–40. November 2005.
- H. Kim, T. Di Giacomo, A. Egges, E. Lyard, S. Garchery, and N. Magnenat-Thalmann. Believable Virtual Environment: Sensory and Perceptual Believability, *CAPTECH Workshop: ENACTIVE Session*, Zermatt, Switzerland, December 2004.
- A. Egges, R. Visser, and N. Magnenat-Thalmann. Example-based Idle Motion Synthesis in a Real-time Application. *CAPTECH Workshop*, Zermatt, Switzerland, pages 13–19, December 2004.

- A. Egges, T. Molet, and N. Magnenat-Thalmann. Personalised Real-time Idle Motion Synthesis. In *Proceedings Pacific Graphics*, pages 121–130. IEEE Computer Society, 2004.
- A. Egges, X. Zhang, S. Kshirsagar, and N. Magnenat-Thalmann. Emotional Communication with Virtual Humans. *Multimedia Modelling*, pages 243–263, Taiwan, 2003.
- A. Egges, S. Kshirsagar, and N. Magnenat-Thalmann. A Model for Personality and Emotion Simulation. *Knowledge-Based Intelligent Information and Engineering Systems (KES2003)*, pages 453–461. Springer-Verlag, 2003.
- A. Egges, S. Kshirsagar, and N. Magnenat-Thalmann . Imparting Individuality to Virtual Humans. *First International Workshop on Virtual Reality Rehabilitation (Mental Health, Neurological, Physical, Vocational)*. Lausanne, Switzerland. November, 2002.

A.3 Book Chapters

- A. Egges, T. Di Giacomo, and N. Magnenat Thalmann. Synthesis of Realistic Idle Motion for Interactive Characters, pages 409–421. *Game Programming Gems 6*, Mike Dickheiser (Ed.). Charles River Media. March 2006.
- S. Kshirsagar, A. Egges and S. Garchery. Expressive Speech Animation and Facial Communication, Chapter 10, pages 230–259. In *Handbook of Virtual Humans*, N. Magnenat Thalmann and D. Thalmann (Eds.). John Wiley and Sons, August 2004.

APPENDIX B

The `animblender` Manual

This document describes in short how to use the animblender tool to mix and edit WRK animations. In the same folder you'll also find some examples of how to use the blender tool. The main approach of this tool is as follows:

1. There is a number of animations that are going to be used to create the final animation

2. There is a schedule defining exactly how the animations are going to be edited/-mixed to produce the final animation

From these two things, the animblender tool creates a new animation file. There are different options to save the animation. You can save it in different formats (WRK, XML, etc.) and with different framerate. The animblender tool has the following syntax:

```
animblender <blendingschedule(*.xml)>
            -outputformat <outputfile(*.wrk;*.xml)>
            [framerate]
```

There are 4 parameters. The first one is the xml file defining the blending parameters (or: blending schedule). The second one specifies the output format. The third parameter is the filename of the output file. Finally, you can give an optional framerate (default is 25). You will see how this works in the different examples that are provided. We'll now briefly discuss the structure of the blending parameters (blending schedule).

B.1 The Blending Schedule

The blending schedule XML file has the following main structure:

```
<blendingschedule>
   <actionpool>
      Here we define all the blending actions
   </actionpool>
   <activate id="..." offset="..."/>
</blendingschedule>
```

The blending schedule consists of two components, the action pool and a list of ¡activate¿ tags. The action pool specifies all the blending actions that are available. I'll discuss in more detail what is exactly a blending action later on. The activate tags specify when each blending action is starting.

B.2 Blending actions

A blending action is the main unit used in the system for blending and editing an animation. It consists of two parts:

1. An animation (a so-called ”blendable object”)
2. A set of parameters that specify how the animation should be blended with other animations

A very simple example of a blending action is as follows:

```
<blendingaction id="action1">
   <blendingparams type="none"/>
   <blendableobject type="keyframebodyanimation">
      neutral_idle.wrk
   </blendableobject>
</blendingaction>
```

Every blending action should have a unique ID. In this case, I chose the simple id ”action1”. As you can see, there are two tags inside the blending action. The first one specifies the blending parameters, in this example, there are none. The second one specifies the animation. In this case, it is a keyframe animation loaded from a file. Now we place this blending action inside a blending schedule and activate it. The final blending schedule looks like this:

```
<blendingschedule>
   <actionpool>
      <blendingaction id="action1">
```

```
         <blendingparams type="none"/>
         <blendableobject type="keyframebodyanimation">
            neutral_idle.wrk
         </blendableobject>
      </blendingaction>
   </actionpool>
   <activate id="action1" offset="0.0f"/>
</blendingschedule>
```

This blending schedule loads the keyframe animation ”neutral_idle.wrk”. It then activates this action at time 0.0. You can run example1.bat, which loads this blending schedule and saves the resulting animation in ”example1.wrk”. Since this blending schedule only activates the animation at time 0.0, the resulting animation will be exactly the same as the neutral_idle.wrk animation.

The animblender tool saves animations by default with a framerate of 25. However, you can also save at other framerates. For example, example1a.bat uses the same blending schedule, but it saves the animation at a framerate 10. You can open the example1a.bat file in notepad to see the syntax for doing this.

B.3 Animation modifiers

Instead of loading only an animation, we can place so-called ”modifiers” on top of an animation, before being used to construct the new animation. There are two modifiers, that can be applied: stretch and window. The stretch modifier resizes an animation according to a new duration. This is an example of an animation stretched to a duration of 5.0 seconds:

```
<blendableobject type="modifier">
   <modifier type="stretch" duration="5.0"/>
   <blendableobject type="keyframebodyanimation">
      neutral_idle.wrk
   </blendableobject>
</blendableobject>
```

The window modifier only loads a part of an animation. An example of a window modifier that only loads an animation in the interval [2.0,3.0] seconds:

```
<blendableobject type="modifier">
   <modifier type="window" begin="2.0" end="3.0"/>
   <blendableobject type="keyframebodyanimation">
      neutral_idle.wrk
   </blendableobject>
</blendableobject>
```

These blendable object modifiers can be nested as well. In the following example, first, the animation is loaded between interval [2.0,3.0] only and then this animation segment is resized to a duration of 5.0 seconds:

```
<blendableobject type="modifier">
   <modifier type="stretch" duration="5.0"/>
   <blendableobject type="modifier">
      <modifier type="window" begin="2.0" end="3.0"/>
      <blendableobject type="keyframebodyanimation">
         neutral_idle.wrk
      </blendableobject>
   </blendableobject>
</blendableobject>
```

You can combine and nest modifiers in this way as often as you want. When you run example2.bat, it will load a WRK animation, stretch it to 5.0 seconds and save it in example2.wrk. You can see the complete blending schedule that is used in the file blendingschedule_example2.xml.

B.4 Blending animations

Now let's take a look at blending two animations. If we want to blend two animations, we need to create two blending actions. The action pool will look something like this:

```
<actionpool>
   <blendingaction id="ba1">
      <blendingparams type="none"/>
      <blendableobject type="keyframebodyanimation">
         neutral_idle.wrk
      </blendableobject>
   </blendingaction>
   <blendingaction id="ba2">
      <blendingparams type="none"/>
      <blendableobject type="keyframebodyanimation">
         NewMouvement.wrk
      </blendableobject>
   </blendingaction>
</actionpool>
```

This simply loads the two animations without any modifiers. Now we can activate these two actions at different times:

```
<activate id="ba1" offset="0.0f"/>
<activate id="ba2" offset="1.0f"/>
```

The first action with start at time index 0.0, action ”ba2” will start one second later. In this particular example, we are blending an full body idle motion sequence (ba1) with a gesture motion sequence (ba2). With the current settings, both animations will have an equal influence on all joints. For some blends this might be okay, be we want the gesture to more or less take over from the idle motion, at least for the upper body! This is where the blendingparams plays a role. For the gesture animation (ba2), we can specify a blending parameters object with different weights for different joints. For a blendingparams object of type ”joints”, we need to specify a default weight and scaling factor that is initially applied to each joint. This is done by the default tag. Then, we can specify a set of rotation and translation joint weights and scalings. For example, here is a blendingparams definition that defines a default weight of 0.0 and that defines a weight of 1.0 for the root rotation and translation:

```
<blendingparams type="joints">
   <default weight="0.0" scaling="1.0"/>
   <rotation id="root" weight="1.0"/>
   <translation id="root" weight="1.0"/>
</blendingparams>
```

When this blendingparams object would be used, only the root translation and rotation would be taken into account as part of the blend. For the gesture animation, we would need all lower joints including the root translation to be zero, and all the upper joints to be 1.0 (for example). This would make a huge list of rotation tags. In order to make things easier, there is also a scope tag, that takes a limited list of joints and that changes the weight for them. In this example, we can use the ”torso” scope (all upper body joints) to put a weight of 1.0 on all the upper body joints. This looks as follows:

```
<blendingparams type="joints">
   <default weight="0.0" scaling="1.0"/>
   <scope id="torso" weight="1.0" scaling="1.0"/>
</blendingparams>
```

Scope, rotation and translation tags can be mixed in a blendingparams object. This generates a weight of 1.0 for the upper body joints, as well as a weight of 0.3 for the root translation and rotation:

```
<blendingparams type="joints">
   <default weight="0.0" scaling="1.0"/>
   <scope id="torso" weight="1.0" scaling="1.0"/>
   <rotation id="root" weight="0.3"/>
   <translation id="root" weight="0.3"/>
</blendingparams>
```

When you open the blendingschedule_example3.xml, you can see how I applied two different blending params objects for the idle motions and the gesture motion. The

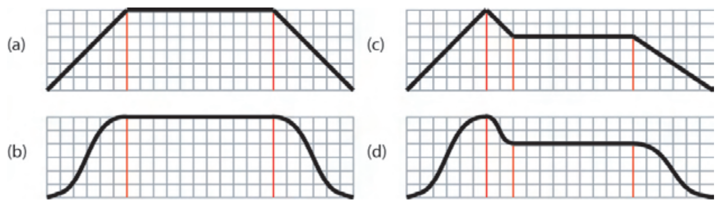


Figure B.1: The four types of blending curves.

resulting animation is example3.wrk. You can define both weights and scalings in a similar way. For the root rotation, we can specify a scaling:

```
<rotation id="root" scaling="0.3"/>
```

Or both a scaling and weight at the same time:

```
<rotation id="root" weight="0.3" scaling="1.0"/>
```

B.5 Smoother blending

When you play the example3.wrk, you have probably noticed that the transition between idle posture and gesture animation is not smooth. This is because the joint weights that we specified earlier jump from 0 to 1 and thus create a non-natural motion. In order to allow for a smoother blend, we can specify a weight fade-in/fade-out curve in combination with the joint weights. There are currently four types of curves (see Figure B.1).

The first one (a) is called ”fade” and does a linear fade-in and fade-out. The second one (b) is called ”goniofade” and it applies a goniometric curve to create a smoother fade. There is also ”adsr” (c) and ”gonioadsr” (d) to simulate an Attack-Decay-Sustain-Release curve, but these curves are not yet available in the XML interface. For the fade curves, you can choose the amount of time that is used for the fade-in and fade-out. Here is an example of a simple fade blending parameter:

```
<blendingparams type="fade">
   <fadein>20</fadein>
   <fadeout>20</fadeout>
</blendingparams>
```

The fadein and fadeout tags define how much time should be taken for the fade in and fade out. The numbers are percentages. So in this example, 20% of the time of the animation in the blending action will be used for the fadein and 20% will be used for the fade out. Practically speaking, if we have an animation with duration of 5.0 seconds linked with these blending parameters, 1.0 second will be used for the fadein as well as the fadeout. In order to mix this curve with the joint weights, a blending parameter object called ”merge” is used. This object can merge as many blendingparams objects as you want. Here is an example that mixes a goniofade curve together with the earlier defined joint weights for the gesture motion:

```
<blendingparams type="merge">
   <blendingparams type="goniofade">
      <fadein>20</fadein>
      <fadeout>20</fadeout>
   </blendingparams>
   <blendingparams type="joints">
      <default weight="0.0" scaling="1.0"/>
      <scope id="torso" weight="1.0" scaling="1.0"/>
   </blendingparams>
</blendingparams>
```

This blending params object is used in the last example to create a smooth blend between the idle motion and the gesture motion. The full blending schedule is contained in the file ”blendingschedule_example4.xml”. You can run the example4.bat and see the resulting animation example4.wrk.

B.6 Idle motion blending

A special class of motions are idle motions. These motions are generated on the fly from motion captured data and they can be dynamically blended in with the other motions. Because such types of motions generally do not have a fixed length, we need to add an additional parameter to the blending action, saying that it should always be included in the frame rendering, because the motion will be generated according to the frame that is rendered. In the fifth example, you will see how such an action looks like:

```
<blendingaction id="idle" alwaysrender="true">
   <blendingparams type="joints">
      <default weight="1.0" scaling="1.0"/>
      <scope id="torso" weight="0.1"/>
   </blendingparams>
   <blendableobject type="idlemotion">
      <balance>balance_alex_raw.xml</balance>
   </blendableobject>
</blendingaction>
```

The file that is included is an XML file containing the raw motion data, as well as the motion graph itself. As you can see, we have defined the idle motion to work on the lower body, and with a low weight on the upper body, to smoothly mix it with a gesture motion.

APPENDIX C

Blending Schedule Examples

C.1 Example 1

```
<blendingschedule>
  <actionpool>
    <blendingaction id="action1">
      <blendingparams type="none"/>
      <blendableobject type="keyframebodyanimation">
        neutral_idle.wrk
      </blendableobject>
    </blendingaction>
  </actionpool>
  <activate id="action1" offset="0.0f"/>
</blendingschedule>
```

C.2 Example 2

```
<blendingschedule>
  <actionpool>
    <blendingaction id="action1">
      <blendingparams type="none"/>
      <blendableobject type="modifier">
        <modifier type="stretch" duration="5.0"/>
        <blendableobject type="keyframebodyanimation">
          neutral_idle.wrk
        </blendableobject>
      </blendableobject>
    </blendingaction>
  </actionpool>
  <activate id="action1" offset="0.0f"/>
</blendingschedule>
```

C.3 Example 3

```
<blendingschedule>
  <actionpool>
    <blendingaction id="ba1">
      <blendingparams type="joints">
        <default weight="1.0" scaling="1.0"/>
        <scope id="torso" weight="0.1"/>
      </blendingparams>
      <blendableobject type="keyframebodyanimation">
        neutral_idle.wrk
      </blendableobject>
    </blendingaction>
    <blendingaction id="ba2">
      <blendingparams type="joints">
        <default weight="0.0" scaling="1.0"/>
        <scope id="torso" weight="1.0" scaling="1.0"/>
      </blendingparams>
      <blendableobject type="keyframebodyanimation">
        NewMouvement.wrk
      </blendableobject>
    </blendingaction>
  </actionpool>
  <activate id="ba1" offset="0.0f"/>
  <activate id="ba2" offset="1.0f"/>
</blendingschedule>
```

C.4 Example 4

```
<blendingschedule>
  <actionpool>
  <blendingaction id="ba1">
    <blendingparams type="joints">
      <default weight="1.0" scaling="1.0"/>
      <scope id="torso" weight="0.1"/>
    </blendingparams>
    <blendableobject type="keyframebodyanimation">
      neutral_idle.wrk
    </blendableobject>
  </blendingaction>
  <blendingaction id="ba2">
```

```
    <blendingparams type="merge">
      <blendingparams type="curve">
        <curveshape type="goniofade">
          <fadein>20</fadein>
          <fadeout>20</fadeout>
        </curveshape>
      </blendingparams>
      <blendingparams type="joints">
        <default weight="0.0" scaling="1.0"/>
        <scope id="torso" weight="1.0" scaling="1.0"/>
      </blendingparams>
    </blendingparams>
    <blendableobject type="keyframebodyanimation">
      NewMouvement.wrk
    </blendableobject>
  </blendingaction>
  </actionpool>
  <activate id="ba1" offset="0.0f"/>
  <activate id="ba2" offset="1.0f"/>
</blendingschedule>
```

C.5 Example 5

```
<blendingschedule>
  <actionpool>
    <blendingaction id="idle" alwaysrender="true">
      <blendingparams type="joints">
        <default weight="1.0" scaling="1.0"/>
        <scope id="torso" weight="0.1"/>
      </blendingparams>
      <blendableobject type="idlemotion">
        <balance>balance_alex_raw.xml</balance>
      </blendableobject>
    </blendingaction>
    <blendingaction id="wrk">
      <blendingparams type="merge">
        <blendingparams type="curve">
          <curveshape type="goniofade">
            <fadein>20</fadein>
            <fadeout>20</fadeout>
          </curveshape>
        </blendingparams>
        <blendingparams type="joints">
          <default weight="0.0" scaling="1.0"/>
          <scope id="torso" weight="1.0" scaling="1.0"/>
        </blendingparams>
      </blendingparams>
      <blendableobject type="keyframebodyanimation">
        NewMouvement.wrk
      </blendableobject>
    </blendingaction>
  </actionpool>
  <activate id="ba1" offset="0.0f"/>
  <activate id="ba2" offset="1.0f"/>
</blendingschedule>
```

APPENDIX D

Visemes

In this chapter, we provide the list of Microsoft SAPI 5.1 visemes, as employed by our facial animation system. Additionally, we indicate the coarticulation curve shapes that we use. These curves are defined in an XML file that can be edited by the animators, ensuring a maximum flexibility in facial animation design and management.

D.1 SAPI5.1 Viseme List

Table D.1 shows an overview of the visemes as defined in SAPI5.1. Viseme 0 represents the silence viseme. SAPI defines 21 different visemes, each related to one or more phonemes. SAPI defines a total of 49 phonemes, including a few miscellaneous characters such as a comma or a hyphen, which are related to the silence viseme.

D.2 Coarticulation Curve definition

The following (simplified) XML specification defines the coarticulation curves for each viseme:

```xml
<visemes>
<vis id="00" spread="2.0" weight="1.0" alpha="1.0"/>
<vis id="01" spread="5.0" weight="1.0" alpha="1.0"/>
<vis id="02" spread="5.0" weight="1.0" alpha="1.0"/>
<vis id="03" spread="5.0" weight="1.0" alpha="1.0"/>
<vis id="04" spread="5.0" weight="1.0" alpha="1.0"/>
<vis id="05" spread="5.0" weight="1.0" alpha="1.0"/>
<vis id="06" spread="5.0" weight="1.0" alpha="1.0"/>
<vis id="07" spread="5.0" weight="1.0" alpha="1.0"/>
```

Viseme	**Corresponding SAPI phonemes**
SP_VISEME_0	silence
SP_VISEME_1	ae, ax, ah
SP_VISEME_2	aa
SP_VISEME_3	ao
SP_VISEME_4	ey, eh, uh
SP_VISEME_5	er
SP_VISEME_6	y, iy, ih, ix
SP_VISEME_7	w, uw
SP_VISEME_8	ow
SP_VISEME_9	aw
SP_VISEME_10	oy
SP_VISEME_11	ay
SP_VISEME_12	h
SP_VISEME_13	r
SP_VISEME_14	l
SP_VISEME_15	s, z
SP_VISEME_16	sh, ch, jh, zh
SP_VISEME_17	th, dh
SP_VISEME_18	f, v
SP_VISEME_19	d, t, n
SP_VISEME_20	k, g, ng
SP_VISEME_21	p, b, m

Table D.1: Overview of the visemes as defined in SAPI5.1 [81]

```
<vis id="08" spread="5.0" weight="1.0" alpha="1.0"/>
<vis id="09" spread="5.0" weight="1.0" alpha="1.0"/>
<vis id="10" spread="5.0" weight="1.0" alpha="1.0"/>
<vis id="11" spread="5.0" weight="1.0" alpha="1.0"/>
<vis id="12" spread="3.0" weight="0.9" alpha="1.0"/>
<vis id="13" spread="3.0" weight="0.9" alpha="1.0"/>
<vis id="14" spread="3.0" weight="0.9" alpha="1.0"/>
<vis id="15" spread="3.0" weight="1.1" alpha="1.0"/>
<vis id="16" spread="3.0" weight="1.2" alpha="1.0"/>
<vis id="17" spread="3.0" weight="0.9" alpha="1.0"/>
<vis id="18" spread="3.0" weight="1.5" alpha="1.0"/>
<vis id="19" spread="3.0" weight="1.1" alpha="1.0"/>
<vis id="20" spread="3.0" weight="1.1" alpha="1.0"/>
<vis id="21" spread="3.0" weight="1.5" alpha="3.0"/>
</visemes>
```

The spread defines the amount of overlap for each viseme. For example, a spread of 2.0 means that the curve length is twice as much as the corresponding phoneme length provided by the text-to-speech software. The weight defines the importance of the viseme in the animation. For example, the weight of viseme 21 (p, b, m) is rather high, in order to assure that the mouth closes properly. We use a standard coarticulation curve as exerted by Cohen and Massaro [26]. The alpha value defines the shape of the curve (see Section 5.3.1).

Index