Physically based avatar animation

Peter Boyen

promotor :
Prof. dr. Wim LAMOTTE

Eindverhandeling voorgedragen tot het bekomen van de graad
Master in de informatica multimedia

universiteit hasselt

Samenvatting

Er bestaan verschillende technieken in de 3D-animatie voor modellering, vervorming en animatie. De weergave van modellen wordt vaak bepaald door de manier waarop de data verkregen wordt. Polygonmodellen zijn populair omdat de andere modellen eerst gepolygonaliseerd moeten worden voor rendering. Vervorming van modellen met een hiërarchisch skelet blijkt in de praktijk de meeste mogelijkheden te bieden. De houding van deze modellen is eenvoudig aan te passen en de modellen bieden toch voldoende controle voor de meeste vormen van animatie. De keuze van animatietechniek wordt vaak bepaald door de applicatie waarin deze nodig is. Zolang er geen sprake is van onverwachte interactie met de omgeving zullen kinematische technieken zeer goed presteren. Als hiervan wel sprake is, doen we er best aan technieken te bekijken die gebruik maken van fysische simulatie.

Ragdolls zijn de eenvoudigste toepassing van dynamische technieken, maar deze bieden niet genoeg controle over het model om in de meeste situaties nuttig te zijn. Er is ook de mogelijkheid om krachten toe te voegen aan het model met behulp van forward en inverse dynamics.

Om dynamische technieken eenvoudiger te kunnen samenvoegen, kunnen deze in controllers gegoten worden. Een controller is een controlealgoritme dat een bepaalde doelstelling probeert te verwezenlijken, rekening houdend met de omgeving. Om verschillende controllers samen te voegen, kan men een hiërarchisch systeem gebruiken waarbij een overkoepelende controller de controle doorgeeft aan één van zijn kindcontrollers. Deze kindcontrollers kunnen dan specifieke bewegingen voor hun rekening nemen zoals balanceren of vallen.

In het implementatie-onderdeel van deze thesis werd het idee van een overkoepelende controller ook geïmplementeerd. Deze implementatie maakt gebruik van de Ageia PhysX engine en van het ExtReAM framework. Deze zijn beide gratis verkrijgbaar en bieden vele mogelijkheden.

Waar mogelijk is de implementatie onafhankelijk van het gebruikte model. Hierdoor zijn controllers eenvoudiger op elk model toe te passen. Voor elk model worden dus automatisch de fysische tegenhangers gegenereerd, maar de beperkingen op het model moeten toch per model aangegeven worden.

Bij de implementatie van een hoofdcontroller horen ook twee kindcontrollers (een ragdollcontroller en een balanscontroller). Hiermee wordt het idee in de praktijk toegepast.

De controllers zijn ook aan een aantal testen onderworpen. Alle controllers kunnen hun taken uitvoeren in real-time. De overkoepelende controller presteert goed en introduceert geen problemen voor de aparte controllers. De balanscontroller kan compenseren voor relatief kleine krachten die op het model uitgeoefend worden door de krachten te zetten op de enkels van het model.

Zowel de ragdollcontroller als de balanscontroller maken gebruik van het fysische model dat de bewegingen beperkt tot mogelijke bewegingen van het menselijk lichaam. In deze zin zijn de animaties dus niet onrealistisch. De bewegingen die de controllers uitvoeren zijn echter niet altijd degene die een normale mens zou doen in deze situaties.

Woord vooraf

Door de snel stijgende CPU-snelheden en verbeterende grafische kaarten wordt er steeds meer mogelijk in animatie. Niet alleen worden de modellen gedetailleerder, maar de animatie wordt ook realistischer gemaakt. Een punt dat de laatste tijd belangrijker is geworden, is de interactie met andere modellen en de omgeving. Om deze interacties geloofwaardig te maken wordt steeds vaker gebruik gemaakt van fysische simulaties. Ik was zeer geïnteresseerd in de opkomst van deze fysische simulaties en in 3D-animaties in het algemeen. Daarom heb ik dit thesisonderwerp gekozen.

Het doel van deze thesis was bekend te raken met deze fysisch gebaseerde technieken. Vervolgens moest er een model geanimeerd worden waarbij gebruik gemaakt moest worden van deze technieken. Ten slotte zou een controlealgoritme geïmplementeerd moeten worden dat automatisch bepaalt welke techniek de juiste is voor dat moment en ervoor zorgt dat het model met die techniek wordt geanimeerd. Tijdens de hele implementatiefase was het de bedoeling om zoveel mogelijk gebruik te maken van reeds aanwezige en vrij verkrijgbare middelen.

Graag zou ik Sara Appeltants en mijn ouders willen bedanken voor het nalezen van mijn thesis en Tamara Vos voor de hulp met LaTeX. Sara ben ik ook dankbaar voor de nodige wiskundige uitleg.

Mijn promotor Prof. Dr. Wim Lamotte en mijn begeleiders Pieter Jorissen en Jeroen Dierckx wil ik danken voor de hulp en uitleg tijdens het maken van mijn implementatie. Pieter en Jeroen dank ik ook voor de nuttige tips bij het schrijven van mijn thesis.

Mijn ouders en Sara wil ik nog in het bijzonder dank zeggen voor hun steun tijdens het maken van mijn thesis.

Peter Boyen

Diepenbeek, februari 2007

Inhoudsopgave

Lijst van figuren

Hoofdstuk 1

Inleiding

Door de snel stijgende CPU-snelheden en verbeterende grafische kaarten wordt er steeds meer mogelijk in animatie. Niet alleen worden de modellen gedetailleerder, maar de animatie wordt ook realistischer gemaakt. De grafische kant van applicaties wordt vaker een verkoopspunt en daarom ook belangrijker.

Een punt dat de laatste tijd meer aandacht heeft gekregen, is de interactie met andere modellen en de omgeving. In veel applicaties wordt interactie steeds belangrijker, omdat het immersieverhogend werkt. Om deze interacties geloofwaardig te maken wordt steeds vaker gebruik gemaakt van fysische simulaties.

Deze technologie kan benut worden in games en in verschillende soorten GVO's (Genetwerkte Virtuele Omgevingen).

Deze thesis omvat een literatuurstudie in verband met physically based avatar animation[1], maar ook een implementatie.

De eerste drie hoofdstukken behandelen uitgebreid de theoretische kant van physically based animation. De andere hoofdstukken verduidelijken de praktische kant en de implementatie die bij deze thesis hoort.

Het tweede hoofdstuk is een inleiding over enkele topics in de computeranimatie. Het bevat informatie over modellering, deformation en kinematische animatie.

Hoofdstuk 3 legt een basis voor het begrijpen van fysisch gebaseerde of dynamische animatietechnieken.

Vervolgens wordt in hoofdstuk 4 het concept van controllers uitgelegd. We proberen hier ook een gedetailleerde definitie van een controller te geven. Daarna verdiepen we ons in manieren om verschillende controllers samen te voegen onder één hoofdcontroller. Het hoodstuk wordt afgesloten met een bespreking van een specifieke controller, namelijk een balanscontroller.

Hierna volgt er nog een conclusie van de literatuurstudie in hoofdstuk 5.

[1] Een avatar is een model dat een gebruiker voorstelt in een virtuele omgeving.

Vervolgens volgt het deel over de Implementatie. Hoofdstuk 6 beschrijft de reeds aanwezige middelen om de implementatie uit te voeren.

Hoofdstukken 7 en 8 beschrijven respectievelijk hoe we de fysische simulatie voor een model voorzien en de aanmaak van twee controllers, een ragdollcontroller en een balanscontroller, die hiervan gebruik maken.

In het volgende hoofdstuk wordt aangegeven hoe een gebruiker de bestaande implementatie kan uitbreiden met behulp van plugins.

Daarna worden in hoofdstuk 10 enkele resultaten van tests met de gemaakte balanscontroller voorgesteld.

Ten slotte bevat hoofdstuk 11 een aantal mogelijke uitbreidingen waarna in hoofdstuk 12 een conclusie van de implementatie geleverd wordt.

Daarna wordt in hoofdstuk 13 de algemene conlusie geformuleerd.

Deel I

Literatuurstudie

Hoofdstuk 2

3D-computeranimatie

Dit hoofdstuk bevat een achtergrond van 3D-computeranimatie. Dit hoofdstuk is slechts een inleiding, zonder de bedoeling een volledig overzicht te zijn.

Eerst geven we een aantal definities, waarna we drie verschillende onderdelen in de 3D-computeranimatie bespreken: modellering, deformation en animatie. Ten slotte geven we nog een idee van de verdere indeling van deze tekst.

2.1 Definities

Een 3D-model is een representatie van een object, gemaakt voor weergave op een computerscherm. Dit kan een bestaand object of een fictioneel figuur zijn, zo klein als een atoom of zo groot als een wolkenkrabber. Alles wat in de fysische wereld kan bestaan, kan worden voorgesteld met een 3D-model. ([Answers.com, 2006a])

Computeranimatie is het maken van bewegende beelden met behulp van een computer. Het is een onderdeel van computer graphics en animatie. Meer en meer wordt computeranimatie gemaakt met 3D computer graphics (3D-computeranimatie), maar ook 2D graphics worden nog vaak gebruikt. Computeranimatie wordt ook wel CGI (Computer Generated Imagery) genoemd, vooral wanneer het in films gebruikt wordt. ([Answers.com, 2006b])

2.2 Modellering

Computeranimatie begint bij het maken van het model (zie 2.1). Hierbij horen de keuzes hoe de data te verkrijgen (2.2.1) en uit welke onderdelen het model opgebouwd wordt (2.2.2).

2.2.1 Data verkrijgen

Een 3D-model kan op verschillende manieren verkregen worden. Een veel gebruikte manier is het handmatig maken, een andere mogelijkheid is het model te verkrijgen uit een scan van een object.

Handmatig modelleren

Men kan een model maken met één van de vele 3D-modeling tools die te verkrijgen zijn. De programma's 3ds Max ([Autodesk, 2006a]), Cinema 4D ([MAXON Computer GmbH, 2006]), Maya ([Autodesk, 2006b]), SoftImage XSI ([SoftImage, 2006]), LightWave 3D ([Newtek, 2006]) zijn hierbij de standaard ([Collins and Hilton, 2001], [Thacker, 2005]).

Het is echter niet eenvoudig om een 3D-model te maken op een scherm dat een 2D-weergave toont. Er wordt dan ook onderzoek gedaan naar manieren om de interactie tussen de maker en het model in wording natuurlijker te maken zoals 3D-brillen, head tracking sensoren ([Turner and Gobbetti,]) en haptic devices ([Klosowski et al., 1998]).

Scannen

Een eenvoudigere manier van werken voor een kunstenaar is het model te maken, bijvoorbeeld in klei, en dan met behulp van een 3D-scanner om te zetten naar een computermodel ([Collins and Hilton, 2001]).

Er zijn verschillende methodes om een model in klei met behulp van een 3D-scanner om te zetten in een computermodel. De 3D-scanner kan bijvoorbeeld gebruik maken van een drukgevoelige sensor die het model aftast of van een optische sensor. Voor de optische scanner worden er een aantal lijnen op het model geprojecteerd. De scanner vormt een driedimensioneel model met behulp van het resulterende patroon dat hij scant in twee dimensies. Het voordeel van de optische scanner is dat hij snel is en dat er geen contact met het model nodig is. De optische scanner heeft als nadeel dat het nodig is om met behulp van een verzameling van tweedimensionele beelden een driedimensioneel model te vormen ([Collins and Hilton, 2001]).

2.2.2 Voorstelling

Er zijn verschillende manieren om de data die een model bepalen voor te stellen. De belangrijkste zijn polygonmodellen, parametric surfaces en implicit surfaces. Deze worden verder besproken.

Polygonen

Een polygonmodel ([Collins and Hilton, 2001]) is een model dat bestaat uit punten. Deze punten zijn de hoekpunten van de veelhoeken die het oppervlak van het model vormen. Voor deze veelhoeken worden meestal driehoeken gebruikt. De punten, zijden en veelhoeken in een polygonmodel noemen we respectievelijk vertices, edges en polygonen.

Het grootste voordeel van deze representatie is dat een polygon een zeer eenvoudig bouwblok is en dat veel soorten oppervlakken benaderd kunnen worden door polygonen. Een polygon renderen is ook zeer eenvoudig en efficiënt omdat dit rechtstreeks ondersteund wordt door de huidige hardware. Het grootste nadeel is dat er veel polygonen nodig zijn om gekromde of gladde oppervlakken te benaderen.

Parametric surfaces

Parametric surfaces ([Collins and Hilton, 2001]) zijn oppervlakken waarvan alle punten bepaald worden door een formule met daarin een klein aantal controlepunten. Deze controlepunten hoeven niet noodzakelijk op het oppervlak te liggen. Ze zijn eenvoudig te gebruiken omdat met een paar controlepunten een groot gerond oppervlak gedefinieerd kan worden.

Het probleem met parametric surfaces is het aaneensluiten van verschillende patches zodat bij de aaneensluiting het oppervlak glad blijft. Ook moeten ze gediscretiseerd worden voor men ze kan renderen (bijvoorbeeld naar een polygonalisatie).

Het meest bekende parametric surface is misschien het Bezier surface. De NURBS (Non-Uniform Rational B-Splines) zijn de computer graphics standaard.

Implicit surfaces

Een implicit surface is een oppervlak bestaande uit alle punten die aan een bepaalde voorwaarde voldoen. Wiskundig wordt deze voorwaarde voorgesteld met behulp van een functie f, met een punt p als argument ([Bloomenthal, 2001], [Bourke, 1997], [Collins and Hilton, 2001], [Giang et al., 2000]).

Per definitie, als $f(p) = 0$ dan ligt p op het oppervlak. De functiewaarde wordt bepaald aan de hand van de afstand van het punt p tot onderliggende controleobjecten zoals punten, lijnen, ...

Een voordeel van deze methode is dat $f(p)$ drie mogelijke uitkomsten heeft voor elk punt p:

- 0, wat betekent dat p op het oppervlak ligt.
- Een getal kleiner dan 0, wat betekent dat p zich binnen het volume bevindt.
- Een getal groter dan 0, wat betekent dat p buiten het volume valt.

Dit zorgt ervoor dat deze methode zeer geschikt is om collision detection mee uit te voeren ([Giang et al., 2000]).

Renderen en modelconstructie zijn echter verre van eenvoudig omwille van de moeilijkheid om deze oppervlakken te discretiseren en het feit dat deze methode niet intuïtief is in het gebruik zoals de voorgaande methodes ([Collins and Hilton, 2001]).

De manier waarop de data verkregen zijn, geeft vaak ook aan in welke voorstelling deze opgeslagen worden. Gescande modellen zijn gewoonlijk een puntenwolk die gemakkelijk in polygonen omgezet kan worden. Handgemaakte modellen daarentegen bestaan vaak uit parametric surfaces ([Collins and Hilton, 2001]).

2.3 Deformation

Deformation bepaalt hoe een model vervormt als een deel van het model beweegt. Als een model zijn arm buigt, kan je de bovenarm laten opzwellen om de illusie te wekken dat het

model realistisch zijn spieren gebruikt. Om dit te bekomen zijn er verschillende technieken, allemaal met een verschillende graad van realisme, snelheid en controle.

2.3.1 Directe methodes

In elke voorstelling van een model kan men wel zelf alle vertices of controlepunten één voor één verplaatsen. Deze directe methodes geven in het algemeen de meeste controle over de vervorming.

Het nadeel is echter dat dit een onmogelijk werk wordt als men een model met duizenden punten wil aanpassen ([Collins and Hilton, 2001]).

2.3.2 Interpoleren tussen houdingen

Een andere methode bestaat erin een aantal houdingen te maken voor het model met behulp van directe methodes. Andere houdingen worden dan bekomen door te interpoleren tussen deze bestaande houdingen, Shape interpolation genoemd in [Collins and Hilton, 2001] en Inbetweening in [Anderson, 2001].

Deze methode heeft verschillende nadelen. Een eerste nadeel is dat de mogelijkheden beperkt zijn door het aantal reeds bestaande houdingen en door de gebruikte interpolatie. Ook is het nodig om veel modellen in verschillende posities bij te houden om de beweging vlot te doen lijken. Hierdoor is de benodigde opslag en geheugen heel groot.

Deze methode kan wel goede resultaten opleveren op plaatsen waar andere methodes problemen ondervinden. Waar een gelaagde methode (zie 2.3.5) bijvoorbeeld problemen kan ondervinden door intersecties met zichzelf, kan de shape interpolation toch een correcte vervorming geven.

2.3.3 Free form deformation

Men kan vertices of controlepunten in een lokaal assenstelsel plaatsen en vervolgens het assenstelsel vervormen. De nieuwe posities van de punten kunnen dan geïnterpoleerd worden ten opzichte van controlepunten in dat assenstelsel ([Collins and Hilton, 2001], [Giang et al., 2000]). Deze methode noemt men free form deformation of FFD.

Men kan deze methode vergelijken met het object in vervormbaar plastic steken. Als het plastic vervormd wordt, vervormt het object erin mee.

Het voordeel van deze methode is dat ze zeer algemeen is: aangezien men het assenstelsel vervormt, is het niet nodig iets te weten over het eigenlijke object dat zich in het assenstelsel bevindt ([Giang et al., 2000]).

2.3.4 Hiërarchische methodes

We kunnen ons model opdelen in verschillende submodellen en deze modellen met elkaar op een hiërarchische manier verbinden met scharnieren ([Anderson, 2001]). Als een object bewogen wordt, bewegen de objecten die zich in de hiërarchie onder dat object bevinden

automatisch mee. Dit wordt verkregen door voor elk object de transformatie bij te houden ten opzichte van het bovenliggende object, zijn parent.

Het nadeel van deze methode is dat er afzonderlijke objecten zijn, en dat ze afzonderlijk bewegen. Tijdens het bewegen kunnen er dus spleten zichtbaar worden. Dit kan opgelost worden door de objecten te laten overlappen, maar dan kunnen er weer zichtbare naden tevoorschijn komen.

2.3.5 Gelaagde methodes

Een eenvoudig model met weinig vertices of controlepunten is veel gemakkelijker te vervormen, dan een ingewikkeld model met veel punten. Maar we kunnen een eenvoudiger model wel gebruiken om de punten van een ingewikkelder model op een automatische manier te bewegen. Men spreekt dan over lagen: het eenvoudige model is een onderliggende laag voor het ingewikkelde model. Dit lost het probleem van directe methodes (zie 2.3.1) op. Deze gelaagde aanpak is een zeer belangrijk concept dat nu bijna overal gebruikt wordt ([Anderson, 2001], [Collins and Hilton, 2001], [Giang et al., 2000]).

Men kan bijvoorbeeld een punt op de gedetailleerde laag zijn afstand tot een controlepunt op de lagere laag laten behouden tijdens het animeren. Het is ook mogelijk om het punt van verschillende controlepunten te laten afhangen. Gewichten bepalen dan de afhankelijkheid van elk punt. De uiteindelijke positie wordt dan geïnterpoleerd uit de oorspronkelijke afstanden en de gewichten ([Anderson, 2001]).

Gewoonlijk plaatst men een skelet in een model zodat men dit eenvoudige draadpoppetje kan bewegen in plaats van het gehele model. Het is ook mogelijk om hierbij verschillende lagen te gebruiken waarbij de ene laag telkens een hogere laag beweegt. Zo kan men bijvoorbeeld modellen maken die een skelet-, spier-, vet- en huidlaag hebben ([Collins and Hilton, 2001], [Giang et al., 2000]).

Het probleem hierbij is beslissen welke vertices of controlepunten door welke punten op de lagere laag bewogen worden. Een foute toekenning kan voor problemen zorgen die zichtbaar worden bij het renderen. Ook kan de manier om de bovenliggende laag te animeren voor problemen zorgen. Het model kan zichzelf gaan snijden als men niet voorzichtig is ([Collins and Hilton, 2001]).

Men kan ook andere methodes gebruiken om dit te vermijden, zoals physically based deformation (zie 2.3.7).

2.3.6 Hiërarchische skeletmethodes

Hiërarchische skeletmethodes zijn een samenvoeging van de twee voorgaande methodes en hebben alle voordelen en geen van de nadelen van deze methodes.

Gelaagde methodes (zie 2.3.5) hebben het nadeel dat de onderliggende lagen nog altijd met directe methodes veranderd moeten worden, en ook al is dit een verbetering ten opzichte van directe methodes, het is nog verre van ideaal.

Als we echter op het onderliggende skelet van het model de hiërarchische methode toepassen, dan kunnen we ons model al veel natuurlijker animeren. Om de arm te bewegen, dienen

we niet meer elk punt in het armskelet aan te passen, één object bewegen is voldoende. Bij deze methode noemen we de scharnierpunten joints en de objecten bones ([Anderson, 2001], [Collins and Hilton, 2001]).

En aangezien de bovenste laag één geheel is in plaats van afzonderlijke objecten, vermijdt deze methode ook de spleten en zichtbare naden die het probleem vormen van de hiërarchische methodes (zie 2.3.4).

Het skelet is een hiërarchie van lokale frames die een positie en oriëntatie aangeven ten opzichte van het bovenliggende frame, de parent. Het bovenste frame geeft de positie en oriëntatie ten opzichte van de virtuele wereld. Elke mogelijke richting om te bewegen noemen we een vrijheidsgraad. Per joint zijn er maximaal 6 vrijheidsgraden: beweging volgens de X-, Y- en Z-as, rotatie rond de X-, Y- en Z-as. Bij een menselijke figuur zijn er vaak minder vrijheidsgraden omdat de meeste joints enkel rotatie en geen of heel weinig beweging toelaten ([Multon et al., 1999]).

2.3.7 Physically based deformation

Deze methode ([Collins and Hilton, 2001]) wordt vaak physically based animation genoemd. Hier wordt ze echter vermeld als physically based deformation omdat deze term preciezer aangeeft wat de methode inhoudt en om duidelijk aan te geven dat dit een ander onderwerp is dan het onderwerp van deze thesis.

In plaats van de hogerliggende lagen te besturen door interpolatie van afstanden, kan men in een gelaagd systeem ook de vertices laten bewegen door een fysisch bepaalde kracht. Op deze manier is het mogelijk om zachte objecten voor te stellen ([Giang et al., 2000]).

Men stelt daarvoor een systeem van veren op dat voldoet aan de wetten van Newton en aan de veerwet van Hooks. Deze wetten simuleert men door een stel differentiaalvergelijkingen (zie 3.4.1) op te lossen. De veren worden zodanig geplaatst dat ze de huid van het skelet wegduwen, maar er wordt op gelet dat de huid zichzelf niet snijdt. Hiermee maakt men een model voor spieren en huid ([Giang et al., 2000]).

Spijtig genoeg vergt deze methode zodanig veel berekeningen dat deze voorlopig niet geschikt is voor real-time applicaties. Ook zijn er nog verschillende open problemen rond dit soort animatie. Het is bijvoorbeeld niet vanzelfsprekend om grote hoeveelheden differentiaalvergelijkingen snel op te lossen. Nog minder vanzelfsprekend is het om de simulatie daarbij stabiel te houden. Het ontdekken van botsingen tussen huidgedeeltes is een zeer moeilijke en tijdrovende taak.

2.3.8 Hybride methodes

Het is ook mogelijk om verschillende van de reeds beschreven methodes in één model te combineren ([Collins and Hilton, 2001]).

Voorbeeld. [Chadwick et al., 1989] stelt een systeem voor waarin een skelet de controlepunten van een FFD (zie 2.3.3) vervormt. Deze FFD vervormt de spieren en gebruikt fysische veerwetten om controlepunten op vetgedeeltes zoals de wangen te bewegen.

Figuur 2.1: Enkele keyframes geven een hele animatie aan

De keuze van methode, en in het geval van gelaagde methodes het aantal lagen, hangt af van de snelheid waarin de berekeningen moeten gebeuren, voornamelijk of deze berekeningen al dan niet in real-time moeten gebeuren. Er hangt ook veel af van de gewenste graad van realisme. Het is ook niet verbazingwekkend dat deze twee, snelheid en realisme, vaak tegen elkaar afgewogen moeten worden ([Giang et al., 2000]).

2.4 Animatie

Als animatie beschouwen we alle methodes die de bewegingen van een model doorheen de tijd bepalen. Wij zijn voornamelijk geïnteresseerd in de beweging van een virtuele mens, maar de technieken in dit hoofdstuk zijn op elk model van toepassing.

We veronderstellen wel dat elk model uitgerust is met een bijbehorende gelaagde structuur (zie 2.3.6). Dit doen we omdat de skeletanimatie de tegenwoordige standaard is in computer graphics. Dit is te merken in de meeste nieuwe games.

We veronderstellen dat alle lagen de skeletlaag volgen en bekijken nu hoe we de beweging van deze skeletlaag voorschrijven.

2.4.1 Keyframing

Een methode om een model te animeren, is voor alle belangrijke momenten in de tijd de positie aan te geven en voor de andere tijdstippen te interpoleren tussen deze posities. Dit noemt men keyframing en één zo'n belangrijk moment noemt men een keyframe. Dit principe zien we in figuur 2.1.

De meest eenvoudige methode om keyframes te maken is handmatig aan te geven in welke positie het model zich dient te bevinden ([Giang et al., 2000]).

Deze methode wordt vaak gebruikt omwille van zijn eenvoud, maar laat nog veel werk over aan de animator. Ook is deze methode niet erg geschikt om gebruikt te worden in een interactieve omgeving omdat niet alle houdingen vooraf gemaakt kunnen worden. Het kan ook moeilijk zijn om realistisch ogende animaties te maken ([Giang et al., 2000]).

2.4.2 Motion capture

In plaats van keyframes met de hand te maken is het ook mogelijk om bewegingen van een echte persoon op te nemen en deze te gebruiken. Deze techniek, die motion captu-

re genoemd wordt, is heel populair in de film- en spellenindustrie ([Multon et al., 1999], [Giang et al., 2000]).

Het voordeel van deze techniek is dat de bewegingen intrinsiek realistisch zijn omdat ze echt zijn. Vaak worden uit deze bewegingen verschillende specifieke bewegingen gegenereerd die de virtuele omgeving mee in rekening brengen. Dit is nodig om realistisch ogende animaties te maken in het geval van contact met de omgeving. Hierbij rekenen we bij de omgeving alle andere objecten, dus ook andere modellen ([Multon et al., 1999]).

Motion blending

Om de bewegingen van motion capture aan te passen kan men motion blending gebruiken. Dit maakt gebruik van een database van karakteristieke bewegingen en maakt nieuwe bewegingen door te interpoleren tussen de parameters van deze bewegingen. Het voordeel hiervan is dat de methode heel weinig berekeningen nodig heeft, maar de mogelijkheden zijn beperkt door het aantal originele bewegingen. Nog een nadeel is dat er geen garantie is op het realisme van de geïnterpoleerde bewegingen, ook al zijn de originele bewegingen intrinsiek realistisch ([Multon et al., 1999]).

Motion graphs

Het normale gebruik van motion capture data is om volledige bewegingen te recreëren. Motion graphs breken deze opgenomen clips in kleine delen en laten toe om deze delen achter elkaar te zetten zolang dit geen te groot visueel kwaliteitsverlies oplevert ([Reitsma, 2006]).

Deze methode heeft al de voordelen van motion capture (2.4.2), maar heeft de mogelijkheid om extra bewegingen te genereren. Die bewegingen blijven nog wel beperkt door de originele bewegingen.

Motion editing

De data verkregen door motion capture is gedetailleerd en realistisch, maar is statisch. Je kan enkel exact de opgenomen data afspelen. Daarom wordt motion capture data gewoonlijk aangepast aan de noden bij het gebruik ([Reitsma, 2006]).

Om deze data aan te passen worden verschillende technieken gebruikt. Men kan één clip aanpassen of twee (of meer) verschillende clips op verschillende manieren combineren. Men kan bijvoorbeeld lichaamsdelen draaien, of ledematen van verschillende clips combineren. Deze methodes hebben als voordeel dat je veel verschillende animaties kunt maken met motion capture data. Spijtig genoeg heeft motion editing als grote probleem dat het voordeel van motion capture hierdoor snel teniet gedaan kan worden. Als de data van de motion capture zomaar aangepast wordt, is er geen garantie dat het inherente realisme van de data bewaard blijft ([Reitsma, 2006]). Er wordt vooruitgang geboekt in het bewaren van realisme bij het aanpassen van motion capture data. In [Yan li, 2002] wordt een techniek gebruikt, motion texturing, die de beweging aanpast met behulp van een statistisch model.

2.4.3 Forward en inverse kinematics

Technieken die aangegeven worden met de term kinematic, werken zonder de krachten die de beweging veroorzaken in rekening te brengen.

Forward kinematics

Forward kinematics stelt de positie en rotatie van alle joints in de hiërarchie in en bekomt zo de positie van het uiteinde van een ketting van joints, de end-effector genaamd ([Multon et al., 1999], [Giang et al., 2000]).

Deze methode kan gebruikt worden om externe informatie om te vormen naar keyframes. Men kan bijvoorbeeld een biomechanisch model omzetten. Dit heeft als voordeel dat de kwaliteit van de beweging afhankelijk is van de gebruikte informatie en niet van het kunnen van de animator ([Multon et al., 1999]).

Inverse kinematics

Inverse kinematics specifieert de gewenste positie van de end-effector en probeert die te bereiken door de positie en rotatie van de joints in te stellen. Het voordeel hierbij is dat er nog minder informatie door de animator aangegeven moet worden. Slechts een paar posities moeten aangegeven worden en de hele hiërarchie volgt ([Anderson, 2001], [Multon et al., 1999], [Giang et al., 2000]).

Dit is een handige methode om interactie met de omgeving te verzorgen. Als een model een voorwerp moet oprapen, kan de hand naar die positie gebracht worden met inverse kinematics. Dit is een actie die met forward kinematics niet zo eenvoudig is, omdat men dan alle rotaties van de joints moet bepalen waarmee men de hand op de juiste positie krijgt.

Inverse kinematics wordt ook regelmatig gebruikt in andere methodes als postprocessing om contactconstraints op te lossen, zoals bijvoorbeeld voeten die op de grond moeten staan ([Multon et al., 1999]).

Er zijn verschillende methodes om inverse kinematics te realiseren. Deze methodes hebben allemaal andere voordelen zoals realisme, snelheid, ... ([Michael Meredith, 2004])

Inverse kinematics is een zeer beperkte methode die niet veel mogelijkheden biedt. Het is echter wel precies. Als enkel een been verplaatst wordt om de voet op de grond te krijgen of een hand om rond een object te passen, is het dus een goede methode ([Multon et al., 1999]).

2.4.4 Constraint based methodes

Men kan ook de gewenste resultaten bekomen door de gewenste beweging, de structuur van het model en fysische bronnen die aanwezig zijn te specifiëren en daarna de animatie als een optimalisatieprobleem op te lossen. Voor deze techniek wordt vaak de gebruikte energie als de te optimaliseren kostfunctie gebruikt ([Giang et al., 2000], [Mandel, 2004]). Deze technieken optimaliseren de baan voor alle lichaamsdelen van het model. Ze staan daarom ook bekend als trajectory optimalisation of trajectory based technieken.

Deze technieken hebben echter enkele nadelen. Ze zijn beperkt tot zeer eenvoudige systemen omdat de hoeveelheid berekeningen enorm is. Er is wel de mogelijkheid om deze methodes toe te passen op enkel de belangrijke vrijheidsgraden van een model en daarna de bekomen oplossing te gebruiken voor een gecompliceerder model ([Mandel, 2004]). Ook levert een fysisch optimale beweging niet altijd een realistisch ogende beweging. Omdat een convergentiealgoritme wordt gebruikt om te optimaliseren, is het mogelijk dat de fysische wetten niet gerespecteerd worden. Elke constraint moet vooraf ingegeven worden en dus kunnen collisions niet automatisch gedetecteerd en behandeld worden ([Giang et al., 2000]).

2.4.5 Parametric generation

Men kan de beweging van het model volledig laten voorschrijven door verschillende algoritmes. Deze algoritmes kunnen gebaseerd zijn op intuïtie, biomechanische kennis of de hiervoor beschreven methodes. Verschillende van deze methodes zijn beschreven in [Giang et al., 2000].

Men kan bijvoorbeeld, zich baserend op louter intuïtie, een algoritme schrijven voor een wandelend karakter dat eerst zijn ene been opheft en dat verder naar voor neerzet en vervolgens dit herhaalt voor het andere been. Hierbij kan men gebruik maken van inverse kinematics om het contact met de grond te bewaren waar nodig. Ondertussen zorgt een ander onderdeel van het algoritme ervoor dat de torso rechtop blijft en dat de armen bewegen op maat met de benen.

Deze methodes zorgen voor nog minder werk, maar ook nog minder controle voor de animator. Ze zijn een eerste stap in de richting van het maken van controllers. Meer informatie over controllers is te vinden in hoofdstuk 4.

2.4.6 Besluit

Het is wel duidelijk dat er veel verschillende mogelijkheden zijn om animatie van een object te verwezenlijken, met elk hun eigen voor- en nadelen. Wanneer men een animatietechniek kiest, dan doet men dat best vanuit de noden van de applicatie. Hierbij dient men voornamelijk rekening te houden met de noden inzake processorsnelheid, realisme en dynamische mogelijkheden ([Collins and Hilton, 2001]). Een game dat real-time interactiemogelijkheden nodig heeft (real-time, dynamisch), zal totaal andere animatietechnieken gebruiken dan een applicatie die offline animaties voor films maakt (offline, realisme).

De tot hiertoe vernoemde technieken kunnen erg realistisch ogende bewegingen voortbrengen, maar het zijn vaak zeer specifieke bewegingen. Als men nieuw gedrag wil verkrijgen, is de hulp van een animator nodig. De technieken kunnen ook geen onverwachte gebeurtenissen aan en dit is een noodzaak als je realistische interactie tussen twee real-time bestuurde modellen wil verkrijgen. Er bestaan ook methodes die hiervoor wel dynamisch aanpasbaar genoeg zijn en daarvoor gebruik maken van fysische simulatie. Deze worden in de verdere hoofdstukken uitgebreid besproken.

Hoofdstuk 3

Dynamische animatietechnieken

Dynamische animatietechnieken zijn technieken die kunnen reageren op veranderende omgevingen. Het zijn technieken die geschikt zijn voor interactiviteit. In het algemeen houden deze technieken rekening met de krachten nodig om bewegingen te veroorzaken. Ze worden ook vaak physically based technieken genoemd.

In dit hoofdstuk bekijken we eerst de eenvoudigste toepassing van dynamische technieken, ragdoll physics. Vervolgens bekijken we forward en inverse dynamics. Ten slotte geven we een inleiding in hoe zulke fysische simulatie uitgevoerd wordt.

3.1 Gebruik van dynamische technieken

Bij bepaalde complexe interacties met de omgeving is het noodzakelijk rekening te houden met dynamics om realistische bewegingen te genereren ([Multon et al., 1999], [Giang et al., 2000]):

- Het model moet zware objecten dragen.
- Het model moet reageren op externe krachten zoals wind of zwaartekracht.
- De ondergrond is complex.
- Er moet nagegaan worden of de gebruikte energie zich in een realistisch interval bevindt.

De constraint based of trajectory based technieken (zie 2.4.4) zijn ook gebaseerd op fysische interacties, maar ze hebben weinig of geen mogelijkheden tot interactiviteit [Mandel, 2004]. Ze worden hier dus niet tot de dynamische technieken gerekend.

3.2 Ragdoll physics

Vroege computerspelletjes gebruikten met de hand gemaakte kinematische doodsanimaties. Dit was weinig belastend voor de CPU maar had enkele nadelen. Het was bijvoorbeeld mogelijk dat het lichaam door muren heenviel of dat het boven een afgrond in de lucht bleef zweven. Dit is bijvoorbeeld te zien in het spel Oni [Games, 2006b] zoals te zien in figuur 3.1.

Figuur 3.1: Oni ([Games, 2006b]) bevat geen ragdolls

Met de stijgende CPU snelheden werd het mogelijk om real-time fysische simulatie van objecten te doen. De modellen werden nog altijd bestuurd met kinematische technieken, maar als het karakter door een grote fysische kracht geraakt werd, kon deze levenloos ter aarde vallen door aan zijn ledematen fysische objecten te koppelen. Wanneer het karakter doodging, nam de physics engine het over en het lichaam viel neer op een fysisch realistische manier, in de richting van de kracht. Deze manier van vallen is echter nog verschillend van een menselijke manier. Het karakter grijpt bijvoorbeeld niet naar een geraakt lichaamsdeel of beschermt zich niet tijdens zijn val. De ragdoll (letterlijk vertaald lappenpop) dankt zijn naam aan het feit dat de persoon gewoonlijk instort zonder enige spierstijfheid en met rondzwaaiende ledematen. Hij valt dus neer als een stoffen mannetje, een ragdoll([Shapiro et al., 2003]).

Een ragdoll (een voorbeeld is te zien in figuur 3.2) is een verzameling van objecten die elk een bone van het model voorstellen. Deze objecten zijn onderhevig aan de wetten van de zwaartekracht en worden beïnvloed door andere krachten. De objecten moeten ook voldoen aan een bepaalde set constraints zodat ze enkel in fysiek haalbare configuraties terecht komen ([Mandel, 2004], [Shapiro et al., 2003]).

Deze manier om ragdolls te implementeren, waarbij men gebruik maakt van fysische simulatie op vaste objecten, noemt men de constrained rigid bodies methode. Er zijn echter verschillende pseudo-ragdoll implementaties.

3.2.1 Pseudo-ragdolls

Verlet integratie

Jakobsen ontwikkelde in [Jakobsen, 2001] een bepaalde methode voor het spel Hitman: Codename 47 van IO Interactive ([Interactive, 2000]). De methode was voornamelijk gericht op snelheid en geloofwaardigheid. Hieronder valt ook stabiliteit (zie 3.4.1), want objecten die door elkaar drijven en dergelijke, zijn niet geloofwaardig. De accuraatheid van de fysische

Figuur 3.2: Unreal3 ([Games, 2006a]) bevat wel ragdolls

simulatie was echter niet van groot belang. De auteur gebruikt verschillende technieken die allemaal bijdragen aan het succes van de methode. Een belangrijk onderdeel is het feit dat hij rigid bodies, vaste objecten, voorstelt als particles en deze simuleert met Verlet integratie. Deze integratietechniek wordt verder besproken in 3.4.3. Deze en de andere methodes worden uitgebreid besproken in [Jakobsen, 2001].

Inverse kinematics post-processing

Deze techniek, die gebruikt wordt in Halo van Bungie games ([Bungie, 2003]), speelt eerst een kinematische sterfanimatie en maakt dan gebruik van inverse kinematics om het karakter naar een mogelijke positie te brengen. Hierdoor kan een karakter nog wel tijdens de animatie door muren gaan, maar op het einde van de animatie zal het lichaam op een fysisch realistische manier tot rust komen.

Dit zorgt ervoor dat de sterfanimaties menselijker overkomen dan bij gewone ragdolls, maar heeft natuurlijk het nadeel dat er zich nog onrealistische situaties kunnen voordoen, zoals een karakter dat doorheen een muur doodvalt en aan de andere kant tot stilstand komt.

Blended ragdoll

Deze techniek voert een kinematische animatiesequentie uit, maar gaat hierbij na of deze binnen fysische constraints valt. Dit vermindert het poppetjesgevoel van ragdolls en voorziet ook interactie met de omgeving. Deze methode vereist echter kinematische en dynamische processing, wat ze nog zwaarder maakt dan gewone ragdolls.

3.3 Forward en inverse dynamics

Forward dynamics berekent de beweging die veroorzaakt wordt door een kracht: hoe reageert een model als het geraakt wordt door een object van buitenaf ([Multon et al., 1999], [Giang et al., 2000]).

Men kan dynamics gebruiken om beweging te simuleren, maar ook om het dynamisch realisme van de beweging te controleren. Als men een kinematische animatie maakt, kunnen met behulp van inverse dynamics de krachten berekend worden die nodig waren om deze beweging te veroorzaken. Men kan dan nagaan of de krachten realistisch zijn en op basis hiervan de animatie aanpassen.

Als men bijvoorbeeld een model zijn arm wil laten opheffen tot een bepaalde hoogte, gebruikt men inverse dynamics. Inverse dynamics berekent de krachten nodig om een bepaalde beweging te veroorzaken ([Multon et al., 1999], [Giang et al., 2000]).

Inverse dynamics laat de animator toe om op de gewone manier animaties te maken en toch realisme toe te voegen. Het nadeel is dat er wel veel berekeningen worden toegevoegd. Het is mogelijk dat er verschillende iteraties nodig zijn vooraleer de kinematische animatie aan de dynamics controle voldoet. Het is dan ook niet meer gegarandeerd dat de resulterende animatie nog lijkt op de originele ([Multon et al., 1999]).

Een algoritme dat een menselijk model voortbeweegt met behulp van forward dynamics noemen we een controller. De moeilijkheid bij een controller is het vinden van de krachten die de gewenste beweging zullen veroorzaken. En als je deze krachten eenmaal gevonden hebt, zijn deze krachten vaak enkel van toepassing op dit model. De krachten zijn niet zo eenvoudig aan te passen om door elk model gebruikt te worden ([Michiel van de Panne, 1990]).Meer uitleg over controllers vind je in 4.1.

Om deze krachten te vinden wordt de gewenste animatie gewoonlijk met kinematics gemaakt en worden dan de krachten bepaald met inverse dynamics. Het kinematisch maken van de animatie zorgt voor controle voor de animator en de inverse dynamics garanderen fysisch realisme en realistische interactie met de omgeving.

Het is ook mogelijk om de computer een einddoel te geven en daaruit een controller te laten berekenen en die te laten optimaliseren. Deze methode is gelijkaardig aan de constraint based methodes uit 2.4.4, maar deze methode zal tijdens de uitvoering ook rekening houden met de omgeving.

Dynamics technieken laten zeer gedetailleerde interacties met de omgeving toe en zijn dus zeer interessant voor virtual reality toepassingen. Hun grotere kost qua berekeningen wordt met de sterkere processoren en nieuwe physics kaarten steeds haalbaarder.

3.4 Simulatie

Tot nu toe is fysische simulatie als een zwarte doos beschouwd. We gingen er van uit dat er een algoritme was dat dit berekende, maar hoe dit dan zou werken, werd nog niet besproken. In dit deel zullen enkele mogelijkheden besproken worden, maar we gaan hier niet diep op in, omdat het doel van deze thesis het gebruik van bestaande implementaties omvat. Meer informatie is te vinden in [Dierckx, 2004] en [Mandel, 2004].

3.4.1 Differentiaalvergelijkingen

In [Dierckx, 2004] vinden we: "Een differentiaalvergelijking is een vergelijking die de afgeleiden van een onbekende functie bevat. Deze vergelijking heeft een orde (bijvoorbeeld van tweede orde, als er afgeleiden van hoogstens tweede orde voorkomen), en een graad (bijvoorbeeld van tweede graad, als er afgeleiden tot hoogstens de tweede macht voorkomen)".
In fysische simulatie is deze onbekende functie normaal een positie of een gezochte waarde van het te simuleren object. De afgeleide is in dat geval de snelheid. Als er een tweede afgeleide is, is dat de versnelling.

De versnelling kunnen we eenvoudig berekenen aan de hand van krachten door gebruik te maken van de wet van Newton:

$f = m.a$ geeft $a = f/m$.[1]

Numerieke methodes

Aangezien we bepaalde waardes aan het zoeken zijn, hebben we geen wiskundige functies voor deze waardes of hun afgeleiden. De enige berekenbare waarde is de versnelling. Daarom kunnen we nooit een exacte oplossing voor deze differentiaalvergelijkingen bekomen. Wat we wel kunnen doen, is het resultaat benaderen met zogeheten numerieke methodes.

Numerieke methodes vervangen het probleem door een eenvoudiger probleem op twee niveau's:

- Discretiseren: door het continu probleem om te zetten in een discreet probleem wordt het makkelijker op te lossen. Conreet bij simulatie wil dit zeggen dat we, in plaats van de waardes te zoeken in functie van de tijd, enkel waardes zoeken voor bepaalde tijdstippen.
- Iteratieve methodes gebruiken: gebruik maken van een schatting en benaderingen zoeken die (hopelijk) naar de oplossing convergeren.

Deze methodes zijn niet zonder problemen. Om te beginnen is de oplossing maar een benadering van de echte. In computeranimatie is dit echter niet zo erg, zolang het resultaat er maar goed blijft uitzien. Er kunnen ook op verschillende manieren fouten in de oplossingen verschijnen.

- Afrondingsfouten: een computer gebruikt getallen met beperkte precisie, dus hierdoor kan een fout ontstaan.
- Benaderingsfouten: door het gebruik van iteratieve methodes wijkt de oplossing af van de echte oplossing.
- Discretisatiefouten: de oplossing van een discreet probleem is niet noodzakelijk de oplossing van het continu probleem.

[1] Hierbij is f de kracht, m de massa en a de versnelling.

Als er een fout opduikt, propageert deze: als er een kleine fout in een oplossing zit, kan deze leiden tot grotere fouten in volgende oplossingen en zo de oplossing onstabiel maken. Hierover worden duidelijke voorbeelden gegeven in [Dierckx, 2004].

Bij fysische simulatie hebben we niet het volledige resultaat van de vergelijkingen nodig. We hebben een beginsituatie en van daaruit willen we op bepaalde tijdstippen de nieuwe situatie bepalen. Om dit uit te werken zijn er verschillende numerieke methodes beschreven in [Dierckx, 2004], allemaal met verschillende resultaten qua snelheid en stabiliteit.

3.4.2 Constrained rigid body simulation

Om de volledige houding van een model te bepalen moeten we een waarde vinden voor elk van zijn vrijheidsgraden (zie 2.3.6). In [Mandel, 2004] lezen we dat we tweede orde differentiaalvergelijkingen kunnen gebruiken om deze vrijheidsgraden te bepalen. De onbekende functie uit de definitie in 3.4.1 is in dit geval de waarde van de vrijheidsgraden. De eerste en tweede afgeleide zijn de snelheid en versnelling van deze waardes.

De precieze vergelijkingen zijn te vinden in [Mandel, 2004] en meer informatie is te vinden in [Faloutsos, 2002]. Deze vergelijkingen omvatten dan de wetten van Newton, de versnellingen van de vrijheidsgraden en interne/externe krachten die het model beïnvloeden. Onder de externe krachten rekenen we onder andere botsingen met de omgeving en de zwaartekracht. De interne krachten zijn de krachten die het model zelf op zijn joints zet, gegenereerd door een controller (zie 4).

3.4.3 Verlet integratie

In [Jakobsen, 2001] maakt Jakobsen gebruik van Verlet integratie, een numerieke methode (zie 3.4.1) die in tegenstelling tot de meeste geen gebruik maakt van de snelheid. Hij benadert bij elke tijdstap de snelheidsvector door het verschil te nemen tussen de huidige positie en de vorige positie. Het niet gebruiken van de snelheid biedt enkele voordelen. Het systeem blijft stabieler omdat de snelheid en posities elkaar niet beginnen tegen te spreken naarmate fouten accumuleren. De methode werkt ook goed samen met de manier waarop Jakobsen rigid bodies voorstelt en constraints oplost.

Rigid bodies worden voorgesteld met punten (de hoekpunten) waarop afstandsconstraints (overeenkomend met de zijden) liggen. Constraints worden opgelost door de punten te verplaatsen naar de dichtstbijzijnde plaats waar de constraint geldt. Omwille van de Verlet integratie hoeft de snelheid niet aangepast te worden in dit geval, maar gebeurt dit automatisch. Dit levert een snelheidswinst op.

In feite zijn de methodes die Jakobsen toepast, een manier om de ingewikkelde differentiaalvergelijkingen die normaal gebruikt worden bij constrained rigid body simulation (zie 3.4.2) te vereenvoudigen.

De methodes die Jakobsen beschrijft in [Jakobsen, 2001] zijn geschreven met snelheid in het achterhoofd. De technieken zijn enkel geschikt als snelheid en geloofwaardigheid belangrijker zijn dan fysische accuraatheid.

Hoofdstuk 4

Dynamische controllers

In dit hoofdstuk wordt het gebruik van controllers voor animatie behandeld. We proberen duidelijk te maken wat een controller precies is en leggen ook enkele interessante ideeën uit om deze controllers te combineren.

Eerst verduidelijken we wat controllers zijn en waarom deze nuttig zijn (in 4.1). Vervolgens wordt het idee van een overkoepelende controller voor verschillende controllers onderzocht (in 4.2). Ten slotte wordt er een specifieke controller, namelijk de balanscontroller, onder de loep genomen (in 4.3).

4.1 Controllers

Een controller voor een object is een interface tussen de high level commando's die gebruikt worden door de gebruiker of AI en de low-level acties en commando's die naar het model gestuurd worden. Een commando van de gebruiker aan de controller kan een bepaalde animatie van het object doen uitvoeren [Jorissen et al., 2005b]. Een wandelcontroller zou het high-level commando 'vooruit' kunnen uitvoeren door de benen van het model één voor één te bewegen.

Een dynamische controller is een controller die gebruik maakt van fysische simulatie. Het is wel belangrijk op te merken dat voor de gebruiker deze controller gelijkwaardig is aan een controller die daarvan geen gebruik maakt.

Een posecontroller is een controller die een model naar een gewenste positie voert. Deze positie is de input van de controller. Een continue controller genereert automatisch de gewenste poses vanuit het systeem ([Mandel, 2004]).

Hoewel deze beschrijving sterk aan keyframing (zie 2.4.1) doet denken, zijn er een aantal belangrijke verschillen [Mandel, 2004]:

- De controller kan falen. De input bestaat uit het gewenste resultaat, en niet het echte resultaat. De controller zal proberen dit resultaat te bekomen, maar het is mogelijk dat de controller dit niet bereikt.

- Het gewenste resultaat wordt vaak niet ingegeven als een statische keyframe (eventueel afkomstig van motion capture data (zie 2.4.2)). De controller kan deze eindpositie zelf kiezen in functie van de feedback sensoren van het systeem.
- De input bestaat uit relatieve informatie. De globale positie en oriëntatie wordt niet gespecifieerd. Een niet ondersteund karakter zal bijvoorbeeld altijd vallen door de zwaartekracht.

De mogelijkheid om een controller te gebruiken als een afscherming tussen de gebruiker en de low-level commando's van het model wordt in 4.2 verder benut. Aangezien de gebruiker niet hoeft te weten op welke manier het karakter geanimeerd wordt, wordt het ook mogelijk om een controller te laten kiezen op welke manier hij de animatie wenst aan te pakken in functie van de huidige situatie.

Het gebruik maken van de low-level commando's van een model in een controller heeft als nadeel dat de controller afhankelijk is van deze low-level attributen. Een controller is dus gewoonlijk afhankelijk van het model waarop hij werkt. In [J. K. Hodgins, 1997] wordt onderzocht hoe men met behulp van schalering controllers kan aanpassen voor nieuwe karakters.

De controlealgoritmes beschreven in 2.4.5 lijken ook sterk op controllers, maar er is één groot verschil: een controller kan zijn gedrag aanpassen met behulp van feedback.

Een controller heeft de mogelijkheid om informatie over zijn model en de omgeving op te vragen. Met behulp van deze gegevens kan hij zijn gedrag bepalen of aanpassen. Hoe meer feedback de controller gebruikt hoe complexer de controller wordt.

Het is ook met behulp van deze feedback dat een continue controller zijn gewenste poses genereert.

Hierdoor kan een controller zijn gedrag opdelen in verschillende fases, of dynamische reacties voorzien op onvoorziene omstandigheden zoals collisions ([Faloutsos et al., 2001b]).

Voorbeeld. Een wandelcontroller kan 1 wandelcyclus opdelen in verschillende delen, gescheiden door 4 gebeurtenissen ([Multon et al., 1999]):

- De linkervoet verlaat de grond (left takeoff).
- De linkervoet raakt de grond (left footstrike).
- De rechtervoet verlaat de grond (right takeoff).
- De rechtervoet raakt de grond (right footstrike).

Hiermee kan men een wandelcyclus opdelen, bijvoorbeeld in stappen (de periode tussen twee footstrikes). Deze takeoffs en footstrikes kunnen op onregelmatige tijdstippen gebeuren als het karakter over oneffen terrein wandelt, en tijdens elke fase van de cyclus kan de controller op een andere manier werken. In [J. Auslander, 1994] maken de auteurs een wandelcontroller voor een zeer eenvoudig skelet met twee benen, die het hen niet gelukt was te maken zonder gebruik van feedback.

4.1.1 Definitie

Een precieze definitie geven voor een controller is moeilijk, omdat de term in de literatuur nogal vrij gebruikt wordt, maar wij definiëren een controller in dit werk als:

Een controller is een *controlealgoritme* voor het *animeren* van een model. Dit algoritme krijgt als input een *doelstelling* (aangegeven als een *high-level* commando) en *poogt* deze doelstelling te verwezenlijken (gebruik makende van de *low-level* mogelijkheden van het model). Hierbij houdt het in mindere of meerdere mate rekening met de *feedback* die het krijgt van de omgeving.

4.2 Overkoepelende controller

Omdat menselijke beweging zeer ingewikkeld is en er zo veel verschillende soorten bewegingen bestaan, zou het interessant zijn als we voor elke beweging een controller konden maken. Deze bewegingen zouden we dan kunnen samenvoegen om de animatie van een model te maken.

Het samenstellen van eenvoudige controllers om ingewikkeld gedrag te bekomen bevordert het hergebruik van controllers. Men kan zo eigenlijk één grote controller maken die bestaat uit verschillende andere controllers. Dit maakt het ook mogelijk voor onderzoekers om hun werk te delen: als twee onderzoekers hun controllers samenvoegen hoeven ze geen controllers te maken die al door de andere geïmplementeerd zijn.

Een mogelijkheid waarvan we geen vermelding vonden in de literatuur, is het maken van controllers voor verschillende onderdelen van het lichaam. Men zou verschillende bovenlichaam- en onderlichaamcontrollers kunnen combineren om zo meer gevarieerde gehele lichaamscontrollers te verkrijgen. De voordelen hiervan zijn eenvoudig in te zien als je het eenvoudige voorbeeld bekijkt van een boven- en onderlichaamcontroller. Met 3 bovenlichaamcontrollers en 3 onderlichaamcontrollers, kunnen we al 9 full bodycontrollers maken. Het maken van dergelijke controllers wordt eenvoudiger omdat er minder joints bestuurd moeten worden. In het geval van dynamische controllers moeten we wel rekening houden met de balans van het karakter die bepaald wordt door de volledige houding (zie 4.3). Hiervoor is mogelijkerwijs enige coördinatie nodig die via een overkoepelende controller geregeld kan worden.

In het eenvoudige voorbeeld werd het lichaam in twee gesplitst. Kleinere of willekeurige selecties zorgen er voor dat het aantal mogelijke controllers nog sneller groeit, maar het brengt ook problemen met zich mee. Als je willekeurige groeperingen van lichaamsdelen wil toelaten, zal het moeilijk worden om een controller samen te stellen die alle delen controleert en toch elk deel maar één keer. Als er meerdere controllers hetzelfde lichaamsdeel willen beïnvloeden, is er de mogelijkheid om de animaties van controllers die gelijktijdig hetzelfde lichaamsdeel beheersen te blenden.

In de praktijk lijkt het onwaarschijnlijk dat controllers veel met elkaar zullen conflicteren. Als we twee bewegingen willen simuleren met behulp van controllers, dan zullen die bewegingen normaal in één van de volgende drie groepen vallen:

- Ze gebruiken verschillende lichaamsdelen en kunnen dus naast elkaar gebruikt worden.
- Ze gebruiken precies dezelfde lichaamsdelen en er moet dus gewoon gekozen worden tussen de twee.

- De ene controller gebruikt een subset van de lichaamsdelen van de andere. In dit geval lijkt het het beste de controle aan de specifiekere controller te geven. Als deze geen nut had, zou deze niet geactiveerd worden.

Als dit niet zo is, moet er een andere manier gezocht worden zoals bijvoorbeeld een biedproces gelijkaardig aan dat in 4.2.2.

In [W. L. Wooten, 1996] vinden we een aaneenschakeling van verschillende controllers om verschillende bewegingen van een duiker te bekomen. De individuele controllers brengen het model op een vast tijdstip in een vaste positie, waarna de volgende controller het overneemt. Zo kan men een duik samenstellen door uit verschillende mogelijkheden te kiezen voor elk onderdeel van de duik.

De auteurs van [J. Auslander, 1994] hebben een algoritme gemaakt dat een selectie maakt uit een aantal automatisch gegenereerde controllers. De controllers konden daar wel niet in real-time gemaakt worden. De keuze van de opeenvolging van controllers werd deels beïnvloed door een animator en kon dus ook niet in real-time gebeuren.

Michael Mandel beschrijft in [Mandel, 2004] een interessante mengeling van de motion capture methode (zie 2.4.2) en fysische simulatie (zie 3.3). Het programma wisselt tussen de technieken als het nood heeft aan realistische interactie of aan statische animatie. Deze hybride methode heeft als grote voordeel dat je op elk moment de meest toepasselijke methode kunt gebruiken.

Michiel van de Panne stelt in [Michiel van de Panne, 1990] dat het interessant zou zijn om controllers in een hiërarchische structuur te hebben, waarbij controllers meer low-level controllers op een lager niveau zouden aanroepen.

Een algemenere vorm van de twee voorgaande ideeën vinden we in [Shapiro et al., 2003], [Faloutsos et al., 2001b] en [Faloutsos et al., 2001a] waar een overkoepelende controller wordt voorgesteld. Hierin wordt een framework voorgesteld waar een overkoepelende controller kiest uit verschillende onderliggende controllers die voor hem een zwarte doos zijn (hij weet niets van de interne werking).

De overkoepelende controller kan op elk moment de controller kiezen wiens animatie het meest van toepassing is, of hij zou eventueel gebruik kunnen maken van een planningsalgoritme om een bepaald doel te bereiken door verschillende controllers in sequentie aan te roepen.

Elke subcontroller voert op zijn manier een bepaalde animatie uit en is dus een controller zoals beschreven in 4.1, de hoofdcontroller heeft als taak uit alle subcontrollers de meest geschikte te zoeken voor de huidige situatie en deze controller aan te roepen.

De methode heeft enkele belangrijke eigenschappen ([Faloutsos et al., 2001a]):

- Eenvoud: deze manier van controllers samenvoegen is eenvoudig te implementeren en bemoeilijkt de taak van het ontwikkelen van controllers nauwelijks.
- Algemeenheid: het samenvoegen van controllers legt geen beperkingen op de mogelijkheden van de controllers zelf, deze mogen zo eenvoudig of ingewikkeld zijn als de designer wenst.

Het laten kiezen van een geschikte controller door de hoofdcontroller, heeft wel als gevolg dat het aan de animator geen (of weinig, zie 4.2.1) controle biedt over de animatie ([J. Auslander, 1994]). Dit kan je beschouwen als een nadeel van de methode, maar eventueel ook als een voordeel.

4.2.1 Subcontroller

Theoretisch

We beschrijven hier het systeem zoals dat voorgesteld is in [Faloutsos et al., 2001a].

De individuele controllers moeten slechts een aantal dingen melden bij hun hoofdcontroller (zie 4.2.2):

- Zijn precondities.
- Zijn postcondities.
- Zijn verwachte werking.

De precondities zijn de voorwaarden waaraan het model en de omgeving moeten voldoen zodat de controller zijn uitvoering kan beginnen. Bijvoorbeeld een balanscontroller (zie 4.3) kan beginnen te werken als het karakter met zijn voeten op de grond staat en rechtop staat.

Postcondities zijn een aantal doelen die de controller belooft waar te maken als aan zijn precondities voldaan was toen hij begon, en als hij ongestoord kan werken. De balanscontroller bijvoorbeeld belooft het karakter met beide voeten op de grond en rechtopstaand te houden (en dat was zo omwille van de precondities) zolang er niets onverwachts gebeurt, bijvoorbeeld een zware duw tegen het karakter.

Dit brengt ons bij de verwachte werking. Een controller moet kunnen nagaan of er iets onverwachts gebeurd is en of dit hem ervan weerhoudt zijn beloftes, de postcondities, na te komen. Als dit zo is, dan moet hij dit aan de hoofdcontroller melden, zodat deze het probleem zo snel mogelijk kan opvangen door een nieuwe biedronde te starten (zie 4.2.2).

Iets onverwachts kan zich voordoen in de vorm van een onverwachte interactie met de omgeving, maar kan ook voortkomen uit slecht gestelde precondities. Een controllerdesigner is niet onfeilbaar en kan een fout maken in zijn precondities, zodat de controller biedt op een moment dat hij toch niet zijn postcondities kan waarmaken. In dit geval moet de controller ook, zodra hij dit merkt, aangeven dat hij niet meer correct werkt. Om dit te vermijden zijn er mogelijkheden om controllers automatisch hun precondities te laten bepalen met behulp van leeralgoritmes, zoals SVM's[1].

Het is ook mogelijk dat een controller zijn werk succesvol beëindigt in plaats van te falen. Een opsta controller heeft zijn werk volbracht als het karakter rechtstaat en geeft dan de controle terug aan de hoofdcontroller.

Het is niet altijd even eenvoudig om de drie eigenschappen te berekenen voor ingewikkelde karakters, bewegingen en omgevingen. Maar volgens [Faloutsos et al., 2001a] is dit haalbaar en noodzakelijk om het samenstellen van controllers mogelijk te maken.

We kunnen bepaalde elementen definiëren die ons kunnen helpen om de pre- en postcondities wiskundig voor te stellen.

De state $\mathbf{q} = [\mathbf{x}\ \dot{\mathbf{x}}]'$ is een vector van posities en snelheden (de dot duidt op de afgeleide naar de tijd). De positie en snelheid van het zwaartepunt worden aangegeven met $\mathbf{c}$ en $\dot{\mathbf{c}}$. De base of support of support polygon is een veelhoek die de voet of voeten die de grond raken, omringt. Deze wordt weergegeven door $\mathbf{S}$.

[1] Support Vector Machines, een populaire artificiële intelligentietechniek ([Fayyad, 1998]).

Voorbeeld Met behulp van deze elementen kunnen we een deel van de precondities van onze balanscontroller voorstellen:

- $\ddot{\mathbf{c}} < 0.1 m/sec^2$: het zwaartepunt versnelt niet te hard.
- $\dot{\mathbf{c}} < 0.3 m/sec$: het zwaartepunt gaat niet te snel.
- $projection(\mathbf{c}) \in \mathbf{S}$: het zwaartepunt bevindt zich boven de support polygon.

Praktisch

De theoretische ideeën van pre- en postcondities worden praktisch gezien geïmplementeerd in de subcontroller zelf, aan de hand van een biedfunctie. De biedfunctie hoort een getal terug te sturen dat voorstelt hoe goed de controller de situatie aankan. Aangezien de controller toegang heeft tot de scène (Hij heeft de toegang nodig voor zijn feedback, zie 4.1), kan deze zelf nagaan of er aan zijn precondities voldaan is. De gewenste postcondities worden op een of andere manier meegegeven aan de biedfunctie, de controller geeft dan een waarde terug die aangeeft hoe goed hij eraan zal voldoen. Dit geeft de mogelijkheid aan de controller om zelf evaluaties te maken aan de hand van energieverbruik of kans op succes ([Faloutsos et al., 2001a]).

In [Faloutsos et al., 2001a] stellen de auteurs echter dat ze zelf een eenvoudigere methode gebruiken, namelijk dat, als de controller iets nuttigs kan uitvoeren gegeven de precondities, hij een vast gewicht biedt. Dit gewicht kan dan eventueel aangepast worden aan de hand van gebruikersinput.

Met behulp van deze aanpassingen aan het theoretische idee is het niet meer nodig in een implementatie een expliciete representatie te hebben voor de pre- en postcondities van een controller. Dit laat de designer dan ook toe van zijn pre- en postcondities zo eenvoudig of ingewikkeld te implementeren als hij wil, al dan niet gebruik makend van de wiskundige notaties. Hierbij negeert men eigenlijk wel de postcondities en daardoor gaat er een deel algemeenheid verloren. Hierover wordt later nog meer gezegd.

De informatie uit [Shapiro et al., 2003], [Faloutsos et al., 2001b] en [Faloutsos et al., 2001a] leidt ons tot de volgende conclusies over elke klasse die een controller zou implementeren:

- De controller moet toegang hebben tot het model en tot de scène.
- De controller moet zeker de volgende functies hebben:
  - Een biedfunctie, die eventueel een aantal gewenste postcondities meekrijgt.
  - Een controlefunctie, die aangeeft of de controller nog functioneert zoals verwacht.
  - Een stapfunctie, die de controller zijn werk laat uitvoeren.

Hoe de controller toegang heeft tot de scène is van ondergeschikt belang. Men kan deze als pointer meegegeven aan de constructor. Het is natuurlijk ook mogelijk om deze globaal te maken of in het geval van slechts één scène deze aanspreekbaar te maken via het Singleton pattern ([Erich Gamma, 1995]). Het is belangrijk dat de controller toegang heeft tot de scène zodat hij kan nagaan of hij nog optimaal werkt.

Ook in het geval van de gewenste postcondities is het nodig dat de controller deze kent op het moment dat hij biedt. Deze kunnen ook meegegeven worden of ergens opvraagbaar gemaakt worden op een gelijkaardige manier als de scène.

Er zijn verschillende mogelijkheden om met de gewenste postcondities om te gaan in een controller.

- Er is de mogelijkheid om de gewenste postcondities expliciet mee te geven. In dat geval moet de controller designer ervoor zorgen dat de controller voor elke mogelijke set van condities kan nagaan hoe goed hij eraan zal voldoen (gegeven een aantal precondities) en aan de hand hiervan moet hij een waarde teruggeven die dit aangeeft. Maar dit is een taak die te ingewikkeld is om van elke controller designer te vragen, daarom willen we dit vermijden. Deze manier is equivalent met de postcondities opvragen aan het systeem, het probleem blijft hetzelfde.

  Het zou misschien interessant zijn om hierin een zekere mate van LOD (level of detail) te kunnen verwerken. Het systeem zou, als het geen controller vindt die ingewikkelde postcondities kan volbrengen, de postcondities kunnen vereenvoudigen tot er wel een controller gevonden wordt.

- Zoals eerder vermeld, wordt in [Faloutsos et al., 2001a] een vast gewicht toegekend aan de controllers. Als de controller iets zinnigs kan uitvoeren met de huidige condities, dan geeft zijn biedfunctie dat gewicht terug, anders 0. De auteurs gebruiken dan gebruikersinput om de gewichten aan te passen. Hierdoor wordt er ook een nieuwe biedronde gestart (zie 4.2.2). Op deze manier kan er dan ingesteld worden welke bewegingen de voorkeur krijgen.

  Het nadeel hierbij is dat de gebruiker dan expliciet moet weten welke controllers er onder de hoofdcontroller zitten. Om de gewichten aan te passen, moet je weten welke controllers er zijn en welke beweging die voorstellen. Dit schendt niet alleen de afscherming die we voorgesteld hebben in 4.1, maar als de hoeveelheid mogelijke controllers stijgt, wordt het kiezen voor de gebruiker steeds moeilijker. De gebruiker gaat dan eigenlijk de taak van de hoofdcontroller overnemen.

  De taak om vaste gewichten toe te kennen aan de verschillende controllers zou wel door een planningsalgoritme overgenomen kunnen worden, maar dan is het nodig dat de controllers hun postcondities expliciet kunnen aangeven.

  Het is ook mogelijk om de gebruikersinput de gewenste postcondities te laten aanpassen, maar dan enkel als deze op een of andere manier expliciet gemaakt worden.

- De laatste mogelijkheid die hier gegeven wordt, is om de postcondities vast te leggen en invariabel te maken. Je kan de controllers altijd naar een bepaalde situatie laten streven. Dit zou in veel toepassingen voldoende zijn, maar er gaat veel algemeenheid verloren. Als je de controllers naar een ander doel wil laten streven, moet hun volledige biedprocedure herschreven worden.

De aandachtige lezer heeft gemerkt dat we bij de noodzakelijke functies geen functie vermeld hebben om op te vragen of de controller succesvol beëindigd is. Aangezien het in het eenvoudige idee van een hoofdcontroller zoals beschreven in 4.2.2 geen verschil maakt of een controller

beëindigd wordt omdat hij faalt of omdat hij slaagt, kunnen we gewoon de controlefunctie gebruiken om het stoppen aan te geven. Als men ingewikkeldere hoofdcontrollers wil gebruiken, zoals een hoofdcontroller met planningsalgoritme, dan is het best om daar toch een aparte functie voor te schrijven, en functies voor het expliciet opvragen van pre- en postcondities toe te voegen (zie 4.2.2).

De tot hiertoe beschreven methode is een interessante oplossing voor gewone verplaatsingsbewegingen, maar er zijn een aantal soorten bewegingen die dit systeem niet aankan. Er zijn bijvoorbeeld controllers die een parameter nodig hebben. Men kan hierbij denken aan een spring controller die een object heeft om over te springen of een controller die een bepaald object laat opnemen en verplaatst.

Je zou zulke parameters kunnen doorgeven aan de controllers, maar alleen als je weet welke subcontroller aangeroepen gaat worden. Een andere controller heeft misschien andere parameters of helemaal geen parameters nodig. Een mogelijke oplossing is deze parameters in de postcondities op te nemen. Men kan bijvoorbeeld voor een object dat verplaatst moet worden, in de postcondities opnemen dat het object op een andere plaats moet staan.

Je kan de controllers zeggen dat je wil dat het karakter van zijn huidige positie (precondities) naar een andere positie (postcondities) gaat. Maar hoe geef je aan dat je wil dat het karakter ernaartoe huppelt? Dit kan niet worden aangeduid met pre- of postcondities, want of het nu loopt, huppelt of kruipt, die zijn gelijk. Als je alle bewegingen tussen het begin- en eindpunt gaat aangeven, dan heeft het gebruik van een controller geen nut meer en kan je beter keyframe (zie 2.4.1) animatie gebruiken.

Het is mogelijk, zoals eerder vermeld, om de subcontrollers op te delen onder nieuwe samengestelde controllers per type van beweging. De controller geeft dan telkens door aan de bovenliggende controller welke types van bewegingen hij mogelijk maakt. Als er geen bewegingstype gespecifieerd wordt, dan kan de controller aan al zijn kinderen vragen om te bieden (zie 4.2.2). Wanneer er geen goede controller gevonden wordt van het juiste type, kan de controller nagaan of een ander type beweging wel aan de rest van de postcondities voldoet.

In [J. Auslander, 1994] stellen de auteurs een aantal secundaire termen[2] voor die gebruikt kunnen worden in de biedfunctie van controllers. Aan deze termen kan een positieve of negatieve waarde meegegeven worden die aanduidt of we dit gedrag verlangen of willen vermijden.

Deze termen zijn:

- max_cm_height, de maximale hoogte van het zwaartepunt tijdens de beweging.
- slip_sum, de afstand die lichaamsdelen over de grond schuivend afleggen.
- rotations, het aantal keer dat het lichaam ronddraait tijdens de beweging.

De eerste term max_cm_height kan de beweging aanzetten tot springen of huppelen. Slip_sum kan de keuze aanduiden tussen stappen of glijden. Ten slotte kan de laatste term, rotations, het model doen rollen of andere acrobatische bewegingen doen uitvoeren.

[2] de primaire termen zijn te vergelijken met de postcondities zoals we ze hier geven

4.2.2 Hoofdcontroller

Wanneer de hoofdcontroller opgestart wordt, wordt de actieve controller bepaald door een eerste biedronde. Onder normale omstandigheden geeft de hoofdcontroller gewoon de controle door aan de actieve subcontroller tot er een nieuwe biedronde gestart wordt.

Er zijn drie redenen waarom een hoofdcontroller een biedronde zou starten onder zijn subcontrollers ([Faloutsos et al., 2001a], [Faloutsos et al., 2001b]):

- Er is geen actieve controller, tenzij een defaultcontroller.
- De actieve controller is succesvol beëindigd of heeft gefaald (dit kan hij opvragen aan de controller, zie 4.2.1).
- De gebruiker heeft de doelstelling van de controller aangepast.

Tijdens de biedronde beslissen alle controllers of ze de huidige situatie aankunnen, en bieden dan om de controle (zie 4.2.1). We gaan er van uit dat er zich onder de subcontrollers een defaultcontroller bevindt. Als er geen enkele andere controller geschikt is voor de huidige situatie, dan wordt de controle aan deze defaultcontroller toegewezen. Van deze controller wordt verwacht dat hij altijd een logische animatie heeft voor het model, maar deze animatie hoeft niet noodzakelijk de gestelde doelstelling te bewerkstelligen. Wat deze logische animatie inhoudt, hangt af van de applicatie, dit kan bijvoorbeeld een ragdoll (zie 3.2) of een stilstaande balansanimatie (zie 4.3) zijn. Op deze manier is er na een biedronde altijd een controller die het model animeert.

Dit brengt ons tot het volgende algoritme in C++[3]:

```
void step(float seconds)
{
    if( active_controller == default_controller )
        bid();

    active_controller.step(seconds);

    if( !active_controller.workingAsExpected() )
        active_controller = default_controller;
}
```

[3] C++ is hier genomen omdat de implementatie bij deze thesis in C++ gebeurt, een omzetting naar Java of een andere programmeertaal is triviaal.

```
void bid()
{
    int max_bid = 0;
    active_controller = default_controller;

    for(int i = 0; i < controllers.size(); i++)
    {
        int current_bid = controllers[i].bid(postConditions)
        if( current_bid > max_bid )
        {
            max_bid = current_bid;
            active_controller =  controllers[i];
        }
    }
}
```

Het is mogelijk dat een subcontroller ook een samengestelde controller is, maar zolang de controller maar van de vorm is zoals beschreven in 4.2.1 is dit onbelangrijk voor de hoofdcontroller. In [Faloutsos et al., 2001a] stelt men voor dat, naarmate de hoeveelheid controllers in het systeem groeit, het misschien wenselijk zou zijn om de gelijkaardige controllers te groeperen in onderliggende controllers, zodat bijvoorbeeld alle wandelbewegingen door één controller uitgevoerd worden.

Dit is het eenvoudig voorbeeld van een hoofdcontroller uit [Faloutsos et al., 2001a]. Er wordt in [Faloutsos et al., 2001b] een iets ingewikkelder beeld geschetst waar de subcontrollers buiten hun status ook waardes teruggegeven die doelposities aangeven voor elke vrijheidsgraad. Deze waardes worden dan door de hoofdcontroller met behulp van proportional-derivative controllers (zie 4.3.1) omgezet in krachten en vervolgens op het model toegepast. De subcontroller kan ook zelf kiezen om krachten toe te passen en dan geen waardes terug te geven voor de hoofdcontroller.

Dit algoritme gaat uit van de naïeve veronderstelling dat een controller correct gemaakt is, maar zoals aangegeven in 4.2.1 is het mogelijk dat de precondities van een controller incorrect zijn, waardoor deze biedt op momenten dat deze niet hoort te bieden. Hierdoor is het mogelijk dat de hoofdcontroller vastraakt in een lus waarbij een controller telkens biedt om de controle en dan onmiddellijk faalt. Er moet ook een veiligheid ingebouwd worden om dit te voorkomen ([Faloutsos et al., 2001a]).

Het is ook mogelijk om het beslissingsalgoritme van de hoofdcontroller uit te breiden. De controller in [Faloutsos et al., 2001a] kiest gewoon de beste controller uit voor het moment, maar als je expliciet de pre- en postcondities van een subcontroller kan opvragen, dan wordt het ook mogelijk om een planningsalgoritme toe te passen. Met het huidige voorbeeld is het mogelijk dat er geen controller is die de postcondities haalt, maar verschillende waarvan de precondities gehaald zijn. Voorlopig wordt dan de controller die het meeste biedt gekozen, maar misschien is er een controller wiens postcondities de precondities van een andere controller bevatten die wel de gewenste postcondities kan bereiken.

Dit zou de hoofdcontroller krachtiger maar ook ingewikkelder maken. Niet alleen moet er dan een planningsalgoritme gezocht worden, maar ook wordt het dan nodig om de pre- en

postcondities expliciet te kunnen weergeven, hoewel we er tot nu toe voor kozen om deze in de subcontroller te capsuleren (zie 4.2.1).

4.2.3 Wisselen tussen controllers

Er zijn verschillende redenen waarom de controle van een model zou overgaan van één controller naar de andere ([Faloutsos et al., 2001a]):

- De controller is succesvol beëindigd;
- De controller heeft gefaald;
- De gebruiker heeft andere wensen ingegeven.

Ongeacht de reden is het nodig om de overgang tussen verschillende soorten controllers vlot te laten verlopen.

Er zijn twee belangrijke soorten controllers:

- Dynamische controllers;
- Kinematische controllers.

Dynamische controllers zijn de controllers die gebruik maken van fysische simulatie om hun werk te doen. Hierdoor zijn ze zeer geschikt om interactie met de omgeving weer te geven.

Kinematische controllers maken gebruik van statische animatie om hun doelen te verwezenlijken, ze hebben de voor- en nadelen van kinematische animatie zoals beschreven in 2.4. Een kinematische controller verliest de controle om dezelfde redenen als elke controller, behalve dat de controller onmiddellijk faalt bij onverwachte interactie met de omgeving en niet nog probeert het probleem op te lossen.

Het is hier heel belangrijk om een onderscheid te maken tussen onverwachte en niet onverwachte interactie met de omgeving. In [Shapiro et al., 2003] wordt er een onderscheid gemaakt tussen actieve en passieve objecten. Objecten die geen deel uitmaken van de animatie zijn actief en kunnen door een botsing de controller doen falen. Objecten die deel uitmaken van de animatie, zoals een baseball bat bij een karakter in zijn zwaaibeweging, zijn passief en doen de controller niet falen.

Om te kunnen wisselen tussen controllers moeten de precondities van de nieuwe controller waar zijn waneer de oude controller beëindigd is. In [J. Auslander, 1994] worden overgangspunten gezocht door een overkoepelende controller, maar wij kunnen deze verantwoordelijkheid doorgeven aan de subcontrollers, door onze subcontrollers hun precondities te laten controleren in hun biedfunctie.

Tot nu toe hebben we de subcontrollers als een zwarte doos beschouwd, de hoofdcontroller heeft geen idee wat er zich binnenin afspeelt. Maar deze veronderstelling is niet altijd realistisch. Soms moeten er speciale maatregelen getroffen worden om tussen verschillende soorten controllers te wisselen.

Tussen controllers van hetzelfde type is het wisselen normaal geen probleem. Tussen kinematische controllers kunnen de eindtoestand van de eerste en de begintoestand van de tweede gecombineerd worden met blending als deze dicht genoeg bij elkaar liggen. Tussen dynamische controllers kan de tweede de simulatie gewoon overnemen, zolang de precondities van de tweede voldaan zijn als de eerste stopt ([Shapiro et al., 2003]).

Om van dynamische naar kinematische controle over te gaan, kunnen we ook gebruik maken van blending, als de eindtoestand dicht genoeg bij een keyframe van de kinematische controller ligt ([Shapiro et al., 2003]). Als we echter bepaalde kinematische controllers willen gebruiken voor bepaalde doeleinden, moeten we ervoor zorgen dat de fysische simulatie in de buurt van een keyframe komt. Mandel gebruikt dit in [Mandel, 2004] om kinematische opsta-animaties te kunnen gebruiken na een dynamische val.

Om van kinematische naar dynamische controle over te gaan is het nodig dat aan de objecten die de ledematen voorstellen, de juiste snelheden en dergelijke toegewezen worden. Voor dit doel gebruikt men in [Shapiro et al., 2003] dynamic tracking controllers: controllers ontworpen om op een fysische manier een kinematische animatie uit te voeren. Dit maakt het ook mogelijk om de kinematische animatie aan te passen met kleine interactiekrachten.

4.3 Balanscontroller

Het bewaren van balans is een onderwerp van verschillende onderzoeksvelden: biomechanica, robotica, informatica (waaronder computer graphics). Het is dan ook een zeer complex onderwerp. Het zou mogelijk zijn om over dit onderwerp verschillende thesissen te schrijven, we geven hier dus ook zeker geen volledig overzicht. Hier wordt enkel voldoende achtergrond gegeven om de ideeën bij de implementatie te kunnen volgen.

Balanceren, stilstaan in rechtopstaande houding, is een complex gebeuren dat door verschillende factoren beïnvloed wordt ([Faloutsos et al., 2001b]). Er zijn verschillende strategieën die een mens gebruikt om zijn balans te bewaren in het geval dat de balans verstoord wordt. Naarmate de storingen in kracht toenemen, worden bepaalde strategieën meer gebruikt ([Faloutsos et al., 2001b], [Kudoh, 2001]):

- Enkelstrategieën worden het meest gebruikt bij kleine storingen.
- Heupstrategieën worden gebruikt bij grotere storingen.
- Armstrategieën worden gebruikt bij een grote storing als er geen mogelijkheid is om een voet te verzetten.

De namen van de strategieën duiden op de gewrichten die het meest gebruikt worden om de balans te herstellen.

Wij richten onze aandacht hier op de enkelstrategie. Voor het benaderen van dit probleem kunnen we gebruik maken van het inverted pendulum model ([Faloutsos et al., 2001a]). De inverted pendulum is een klassiek probleem waar een gewicht op een paal op de bodem verbonden is met een kar door middel van een scharnier. Dit is een onstabiel evenwicht en om deze constructie in evenwicht te houden moet er constant actief voor evenwicht gezorgd worden. Het scharnier stelt de enkels voor en het gewicht de rest van het lichaam boven de

enkels ([Richard C. Fitzpatrick, 1992]). Dit probleem kunnen we oplossen door met behulp van proportional-derivative controllers (zie 4.3.1) het gewicht boven het scharnier te houden, en de oplossing kunnen we dan gebruiken als afschatting voor het echte probleem. Hoewel het lichaam in werkelijkheid niet zo stijf is als een inverted pendulum, kan in de praktijk de afschatting van het model goed gebruikt worden ([Faloutsos et al., 2001a]).

Op deze manier beweegt de balanscontroller de projectie van het zwaartepunt op de grond naar de gewenste locatie, namelijk het middelpunt van de support polygon ([Kudoh, 2001], [W. L. Wooten, 1996]).

4.3.1 Proportional-derivative controller

Proportional-derivative controllers of PD-controllers zijn controllers die trachten de houding van het model naar een bepaalde positie te brengen. In het geval van de balanscontroller is deze positie de rustpositie.

De input van de controller zijn de gewenste hoeken voor elk gewricht van het model, de output zijn de benodigde krachten om het model daar te krijgen. Dit is dus eigenlijk een vorm van inverse dynamics (zie 3.3).

De volgende formule kan door een PD-controller uitgevoerd worden voor elke vrijheidsgraad. Het resultaat van deze formule is de benodigde kracht op die vrijheidsgraad ([Mandel, 2004], [W. L. Wooten, 1996]): In [Mandel, 2004] wordt een formule gegeven die een PD-controller kan uitvoeren voor elke vrijheidsgraad:

$\tau = k_s(\theta_{des} - \theta) + k_d(-\dot{\theta})$

Hierin zien we duidelijk dat, zoals we intuïtief zouden verwachten, de resulterende kracht τ een functie is van de gewenste staat θ_{des} min de huidige staat θ en de snelheid van de vrijheidsgraad $\dot{\theta}$. We houden dus rekening met de afstand die moet afgelegd worden en de richting waarin reeds bewogen wordt.

k_s en k_d zijn stijfheid en dampingcoëfficiënten. Deze veerconstanten zorgen ervoor dat de vrijheidsgraad zo snel mogelijk naar zijn doel beweegt, maar er niet (al te veel) aan voorbij gaat. Als de dampingcoëfficiënt te laag is, zal de vrijheidsgraad schommelen rond de gewenste waarde in plaats van tot stilstand te komen. Als deze coëfficiënt te hoog is, zal de vrijheidsgraad maar langzaam naar zijn einddoel gaan ([Mandel, 2004]).

Hoofdstuk 5

Conclusie van de literatuurstudie

We bespraken drie onderdelen van 3D-computeranimatie: modellering, vervorming en animatie. Bij modellering merkten we dat de voorstelling van het model sterk verbonden was aan de manier waarop het geconstrueerd was. Een gescand model leent zich eenvoudig tot het maken van een polygonmodel omdat het scannen een puntenwolk oplevert. Een met de hand gemaakt model wordt vaak met parametric surfaces gemaakt omdat dit op eenvoudige wijze geronde oppervlakken kan weergeven. Voordat deze modellen gerenderd kunnen worden, moeten ze vaak nog wel gepolygonaliseerd worden.

Naïeve vervormingsmethodes vragen vaak een onmogelijke hoeveelheid werk als het model uit heel veel punten bestaat, maar ze bieden wel heel veel controle. We zochten een techniek die toeliet om de hoeveelheid werk te verminderen en toch zo veel mogelijk controle te behouden. We hebben die gevonden in de hiërarchische skeletmethodes. Deze laten toe het model met behulp van slechts enkele punten aan te passen en bieden toch voldoende controle voor de meeste animaties.

Er zijn verschillende animatietechnieken en deze hebben elk hun toepassing in bepaalde applicaties. De meeste huidige animatietechnieken gaan er van uit dat er een gelaagde structuur gebruikt wordt voor het model. Deze technieken moeten dan enkel de beweging voorzien voor de onderste laag van het model, meestal de skeletlaag.

Kinematische technieken kunnen ons intrinsiek realististische animaties bezorgen, maar ze zijn weinig geschikt voor applicaties waar er veel (onverwachte of variërende) interactie is tussen het model en de omgeving (zijnde alles behalve dat model).

Voor de applicaties waar we wel nood hebben aan interactie richten we ons tot de dynamische animatietechnieken. Ragdolls zijn de meest eenvoudige toepassing van dynamische technieken. Ze worden fysisch gesimuleerd, maar zijn totaal krachteloos. Ze bieden echter maar één mogelijke animatie. Forward en inverse dynamics worden gebruikt om krachten te zetten op het model, waardoor dit beweegt. Inverse dynamics zijn nuttig als we het model naar een specifieke houding willen brengen. Deze krachten worden dan toegepast met forward dynamics, waardoor er toch rekening gehouden wordt met mogelijke interactie met de omgeving. Dynamische technieken zijn zeer interessant voor virtual reality toepassingen omdat ze deze interactiemogelijkheden ondersteunen. Ze hebben wel een grotere kost qua berekeningen, maar deze wordt met de sterkere processoren en nieuwe physics kaarten steeds haalbaarder. Het is echter vaak niet zo eenvoudig om een eenmaal berekende beweging te hergebruiken

voor andere modellen.

De meest bekende technieken lenen zich zeer goed tot uitvoer met behulp van controllers. Controllers hebben enkele belangrijke voordelen:

- Het is eenvoudig om controllers te delen als er duidelijke afspraken over hun opbouw gemaakt worden. Hierdoor zouden designers hun controllers kunnen delen en zo werk uitsparen.
- Een controller zal opgeven voor hij onrealistische zaken doet. Als de controller opgeeft, kan een andere controller geactiveerd worden.
- Controllers kunnen gebruik maken van feedback. Dit laat ingewikkeldere dynamische technieken toe.
- Het principe van controllers laat een vorm van afscherming toe. De gebruiker kan een high-level commando geven, wat resulteert in de aanroep van een controller. De controller neemt dan de low-level commando's voor zijn rekening.

Nog een voordeel is dat we controllers kunnen samenvoegen om één overkoepelende controller te maken die al de bewegingen kan animeren die zijn subcontrollers kunnen.

We bekeken een methode om controllers op een hiërarchische wijze onder te verdelen. De overkoepelende controller geeft de controle door aan zijn kind dat het meest geschikt is voor de huidige situatie. We hebben het expliciet moeten maken van pre- en postcondities vermeden, zodat de designers deze zo eenvoudig of ingewikkeld konden maken als ze wilden. De precondities konden vermeden worden door deze taak door de subcontrollers zelf te laten uitvoeren in hun biedfunctie. De postcondities konden we niet vermijden, tenzij door ze te negeren. Hiervoor zouden we nog graag een oplossing vinden.

Deel II

Implementatie

Hoofdstuk 6

Uitgangspunten

Bij deze thesis hoort de implementatie van een balanscontroller (zie 4.3). Hierbij was het de bedoeling om eerst een onderzoek te doen naar de bestaande middelen en daarvan zo veel mogelijk te gebruiken. Er is bijvoorbeeld keuze uit vrij verkrijgbare software om fysisch-gebaseerde systemen te simuleren.

Hieronder worden voor een aantal onderdelen kort de onderzochte mogelijkheden beschreven en hierbij de redenen die de keuze het meeste beïnvloed hebben.

6.1 Ageia PhysX

In de nu volgende hoofstukken wordt de opbouw van de implementatie stap voor stap beschreven. De eerste stap in deze implementatie zou normaal het maken van een fysisch simulatiesysteem (zie 3.4) zijn, maar we hebben ervoor gekozen om het reeds bestaande systeem Ageia PhysX [Ageia PhysX, 2006b] te benutten. Dit is een commerciële bibliotheek, maar ze is gratis voor niet-commercieel gebruik. Deze bibliotheek staat bekend om zijn snelheid en heeft uitgebreide mogelijkheden om de instellingen van zijn engine aan te passen. Tests hebben uitgewezen dat de Ageia PhysX engine voldoende stabiliteit en snelheid bezit ([Jorissen et al., 2005b]). Ageia PhysX is ook een belangrijk deel van de engine van verschillende spellen [gamesindustry, 2005]. Een ander voordeel van Ageia PhysX is dat het reeds aanwezig was in het ExtReAM framework (zie 6.3), waar het gebruikt werd voor rigid body simulation.

Ageia PhysX biedt verschillende interessante mogelijkheden, waaronder:

- Het maken van fysische objecten.
- Het maken van joints tussen fysische objecten.
- Het zetten van krachten op joints en objecten.
- Het simuleren van de fysische krachten.

Ageia biedt nog veel andere mogelijkheden, maar deze vier zijn degene waarvan het duidelijk was van in het begin dat ze nodig zouden zijn om aan de implementatie te kunnen beginnen.

Een aantal van de andere mogelijkheden wordt verder nog besproken als ze gebruikt worden in de implementatie.

Andere mogelijkheden voor physics engines zijn Havok [havok, 2006] en ODE [ODE, 2006]. Havok is echter niet gratis verkrijgbaar, dus was hier niet bruikbaar. Voor een verdere bespreking van physics engines verwijzen we naar [Jorissen et al., 2005b] en [Mandel, 2004].

6.2 Model

Er was nood aan een 3D-model om de mogelijkheden van de implementatie te demonstreren. De implementatie is wel op zoveel mogelijk vlakken geschreven om onafhankelijk te zijn van het model. Op bepaalde plaatsen wordt er wel uitgegaan van specifieke informatie van het model, maar daar worden mogelijke technieken vermeld om dit in de toekomst te vermijden.

Er zijn natuurlijk onnoemelijk veel keuzes voor een 3D-model, de keuze werd echter beperkt door een aantal voorwaarden waaraan het moest voldoen:

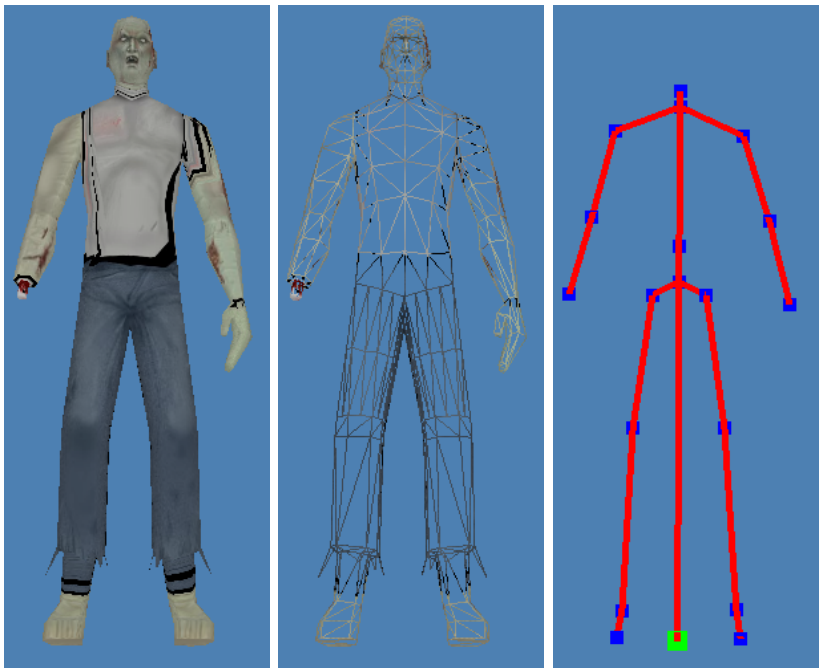
- Het moest gratis te downloaden, te gebruiken en aan te passen zijn.
- Het moest menselijk van vorm zijn.
- Het moest voldoende bones hebben, maar toch niet te ingewikkeld zijn.

De laatste voorwaarde is nogal vaag en vereist extra uitleg. Het model moet over de bones beschikken om bepaalde controllers (we denken hier aan de balanscontroller uit 4.3) mogelijk te maken. Als we echter een bone in het skelet zouden hebben voor elk bot in het menselijk lichaam zou de controller te complex worden.

De belangrijkste joints voor de balanscontroller zijn de enkels, de heupen en de armen. Natuurlijk moeten er nog andere joints in het model zitten om menselijk ogende animaties te kunnen maken.

Het model dat in de voorbeelden gebruikt wordt, is een model van een zombie. Het is een polygon model, gevonden op [Psionic, 2006]. Het werd met de hand gemaakt in het 3D-modeling programma milkshape 3d ([chUmbaLum sOft, 2006]). In figuur 6.1 zijn de verschillende onderdelen van het model weergegeven. In dit model bevindt zich een eenvoudig hiërarchisch skelet, waarvan elke vertex verbonden is met één bone. Dit feit wordt benut in 7.1.2, maar is niet noodzakelijk voor de werking van het programma.

We hebben het model van het internet gehaald, maar het is nog wel aangepast om het beter te gebruiken met de physics engine (zie 7.1.2). Het is niet het meest realistisch uitziende model en ook de vervormingsmethode is niet de best mogelijke, maar deze zaken zijn onbelangrijk voor de relevantie van de implementatie, aangezien ze zich op andere lagen afspelen. Het belangrijke punt is hier de animatie, en daar hebben we in principe enkel het skelet voor nodig. We gebruiken het model echter wel om de vorm en grootte van de fysische objecten te bepalen (zie 7.1.2).



Figuur 6.1: zombie model: a) model b) polygonen c) skelet

6.3 ExtReAM Framework

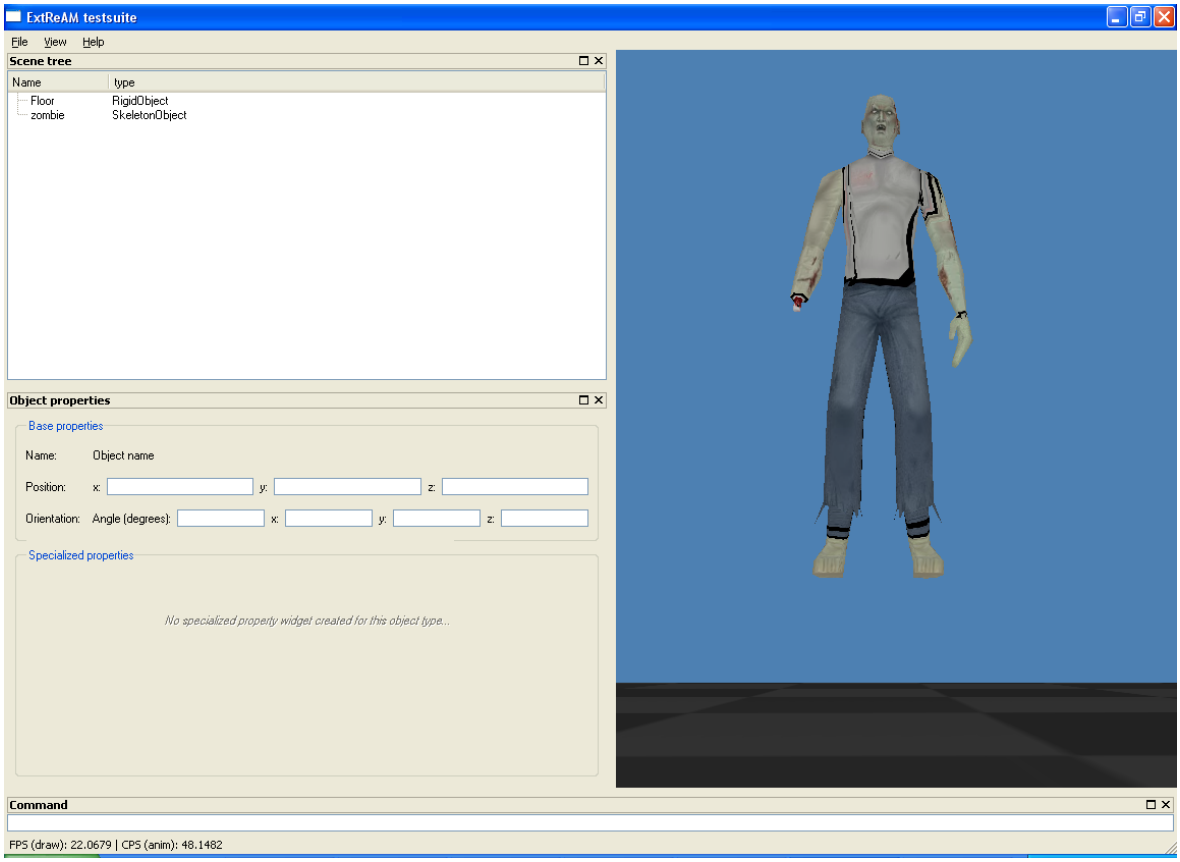
De implementatie is gebeurd in het ExtReAM framework ([Jorissen et al., 2005a]). Het was de bedoeling dit systeem uit te breiden met een basis voor physically based character animation.

Dit framework bevatte reeds enkele essentiële zaken:

- Het inlezen van 3D-modellen in verschillende formaten (zoals ms3d).
- Een standaard vervormingsmethode.
- Rendering (met behulp van OpenGL [SGI, 2006]).
- Een GUI (met behulp van Qt [trolltech, 2006]).
- Skeleton animation.
- Kinematic animation (keyframes).
- Rigid body simulation (het simuleren van onverbonden objecten).

De rigid body simulation maakt ook gebruik van Ageia PhysX (zie 6.1).

Een andere interessante eigenschap van het ExtReAM framework is dat het gebaseerd is op plugins. Het is geschreven om uitbreidbaarheid eenvoudig te maken en dit is een eigenschap die we ook willen bij het ontwikkelen van controllers. De mogelijkheid tot het uitwisselen van controllers als plugins sluit prachtig aan bij het idee van de overkoepelende controller in 4.2.



Figuur 6.2: zombie model in het ExtReAM framework

Hierdoor zien we des te duidelijker hoe belangrijk het is dat de overkoepelende controller met de onderliggende controllers kan omgaan zonder iets van de interne werking te kennen.

Het DANCE framework ([Victor Ng-Thow-Hing, 2005]) en het PhysX Rocket programma ([Ageia PhysX, 2006a]) zijn twee gratis verkrijgbare alternatieven waarin deze implementatie ingevoegd had kunnen worden. Op het moment van implementatie was voor het DANCE framework echter nog geen broncode vrijgegeven. Voor PhysX Rocket is uiteindelijk niet gekozen omwille van de complexe klassenstructuur waaruit het framework is opgebouwd.

6.3.1 Gevolgen

De keuze van het framework had nog enkele andere gevolgen:

- De programmeertaal werd door het framework vastgelegd op C++.
- Het programma zou een plugin worden, maar ook de verschillende controllers werden uiteindelijk als aparte plugins geïmplementeerd (zie 8.1).

In de volgende hoofstukken zullen we verder werken op de basis beschreven in dit hoofdstuk.

Hoofdstuk 7

Het aanmaken van fysische objecten

Om voor een model een dynamische controller te maken heb je voor de verschillende onderdelen van het model (bijvoorbeeld ledematen en torso) objecten nodig waarop krachten uitgeoefend kunnen worden. Deze moeten ervoor zorgen dat het 3D-model de simulatie volgt. Ook heb je een fysische representatie nodig van de gewrichten die het model bij elkaar houden. Anders zou het model gewoon uit elkaar vallen.

Over het aanmaken van deze twee zaken gaat dit hoofdstuk. Eerst worden fysische objecten besproken, daarna komen de joints aan de beurt.

7.1 Het aanmaken van fysische lichaamsdelen

In 2.3.5 bespraken we een gelaagde structuur waarbij de bewegingen in een lager liggende laag de bewegingen op hoger liggende lagen bepaalden. Daar werd de skeletlaag naar voren gebracht als onderste laag die alle posities in de bovenliggende lagen bepaalde. Impliciet gingen we er daar van uit dat deze laag gecontroleerd werd door een animator.

Nu maken we onder deze laag nog een laag, de fysische laag. Voor elk bone in de skeletlaag maken we een object in de fysische laag. Deze fysische objecten zijn de voorstelling van de verschillende lichaamsdelen van het model. De bones van de skeletlaag moeten gewoon deze objecten volgen.

Het verschil tussen de twee lagen is het feit dat de fysische laag niet wordt bestuurd door een animator. De bones uit de skeletlaag horen nog bij het model zelf en zijn delen die (al dan niet door een animator) bestuurd moeten worden. De fysische objecten uit de fysische laag maken deel uit van het controlealgoritme dat het model animeert en zorgen dus voor de besturing. Een fysisch object heeft ook veel meer eigenschappen dan de bones op de skeletlaag. Waar een bone eigenlijk volledig beschreven is door een positie, een oriëntatie en eventuele ouder en kinderen, heeft een fysisch object ook nog een vorm, massa, volume, dichtheid,... Dit is het grote voordeel van bones, zoals gezien in 2.3.5, want dit maakt ze ideaal en eenvoudig om geanimeerd te worden, maar om de fysische simulatie te kunnen uitvoeren hebben we nood aan deze extra eigenschappen.

De posities van de objecten worden tijdens de simulatie bepaald door hun eigenschappen (zoals bijvoorbeeld dichtheid) en door de fysische krachten die erop uitgeoefend worden. Deze

krachten zijn onder andere externe krachten zoals zwaartekracht, maar ook interne krachten die door joints (zie 7.2) uitgeoefend worden.

Bij het aanmaken van de fysische objecten die de lichaamsdelen van het model voorstellen, kunnen we ervoor kiezen de fysische objecten handmatig aan te maken per model of een zekere graad van automatisatie toe te voegen.

7.1.1 Handmatig objecten maken

Het is natuurlijk mogelijk om voor elk model handmatig fysische objecten te definiëren: men kan in een aparte file de nodige informatie van het model opslaan. Dit wordt dan de voorstelling van het model op de fysische laag, met extra informatie zoals de dichtheid en massa van de ledematen.

Om deze taak te verlichten kan men gebruik maken van tools, waardoor deze taak veel op het gewone 3D modelleren begint te lijken. Een mogelijke tool hiervoor is PhysX Create [Ageia PhysX, 2006a]. Deze tool is een plugin voor 3ds max [Autodesk, 2006a] of Maya [Autodesk, 2006b], die toelaat om modellen, gemaakt in deze programma's, te voorzien van fysische objecten in een specifiek Ageia PhysX formaat.

Het grote voordeel van deze methode is natuurlijk de grote mate van controle die je hebt over de vorm en eigenschappen van elk object. Je kan hierbij bepalen hoe precies de objecten de vormen van je model moeten volgen. Je moet hierbij wel opletten: hoe ingewikkelder de vorm die je kiest, hoe trager de simulatie zal lopen, omdat collision detection bemoeilijkt wordt. Maar hoe eenvoudiger de vorm, des te meer kans is er dat de collision detection niet realistisch genoeg werkt, waardoor er lichaamsdelen met elkaar of de omgeving kunnen gaan snijden.

Het nadeel van deze methode is echter dat er voor elk model specifiek andere objecten met de hand gemaakt moeten worden. Spijtig genoeg zit er niet bij elk 3D-model een fysische representatie, dus telkens als men een nieuw model aan het programma zou willen toevoegen, zou het noodzakelijk zijn om hierbij extra informatie aan te maken (en dit kan een langdurig proces zijn).

7.1.2 Automatisch objecten genereren

Als je de fysische objecten handmatig zou maken zoals beschreven in 7.1.1, zou je opmerken dat de informatie (specifieker de vorm) van de fysische objecten sterk lijkt op die van de bovenste laag in je model file. De fysische objecten doen immers dienst als benadering van deze lichaamsdelen. Je zou deze gelijkenis kunnen uitbuiten om de fysische objecten automatisch te genereren.

Voor elk object moeten er nog wel enkele fysische eigenschappen bepaald worden. Deze zouden uit een file ingelezen kunnen worden, maar dit lijkt ons enkel grote voordelen te bieden bij gespecialiseerde modellen, waarbij de massadichtheid enorm verschilt en van belang is. Bij gewone modellen is het vaak even goed om overal een standaardwaarde te gebruiken.

De voor- en nadelen van deze methode zijn respectievelijk de na- en voordelen van het handmatig genereren: Je hebt minder of geen extra werk per model (dit is afhankelijk van het al dan niet gebruiken van standaardwaardes voor fysische eigenschappen), maar je hebt geen

controle over de toegekende eigenschappen. Je zou natuurlijk een standaardwaarde file kunnen laten genereren waarin je dan later eventuele aanpassingen zou kunnen doen. Op deze manier heb je geen extra werk, tenzij je ondervindt dat de standaardwaardes niet voldoen voor een bepaald model (we hebben deze methode niet gebruikt bij objecten, maar wel bij joints, zie 7.2.1).

In de implementatie is er gekozen om de objecten automatisch te genereren om zoveel mogelijk te voorkomen dat er voor elk model nieuw werk verricht zou moeten worden.

De objecten zouden zo goed mogelijk de vormen van het model moeten benaderen zonder te ingewikkeld te zijn om nog collision detection en fysische simulatie mee uit te voeren. Elk lichaamsdeel zou benaderd kunnen worden door een eenvoudige capsule. Dit is zeer efficiënt, maar vormt geen precieze benadering voor de echte vormen. Vaak is de breedte van de capsule niet zo gemakkelijk te schatten, en die breedte kan ook veel variëren, zelfs binnen eenzelfde deel van een ledemaat (Het onderbeen is bijvoorbeeld meestal een stuk breder aan de knie dan aan de enkel).

Hier is er gekozen om de objecten te genereren als de convexe[1] hull van de vertices die bewogen worden door een bone. Deze keuze is gemaakt om verschillende redenen:

- We gaan er van uit dat een bone geen vertices beweegt die niet tot zijn lichaamsdeel behoren, maar dit lijkt een redelijk aanvaardbare veronderstelling.
- Ageia PhysX voorziet zelf de mogelijkheid om een convexe hull te genereren uit een verzameling punten.
- Een convex object zou een redelijk goede benadering moeten zijn voor alle lichaamsdelen (toch zeker zolang het model menselijk is).
- Convexe objecten kunnen redelijk complex worden, maar blijven efficiënt om collision detection mee uit te voeren.

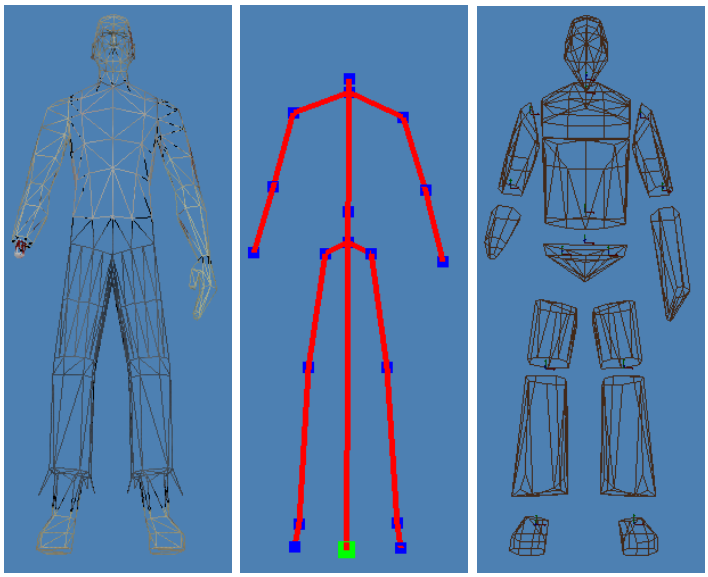
In deze implementatie wordt geen rekening gehouden met het feit dat een vertex beïnvloed kan worden door meerdere bones. In het model dat gebruikt wordt, is elke vertex maar aan één enkele bone gekoppeld (zie 6.2).

Moest men modellen willen toevoegen waarbij dit toch het geval is, zijn er een aantal mogelijkheden om dit probleem op te lossen:

- Elke vertex toekennen aan de bone die hem het meeste beïnvloedt.
- Elke vertex toekennen aan alle bones die hem beïnvloeden.

De eerste mogelijkheid zorgt dat de punten normaal gezien toegekend worden aan de objecten waartoe ze behoren. Als er twee punten gelijk beïnvloed worden, kunnen we het punt negeren, een willekeurige keuze maken of overstappen op de tweede mogelijkheid.

[1] Dit wil zeggen dat als je een lijn trekt tussen twee willekeurige punten binnen het object, dat je dan ook binnen het object blijft. Een bol is een convex object. Een hol object of een object met een deuk erin is niet convex.



Figuur 7.1: zombie model: a) polygonen b) skelet c) fysische objecten

Bij de tweede mogelijkheid moet je wel opletten, omdat je dan objecten genereert die elkaar snijden en dit kan tot vreemde situaties leiden als de simulatie eenmaal begint. Vaak worden deze objecten dan onmiddellijk met grote kracht uit elkaar gesmeten. Het is dan het beste om bij de creatie te specifiëren dat de objecten die vertices delen, niet met elkaar kunnen botsen in de simulatie. Ageia PhysX voorziet hiervoor mogelijkheden in de vorm van collision groups: groepen van objecten waarvoor je de groepen van objecten kan aangeven waarmee ze botsen. De onderlinge constraints tussen zulke objecten moeten dan wel volledig bepaald zijn door de joints tussen de objecten. De joints mogen de objecten elkaar niet (te ver) laten snijden.

De fysische objecten worden weergegeven in figuur 7.1. In de figuur zijn het model (weergegeven in polygonen) en het skelet uit figuur 6.1 herhaald om de overeenkomst met de fysische objecten duidelijker te kunnen zien.

In de figuur merken we ook dat er zich tussen de fysische objecten gaten bevinden. Deze gaten zijn nodig om de objecten te kunnen laten draaien ten opzichte van elkaar. Het gevolg hiervan kan wle nadelig zijn. Het is immers mogelijk dat botsingen met het model niet opgemerkt worden of dat er voorwerpen door het model heengaan omdat er zich geen fysische objecten bevinden. Als dit een probleem is, dan moeten er complexere collision detection proeven gedaan worden, maar in het geval van een ragdoll- of een balanscontroller vormen deze gaten normaal geen probleem.

Als we de objecten met de hand gemaakt hadden, hadden we misschien enkele wijzigingen aangebracht. We zouden waarschijnlijk de uitsteeksels van de broek genegeerd hebben, de onderkant van het hoofd ronder gemaakt hebben en meer detail aan de hand van het object hebben toegevoegd. Als het object aparte bones had gehad om de hand te besturen, zouden deze objecten ook gedetailleerder geweest zijn. Maar ook zonder deze mogelijke aanpassingen zien we toch duidelijk dat de objecten een goede benadering zijn voor het model.

Het 3D-model aanpassen

Nadat de beslissing genomen was om de objecten automatisch te generen, werden er nog een aantal problemen met deze methode ontdekt:

- Er waren bones die geen enkele vertex beïnvloeden. De root is hiervan een speciaal geval.
- Ageia PhysX heeft stabiliteitsproblemen als een object te klein of te smal is.
- Er is geen informatie over fysische eigenschappen van de ledematen zoals dichtheid, wrijving, elasticiteit,...

Bij het eerste geval lijkt het probleem zichzelf op te lossen door het te negeren, maar hierdoor zouden bepaalde bones geen fysische objecten toegewezen krijgen. Hierdoor worden de bones zelf niet bewogen. Dit is geen probleem aangezien deze bones toch geen vertices beïnvloeden, maar de ontbrekende lichaamsdelen zorgen ook voor ontbrekende verbindingen tussen onderdelen. Dit probleem wordt pas echt duidelijk wanneer er tussen al de objecten verbindingen, joints, gemaakt moeten worden (zie 7.2). Maar het resultaat is dat, als er een object ontbreekt, het lichaam niet meer volledig verbonden is en uit elkaar valt tijdens de simulatie.

De root bone is hier een speciaal geval. Deze bone ligt buiten het model (zie 6.2) en heeft geen lichaamsdeel waarmee het overeenkomt. Maar daaruit volgt ook dat het geen bone heeft dat het moet besturen op de bovenliggende laag, en dus is het niet van belang in onze implementatie. We negeren de root bone dan ook, waardoor deze tijdens de animatie op dezelfde plaats blijft. Als het toch gewenst zou zijn om de root bone te verplaatsen met het object (om de positie op te vragen bijvoorbeeld) kan men ervoor zorgen dat de root bone meebeweegt met het eerste fysisch object in de hiërarchie.

In het geval dat er bones zijn die geen vertices beïnvloeden en die ook geen kinderen hebben die vertices beïnvloeden, dan kunnen we deze gewoon negeren, omdat deze geen invloed hebben op de animatie.

Het tweede probleem resulteerde in knieschijven die met enorme snelheid weggeschoten werden. De knieschijven werden voorgesteld met aparte objecten, maar in het model hadden deze enkel vertices die (bijna) op dezelfde hoogte stonden. Dit resulteerde in objecten met een hoogte van (bijna) nul. Als de engine berekeningen uitvoerde voor verbonden objecten met zowel heel grote als heel kleine waardes, resulteerde dit in een precisieprobleem. Dit was de oorzaak van de grote krachten die op de objecten uitgevoerd werden.

Dit en nog een aantal andere probleempjes van de Ageia PhysX engine worden besproken in [melax, 2007].

Het derde probleem is het ontbreken van informatie die essentieel is voor de succesvolle uitvoering van de simulatie.

De eerste twee problemen zijn op te lossen met eenvoudige aanpassingen aan het model en dat is de aanpak die hier gekozen is. Door de vertices te herverdelen, of door nieuwe vertices toe te voegen, kan het model aangepast worden zodat elk object vertices heeft. Als het fysische object te klein blijkt te zijn, kan het object geschaleerd worden waardoor de vertices verder uit elkaar liggen.

Om de implementatie te doen werken voor elk model zonder aanpassingen, zijn de volgende aanpassingen mogelijk:

- Je kan, als er geen vertices zijn, toch generieke capsules gebruiken om de bones voor te stellen. Je hebt de begin- en eindposities van de bones, en daar tussen kan je een capsule van een bepaalde breedte zetten. Die breedte zou je kunnen bepalen aan de hand van een ratio van de hoogte. De kans dat dit geen goede benadering geeft voor het bone is groot. De kans is echter wel groot dat de bones zonder vertices maar klein zijn, en deze zullen dus vaak te kleine capsules worden.
- Je kan van de gewone methode gebruik maken, maar dan voor elk object een groottetest invoeren. Als het te klein is, kan je een object maken dat geen botsingen veroorzaakt, maar dat enkel dient om er joints aan vast te maken. Dit geeft natuurlijk problemen als er veel bones te klein zijn.
- Je kan de physics engine verbeteren. Dit was in dit geval geen optie omdat de physics engine niet open source is. Ook is het mogelijk dat dit een noodzakelijk kwaad is om de nodige berekeningen real-time te kunnen doen.

Het derde probleem is eenvoudig op te lossen door, zoals eerder vermeld, voor alle modellen standaardwaardes te nemen voor elk object. Als je echter specifiekere waardes wil, is het misschien beter om de objecten handmatig te maken.

We hebben nu fysische objecten voor de lichaamsdelen. Dit is een eerste stap in de goede richting. Als we ons programma nu zouden testen, zou het model ineenzakken en in verschillende stukken uiteenvallen (hoewel het toch door de bovenliggende lagen verbonden getekend wordt, wat resulteert in zeer lelijke en voornamelijk onrealistische animatie). Dit is logisch aangezien we de lichaamsdelen nu fysisch simuleren als aparte objecten zonder enige constraint buiten onderling botsen. De nodige constraints gaan we nu toevoegen in de vorm van joints.

7.2 Het aanmaken van fysische joints

Zoals de fysische objecten de voorstelling van de lichaamsdelen zijn op een lagere laag (zie 7.1), zo zijn de joints de voorstelling van de gewrichten van het model. In tegenstelling tot de objecten hebben de joints geen tegenhanger op de skeletlaag of een bovenliggende laag. De joints stellen eigenlijk een aantal constraints voor die nodig (maar niet noodzakelijk voldoende) zijn om de animatie realistisch te maken. Ze hebben de belangrijke taak om de beweging van de fysische objecten te beperken door ze bij elkaar te houden en ook niet te ver te laten draaien (zie 7.2.1). Ze kunnen ook krachten uitvoeren op de objecten die ze met elkaar verbinden.

De joints beperken dus de mogelijke bewegingen die de objecten kunnen uitvoeren. Ze doen dit door krachten uit te oefenen op de objecten, dit zijn de zogenaamde interne krachten. Hiertoe behoren ook de krachten die door eventuele controllers op de joints gezet worden om de objecten in beweging te brengen.

Een joint is dus een verzameling constraints tussen twee fysische objecten waarvan de bones ouder en kind zijn. Bij het automatisch genereren van de fysische objecten (zie 7.1.2) werd vermeld dat er een probleem was waardoor er bepaalde fysische objecten niet aangemaakt werden. Dit was alleenstaand geen probleem, alle bones die bewogen moesten worden, werden ook bewogen. Enkel wanneer de joints toegevoegd werden, werd het duidelijk dat er iets mis was. Omdat er bepaalde objecten ontbraken, konden er met die objecten ook geen joints aangemaakt worden. Hierdoor werden bepaalde constraints niet correct uitgevoerd: de constraints die de ouder met het kind verbonden waren er niet, en die tussen het kind en zijn kind ook niet.

Dit probleem werd opgelost door het model aan te passen (zie 7.1.2), maar dit had ook gekund door een dummy object te plaatsen voor het kind, dat enkel diende om de constraints door te geven. Ook zou men de constraints van de twee joints naar één joint kunnen omrekenen, maar exacte formules daarvoor zijn niet zo eenvoudig.

Je hebt verschillende soorten joints, afhankelijk van de vrijheidsgraden waarin ze beweging toelaten:

- Fixed joint, dit joint laat geen enkele beweging toe.
- Hinge joint, dit joint laat rotatie in één richting toe zoals een scharnier.
- Ball-and-Socket joint, dit joint laat rotatie in twee richtingen toe.
- Slider joint, dit laat beweging langs een lijn toe.
- 6 degree-of-freedom joints, dit type van joints laat alle bewegingen toe.

Op deze joints kan je nog beperkingen toevoegen in welke mate ze beweging toestaan in hun verschillende vrijheidsgraden. Als je het 6 degree-of-freedom joint bekijkt, merk je dat het mogelijk is de andere soorten van joints te simuleren met behulp van een 6 degree-of-freedom joint door op de juiste vrijheidsgraden strikte beperkingen toe te voegen. Binnen Ageia zelf is er ook een tendens om meer en meer 6 degree-of-freedom joints te gebruiken om zo de andere joints te bekomen. Wij hebben ook voor deze aanpak gekozen om te vermijden dat we bij elke joint zouden moeten opgeven welk type hij is.

We hebben dus overal een standaard 6 degree-of-freedom joint (de NxD6Joint klasse), maar hierop moeten we nog de beperkingen aangeven.

7.2.1 Joint limits

De beperkingen die joints opleggen worden joint limits genoemd. Men denkt hierbij voornamelijk aan de rotatiebeperkingen die een joint oplegt, zoals de beperkingen van een gewricht. Je elleboog kan bijvoorbeeld je arm meestal niet verder laten gaan dan gestrekt. De joint limits worden echter aangegeven in alle vrijheidsgraden, dus er moet ook rekening gehouden worden met translaties. Menselijke gewrichten laten meestal geen of verwaarloosbare translatie toe. Het is dus bij deze toepassing vaak mogelijk om translaties te negeren.

In wat volgt bespreken we de implementatiespecifieke manier waarop deze joint limits geïmplementeerd zijn, namelijk limit planes.

Limit planes

In Ageia PhysX [Ageia PhysX, 2006b] is het mogelijk om de beperkingen rechtstreeks aan de joint toe te kennen, maar dit heeft enkele belangrijke nadelen. Het belangrijkste is dat op één vrijheidsgraad na, de beperkingen enkel symmetrisch opgegeven kunnen worden. Men kan bijvoorbeeld enkel zeggen dat een joint dertig graden mag afwijken. Hierdoor zou het model zo gezet moeten worden dat het voor alle vrijheidsgraden bij elk gewricht in het midden van zijn bereik staat. Als we het kunnen vermijden, willen we echter het model niet aanpassen, dus willen we een andere oplossing.

Deze oplossing werd gevonden in limit planes. Een limit plane is een vlak in de 3D-ruimte waarvoor je een beperking instelt dat een gekozen punt altijd aan dezelfde kant moet blijven liggen. Dit punt noemen we het limit point. Als je een dergelijk vlak bevestigt aan het object van de ouder en een dergelijk punt aan het object van het kind, kan je de bewegingen van het kind laten beperken door het vlak.

Als je je de meest eenvoudige implementatie hiervan indenkt, neem je als limit point een punt in het kindobject, gelegen op de as die gevormd wordt door een rechte te trekken tussen de posities van het ouder- en het kindobject. De benodigde limit planes vorm je dan door een vlak te nemen dat loodrecht staat op deze as en door de positie van de ouder gaat. Dit vlak roteer je dan rond dat punt tot het de gewenste beperking oplegt.

Voorbeeld Als we bij de elleboog de pols als limit point kiezen, willen we een vlak plaatsen dat evenwijdig loopt met onze bovenarm. Op deze manier kan de elleboog niet verder gaan dan wanneer de arm volledig gestrekt is.

Dit biedt ons een oplossing voor twee van de zes vrijheidsgraden. We hebben gekozen de translatievrijheidsgraden vast te leggen omdat deze normaal nauwelijks invloed hebben op een model en omdat deze de implementatie met limit planes zouden bemoeilijken. De laatste vrijheidsgraad is de rotatie rond de as die gevormd wordt door een rechte te trekken tussen de posities van het ouder- en het kindobject. Met het limit point gekozen zoals hierboven, valt dit niet op te lossen met limit planes, aangezien de rotatie rond die as de positie van het limit point ongemoeid laat. We zouden dit kunnen oplossen door het limit point te verplaatsen langs die as, maar dat zou de implementatie van de limit planes veel ingewikkelder maken. En omdat dit de vrijheidsgraad is die eerder vermeld werd, waarbij de symmetrie niet noodzakelijk was, is er dan ook geen reden om de rotatie niet in te stellen door het gewoon aan de joint toe te kennen.

Onze keuze van de plaatsing van het limit point en de limit planes heeft nog een ander gevolg: Een rotatie is altijd ofwel helemaal vrij (geen limit plane in die richting) ofwel is ze beperkt tot hoogstens 180 graden. Dit wordt veroorzaakt door het vlak dat zich vanuit de positie van de ouder in twee richtingen uitstrekt, waar we eigenlijk liever een halfvlak zouden willen. Het is weer mogelijk om dit op te lossen door het punt en de vlakken op een ingewikkeldere manier te plaatsen, maar er is hier voor de eenvoudigere oplossing gekozen. In de implementatie heeft een joint dus hoogstens 180 graden rotationele beweging.

Als de physics engine aangepast zou kunnen worden, zou het toevoegen van half limit planes een handige uitbreiding zijn, maar hier is ook geen eenvoudige uitvoering voor. Hierdoor wordt de rotatie wel nog altijd beperkt tot hoogstens 360 graden of volledig vrij, maar dit zou voor de meeste modellen meer dan voldoende moeten zijn. Het zou ook handig zijn als

je meerdere limit points zou kunnen toevoegen, elk met hun eigen vlakken. Dit zou toelaten om de verschillende vrijheidsgraden eenvoudiger met elkaar te combineren.

Ook bij het aanmaken van joints hebben we de keuze tussen automatisch genereren of handmatig aanmaken. Voor het automatisch genereren hebben we wel nog extra informatie nodig (zie 7.2.3), maar we kunnen deze op een andere manier verstrekken dan handmatig een file invullen.

7.2.2 Handmatig joints maken

Je kan voor elke joint voor elke vrijheidsgraad een minimum- en maximumwaarde in een file schrijven en deze dan inlezen en gebruiken. Dit is de aanpak die in deze implementatie gekozen is. Je moet er hierbij wel voor zorgen dat de joint van het model zich al tussen deze waardes bevindt. Je kan dit controleren met het model of je kan afspreken om de huidige positie van het model als nulwaarde voor alle vrijheidsgraden te gebruiken, dan moet je enkel nagaan of de minimum- en maximumwaarde respectievelijk negatief en positief zijn.

In deze implementatie is een eigen xml notatie gebruikt voor deze informatie en die is van de volgende vorm:

```
<?xml version="1.0" ?>
<!-- possible tags for joints are: -->
<!-- name (string) -->
<!-- minRotX maxRotX minRotY maxRotY minRotZ maxRotZ (string "none" or a
 float) -->
<!-- minTransX maxTransX minTransY maxTransY minTransZ maxTransZ (string
 "none" or a float) -->
<joint name="LeftShoulder" minRotX = "0" maxRotX = "0" minRotY = "0" maxRotY
 = "0" minRotZ = "0" maxRotZ = "0" >

        // any children of the joint are put here

</joint>
```

Als een model voor het eerst gebruikt wordt, en er is geen xml file aanwezig voor de joint limits, zal het programma een file schrijven die voor alle joints alle rotaties toelaat, maar geen translaties. Deze file kan dan nadien aangepast worden als men restricties wil opleggen, maar men moet niet de volledige xml structuur zelf uitschrijven.

De namen van de joints van het model zijn aangepast zodat deze intuïtief te gebruiken waren in de implementatie. Als men andere modellen toevoegt, moet men ofwel de namen aanpassen zodat deze overeenkomen met die in de implementatie ofwel een file maken die de overeenkomst tussen de modelnamen en de implementatienamen aangeeft.

De informatie die gebruikt is voor onze joint limits werd oorspronkelijk uit de biomechanische literatuur gehaald. Maar deze joint limits zijn uiteindelijk veel vereenvoudigd omdat de testen erdoor te ingewikkeld werden, en de eenvoudigere limits leverden realistischer ogende animaties op.

7.2.3 Automatisch joints genereren

Je kan joints ook in zekere mate automatisch genereren. In plaats van handmatig files in te vullen, zou je de benodigde informatie ook uit een representatieve animatie kunnen halen. Met een representatieve animatie bedoelen we een animatie waarin het volledige bereik van elke joint gedemonstreerd wordt. Als de animatie aan deze voorwaarde voldoet kunnen we gewoon bij elk joint nagaan wat de minimum- en maximumwaarde voor elke vrijheidsgraad is in de animatie, en hieruit onze joints genereren.

Het is hier nog altijd nodig de vereiste informatie te geven, maar dat kan op een manier gebeuren die voor sommigen eenvoudiger is, dan het invullen met waardes van files. Ook zouden er reeds een aantal animaties voor een model kunnen bestaan, en dan zou er op deze manier veel werk uitgespaard kunnen worden.

Ook hier zouden we met behulp van bestaande animaties een initiële file kunnen maken en deze daarna eventueel met de hand aanpassen.

Nu hebben we de fysische objecten die we nodig hebben; hiermee hebben we in principe al een ragdollcontroller (zie 3.2) gemaakt. In de verdere hoofdstukken bespreken we hoe we aan deze simulatie extra controle toevoegen om zo ingewikkeldere animaties te maken.

Hoofdstuk 8

Het maken van dynamische controllers

In het vorige hoofdstuk hebben we ervoor gezorgd dat we de bouwstenen hebben om controllers te bouwen. Deze twee bouwstenen waren fysische objecten en joints om deze objecten bijeen te houden en te bewegen. Hier breiden we de implementatie uit door op deze objecten krachten toe te passen. Dit kunnen we vergelijken met de spieren van het menselijk lichaam die de gewrichten in beweging brengen.

In dit hoofdstuk zullen we met deze bouwstenen twee controllers maken, een ragdollcontroller (zie 8.3) en een balanscontroller (zie 8.4). We hebben ook een overkoepelende controller gemaakt die beslist welke controller het model controleert (zie 8.2).

8.1 Plugin systeem

De implementatie is geschreven als een plugin in het ExtReAM framework (zie 6.3). De plugin bestaat uit de fysische simulatie en de overkoepelende controller. De aparte subcontrollers vormen elk op zich afzonderlijke plugins om het programma eenvoudig te kunnen uitbreiden met nieuwe controllers. Hoe men deze uitbreiding kan uitvoeren, wordt behandeld in hoofdstuk 9, waarin de aanmaak van nieuwe plugins en nieuwe controllers beschreven wordt (9.2 en 9.3 respectievelijk).

Elke aparte controller is een plugin in het ExtReAM framework. Hierdoor kan de controller, als hij gebruik wil maken van feedback, het model aanspreken dat aan de controller doorgegeven wordt (zie 4.2.1), of hij kan alle gewenste informatie rechtstreeks aan het hoofdprogramma opvragen.

Deze plugins moeten bij het openen van een scène ingeladen worden. Om dit te bekomen wordt er voor elke scène een xml file gemaakt, waarin de op te roepen controllers ingevuld worden (zie hoofdstuk 9). In deze xml file moeten dus de overkoepelende controller en alle kindcontrollers die we willen gebruiken toegevoegd worden. De kindcontrollers moeten wel na de overkoepelende controller voorkomen: als een kindcontroller aangemaakt wordt, zal deze proberen zich te registreren bij de overkoepelende controller, en als deze er nog niet is, zal dit dus falen.

Het ExtReAM framework biedt ook de mogelijkheid om in de code aan te geven dat een plugin afhankelijk is van een andere plugin. Deze wordt dan automatisch geladen. Als men dit doet, is het dus niet noodzakelijk de overkoepelende plugin te vermelden in de xml file.

8.2 Overkoepelende controller

De overkoepelende controller is geschreven naar het model uit 4.2. De hoofdcontroller maakt in deze implementatie geen gebruik van postcondities en gaat er van uit dat de controllers altijd het best mogelijke voor het model proberen uit te voeren (de redenen hiervoor zijn besproken in 4.2.1).

De ragdollcontroller in 8.3 kan men gebruiken als de default controller beschreven in 4.2.2, omdat deze altijd een geldige animatie voor het model oplevert.

We hebben in de implementatie wel een mogelijkheid voorzien waarmee de gebruiker de biedwaardes (zie 4.2.2) van de aparte controllers kan aanpassen. Dit is gebeurd om testredenen, want eigenlijk wilden we voor de gebruiker deze mogelijkheid niet voorzien omwille van de afscherming (zie 4.2.1).

De overkoepelende controller heeft een aantal extra verantwoordelijkheden toegewezen gekregen, waaronder het aanmaken van de fysische objecten en joints (zie hoofdstuk 7). Ook het aanroepen van de fysische simulatie behoort tot deze verantwoordelijkheden.

De overkoepelende controller moet ook nieuwe subcontrollers toelaten zich te registreren en zich aan de verzameling van mogelijke controllers toe te voegen. Eventueel kan de mogelijkheid voorzien worden om controllers te onregistreren, maar we gaan er hier van uit dat de verzameling mogelijke controllers niet tijdens de uitvoering aangepast kan worden. Dit kan eventueel handig zijn als er een serieuze verandering van situatie is, maar tenzij er veel controllers verwijderd moeten worden, kan dit aangepakt worden door het gewicht van die controllers op 0 te zetten.

8.3 Ragdollcontroller

We hebben reeds al de nodige bouwstenen om een ragdollcontroller zoals beschreven in 3.2 te maken. Al wat we moeten doen, is de fysische simulatie uitvoeren op de fysische objecten en joints die we gemaakt hebben in respectievelijk 7.1 en 7.2. Als er door de controller geen krachten gezet worden op de joints, dan zal het model ineenzakken, maar door de beperkingen die de joints leggen, kan het model niet uit elkaar vallen en wordt de animatie een beetje realistisch gehouden. Het model valt dus krachteloos naar beneden net als een ragdoll, en dit is net wat we wilden bekomen.

Het is bij deze specifieke controller onnodig om na te gaan of hij gefaald heeft of succesvol beëindigd is. De controller zal altijd succesvol kunnen blijven lopen. Het is dus het beste om altijd uit te kijken naar een controller die meer biedt voor de controle over het model, ook al faalt deze controller niet in de strikte zin.

In het geval dat de overkoepelende controller een biedronde start, moet de ragdollcontroller altijd een positieve waarde terug sturen, maar dit moet wel een kleine waarde zijn. De waarde

moet positief zijn, omdat de controller elke situatie aankan; ze mag echter niet te groot zijn, omdat bijna elke andere controller, als deze de situatie aankan, nuttiger zal zijn dan de ragdollcontroller.

Als we een controller vinden die meer biedt voor de controle over het model, moeten we er de controle aan overdragen. Meer specifieke informatie hierover vind je in 8.4.5.

8.4 Balanscontroller

Vervolgens hebben we een balanscontroller geïmplementeerd zoals deze beschreven is in 4.3. De balanscontroller heeft verschillende taken en deze worden hier één voor één besproken. Eerst en vooral moet hij er voor zorgen dat het model niet gewoon ineenzakt zoals het model van de ragdollcontroller dat wel doet. Hoe dit wordt bekomen, wordt beschreven in 8.4.1.

De belangrijkste verantwoordelijkheid van de balanscontroller is natuurlijk er voor te zorgen dat het model in evenwicht blijf terwijl het zijn algemene houding zo goed mogelijk bewaart. Dit wordt besproken in 8.4.2.

Het is ook nodig dat de controller kan nagaan of hij succesvol aan het uitvoeren is en of hij succesvol beëindigd is. Maar aangezien het doel van deze controller is het model in een bepaalde staat te houden, zal deze controller nooit succesvol eindigen. Net als bij de ragdollcontroller (zie 8.3), gaan we hier dus best altijd na of er een controller is die meer biedt voor de controle over het model. We moeten dus enkel nog nagaan of hij faalt. Hoe we dit doen wordt uitgelegd in 8.4.3.

8.4.1 Voorkomen dat het model ineenzakt

We gaan er hier van uit dat, als de controller geactiveerd wordt, het model zich in een balanspositie bevindt en dat het de enige taak van de controller is om deze balans te bewaren. We kunnen hier van uitgaan omdat anders de controller niet geboden zou hebben in de biedronde van de hoofdcontroller (zie 8.4.4).

Om ervoor te zorgen dat het model niet gewoon ineenzakt, worden er krachten op alle joints gezet die de objecten duwen naar de positie waarin ze stonden toen de controller activeerde. Deze krachten worden zo klein mogelijk gemaakt zodat ze toch de positie bewaren, maar het model niet onnodig stijf maken, moest er een onverwachte interactie met de omgeving zijn.

Als de controller een kracht wil zetten op een joint, wordt deze kleine kracht verwijderd omdat deze anders de controller zou kunnen tegenwerken. De controller beslist dan om de joint naar een andere positie te duwen dan de oorspronkelijke positie.

8.4.2 Zorgen dat het model zijn evenwicht behoudt

We hebben bij deze implementatie in de klasse voor de fysische representatie van het model mogelijkheden voorzien om de state, de support polygon en het zwaartepunt op te vragen (zie 4.2.1). Dit zijn de essentiële eigenschappen die in deze en eventuele andere controllers nodig zijn om het werk te kunnen verrichten. De snelheid en versnelling van de state en het zwaartepunt benaderen we met behulp van de voorgaande posities (zie 3.4.1).

Telkens als de controller aangeroepen wordt, bepalen we de positie van het centrum van de support polygon en de positie van de projectie van het zwaartepunt op het grondoppervlak. We bepalen de afstand tussen deze twee punten en aan de hand hiervan kiezen we wat we doen:

- Als deze afstand te groot is voor de controller (zie 8.4.3), sturen we terug dat de controller gefaald heeft.
- Als de afstand tussen de twee punten niet te groot is, zetten we een kracht op de enkels die het zwaartepunt in de richting van het centrum van de support polygon drijft.
- Als deze afstand bijna nul is, zetten we op de enkels een kracht gelijkaardig aan de krachten gebruikt in 8.4.1 om de huidige positie (een evenwichtspositie) te bewaren.

Deze krachten zijn voorlopig nog constant, maar hiermee is het risico groot dat het model het doel voorbijschiet. Een eenvoudige uitbreiding zou zijn om deze krachten te laten afhangen van de afstand tussen het zwaartepunt en zijn gewenste positie. Het kan hier wel nodig zijn om na te gaan of je het model geen onmenselijk grote krachten laat gebruiken.

Dit is een zeer eenvoudige invulling van een balanscontroller, want er wordt geen rekening gehouden met snelheid of versnelling, enkel met positie. Dit is evenzeer waar voor het bepalen of de controller faalt en bij het bieden (zie 8.4.3 en 8.4.4).

Er zijn ook nog andere strategieën die een controller zou kunnen gebruiken om de balans van het model te bewaren. Onder andere kan het model bewegen met de heupen of de armen uitwerpen om zijn balans te bewaren. Als het mogelijk is kan het model ook nog een voet verzetten om te voorkomen dat hij valt (zie 4.3).

8.4.3 Bepalen of de controller faalt

We hebben enkele tests uitgevoerd waarvan de resultaten beschreven worden in hoofdstuk 10. Bij deze tests hebben we ook nagegaan hoe ver de projectie van het zwaartepunt op het grondoppervlak kon afwijken van het midden van de support polygon, voor het model zijn balans verloor en de controller dit niet meer kon verhelpen.

Als we willen bepalen wanneer de controller nuttig werk verricht of wanneer hij gefaald heeft, controleren we bij elke aanroep van de controller of de huidige afstand deze waarde overschrijdt. Als dit zo is, stuurt de controller een fail message terug.

8.4.4 Bieden voor de controle

De balanscontroller zal enkel bieden voor de controle over het model, als hij denkt het model in evenwicht te kunnen houden. Om dit na te gaan voeren we dezelfde test uit als degene die we gebruiken om na te kijken of de controller gefaald heeft (zie 8.4.3). Als deze test aantoont dat we de balans kunnen bewaren, sturen we het vaste gewicht terug, anders sturen we 0 terug. Het vaste gewicht moet een positieve waarde zijn, en in dit geval moet deze waarde groter zijn dan die van de ragdollcontroller. Pas als deze controller faalt, willen we dat de ragdollcontroller het overneemt.

8.4.5 Wisselen tussen controllers

Om te voorkomen dat het model ineenzakt werd er in 8.4.1 aan elke joint een kracht toegevoegd die de joint duwde. In Ageia PhysX (zie 6.1) worden deze krachten aan een joint toegevoegd waarna deze in elke tijdstap uitgevoerd worden. Dit zou geen enkel probleem vormen als er elke tijdstap nieuwe krachten gezet zouden moeten worden, maar deze krachten herhalen automatisch.

Het probleem is dat dit niet gewenst is voor elke controller. Elke controller gaat er van uit dat het model niet automatisch krachten zet, en daarom dienen we deze krachten ook te verwijderen bij het afsluiten van de controller. De controller kan dit echter niet zelf doen omdat er ook een biedronde kan gestart worden zonder dat hij dit weet (bijvoorbeeld door gebruikersinput, zie 4.2.3).

Om dit toch te kunnen verwezenlijken is er in de implementatie aan de controller interface een functie deactivate() toegevoegd, die door de hoofdcontroller aangeroepen wordt op de oude controller, als hij de controle aan een andere controller doorgeeft.

Hier beëindigen we de bespreking van het aanmaken van de twee bestaande controllers. In het volgende hoofdstuk wordt beschreven hoe er nieuwe controllers aangemaakt kunnen worden.

Hoofdstuk 9

Gebruik van de implementatie

Er is in deze implementatie veel tijd besteed aan het idee van uitbreidbaarheid. Zowel het plugin systeem van het ExtReAM framework (zie 6.3) als het idee van subcontrollers uit te wisselen en te integreren in één controller ([Faloutsos et al., 2001a], [Faloutsos et al., 2001b]) moedigden dit idee aan.

Dit is implementatiegewijs zeer eenvoudig, zoals beschreven in 9.2 en 9.3. De moeilijkheid ligt nog voornamelijk in het berekenen van de nodige krachten en het uitzoeken op welke momenten deze krachten uitgevoerd moeten worden. Dit kan een ingewikkelde en langdurige taak zijn.

9.1 Opstarten en gebruik van plugins in het ExtReAM framework

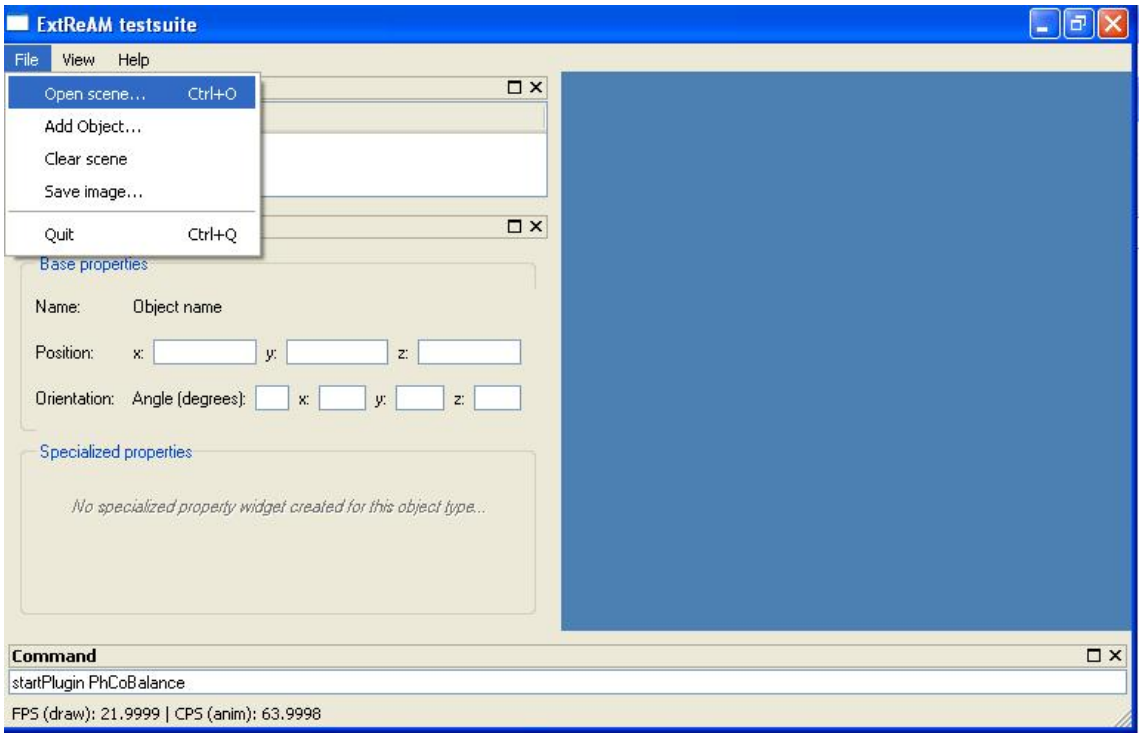
Na het opstarten van het ExtReAM framework (zie 6.3) zie je een venster zoals dat in figuur 9.1. Om een nieuwe scène te openen kun je in het 'File' menu 'Open scene' openen en daar kan je dan een scène selecteren. De scènes worden opgeslagen in een xml file, waarin ook de plugins die erin gebruikt worden, opgeslagen zijn.

Als er een scène geopend is, kunnen er ook nieuwe plugins gestart worden met commando's in de command bar onderaan het scherm. Hierin kunnen ook commando's ingegeven worden voor plugins die geregistreerd hebben dat ze deze commando's kunnen verwerken.

9.2 Het aanmaken van nieuwe plugins

Als een gebruiker nieuwe controllers zou willen maken voor deze implementatie, is dat relatief eenvoudig. Er zijn al verschillende zaken waar de gebruiker geen rekening meer mee moet houden:

- Fysische objecten en joints voor het model moeten niet meer aangemaakt worden, want deze zijn reeds aanwezig.



Figuur 9.1: Een nieuwe scène openen in het ExtReAM framework

- Op deze objecten en joints wordt de fysische simulatie al uitgevoerd met behulp van de Ageia PhysX engine (zie 6.1).
- Het uittekenen gebeurt al met behulp van het ExtReAM framework (zie 6.3). Dit framework voorziet ook al gebruikersinvoer.

Het enige dat de gebruiker nog moet doen, is een nieuwe controller als plugin voor het framework aanmaken, en deze moet op de juiste momenten krachten zetten op de joints.

Om een nieuwe plugin aan het ExtReAM framework toe te voegen moeten er een aantal klasses geschreven worden. Deze klasses zijn onder andere verantwoordelijk voor het opstarten en afsluiten van de plugin en eventueel voor het ontvangen van gebruikersinput. Voor deze klasses zijn er templates voorzien bij het framework om de aanmaak van nieuwe plugins te vergemakkelijken.

Vanzelfsprekend is er ook een zekere kennis van programmeren met C++ nodig om dit te kunnen verwezenlijken.

Er zijn twee manieren om een controller te gebruiken in het framework. De eenvoudigste is om deze in de xml files van de scènes waar je hem wil gebruiken te zetten. Deze xml file ziet eruit als volgt:

```
<?xml version="1.0"?>
<!DOCTYPE SCENE (View Source for full doctype...)>
<SCENE>
    <!-- Scene specific plugins  -->
    <PLUGINS>
        <PLUGIN name="SkeletonObject" timeToLoad="preload" />
        <PLUGIN name="MilkShapeLoader" timeToLoad="preload" />
        <PLUGIN name="Cal3DLoader" timeToLoad="preload" />
        <PLUGIN name="RigidBodies" timeToLoad="postload" />
        <PLUGIN name="PhysicsController" timeToLoad="postload" />
        <PLUGIN name="PhCoRagdoll" timeToLoad="postload" />
        <PLUGIN name="PhCoBalance" timeToLoad="postload" />

        <!-- ADD NEW PLUGINS HERE  -->

    </PLUGINS>
    <DATAPATH path="../data" />

    <!-- Objects of the scene  -->
    <OBJECT ... />

</SCENE>
```

De nieuwe controller moet in deze file bij de andere plugins toegevoegd worden. Hij moet zeker na de PhysicsController (dit is de overkoepelende controller) komen met als timeToLoad attribuut "postload"[1]. Als aangegeven is dat de controller afhankelijk is van de PhysicsController, hoeft de PhysicsController niet in deze file te staan.

De andere manier om een controller op te starten is tijdens de uitvoer van het programma het volgende commando aan te roepen: 'startPlugin NEWPLUGIN' met NEWPLUGIN de naam van de nieuwe plugin.

De <OBJECT/> tags in de xml file worden gebruikt om aan te geven welke modellen in de scène ingeladen moeten worden. Dit kunnen personen zijn, maar ook gebouwen of de vloer.

9.3 Het aanmaken van nieuwe controllers

Om van een plugin een subcontroller te maken, is het nodig om een klasse af te leiden van de Controller klasse. De Controller klasse bevindt zich bij de PhysicsController en is een abstracte klasse die sterk lijkt op de interface in 4.2.1.

De PhCoRagdoll controller is een heel eenvoudig voorbeeld, omdat alle functionaliteit van deze controller al voorzien is in de hoofdcontroller. Men kan de klasses uit deze controller overnemen en dan enkel de nagenoeg lege bid() en step() invullen. Als men de files behorende bij deze controller kopieert, moet men er wel zeker aan denken om ook in de files alle voorkomens van 'ragdoll' te vervangen.

[1] Dit attribuut geeft aan of de controller moet opgestart worden voor of na het openen van de scène

Voor een correcte werking moet de biedfunctie voldoen aan de voorwaarden beschreven in 4.2.1. Deze voorwaarden zijn:

- Als de controller de huidige situatie niet denkt aan te kunnen, dan moet de functie nul terugsturen.
- Als de controller de situatie wel aankan, moet hij een positief getal terugsturen. De grootte van het getal geeft aan hoe goed de controller is voor het model in deze situatie.

De step functie moet de nodige krachten aan joints toevoegen om de acties van de controller uit te voeren. De simulatie zelf neemt de hoofdcontroller voor zijn rekening.

Om deze functies te kunnen invullen is een degelijke kennis van Ageia PhysX (zie 6.1) wel vereist. Via de PhysicsSkeleton klasse kan men de objecten en joints (NxActor en NxD6Joint) horende bij het model van de physics engine aanspreken.

Hoofdstuk 10

Resultaten

In dit hoofdstuk proberen we, onder andere met een aantal testresultaten voor de balanscontroller, aan te geven in welke mate de implementatie geslaagd is.

We gaan elk onderdeel van de implementatie bekijken, beginnende met de overkoepelende controller (zie 10.2). Daarna komen de twee subcontrollers, de ragdollcontroller (zie 10.3) en de balanscontroller (zie 10.4), aan de beurt.

10.1 Specificaties

De testen zijn uitgevoerd op een PC met een Intel Pentium 4 processor (1.8GHz) en Windows XP Professional, gebruik makend van:

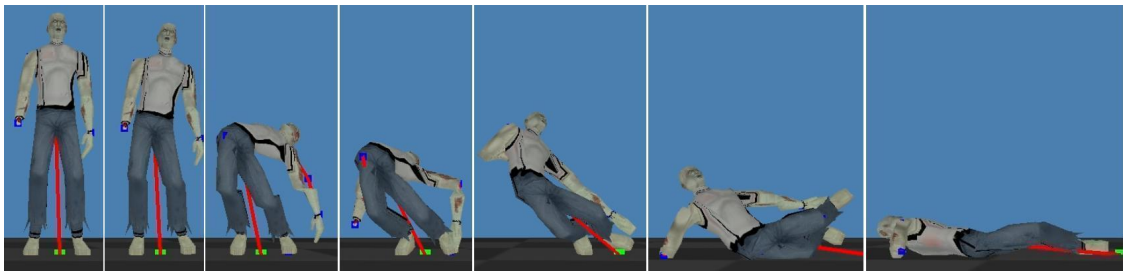
- C++,
- Qt [trolltech, 2006],
- OpenGL [SGI, 2006],
- Ageia PhysX [Ageia PhysX, 2006b]
- Het ExtReAM framework [Jorissen et al., 2005a]

De testen zijn uitgevoerd met het simuleren van één model met 19 joints. Dit model is het model beschreven in 6.2.

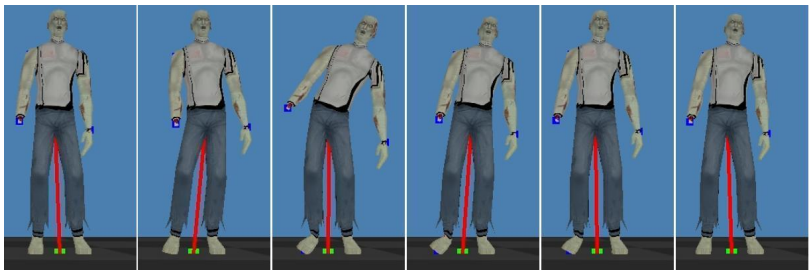
10.2 Overkoepelende controller

De overkoepelende controller geeft gewoon de controle door aan de onderliggende controllers. Het enige waar we rekening mee moeten houden, is dat de biedronde misschien te lang kan duren, zeker als we die telkens moeten uitvoeren als de default controller actief is.

In dit geval leverde dit geen problemen op, maar er waren ook maar twee controllers, en die hadden elk een eenvoudige biedfunctie.



Figuur 10.1: Een voorbeeldanimatie van de ragdollcontroller



Figuur 10.2: Een voorbeeldanimatie van de balanscontroller

De tijd die hiervoor nodig is, stijgt mee met het aantal controllers, maar dat zullen er normaal nooit te veel worden. De programmeurs die controllers implementeren, moeten er alleen rekening mee houden dat de biedfunctie niet te zwaar wordt gemaakt.

10.3 Ragdollcontroller

De uitvoer van de ragdollcontroller bestaat volledig uit de fysische simulatie van objecten en joints. Deze simulatie is dus het enige wat getest zou moeten worden. Aangezien we hiervoor een commercieel programma gebruiken (maar gratis voor niet commercieel gebruik) gingen we er van uit dat dit snel genoeg was en dit bleek ook zo te zijn. Het uitvoeren van het programma gaat perfect in real-time.

Een voorbeeld uitvoering van de ragdollcontroller is te zien in figuur 10.1.

10.4 Balanscontroller

De uitvoer van de controller is nauwelijks zwaarder dan die van de ragdollcontroller. De eenvoudige tests die door de controller zelf worden uitgevoerd, wegen niet op tegen de tijd die nodig is voor de simulatie. De simulatie wordt maar miniem verzwaard door het toevoegen van een aantal krachten aan de joint. Ook hier is het nog perfect mogelijk om alles in real-time uit te voeren.

Een voorbeeld uitvoering van de balanscontroller is te zien in figuur 10.2.

Vervolgens hebben we een aantal testen uitgevoerd om na te gaan hoe effectief de strategie van onze balanscontroller was.

10.4.1 Testresultaten

Als er geen onvoorziene krachten op het model toegepast worden, slaagt de balanscontroller erin het model in balans te houden.

Ageia PhysX [Ageia PhysX, 2006b] werkt zonder vaste eenheden, maar in de implementatie werden de volgende waardes gebruikt:

- Alle objecten hebben de door Ageia PhysX [Ageia PhysX, 2006b] vastgelegde standaard-dichtheid van 1.0.
- De balanscontroller gebruikt krachten die empirisch zijn vastgesteld bij het voorbeeld-model als de beste waardes om stabiliteit te bekomen.

We hebben de testen voor onze balanscontroller gebaseerd op de testen die uitgevoerd worden in [Kudoh, 2001]. We hebben wel geen sinusoïdale krachten toegepast, omdat we hiervoor de voorzieningen niet hadden.

De krachten worden uitgeoefend op het fysisch object dat zich het dichtste bij het zwaartepunt bevindt. De controller begint met corrigeren van de eerste tijdstap waarbij het zwaartepunt buiten een kleine omgeving van het midden van de base of support gaat. Dit is een onrealistisch snel reactievermogen; als dit niet de bedoeling is, kan men in de controller een vertraagde reactie inbouwen. Dit is mogelijk door voor deze test de omgeving te vergroten waarbinnen hij het zwaartepunt als in orde aanvaardt.

De balanscontroller kan relatief kleine krachten zonder problemen corrigeren. Het model bevindt zich na het uitoefenen van deze krachten bijna onmiddellijk terug in een rustpositie. Als we de maximale krachten testen die onze controller aankan is het wel duidelijk merkbaar dat de controller soms zijn doel een aantal keer voorbij schiet en daarna pas tot rust komt in zijn doelpositie.

Hoofdstuk 11

Mogelijke uitbreidingen

De balanscontroller kan nog op een aantal belangrijke manieren verbeterd worden:

- Hij kan rekening houden met de snelheid en versnelling van het zwaartepunt, in plaats van enkel met de positie.
- Er kunnen krachten op de joints gezet worden die variëren naargelang de afstand, in plaats van een vaste kracht.
- Andere balansstrategieën zoals armstrategieën, heupstrategieën of een voet verzetten kunnen toegevoegd worden. Nu wordt er enkel van de enkels gebruik gemaakt.

De andere strategieën van de balanscontroller kunnen eventueel uitgewerkt worden als subcontrollers onder de balanscontroller.

De belangrijkste uitbreiding is natuurlijk het toevoegen van extra controllers. De implementatie is gemaakt met de bedoeling dit zo eenvoudig mogelijk te maken. Met slechts een ragdollcontroller en een balanscontroller is deze implementatie niet erg nuttig. Het is pas als er een voldoende aantal controllers aan toegevoegd wordt, dat het idee van een overkoepelende controller zijn nut bewijst.

Dit zijn de belangrijkste uitbreidingen die nog kunnen gebeuren. Doorheen het gedeelte Implementatie (deel II) van deze thesis zijn er nog andere vermeldingen geweest van uitbreidingen en verbeteringen aan de implementatie:

- We kunnen, als dit gewenst is, de mogelijkheid voorzien om handmatig fysische objecten te maken. Fysische informatie (dichtheid, wrijving, elasticiteit,...) inlezen uit een file is ook een mogelijkheid.
- Er moet bij het handmatig aanmaken van objecten nog rekening gehouden worden met vertices die door verschillende bones beïnvloed worden. We kunnen hier de vertex toekennen aan de bone die het het meeste beïnvloedt. Als dit geen uniek bone oplevert, negeren we gewoon de vertex.
- Objecten die geen collisions veroorzaken, kunnen gemaakt worden als objecten te klein zijn of als er bones zijn die geen vertices beïnvloeden.

- We kunnen ervoor zorgen dat er rotaties van meer dan 180 graden mogelijk zijn, door de limit planes anders te plaatsen. We kunnen ook limit planes toevoegen om de translatievrijheidsgraden te beperken.
- De joint limits kunnen ook automatisch gegenereerd worden uit een representatieve animatie. Deze mogelijkheid kunnen we ook nog toevoegen aan de implementatie.
- We zouden eventueel een expliciete uitwerking van de postcondities kunnen doen, mogelijkerwijs met extra secundaire termen.
- Het onregistreren van controllers bij de hoofdcontroller zou een tijdswinst kunnen opleveren tijdens het biedproces.
- De controllers kunnen aangepast worden zodat de krachten die ze gebruiken afhankelijk zijn van de eigenschappen van het model.

Hoofdstuk 12

Conclusie van de implementatie

Er is bij deze thesis een implementatie gemaakt van een overkoepelende controller en bij deze controller zijn er twee relatief eenvoudige subcontrollers gemaakt om het principe van het samenvoegen van controllers te testen. Deze twee controllers zijn een ragdollcontroller en een balanscontroller:

- De ragdollcontroller laat het model op de grond vallen, hierbij rekening houdend met de constraints die joints op het model leggen.
- De balanscontroller kan het model in balans houden met behulp van krachten op de enkels. De balans blijft bewaard als er geen externe krachten op het model inwerken, en ook nog bij kleine krachten die inwerken op het zwaartepunt van het model.

De overkoepelende controller kan zijn taak, wisselen tussen de beide controllers, succesvol uitvoeren zonder dat er visuele artefacten optreden bij de overgang tussen twee controllers. Er is bijvoorbeeld geen sprake van haperingen van het model tijdens de overgangsfase.

Het aantal controllers is nog niet ideaal, want de kracht van de overkoepelende controller kan met slechts twee controllers niet ten volle gedemonstreerd worden.

De balanscontroller is ook nog niet optimaal. Hij kan het model enkel in balans houden als er slechts kleine krachten uitgeoefend worden en gebruikt maar één van de vele mogelijke balansstrategieën.

Er is dus nog veel ruimte voor verbetering en uitbreiding in de praktische kant van de implementatie, maar enkele belangrijke principiële zaken zijn aangetoond. We denken hier aan de eenvoud van het uitbreiden van het aantal controllers, en het gemak waarmee er tussen deze controllers gewisseld kan worden. Dit zijn essentiële elementen van het principe van de overkoepelende controller en deze elementen zijn goed aanwezig.

Hoofdstuk 13

Algemene conclusie

We maken een onderscheid tussen drie verschillende onderdelen van 3D-computeranimatie: modellering, vervorming en animatie. Modellering wordt gekenmerkt door de wijze waarop het opgeslagen wordt en de manier waarop het geconstrueerd wordt. We merken op dat deze twee vaak met elkaar verbonden zijn. Een scan van een model is vaak een puntenwolk en hieruit kan men eenvoudig een polygonmodel maken. Een handgemaakt model maakt vaak gebruik van parametric surfaces om eenvoudig rondingen te kunnen weergeven. Voordat men deze modellen kan renderen, worden ze vaak nog wel naar polygons omgezet.

Als we een model dat uit veel punten bestaat moeten vervormen, willen we niet elk punt verplaatsen. Hiërarchische skeletmethodes laten toe om het model met behulp van slechts enkele punten aan te passen, zonder hiervoor al te veel controle in te boeten. Deze methode wordt tegenwoordig standaard gebruikt bij de meeste animatietechnieken, die dan de onderste laag, meestal de skeletlaag, besturen.

We onderscheiden twee belangrijke groepen van animatietechnieken: kinematiche en dynamische technieken. Kinematische technieken volgen een voorgeschreven beweging, en dynamische technieken volgen een fysische simulatie.

Kinematische technieken zijn zeer geschikt voor vele applicaties, bijvoorbeeld omdat ze intrinsiek realistische animaties kunnen opleveren. Ze zijn echter niet zo geschikt voor applicaties waarbij het model veel interageert met de omgeving, zeker als er ook onverwachte interactie kan voorkomen.

De applicaties die nood hebben aan interactie kunnen beter gebruik maken van dynamische animatietechnieken. We kunnen gebruik maken van forward en inverse dynamics om het model te bewegen met behulp van krachten. Inverse dynamics kunnen de nodige krachten berekenen om een bepaalde beweging te veroorzaken. Deze krachten kunnen we dan toepassen op het model door gebruik te maken van forward dynamics. Deze methodes zorgen ervoor dat de interactie met de omgeving automatisch mee in rekening gebracht wordt. Dynamische technieken zijn interessant voor virtual reality toepassingen omdat ze deze interactiemogelijkheden ondersteunen. Hun grotere kost qua berekeningen worden steeds haalbaarder dankzij de steeds sterker wordende processoren en de nieuwe physics kaarten. Het hergebruiken van een beweging voor een ander model is echter vaak nog niet zo eenvoudig.

De huidige technieken kunnen eenvoudig als controllers geïmplementeerd worden. Dit biedt verschillende voordelen:

- Mits een aantal duidelijke afspraken over de opbouw van een controller, is het eenvoudig om controllers te delen.
- Een controller kan opgeven en aangeven dat hij de situatie niet aankan. Als dit gebeurt kan de controle over het model overgedragen worden aan een andere controller.
- Het gebruik van feedback laat ingewikkeldere dynamische technieken toe.
- Het principe van controllers laat een vorm van afscherming toe. Een high-level commando kan resulteren in de aanroep van een controller. De controller neemt dan de low-level commando's voor zijn rekening.

Controllers kunnen ook samengevoegd worden om op deze manier één grote controller te vormen. Deze controller kan dan al de bewegingen animeren van zijn onderliggende controllers.

We hebben een methode onderzocht die verschillende subcontrollers op een hiërarchische wijze onderverdeelt onder een overkoepelende controller. De hoofdcontroller zal de controle van het model doorgeven aan de subcontroller die hiervoor het meest geschikt is. Pre- en postcondities bepalen hoe geschikt een controller is, maar het is ingewikkeld om deze expliciet te maken en daarom hebben we dit vermeden. De precondities hebben we vermeden door de subcontrollers deze zelf te laten controleren in hun biedfunctie. Postcondities konden niet vermeden worden. Het enige wat we wel konden doen was deze negeren, maar hier zouden we nog graag een andere oplossing voor vinden.

We hebben een overkoepelende controller geïmplementeerd en hierbij ook twee relatief eenvoudige subcontrollers om het principe van het samenvoegen van controllers te testen. De twee controllers in kwestie zijn een ragdollcontroller en een balanscontroller. De ragdollcontroller laat het model krachteloos neervallen. De balanscontroller oefent krachten uit op de enkels van het model om het model in balans te houden. De controller slaagt er ook in het model in balans te houden als er relatief kleine krachten op het zwaartepunt van het model uitgeoefend worden.

Het wisselen tussen controllers wordt succesvol uitgevoerd zonder dat er hierdoor haperingen in de animatie zichtbaar worden. Er treden geen visuele artefacten op tijdens de overgang.

Er zijn nu slechts twee subcontrollers geïmplementeerd. Dit is verre van ideaal, want de volle kracht van de overkoepelende controller kan pas gedemonstreerd worden als er een grote hoeveelheid controllers is om uit te kiezen.

De balanscontroller kan het model enkel in balans houden zolang de krachten die er op uitgeoefend worden klein zijn. De controller zou nog met andere balansstrategieën uitgebreid moeten worden.

De implementatie is nog niet volledig en kan nog verbeterd en uitgebreid worden. Ze toont wel aan dat het uitbreiden van het aantal controllers eenvoudig is en dat het wisselen tussen deze controllers geen problemen hoeft op te leveren. Deze elementen zijn essentieel voor het principe van de overkoepelende controller en zijn duidelijk in de implementatie aanwezig.

Physically based avatar animation blijkt een techniek te zijn die in de praktijk uitvoerbaar is. Het is mogelijk om verschillende bewegingen door het model te laten uitvoeren met behulp van controllers. De noden van de simulaties worden steeds haalbaarder door sterker wordende processoren en het ontwikkelen van physcis kaarten. Deze technieken zijn niet nuttig in elke applicatie, maar zullen vooral hun nut vinden in virtuele realiteitsomgevingen. In deze omgevingen werkt realistische interactie immersieverhogend. De techniek zou ook kunnen toelaten dat verschillende gebruikers die via een netwerk verbonden zijn, hun modellen in de virtuele omgevingen kunnen laten samenwerken aan één taak.

Bibliografie

[Ageia PhysX, 2006a] Ageia PhysX (2006a). Ageia. http://www.ageia.com/developers/tools.html/. 6.3, 7.1.1

[Ageia PhysX, 2006b] Ageia PhysX (2006b). Ageia PhysX. http://www.ageia.com/. 6.1, 7.2.1, 10.1, 10.4.1

[Anderson, 2001] Anderson, E. F. (2001). Real-time character animation for computer games. 2.3.2, 2.3.4, 2.3.5, 2.3.6, 2.4.3

[Answers.com, 2006a] Answers.com (2006a). 3D model: Information From Answers.com. http://www.answers.com/topic/3d-model. 2.1

[Answers.com, 2006b] Answers.com (2006b). computer animation: Definition and Much More From Answers.com. http://www.answers.com/topic/computer-animation. 2.1

[Autodesk, 2006a] Autodesk (2006a). Autodesk - Autodesk 3ds Max - Product Information. www.autodesk.com/3dsmax. 2.2.1, 7.1.1

[Autodesk, 2006b] Autodesk (2006b). Autodesk - Autodesk Maya - Product Information. www.autodesk.com/maya. 2.2.1, 7.1.1

[Bloomenthal, 2001] Bloomenthal, J. (2001). Implicit surfaces. *Encyclopedia of Computer Science and Technology*. 2.2.2

[Bourke, 1997] Bourke, P. (1997). Implicit surfaces, also known as "metaballs", "blobbies", "soft objects". http://local.wasp.uwa.edu.au/~pbourke/modelling/implicitsurf/. 2.2.2

[Bungie, 2003] Bungie (2003). Bunigie.net: Games: Halo: Xbox. http://www.bungie.net/Games/Halo/. 3.2.1

[Chadwick et al., 1989] Chadwick, J. E., Haumann, D. R., and Parent, R. E. (1989). Layered construction for deformable animated characters. volume 23, pages 243–252. 2.3.8

[chUmbaLum sOft, 2006] chUmbaLum sOft (2006). MilkShape 3D. http://www.milkshape3d.com/. 6.2

[Collins and Hilton, 2001] Collins, G. and Hilton, A. (2001). Models for character animation. 2.2.1, 2.2.1, 2.2.2, 2.2.2, 2.2.2, 2.2.2, 2.3.1, 2.3.2, 2.3.3, 2.3.5, 2.3.6, 2.3.7, 2.3.8, 2.4.6

[Day et al., 1993] Day, B. L., Steiger, M. J., Thompson, P. D., and Marsden, C. D. (1993). effect of vision and stance width on human body motion when standing: implications for afferent control of lateral sway. *the journal of physiology*, pages 479–499.

[Dierckx, 2004] Dierckx, J. (2004). Opencal: Open computer animation library. Master's thesis, Universiteit Hasselt. 3.4, 3.4.1, 3.4.1

[Erich Gamma, 1995] Erich Gamma, Richard Helm, R. J. J. V. (1995). Design patterns: Elements of reusable object-oriented software. 4.2.1

[Faloutsos, 2002] Faloutsos, P. (2002). Composable controllers for physically-based character animation. 3.4.2

[Faloutsos et al., 2001a] Faloutsos, P., van de Panne, M., and Terzopoulos, D. (2001a). Composable controllers for physics-based character animation. In Fiume, E., editor, *SIGGRAPH 2001, Computer Graphics Proceedings*, pages 251–260. ACM Press / ACM SIGGRAPH. 4.2, 4.2.1, 4.2.1, 4.2.2, 4.2.3, 4.3, 9

[Faloutsos et al., 2001b] Faloutsos, P., van de Panne, M., and Terzopoulos, D. (2001b). The virtual stuntman: Dynamic characters with a repertoire of autonomous motor skills. *Computers and Graphics*, 25(6):933–953. 4.1, 4.2, 4.2.1, 4.2.2, 4.3, 9

[Fayyad, 1998] Fayyad, U. (1998). A tutorial on support vector machines for pattern recognition. 1

[Games, 2006a] Games, E. (2006a). Epic games. http://www.epicgames.com/. (document), 3.2

[Games, 2006b] Games, R. (2006b). Rockstar games: Oni. http://www.rockstargames.com/oni/main.html. (document), 3.2, 3.1

[gamesindustry, 2005] gamesindustry (2005). Sony computer entertainment enters into strategic licensing agreemend with ageia. http://www.gamesindustry.biz/content_page.php?aid=10281. 6.1

[Gatev et al., 1999] Gatev, P., Thomas, S., Kepple, T., and Hallett, M. (1999). Feedforward ankle strategy of balance during quiet stance in adults. *the journal of physiology*, pages 915–928.

[Giang et al., 2000] Giang, T., Mooney, R., Peters, C., and O'Sullivan, C. (2000). Real-time character animation techniques. 2.2.2, 2.3.3, 2.3.5, 2.3.7, 2.3.8, 2.4.1, 2.4.2, 2.4.3, 2.4.3, 2.4.4, 2.4.5, 3.1, 3.3

[havok, 2006] havok (2006). Havok. http://www.havok.com. 6.1

[Interactive, 2000] Interactive, I. (2000). Ioi - hitman codename 47. http://www.ioi.dk/games/hitman1.htm. 3.2.1

[J. Auslander, 1994] J. Auslander, A. Fukunaga, H. P. J. C. L. H. P. R. A. S. J. M. J. N. (1994). Towards practical automated motion synthesis. 4.1, 4.2, 4.2.1, 4.2.3

[J. K. Hodgins, 1997] J. K. Hodgins, N. S. P. (1997). Adapting simulated behaviors for new characters. volume 24, pages 153–162. 4.1

[Jakobsen, 2001] Jakobsen, T. (2001). Advanced character physics. 3.2.1, 3.4.3

[Jorissen et al., 2005a] Jorissen, P., Dierckx, J., and Lamotte, W. (2005a). The ExtReAM Library: Extensible Real-time Animations for Multiple Platforms. 6.3, 10.1

[Jorissen et al., 2005b] Jorissen, P., Wijnants, M., and Lamotte, W. (2005b). Dynamic interactions in physically realistic collaborative virtual environments. 4.1, 6.1

[Klosowski et al., 1998] Klosowski, J. T., Held, M., Mitchell, J. S. B., Sowizral, H., and Zikan, K. (1998). Efficient collision detection using bounding volume hierarchies of k-dops. *IEEE Transactions on Visualization and Computer Graphics*, pages 21– 36. 2.2.1

[Kudoh, 2001] Kudoh, S. (2001). The dynamic balance control system for human body model with the quadratic programming method. Master's thesis, The University of Tokyo, Tokyo, Japan. 4.3, 10.4.1

[Kudoh et al., 2003] Kudoh, S., Komura, T., and Ikeuchi, K. (2003). Balance maintenance by stepping for human-like characters against large perturbation.

[Loram and Lakie, 2002] Loram, I. D. and Lakie, M. (2002). Human balancing of an inverted pendulum: position control by small, ballistic-like, throw and catch movements. *the journal of physiology*, pages 1111–1124.

[Mandel, 2004] Mandel, M. J. (2004). Versatile and interactive virtual humans: Hybrid use of data-driven and dynamics-based motion synthesis. Master's thesis, Carnegie Mellon University. 2.4.4, 3.1, 3.2, 3.4, 3.4.2, 4.1, 4.2, 4.2.3, 4.3.1, 6.1

[MAXON Computer GmbH, 2006] MAXON Computer GmbH (2006). MAXON - The makers of CINEMA 4D and BodyPaint 3D. http://www.maxon.net/. 2.2.1

[McKenna and Zeltzer, 1990] McKenna, M. and Zeltzer, D. (1990). Dynamic simulation of autonomous legged locomotion. *Computer Graphics*, 24(4).

[melax, 2007] melax (2007). Novodex 3dsmax tools troubleshooting. http://www.melax.com/3dsmax/trouble/. 7.1.2

[Michael Meredith, 2004] Michael Meredith, S. M. (2004). Using a half-jacobian for real-time inverse kinematics. 2.4.3

[Michiel van de Panne, 1990] Michiel van de Panne, Eugene Fiume, Z. V. (1990). Reusable motion synthesis using state-space controllers. 3.3, 4.2

[Multon et al., 1999] Multon, F., France, L., Cani, M.-P., and Debunne, G. (1999). Computer animation of human walking: a survey. *Journal of Visualization and Computer Animation (JVCA)*, 10:39–54. Published under the name Marie-Paule Cani-Gascuel. 2.3.6, 2.4.2, 2.4.2, 2.4.3, 2.4.3, 3.1, 3.3, 4.1

[Newtek, 2006] Newtek (2006). LightWave® - Go Ahead, Make A Scene. http://www.newtek.com/lightwave/. 2.2.1

[ODE, 2006] ODE (2006). Open Dynamics Engine. http://www.ode.org. 6.1

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

[Psionic, 2006] Psionic (2006). Psionics 3D Game Resources: News. http://www.psionic3d.co.uk/. 6.2

[Reitsma, 2006] Reitsma, P. S. A. (2006). Evaluating data driven character animation. Master's thesis, Carnegie Mellon University. 2.4.2, 2.4.2

[Richard C. Fitzpatrick, 1992] Richard C. Fitzpatrick, Janet L. Taylor, D. I. M. (1992). ankle stifness of standing humans in response to imperceptible perturbation: reflex and task-dependent components. 4.3

[SGI, 2006] SGI (2006). OpenGL, The Industry Standard for High Performance Graphics. http://www.opengl.org/. 6.3, 10.1

[Shapiro et al., 2003] Shapiro, A., Pighin, F. H., and Faloutsos, P. (2003). Hybrid control for interactive character animation. In *Pacific Conference on Computer Graphics and Applications*, pages 455–461. 3.2, 3.2, 4.2, 4.2.1, 4.2.3

[Shen and Thalmann, 1995] Shen, J. and Thalmann, D. (1995). Interactive shape design using metaballs and splines.

[SoftImage, 2006] SoftImage (2006). SOFTIMAGE::Products::XSI. http://www.softimage.com/. 2.2.1

[Thacker, 2005] Thacker, J. (2005). Buyers guide. 3d world. pages 94 – 98. 2.2.1

[trolltech, 2006] trolltech (2006). Qt Homepage - Trolltech. http://www.trolltech.com/products/qt/. 6.3, 10.1

[Turner and Gobbetti,] Turner, R. and Gobbetti, E. Interactive construction and animation of layered elastically deformable characters. 2.2.1

[Victor Ng-Thow-Hing, 2005] Victor Ng-Thow-Hing, P. F. (2005). Dance: Dynamic animation and control environment. www.dgp.toronto.edu/DGP/software/dance/dance.html. 6.3

[W. L. Wooten, 1996] W. L. Wooten, J. K. H. (1996). Animation of human diving. volume 15, pages 3–14. 4.2, 4.3, 4.3.1

[wolfram, 2006a] wolfram (2006a). Jacobian – from wolfram mathworld. http://mathworld.wolfram.com/Jacobian.html.

[wolfram, 2006b] wolfram (2006b). Positive definite matrix – from wolfram mathworld. http://mathworld.wolfram.com/PositiveDefiniteMatrix.html.

[Yan li, 2002] Yan li, Tianshu Wang, H.-Y. S. (2002). Motion texture: A two-level statistical model for character motion synthesis. 2.4.2

[Zhang, 2001] Zhang, X. (2001). Biomechanical realism versus algorithmic efficiency: A trade-off in human motion simulation modeling.

[Zordan et al., 2005] Zordan, V. B., Majkowska, A., Chiu, B., and Fast, M. (2005). Dynamic response for motion capture animation. *j-TOG*, 24(3):697–701.

Auteursrechterlijke overeenkomst

Opdat de Universiteit Hasselt uw eindverhandeling wereldwijd kan reproduceren, vertalen en distribueren is uw akkoord voor deze overeenkomst noodzakelijk. Gelieve de tijd te nemen om deze overeenkomst door te nemen, de gevraagde informatie in te vullen (en de overeenkomst te ondertekenen en af te geven).

Ik/wij verlenen het wereldwijde auteursrecht voor de ingediende eindverhandeling:
Physically based avatar animation
Richting: **Master in de informatica** Jaar: **2007**
in alle mogelijke mediaformaten, - bestaande en in de toekomst te ontwikkelen - , aan de Universiteit Hasselt.

Niet tegenstaand deze toekenning van het auteursrecht aan de Universiteit Hasselt behoud ik als auteur het recht om de eindverhandeling, - in zijn geheel of gedeeltelijk -, vrij te reproduceren, (her)publiceren of distribueren zonder de toelating te moeten verkrijgen van de Universiteit Hasselt.

Ik bevestig dat de eindverhandeling mijn origineel werk is, en dat ik het recht heb om de rechten te verlenen die in deze overeenkomst worden beschreven. Ik verklaar tevens dat de eindverhandeling, naar mijn weten, het auteursrecht van anderen niet overtreedt.

Ik verklaar tevens dat ik voor het materiaal in de eindverhandeling dat beschermd wordt door het auteursrecht, de nodige toelatingen heb verkregen zodat ik deze ook aan de Universiteit Hasselt kan overdragen en dat dit duidelijk in de tekst en inhoud van de eindverhandeling werd genotificeerd.

Universiteit Hasselt zal mij als auteur(s) van de eindverhandeling identificeren en zal geen wijzigingen aanbrengen aan de eindverhandeling, uitgezonderd deze toegelaten door deze overeenkomst.

Ik ga akkoord,

Peter Boyen

Datum: **15.02.2007**