Faking Dynamics of Clothes

Bram Gerits

promotor :
dr. Fabian DI FIORE

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

Universiteit Maastricht

universiteit hasselt

Faking Dynamics of Clothes

Auteur: Gerits Bram

Promotor: Fabian Di Fiore

Universiteit Hasselt
2007

Voorwoord

Een thesis is een werk dat grotendeels individueel gemaakt wordt, maar deze thesis zou nooit tot een goed einde zijn gekomen zonder de hulp en steun van bepaalde mensen, deze zou ik daarom via deze weg willen bedanken.

Ik wil beginnen met het bedanken van mijn promotor Fabian Di Fiore, hij heeft me bij het begin van deze thesis een duidelijk idee gegeven over het doel van de thesis en heeft me daarna op weg geholpen om dit doel te bereiken. Ook in het verloop van de thesis heeft hij me steeds voorzien van nuttige informatie en hints.

Verder wil ik ook mijn vriendin Jill bedanken voor het nalezen van deze thesis en vooral voor de morele steun wanneer er problemen waren.

Als laatste wil ik mijn ouders bedanken, eveneens voor het nalezen van de tekst, maar ook, en vooral voor mij de mogelijkheid te geven om deze studies te volgen en mij hierbij altijd moreel te steunen.

Bram Gerits

Inhoudsopgave

1 Inleiding **5**

I Real dynamics **7**

2 Introductie **8**

3 Mass Spring Systems **10**
3.1 Particles 10
3.2 Veren 12
3.3 Krachten 13
3.3.1 N-aire krachten 13
3.3.2 Unaire krachten 14
3.4 Kleding modelleren 15
3.4.1 Drie soorten veren 18
3.5 Conclusie 19

4 Integratietechnieken **21**
4.1 Ordinary Differential Equation 21
4.2 Expliciete integratie-technieken 24
4.2.1 Forward euler 25
4.2.2 Midpoint methode 28
4.2.3 Runge-Kutta-methode 30
4.3 Grote van de tijdstappen 30
4.4 Conclusie 32

5 Beperkingen en Collision Detection **33**
5.1 Beperkingen 34

5.1.1 Beperken van particles 34
5.1.2 beperkende krachten 38
5.1.3 Conclusie 39
5.2 Collision detection 39
5.2.1 Bounding Volume Hierarchies 40
5.2.2 Spatial subdivision 43
5.2.3 Self-collision 44
5.3 Collision responce 45
5.4 Conclusie 47

II Fake Dynamics 48

6 Introductie 49
6.1 Wat zijn fake dynamics? 49
6.2 Reeds bestaand onderzoek naar 'fake dynamics' 50
6.3 Voor- en nadelen van Fake Dynamics 51
6.4 verdere verloop 53

7 Faking dynamics of Ropes and springs 54
7.1 Inleiding 54
7.2 De 3 lagen 55
7.2.1 Doorhangend touw 57
7.2.2 Loshangend touw 61
7.3 Catenary 62
7.3.1 Numeriek vinden van a 65
7.3.2 Aanpassingen in de formule 66
7.4 Algemene modellen 67
7.5 Resultaten en conclusie 68

8 Uitbreiding naar 2d-modellen 70
8.1 Inleiding 70
8.2 Voorstelling van de doek 71
8.3 Rusttoestand 72
8.3.1 Oppervlakte benaderen binnen de vaste punten 72
8.3.2 Relaxatie 82
8.4 Algemene vervorming 83
8.5 Golf-vervorming 84

8.6 Conclusie 86

9 Animatie **87**
9.1 Overgang tussen rust-vormen 88
9.1.1 Lineaire overgang 89
9.1.2 Overgang waarbij de snelheid verhoogt 89
9.1.3 Overgang met ease-in, ease-out 90
9.2 Golf-vervormingen en algemene vervormingen 91
9.3 User-interface voor het manipuleren en creëren van een animatie 92
9.3.1 Creatie van een doek 93
9.3.2 Produceren van een keyframe 93
9.3.3 De animatie 94

10 Conclusie en resultaten **95**

Hoofdstuk 1

Inleiding

Computergames en computeranimaties zijn de laatste jaren zeer sterk verbeterd op grafisch vlak en er is dan ook steeds meer vraag naar algoritmes voor realistischere en/of mooier uitziende beelden. Een in de laatste jaren sterk opkomend domein binnen deze animaties is het visualiseren van bewegende stoffen. Een domein dat enorm veel rekenkracht van de computer vereist en daarom pas de laatste jaren vernoemenswaardige resultaten heeft opgeleverd.

Er is tot op heden reeds zeer veel onderzoek gedaan naar het visualiseren van bewegende stoffen, en vaak met zeer goede resultaten. Maar bijna altijd word er gebruik gemaakt van de wetten uit de fysica en dan vooral de wetten uit de dynamica om de bewegingen te creëren. Dit wil zeggen dat de stoffen worden opgebouwd uit puntmassa's die verbonden zijn met veren. Op deze puntmassa's worden dan krachten uitgevoerd en op deze manier ontstaat er beweging.

Er zou echter nog een andere manier kunnen zijn om bewegende stoffen te visualiseren. Een manier waarbij we de wetten van de dynamica grotendeels overboord gooien en we proberen door behulp van wiskundige formules het uitzicht en de bewegingen van de stoffen te benaderen. Dit wordt ook wel "faking dynamics"genoemd. Deze techniek is reeds succesvol toegepast bij het visualiseren van touwen en veren maar is zelden gebruikt bij objecten met meerdere dimensies. Het hoofddoel van deze thesis is om te bepalen of het mogelijk is deze technieken uit te breiden naar 2-dimensionale objecten en om te bepalen welke technieken hiervoor gebruikt kunnen worden.

Ik zal in deze thesis eerst de reeds bestaande algoritmes en technieken beschrijven die gebruikt worden voor het visualiseren van kleding, dit zijn met andere woorden de technieken die gebruik maken de wetten van de dynamica.

Hierna zal ik beschrijven hoe het visualiseren van touwen en veren gebeurt met behulp van fake dynamics en ik zal deze technieken hierna uitbreiden naar 2-dimensionale objecten.

Deel I

Real dynamics

Hoofdstuk 2

Introductie

Het eerste grote deel van deze thesis is de literaatuurstudie. Hierin heb ik bestudeerd welke technieken er reeds bestaan voor het simuleren van kleding. Bijna al deze technieken maken gebruik van de wetten van de dynamica, dus spreken we hier van 'Real dynamics'.

'Fake dynamics' en 'Real dynamics' kunnen beide gebruikt worden voor het visualiseren van kleding. Er zijn echter weinig overeenkomsten tussen de 2 methodes, en er zullen bijgevolg weinig algoritmes en werkwijzes van de 'Real Dynamics' terugkeren wanneer we het gaan hebben over 'Fake dynamics'. Toch zal ik het eerste grote deel van dit werk wijden aan 'Real dynamics' en dit omwille van de volgende 3 redenen:

1. Het is onmogelijk om dynamische bewegingen van kleren te gaan benaderen met fake dynamics zonder eerst te weten hoe deze bewegingen er in de werkelijkheid uitzien en hoe deze momenteel gevisualiseerd worden.

2. Veel problemen en moeilijkheden die zich voordoen bij het programmeren van real dynamics van clothes zullen terugkeren bij het implementeren van fake dynamics. (Denk bijvoorbeeld maar aan collision detection en beperkingen)

3. Aangezien we uiteindelijk een relatief realistisch resultaat willen bekomen zullen we altijd in beperkte maten gebruik moeten maken van real dynamics.

Wanneer we onderzoek gaan doen naar de methodes die tot op heden gebruikt worden om kleding te visualiseren zullen we merken dat zo goed als al

deze methodes een verbetering of een uitbreiding zijn van 'Mass spring Systems'. Dit is dan ook het systeem dat verder in dit deel uitgebreid besproken zal worden. Een andere iets minder gebruikte methode is finite elements, maar deze methode vergt zeer veel rekenkracht van de computer en levert meestal mindere resultaten op. We zullen hierover dan ook niet verder uitwijden.

We zullen in dit deel beginnen met een bespreking van de basiscomponenten van een mass spring systeem: particles, veren en krachten. In een volgend hoofdstuk zal besproken worden hoe we kledingstukken kunnen opbouwen aan de hand van deze componenten. Hierbij hechten we vooral belang aan de structuur die we bekomen wanneer we de veren aan de particles gaan vasthechten. De structuur die we op deze manier bekomen willen we nu laten bewegen, hiervoor moeten we krachten gebruiken om versnellingen, snelheden en posities te berekenen. Hiervoor hebben we integratietechnieken nodig. Er bestaan zeer veel integratietechnieken en verbeteringen op bestaande integratietechnieken, en het is onmogelijk om deze allemaal te bespreken, maar in hoofdstuk 4 zullen de belangrijkste vermeld en uitgelegd worden.

Met al de technieken besproken in de hoofdstukken 2 tot en met 4 is het mogelijk om kledingenstukken te construeren en te laten bewegen, maar er zijn nog een paar technieken die ons kunnen helpen om een meer realistisch en bruikbaar systeem te bekomen. Stel bijvoorbeeld dat we een gordijn willen simuleren die logischerwijs aan de bovenkant vastzit aan een stok. Dan willen we dus de bovenste particles van de structuur beperken op een horizontale lijn. Dit kan door beperkingen op te leggen aan bepaalde particles van de structuur. De methodes die hiervoor gebruikt kunnen worden, worden besproken in hoofdstuk 5. Wanneer we nu een bal tegen deze gordijn gooien is het ook handig als er een systeem is dat de botsing tussen de bal en de gordijn detecteert en hier op een passende manier op reageert. Dit systeem noemen we collision detection en zal besproken worden in hoofdstuk 6.

Hoofdstuk 3

Mass Spring Systems

Een mass spring systeem is eigenlijk een uitbreiding van een particle systeem waarbij we de particles onderling verbinden met behulp van veren. Mass spring systemen worden vooral gebruikt in physically based modeling om vervormbare objecten voor te stellen. De manier waarop de veren verbonden worden met de particles (dus eigenlijk de opbouw van het object) en de verschillen in kracht van de veren zal het gedrag van het object als een geheel bepalen. Dankzij dit systeem is het ook mogelijk om krachten van buitenuit te laten inwerken op het object en zal het object hier op een min of meer realistische wijze op reageren.

Mass spring systemen worden vaak gebruikt omdat ze eenvoudig te begrijpen en te implementeren zijn. En omdat ze gebruikt kunnen worden voor een uitgebreid aantal intressante objecten te cre/ëren. Dankzij de eenvoud van mass spring systemen is het ook zeer eenvoudig om ze te integreren in reeds bestaande frameworks zonder de frameworks zwaar te belasten.

In de volgende secties zullen we dieper ingaan op particles en veren, en ook op de verschillende krachten die er op een mass spring systeem kunnen inwerken. We eindigen dit hoofdstuk met een manier voor te stellen waarop we kleding kunnen modelleren aan de hand van een mass spring systeem.

3.1 Particles

Een particle is het kleinste deel van een mass spring systeem, het kan gezien worden als een punt in een 3d-omgeving met een bepaalde massa. (We noemen ze dan ook wel eens puntmassa's) Elke particle zal zich op elk tijdstip

t in een bepaalde toestand bevinden. Deze toestand bevat de volgende 4 elementen:

1. massa
2. positie: x(t)
3. snelheid: v(t)
4. kracht: f(t)

Wat belangrijk is in een particle systeem is het verband tussen de kracht die inwerkt op een particle en de massa en de versnelling van deze particle. Dit verband kan beschreven worden volgens de bewegingswetten van newton, dit wil zeggen dat de kracht die op een particle wordt uitgeoefend gelijk is aan:

$$f = ma \tag{3.1}$$

Het is ook zo dat de versnelling van een particle de afgeleide is van de snelheid, en de snelheid is dan op zijn beurt weer de afgeleide van de positie van een particle. Deze relatie kunnen we dus als volgt beschrijven:

$$f = ma \tag{3.2}$$

$$a = \frac{dv}{dt} = \dot{v} = \ddot{x} = \frac{f}{m} \tag{3.3}$$

$$v = \frac{dx}{dt} = \dot{x} \tag{3.4}$$

Om nu de positie van de particle te bekomen kunnen we in theorie als volgt te werk gaan:

1. Met behulp van vergelijking (3.1) en de kennis van de massa van een particle en de kracht uitgeoefend op die particle berekenen we de versnelling van deze particle.

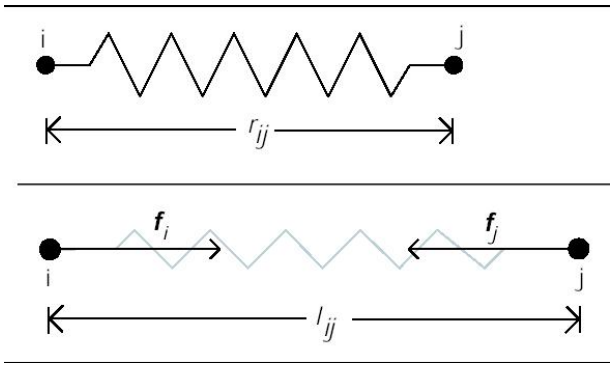
2. Als we deze versnelling nu integreren in respect tot de tijd, dan bekomen we de snelheid van de particle.

3. Wanneer we deze snelheid nu opnieuw integreren in respect tot de tijd, bekomen we de positie van de particle na een bepaalde tijd.

In praktijk is het vaak iets moeilijker om een exacte positie van een particle te bepalen na een bepaalde tijd als hierboven beschreven, maar dit zal uitvoerig besproken worden in hoofdstuk 4.

3.2 Veren

De particles in een mass spring systeem worden verbonden met elkaar door veren. Het eenvoudigste type van veer is een lineaire veer, dit is dan ook de soort veer die meestal gebuikt wordt in een mass spring systeem. Een lineaire veer heeft een bepaalde rustlengte r_{ij} die hij altijd zal proberen te behouden door de nodige krachten uit te oefenen op de 2 particles waaraan hij verbonden is. Dit zal gebeuren wanneer de veer is uitgetrokken of samengeduwd tot een lengte verschillend van r_{ij}.



Figuur 3.1: De krachten die een lineaire veer uitoefent op 2 particles.

De krachten die een lineaire veer uitoefent op 1 van zijn 2 eindpunten is afhankelijk van zijn huidige lengte zoals in de volgende vergelijking:

$$f_i = -k_{ij}(|d_{ij}| - r_{ij})\frac{d_{ij}}{|d_{ij}|} \tag{3.5}$$

$$d_{ij} = x_i - x_j \tag{3.6}$$

De kracht die inwerkt op het andere eindpunt van de veer is dan gelijk aan het omgekeerde:

$$f_j = -f_i \tag{3.7}$$

De term $(d_{ij}/|d_{ij}|)$ in vergelijking (3.5) definieert de richting van de kracht uitgeoefend op particle i. De term $(|d_{ij}|-r_{ij})$ in deze vergelijking is evenredig met het verschil in lengte tussen de huidige toestand van de veer en zijn rusttoestand op een lineaire manier. Vandaar ook de naam, lineaire veren. De factor (k_{ij}) is de stijfheid van de veer en bepaalt hoe sterk de kracht is die de veer terug naar zijn rusttoestand berngt wanneer deze samengeduwd of uitgetrokken is.

3.3 Krachten

We hebben nu de 2 componenten besproken waaruit een mass spring systeem is opgebouwd. Maar om nu beweging te krijgen in ons object moeten er krachten uitgeoefend worden op de particles. Er zijn een heel aantal verschillende krachten die uitgeoefend kunnen worden op particles. We kunnen deze opdelen in 2 grote groepen: de n-aire krachten en de unaire krachten.

3.3.1 N-aire krachten

N-aire krachten zijn krachten die inwerken op verschillende particles tegelijk. Het meest voor de hand liggende voorbeeld is hier een lineaire veer zoals eerder besproken. Een ander voorbeeld zou bijvoorbeeld het afstoten en aantrekken tussen twee atomen kunnen zijn. Maar aangezien voor het visualiseren van kleding enkel veren gebruikt worden als n-aire krachten zullen we niet verder ingaan op deze krachten.

3.3.2 Unaire krachten

Dit zijn krachten die slechts op enkele particle inwerken. Maar vaak zullen deze krachten dezelfde sterkte hebben bij alle particles. Bij het visualiseren van kleding zijn de drie meest voorkomende unaire krachten de zwaartekracht, viscious drag en wind.

1. **Zwaartekracht:** De zwaartekracht of gravitatiekracht is een aantrekkende kracht die twee (punt)massa's op elkaar uitoefenen. Bij het visualiseren van kleding zal vooral de aantrekking van de aarde op het kledingstuk van belang zijn. In zijn meest eenvoudige vorm kunnen we de zwaartekracht voorstellen met een richtingsvector **g**, zodat de zwaartekracht in alle richtingen kan wijzen. We stellen de zwaartekracht op een particle nu voor als de volgende vergelijking:

$$f_{zwaartekracht} = mg \tag{3.8}$$

   Waarbij m de massa is van de betreffende particle.

2. **Viscous drag:** Een 2de unaire kracht die vaak gebruikt wordt bij het visualiseren van kleding is viscous drag, deze kracht heeft als doel om beweging tegen te gaan. Hierdoor zullen particles bij een afwezigheid van andere krachten langzaam tot rust komen. Deze kracht simuleert eigenlijk een beetje de wrijvingskracht uit de realiteit, aangezien dat het inplementeren van een echte wrijvingskracht enorm complex is en dus meestal genegeerd wordt bij computeranimaties. Dankzij deze kracht wordt het mass spring systeem ook stabieler. De viscous drag op een particle wordt voorgesteld door de volgende formule:

$$f_{vd} = -k_d v \tag{3.9}$$

   Waarbij k_d de drag constante is, en v de snelheid van de particle.

3. **Wind:** Een 3de unaire kracht die vaak gebruikt wordt is wind. Wind simuleren zoals deze zich in werkelijkheid voordoet is bijna onbegonnen werk. We zouden dan rekening moeten houden met de manier waarop wind zich rond obstakels beweegt, de verschillende temperaturen van

de lucht, de richting van wind is ook overal verschillend. Dit alles valt buiten het bestek van deze thesis, en daarom geef ik hier enkel een eenvoudige formule die windstoten genereert in 1 richting:

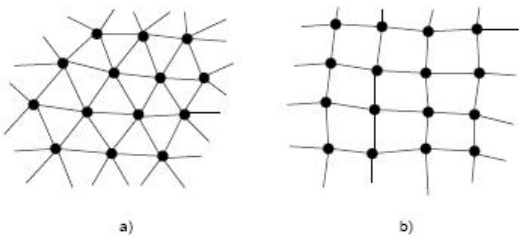
$$f_{wind} = -v_{wind}.(sin(t^2) + 0.5) \tag{3.10}$$

Met v_{wind} als vector voor de richting en de snelheid van de wind en waarbij t lineair geïnterpoleerd wordt over de tijd.

Om nu de kracht op een bepaalde particle te bekomen moeten we enkel de resulterende kracht berekenen die we bekomen door alle krachten op deze bepaalde particle bij elkaar optellen.

3.4 Kleding modelleren

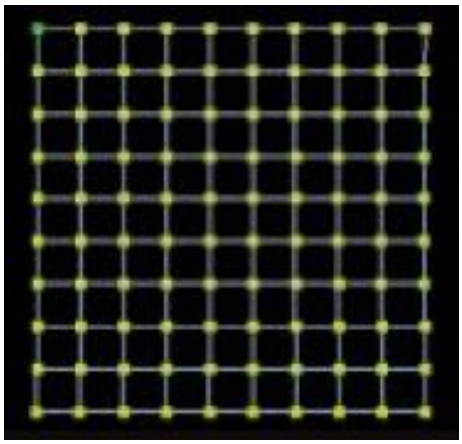
Nu we gezien hebben hoe mass spring systemen worden opgebouwd en hoe we hier krachten op kunnen laten inwerken gaan we proberen om met dit model kledingstukken te creeren die op een zo realistisch mogelijke manier gaan bewegen. Net zoals bij echte kleding kunnen we de particles op enorm veel verschillende manieren met elkaar gaan verbinden. Maar de twee meest voorkomende topologien die gebruikt worden bij de visualisatie van kleding zijn de driehoekige en de vierhoekige mesh zoals te zien in figuur 3.2



Figuur 3.2: Een kledingstuk gemodelleerd met een driehoekige en een vierhoekige mesh.

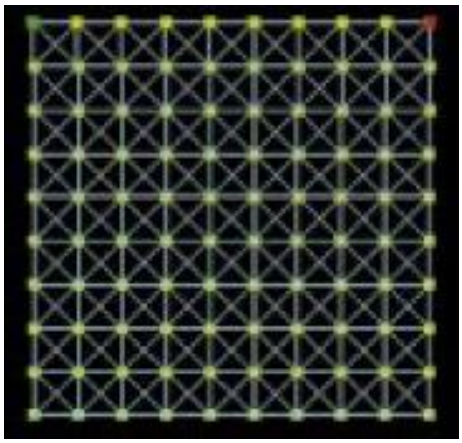
Het gebruik van een mesh met driehoeken heeft als voordeel dat het gemakkelijker is om collision detection te doen met zichzelf. Maar aangezien de mesh met vierkanten de geweven structuur van echte kleding meer benadert, opteren we er toch voor om deze te gebruiken in het verdere verloop van dit hoofdstuk.

We willen dus een mass spring systeem opbouwen dat er uitziet als een mesh opgebouwd uit vierhoeken. We starten met een rechthoekige grid te bouwen van particles, deze verbinden we onderling met hun buren door gebruik te maken van veren zoals weergegeven op figuur 3.3.



Figuur 3.3: Een eenvoudige grid van particles verbonden met veren.

Het resultaat dat we op deze manier bekomen ziet er al goed uit en heeft weinig veren nodig. Maar wanneer we deze structuur echter aan 1 punt zouden ophangen en een simulatie zouden starten, dan zou het volledige oppervlak uiteindelijk 1 lijn worden. Het eenvoudige model uit figuur 3.3 is dus niet uitgebreid genoeg om de vorm van de kleding te behouden. Dit is dus nog niet volledig wat we willen, we willen de afstand tussen de diagonale elementen behouden om zo de vorm te behouden. Dit kunnen we eenvoudig afdwingen door diagonale veren toe te voegen zoals afgebeeld op afbeelding 3.4.

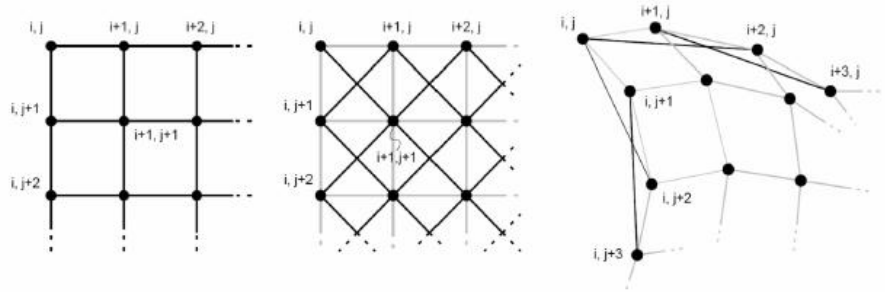


Figuur 3.4: Diagonale veren toegevoegd in de eenvoudige gridstructuur.

We hebben er nu voor gezorgd dat de vorm behouden wordt, maar wanneer we het kledingstuk vanuit een verticale positie op de vloer zouden laten vallen zal dit resulteren in 1 grote hoop veren zonder enige structuur. Dit komt omdat er niets is dat het model er van weerhoud om dubbel te plooien langs de veren die we reeds hebben. Om dit probleem te verhelpen voegen we nog een derde soort veer toe. Deze veren slaan steeds 1 particle over en zorgen er zo voor dat het kledingstuk niet kan dubbelplooien zoals een blad papier.

We concluderen dus dat we uiteindelijk drie soorten veren nodig hebben om de eigenschappen van een kledingstuk te simuleren, deze houden verband met de drie eigenschappen van een kledingstuk: weerstand tot uitrekking, afschuiving en buiging. We noemen deze veren de structurele veren, de afschuif-veren en de buigveren. Hier wordt in de volgende sectie wat dieper op in gegaan.

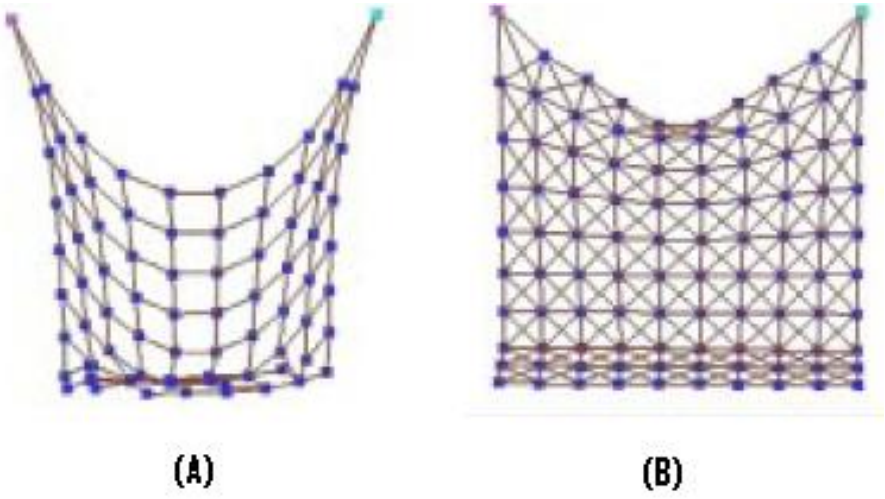
3.4.1 Drie soorten veren



Figuur 3.5: De 3 soorten veren die gebruikt worden om een kledingstuk te modelleren: de structurele veren (a), de afschuif-veren (b) en de buig-veren (c).

1. **Structurele veren:** De structurele veren verbinden de punten (i,j) met de punten (i+1,j) en (i,j+1) zoals te zien in afbeelding 3.5(a). Ze proberen de structuur en de grootte van het kledingstuk in stand te houden en weerspiegelen dus de eigenschap weerstand tot uitrekking. Dit wil zeggen dat wanneer het kledingstuk door een externe kracht zal worden uitgetrokken of samengeduwd, dat deze veren het kledingstuk terug tot zijn originele grootte zullen duwen of trekken.

2. **Afschuif-veren:** De afschuif-veren verbinden de punten (i,j) met (i+1,j+1) en de punten (i+1,j) met (i,j+1) zoals te zien in afbeelding 3.5(b). Ze vormen dus de diagonalen tussen de structurele veren. Deze veren hebben als functie om de rechthoekige vorm van het kledingstuk te behouden. De toevoeging van deze veer zal er voor zorgen dat het kledingstuk er veel realistischer zal gaan uitzien doordat het minder zal uitrekken en doorzakken. Het verschil tussen de aanwezigheid en afwezigheid van deze veren is te zien in afbeelding 3.6.



Figuur 3.6: Het verschil tussen een kledingstuk met(b) en zonder(a) afschuifveren.

3. **Buig-veren:** Deze veren verbinden de punten (i,j) met de punten (i,j+2) en (i+2,j) zoals te zien in afbeelding 3.5(c). Deze veren zorgen ervoor dat het kledingstuk een weerstand heeft tegen buiging en plooiing. Ze zorgen ook voor een stabielere structuur van het kledingstuk.

Met behulp van een mass spring systeem en gebruik makend van deze 3 soorten veren is het nu mogelijk om alle soorten stoffen op een redelijk realistische manier na te bootsen. Om de eigenschappen van een stof na te bootsen is het nu enkel nog een kwestie van de juiste parameters in te stellen voor de 3 types veren, zoals de kracht en de dempingskracht. Aangezien de meeste stoffen heel gemakkelijk buigen is het bijvoorbeeld vanzelfsprekend dat we de kracht van de buig-veren veel lager zetten dan deze van de structurele veren en de afschuif-veren. Op deze manier dienen de buigveren enkel om de stof er van te weerhouden dubbel te plooien zoals een blad papier, en dit is meestal ook wat we willen.

3.5 Conclusie

In dit hoofdstuk hebben we gezien dat een mass spring systeem een uitbreiding is van een particle systeem dat gebruikt wordt voor het visualiseren

van vervormbare objecten. We hebben gezien dat ze opgebouwd werden uit particles of puntmassa's die verbonden worden met veren. Op deze particles kunnen dan zowel unaire als n-aire krachten uitgeoefend worden om het geheel te laten bewegen. Hierna hebben we een methode gezien om aan de hand van deze mass spring systemen kledingstukken op te bouwen door gebruik te maken van 3 soorten veren: structurele veren, afschuif-veren en buig-veren. Op deze manier krijgen we een simulatie van een kledingstuk dat redelijk goed voldoet aan de werkelijke eigenschappen van kledingstukken. Om nu tot een realistische simulatie te komen hebben we nog nood aan integratietechnieken, deze worden besproken in het volgende hoofdstuk.

Hoofdstuk 4

Integratietechnieken

Om ons mass spring systeem te laten bewegen moeten we na elk tijdstip t de mogelijkheid hebben om de posities van alle particles te kunnen berekenen. Zoals we reeds hebben besproken in het vorige hoofdstuk is het bepalen van een positie op tijdstip $(t + h)$ enkel een kwestie van de toestand op tijdstip t te integreren. Integratietechnieken zijn dus nodig en zelfs zeer belangrijk wanneer we beweging willen bekomen. Het probleem is hier echter dat we de krachten niet kennen die over de tijd zullen inwerken op de particle, waardoor we geen functie van deze krachten kunnen opstellen. Hierdoor kunnen we geen analystische integratietechnieken gebruiken en zullen we numerieke integratie-technieken moeten gebruiken.

Numerieke integratietechnieken zijn van enorm belang in cloth simulation en worden ook binnen verschillende andere vakgebieden al lang druk onderzocht. Hierbij wordt vooral aandacht gegeven aan fysische correctheid en stabiliteit met als belangrijkste factor het visuele voorkomen [17]. In dit hoofdstuk zullen we eerst bespreken wat een *Ordinary Differential Equation*, of kortweg ODE is en wat hun toepassing is bij het bewegen van particles. Hierna zullen we enkele van de meest gebruikte integratietechnieken bespreken.

4.1 Ordinary Differential Equation

De definitie van een differentiaalvergelijking beschreven in [20] gaat als volgt: "*Differentiaal vergelijkingen beschrijven de relatie tussen een onbekende functie en zijn afgeleiden.*". Om een differentiaal vergelijking op te lossen moeten we een functie zoeken die aan de relatie voldoet, meestal rekeninghoudend

met enkele bijkomende condities. Voor het visualiseren van kleding zijn we slechts geïntresseerd in ODE's van de eerste orde, dit wil zeggen dat enkel de eerste afgeleide in de vergelijking zal voorkomen.

Deze eerste orde differentiaalergelijkingen kunnen gebruikt worden voor het oplossen van een *initial value problem*. In zo een *initial value problem* wordt het gedrag van het systeem beschreven door een differentiaalvergelijking met de volgende vorm:

$$y'(t) \quad = \quad f(t, y(t)) \text{ met } f : \mathbb{R} \times \mathbb{R} \to \mathbb{R} \tag{4.1}$$

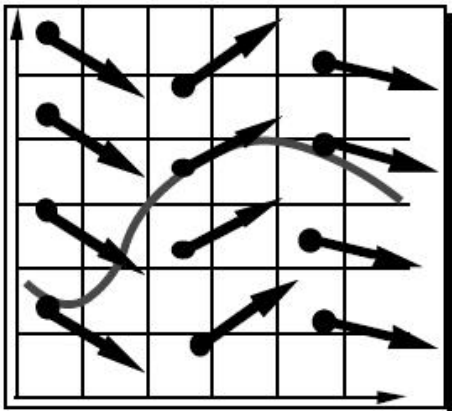
Bij deze differentiaalvergelijking wordt ook de beginconditie gegeven, deze bevind zich in het domein van f:

$$(t_0, y_0) \in \mathbb{R} \times \mathbb{R}$$

Een oplossing van dit *initial value problem* is een functie y die een oplossing is van de differentiaalvergelijking en die voldoet aan

$$y(t_0) \quad = \quad y_0 \tag{4.2}$$

Een standaard *initial value problem* is eenvoudig te visualiseren. In 2D geeft $x(t)$ een curve die de beweging van een punt over een vlak weergeeft. Op elk punt x kan de functie f nu geevalueerd worden om zo een 2d-vector te bekomen, de functie f definieerd dus een vector-veld in het vlak. [20] De bekomen vector zal de snelheid zijn die het punt zal hebben wanneer het al dan niet doorheen het punt x zal gaan. 4.1



Figuur 4.1: Het visualiseren van een initial value problem.

Op deze manier krijgen we een mogelijkheid om het volledige traject van het punt over de curve te bepalen zonder de werkelijke functie van de curve te kennen. We moeten dus enkel beschikken over de startpositie van het punt en een functie die ons op elk moment en op elke positie de mogelijkheid geeft om de snelheid van het punt te geven.

Het *initial value problem* is een ODE van de eerste orde, omdat we een formule kennen voor de eerste afgeleide. Het probleem dat wij nu hebben, waarbij we enkel de krachten op een particle kennen, en dus enkel de versnelling van deze particles kunnen bekomen op elk tijdstip is een ODE van orde 2. De reden hiervoor is dat de versnelling de 2de afgeleide is van de positie zoals we zagen in formule 3.3. Om deze ODE van orde 2 op te lossen moeten we deze converteren naar 2 ODE's van orde 1. Dit kunnen we door gebruik te maken van de snelheid. De 2 ODE's die we dan uiteindelijk moeten oplossen zijn de 2 volgende:

$$\dot{v} = \frac{f}{m} \qquad \dot{x} = v$$

Omdat analytische oplossingen niet of zelden voor handen zijn om dit probleem op te lossen richten we ons enkel op numerieke oplossingen. Dit wil zeggen dat we starten vanuit de begintoestand en dat we werken met discrete tijdstappen om een toestand na een bepaalde tijd te bekomen, waarna we deze toestand weer gebruiken om een volgende toestand te bekomen, enzovoort. Wanneer we een toestand hebben op een tijdstip t en we willen een toestand berekenen na een discrete tijdstap met een lengte h, gebruiken we

de krachten die op tijdstip t op de particle inwerken. Door een ODE van orde 2 op te lossen kunnen we uit deze krachten de verandering in positie($\triangle x$) benaderen, die we dan weer kunnen gebruiken om de nieuwe positie op tijdstip $(t + h)$ te bepalen zoals aangegeven in de volgende formule:

$$x(t + h) \quad = \quad x_t + \triangle x \tag{4.3}$$

Doordat we met numerieke oplossingen werken en dus gebruik maken van discrete tijdstappen, zullen we geen exacte posities kunnen bepalen, maar enkel benaderingen hiervan. Doordat we bij het berekenen van een nieuwe toestand altijd gebruik maken van de vorige toestand zal deze fout steeds meegenomen worden en is het mogelijk dat deze fout zal accumuleren. Hierdoor zal de benaderde toestand steeds harder verschillen van de werkelijke toestand. Dit probleem en oplossingen hiervoor zullen besproken worden in de volgende secties van dit hoofdstuk, waarbij we het zullen hebben over verschillende integratietechnieken.

Er is reeds enorm veel onderzoek gedaan naar integratietechnieken en er bestaan dan ook reeds een heel aantal verschillende technieken. En aangezien er nog steeds veel onderzoek gedaan wordt onstaan er nog dagelijks nieuwe technieken of uitbreidingen of verbeteringen op bestaande technieken. Maar aangezien het in deze literatuurstudie vooral gaat om het creëren van een idee wat er belangrijk is bij het visualiseren van kledingstoffen zullen we ons enkel richten tot de expliciete integratietechnieken. Deze zijn het meest eenvoudig en liggen reeds jaren in de plooi en hier wordt nog weinig tot geen onderzoek naar gedaan [17]. Andere technieken zijn impliciete integratie [1] en verlet integratietechnieken [1]. Ook bestaan er nog enkele technieken die impleciete en expliciete integratie combineren, deze worden IMEX-technieken genoemd.

4.2 Expliciete integratie-technieken

Bij expliciete integratie gebruiken we de snelheid en versnelling op een tijdstip t om de positie van een particle te bepalen op een tijdstip (t+h). De bestaande technieken om deze positie x(t + h) te bepalen zijn allemaal gebouwd op de taylorreeks, ook de nauwkeurigheid en dus de gevoeligheid voor fouten kan uit deze taylorreeks afgeleid worden. De taylorreeks ziet er uit als volgt:

$$x(t+h) \quad = \quad x_t + h\dot{x}(t) + \frac{h^2}{2!}\ddot{(x)} + ... + \frac{h^n}{n!}\frac{d^n x}{dt^n}(t) + Oh^{n+1} \qquad (4.4)$$

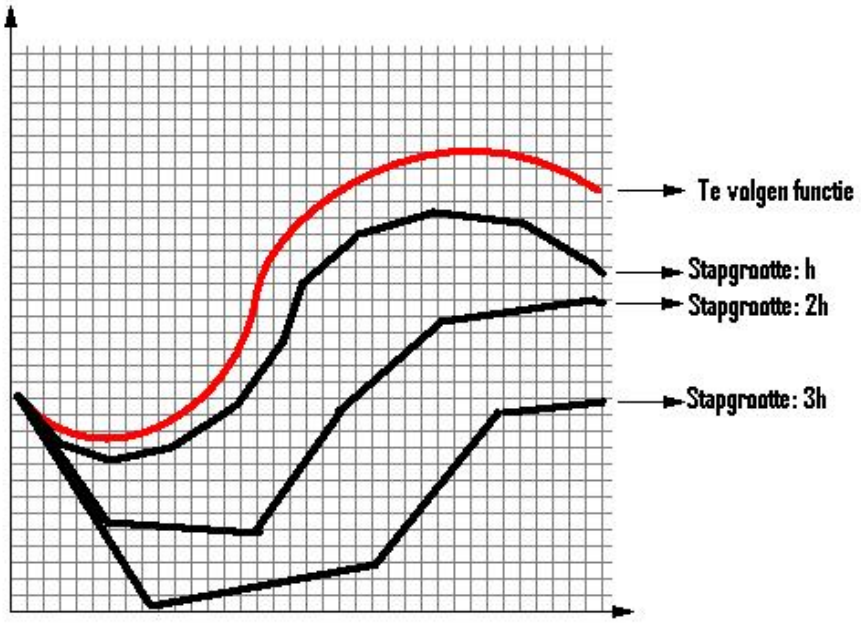
De nauwkeurigheid van een expliciete integratie-techniek zal dus altijd een orde hebben van h^{n+1} en zal bijgevolg nooit gelijk zijn aan nul omdat we met een eindig aantal termen moeten werken [14]. De 3 bekendste expliciete integratie-technieken zijn forward Euler, midpoint en Runge-Kutta. Deze verschillen vooral met elkaar in het aantal termen dat ze gebruiken van de taylorreeks.

4.2.1 Forward euler

De meest eenvoudige methode gebruikt enkel de eerste 2 termen uit de taylorreeks en om dus een toestand te berekenen na een bepaalde tijd h gebruiken we de volgende formule:

$$x(t_0 + h) \quad = \quad x_{t_0} + h\dot{x}(t_0) \qquad (4.5)$$

Zoals we zien gebruikt de euler methode enkel de eerste afgeleide van de positie, dit wil zeggen dat enkel de snelheid van een particle op het tijdstip t_0 in rekening wordt gebracht bij het berekenen van de positie op het tijdstip $t_0 + h$. De eulermethode laat de functie dus als het ware rechtdoor lopen over een tijdspanne h in de richting waarin de functie loopt op het tijdstip t_0 oftewel de afgeleide $\dot{x}(t_0)$. Bij bewegende particles wil dit zeggen dat de particles over een tijdspanne h rechtdoor zullen bewegen met de snelheid die ze hebben op het tijdstip t_0. Dde eulermethode heeft bijgevolg slechts een nauwkeurigheid van $O(h^2)$ zoals we kunnen afleiden uit formule 4.4. Op afbeelding 4.2 zien we het resultaat van een benadering met de eulermethode, ook zien we hier hoe de fout kan accumuleren en op welke manier de stapgrootte een invloed heeft op de grootte van de fout.

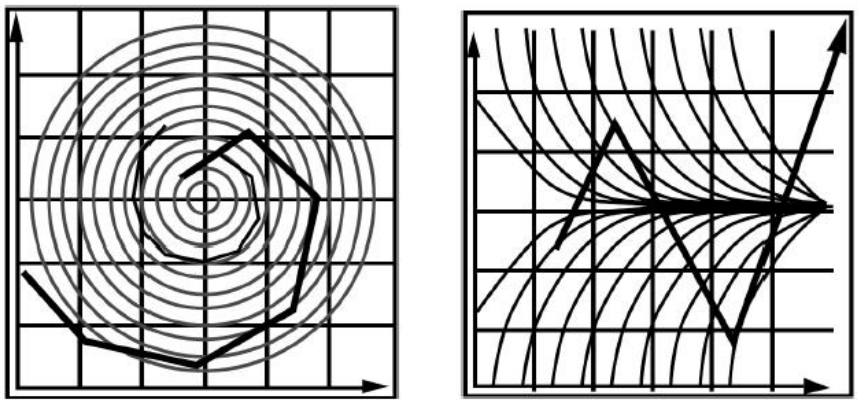


Figuur 4.2: De werking van de eulermethode en zijn nauwkeurigheid.

Om aan te tonen dat de eulermethode niet accuraat is nemen we een voorbeeld uit een paper van Andrew Witkin en David Baraff [20].We beschouwen een 2D-functie f waarvan de integralen concentrische cirkels beschrijven. Een punt p dat dus start op deze cirkel zou in theorie oneindig moeten blijven ronddraaien op de cirkel waarop hij start. Maar omdat we de eulermethode gebruiken zal het punt bij elke stap bewegen naar een cirkel met een grotere straal, en zal dit punt dus een uitwaartse spiraal volgen. We zien dit probleem heel duidelijk op de grafische voorstelling: 4.3(a). Het inkorten van de stapgrootte zal deze uitwaartse beweging vertragen, maar deze zal nooit volledig verdwijnen.

Een tweede probleem van numerieke integratie is e instabiliteit, zoals we kunnen zien op figuur 4.3(b) is het mogelijk dat een punt gaat oscilleren rond de plaats waar het eigenlijk naar toe zou moeten bewegen en hier op die manier zelfs steeds verder vanaf gaat bewegen. De reden hiervoor is dat de tijdstappen te groot zijn waardoor het punt binnen elke tijdstap alweer te ver is. De simulatie kan hierdoor exploderen. Dit probleem zal zich vaak voordoen bij mass spring systemen omdat we hier gebruik maken van stijve veren die grote krachten uitvoeren op de particles en dus onnauwkeurige

verplaatsingen teweeg brengen. [17]



(a) De onnauwkeurigheid bij integratie die tot een probleem leidt

(b) De oplossing oscilleert rond nul en het systeem is instabiel

Figuur 4.3: Twee veel voorkomende problemen bij numerieke integratie.

Het tweede probleem hierboven kan gedeeltelijk verholpen worden door bijvoorbeeld de particles zwaarder te maken, waardoor de stijve veren minder invloed zullen hebben op verplaatsingen van deze particles. Ook het toevoegen van viscous drag, zoals reeds besproken in het vorige hoofdstuk beperkt de instabiliteit omdat deze externe kracht een voordurende remkracht op alle particles uitoefent. Deze twee oplossingen kunnen echter als gevolg hebben dat de simulatie minder realistisch zal ogen. Een kledingstuk waar we het gewicht van de puntmassa's sterk zouden verhogen zal dan bijvoorbeeld een gewicht krijgen dat veel hoger is dan in de realiteit. Ook een te sterke externe kracht kan er voor zorgen dat een simulatie er niet meer realistisch uitziet.

Een betere oplossing voor de instabiliteit van de euler methode is het gebruiken van meerdere termen uit de taylor-reeks. Dit is ook wat de volgende expliciete integratie technieken die we in dit hoofdstuk zullen bespreken gaan doen. We zullen ook zien dat de minieme rekenkost die de eulermethode nodig heeft per stap, niet opweegt tegen het aantal stappen dat er nodig is om een accuraat en stabiel systeem te krijgen. Meer gesofisticeerde methodes, die zelfs 4 of 5 evaluaties nodig hebben per stap zullen toch efficienter werken dan de eulermethode omdat ze een veel grotere stapgrootte toelaten.

4.2.2 Midpoint methode

Zoals reeds gezegd in de vorige sectie kunnen we een nauwkeurigere methode verkrijgen door meer termen van de taylor-reeks te gebruiken. De midpoint methode gebruikt 1 term meer dan de euler-methode, we herhalen hier even voor de duidelijkheid het stuk van de taylor-reeks gebruikt door de midpoint-methode:

$$x(t+h) \quad = \quad x_t + h\dot{x}(t) + \frac{h^2}{2!}\ddot{(x)} + Oh^3 \tag{4.6}$$

Wanneer we dus een mogelijkheid vinden om $\ddot{x}$ en $\dot{x}$ te bepalen, kunnen we een nauwkeurigheid verkrijgen van Oh^3 in plaats van Oh^2, de methode hiervoor is beschreven in [20] en zullen we hieronder reproduceren.

We herhalen even dat de afgeleide over de tijd $\dot{x}$ gegeven wordt door de functie $f(x(t), t)$. Om hetgeen wat volgt eenvoudig te houden gaan we er van uit dat de afgeleide functie f enkel indirect van de tijd afhangt, we krijgen dan dat $\dot{x} = f(x(t))$. De kettingregel zegt dan dat:

$$ddot(x) = \frac{\partial f}{\partial x}\dot{x} = f'f \tag{4.7}$$

Om te vermijden dat we f' moeten evalueren, iets wat zeer complex en rekenintensief is, kunnen we de tweede orde term benaderen in termen van f en substitueren in vergelijking 4.6. Hiervoor moeten ook f ontbinden in een taylor reeks:

$$f(x_0 + \triangle x) = f(x_0 + \triangle x f'(x_0) + \bigcirc(\triangle x^2) \tag{4.8}$$

We introduceren $\ddot{x}$ in deze formule door $\triangle x$ als volgt te kiezen:

$$\triangle\, x \quad = \quad \frac{h}{2} f(x_0) \tag{4.9}$$

Wanneer we nu in vergelijking 4.8 de substitutie voorgesteld in 4.9 uitvoeren krijgen we de volgende vergelijking:

$$f(x_0 + \frac{h}{2} f(x_0)) \quad = \quad f(x_0) + \frac{h}{2}\ddot{x}(t_0) + O(h^2) \tag{4.10}$$

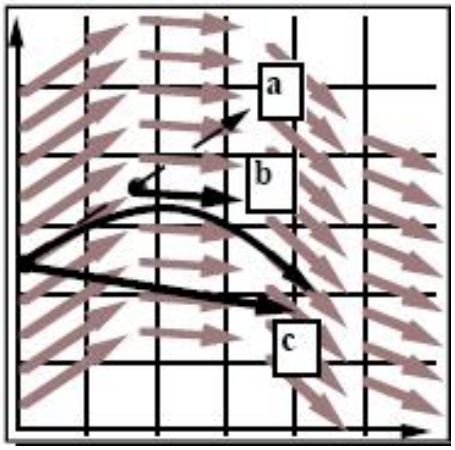
We vermenigvuldigen nu beide kanten met h, zodat we een nauwkeurigheid van $O(h^3)$ krijgen in plaats van $O(h^2)$ en we herschikken zodat we de volgende vergelijking bekomen:

$$\ddot{x} + O(h^3) \quad = \quad h(f(x_0 + \frac{h}{2}f(x_0)) - f(x_0) \tag{4.11}$$

Wanneer we de formule 4.11 nu substitueren in vergelijking 4.7 dan bekomen we het uiteindelijk resultaat, en dit ziet er uit als volgt:

$$x(t_0 + h) \quad = \quad x(t_0) + h(f(x_0 + \frac{h}{2}f(x_0)) \tag{4.12}$$

De vergelijking 4.12 is de uiteindelijk formule van de midpoint-methode. Deze formule berekent eerst een euler stap, waarna er een tweede afgeleide zal berekend worden in het midden van deze stap. Deze evaluatie zal dan uiteindelijk gebruikt worden voor de nieuwe positie van x te berekenen. Omdat we de afgeleide nemen in het midden van een stap wordt deze methode dus de midpoint-methode genoemd. Om een duidelijker beeld te krijgen van wat er juist gebeurt bij de midpoint methode geeft figuur 4.4 nog eens een visueel beeld van de werking en zijn 3 grote stappen.



Figuur 4.4: De werking van de midpoint-methode.

1. (a). We berekenen een euler stap: $\triangle x = \triangle t f(x, t)$

2. (b). We evalueren f in het midden van deze stap: $f_{mid} = f\left(\frac{x+\triangle x}{2}, \frac{t+\triangle t}{2}\right)$

3. (c). We nemen een stap gebruik makend van de midpoint waarde: $x(t + \triangle x) = x(t) + \triangle t f_{mid}$

4.2.3 Runge-Kutta-methode

De midpoint-methode gebruikt zoals in de vorige sectie besproken 1 term meer uit de taylor-reeks waardoor we een nauwkeurigheid bekomen van $O(h^3)$. We kunnen natuurlijk ook meerdere termen gebruiken uit de taylor-reeks waardoor we logischerwijs ook een hogere nauwkeurigheid krijgen. Een van de meest populaire methodes hiervoor is Runge-Kutta, en dan vooral deze met orde 4, deze geeft een nauwkeurigheid van $O(h^5)$. Het afleiden van de formule voor de runge-kutta-methode van orde 4 is zeer complex en uitgebreid en valt dus buiten het bestek van deze thesis, maar we geven hier wel even de formule zelf voor het berekenen van $x(t_0 + h)$:

$$k_1 = hf(x_0, t_0)$$
$$k_2 = hf(x_0 + \frac{k_1}{2}, t_0 + \frac{h}{2})$$
$$k_3 = hf(x_0 + \frac{k_2}{2}, t_0 + \frac{h}{2})$$
$$k_4 = hf(x_0 + k_3, t_0 + h)$$
$$x(t_0 + h) = x(t_0) + \frac{1}{6}k_1 + \frac{1}{3}k_2 + \frac{1}{3}k_3 + \frac{1}{6}k_4$$

4.3 Grote van de tijdstappen

We hebben nu 3 verschillende integratie-methodes gezien met allemaal een verschillende nauwkeurigheid, en we hebben hieruit kunnen concluderen dat we een grotere stapgrootte h kunnen gebruiken wanneer we een hogere nauwkeurigheid hebben. Maar nu rest ons nog steeds het probleem hoe groot we deze stapgrootte nu juist moeten nemen. Wanneer we te kleine tijdsprongen zullen nemen zal de functie f te vaak geëvalueerd moeten worden en zal dit teveel berekeningen met zich mee brengen. Wanneer we te grote tijdsprongen nemen zal dit waarschijnlijk leiden tot een te grote onnauwkeurigheid of zelfs tot het exploderen van de simulatie.

Het is ook eenvoudig in te zien dat we hele grote stappen kunnen nemen wanneer er weinig of geen veranderingen zijn in de baan die een particle volgt, zoals bijvoorbeeld wanneer er enkel zwaartekracht op een particle inwerkt en deze in een rechte lijn naar onder zal vallen. Maar natuurlijk is ook het omgekeerde waar: wanneer we een particle beschouwen waar zeer veel krachten op inwerken en die dus bijgevolg een zeer ingewikkelde baan zal volgen, is het nodig om kleine tijdstappen te nemen om deze baan zonder een te grote fout te laten volgen.

Wanneer we dus een vaste stapgrootte willen nemen zal deze maximaal de grootte kunnen hebben die nodig is om nauwkeurig te blijven op de moeilijkste momenten. Hiervoor wordt een oplossing voorgesteld in [20]: We willen de stapgrootte h laten varieren tijdens het vooruitgaan in de tijd. We willen dus h groter maken op het moment dat de bewegingen dat toelaten zonder dat dat een te grote fout met zich meebrengt, en we willen h inkorten wanneer we merken dat de fout te groot wordt. We zullen het idee van het aanpassen van de stapgrootte introduceren in de euler-methode.

We hebben dus een bepaalde stapgrootte h_{oud} en we willen weten hoeveel we deze moeten of kunnen aanpassen om binnen een gewenste foutmarge e_{marge}) te blijven. We berekenen nu 2 schattingen van $x(t_0 + h)$:

1. De eerste schatting (x_1) bekomen we door een eulerstap te nemen van grootte h.

2. De tweede schatting (x_2) bekomen we door 2 eulerstappen te nemen met een grootte van h/2.

De twee waardes (x_1) en (x_2) verschillen nu van de echte waarde $x(t_0+h)$ met een nauwkeurigheid van $O(h^2)$. We kunnen de schaal van de huidige fout als volgt berekenen:

$$e = |x_a - x_b| \tag{4.13}$$

De nieuwe stapgrootte h_{nieuw} kunnen we nu bekomen op de volgende manier:

$$h_{nieuw} = \left(\frac{e_{marge}}{e}\right)^{\frac{1}{2}} h_{oud} \tag{4.14}$$

Het gebruik van veranderlijke tijdstappen is een zeer sterke verbetering bij het gebruik van integratie-technieken en er is reeds veel onderzoek naar gedaan. Maar voor deze thesis houden we het bij de basistechniek die we hierboven besproken hebben.

4.4 Conclusie

In dit hoofdstuk hebben we integratie-technieken besproken. Deze zijn noodzakelijk om een mass-spring systeem te laten bewegen over de tijd. We zijn begonnen met uit te leggen wat een ODE is en meer specifiek wat een initial value probleem is. We hebben dit hierna toegepast op een mass-spring systeem en aangetoond dat het laten bewegen hiervan eigenlijk overeen komt met het oplossen van een initial value probleem.

Hierna hebben we de drie meest gebruikte expliciete integratietechnieken gezien, deze zijn euler, midpoint en runge kutta. Deze zijn alledrie gebaseerd op de taylorreeks en hun nauwkeurigheid hangt af van het aantal elementen dat ze gebruiken uit deze taylorreeks. De geziene integratietechnieken zijn allemaal slechts benaderingen van de werkelijke oplossing en de nauwkeurigheid van deze benadering hangt af van de stapgrootte. Daarom zijn we dit hoofdstuk geëndigd met een sectie over de stapgrootte en hoe we een sterke stijging in efficientie kunnen bekomen door deze stapgrootte te laten varieren in tijd.

Nu we de werking van een mass-spring systeem kennen en weten hoe we deze kunnen laten bewegen met behulp van de in dit hoofdstuk geziene integratietechnieken hebben we alle elementen die we nodig hebben om bewegend kledingstuk te simuleren. De volgende 2 hoofdstukken zullen nu gaan over beperkingen en collision detection. Dit zijn enkel twee hulpmiddelen om het model dat we reeds hebben een meer realistisch karakter te geven.

Hoofdstuk 5

Beperkingen en Collision Detection

Omdat kledingstukken of andere stoffen zelden of nooit gewoon in een open ruimte zullen rondvliegen, zullen we in dit hoofdstuk enkele technieken bespreken die ons helpen om ons model te later interageren met andere objecten in een ruimte. Het eerste deel van dit hoofdstuk zal ons leren hoe we bepaalde particles in ons systeem kunnen beperken in hun bewegingsvrijheid, dit kan bijvoorbeeld nuttig zijn om bepaalde punten van ons kledingstuk vast te zetten (bijvoorbeeld om te simuleren dat dit vasthangt met wasknijpers). Een ander voorbeeld is een gordijn waarbij de bovenste punten enkel op een horizontale lijn mogen bewegen om zo te simuleren dat deze gordijn vast hangt aan een stok.

Het tweede deel van dit hoofdstuk gaat over collision detection, dit is het detecteren van een botsing tussen twee objecten, in ons geval het kledindstuk en de andere objecten in een omgeving. Zonder collision detection is het bijna onmogelijk om een realistische simulatie te maken. Het kledingstuk zal door de zwaartekracht meteen naar beneden vallen, en omdat we geen systeem hebben dat detecteert wanneer dit kledingstuk de vloer zal bereiken zal dit ook oneindig lang blijven vallen. Ook voor te vermijden dat andere objecten, zoals bijvoorbeeld een persoon het kledingstuk kan infiltreren, is collision detection broodnodig.

5.1 Beperkingen

We hebben in de vorige hoofdstukken gezien hoe een mass-spring systeem werkt en hoe we deze kunnen laten bewegen met behulp van integratietechnieken, maar tot nu toe heeft de gebruiker nog weinig mogelijkheden om de simulatie te controleren en te beinvloeden. De gebruiker heeft wel de mogelijkheid om de vorm van het mass-spring systeem te bepalen, en ook de krachten van de veren kunnen door de gebruiker ingesteld worden aan de hand van bepaalde parameters. We hebben ook mogelijkheden gegeven om bepaalde krachten zoals zwaartekracht of wind toe te voegen aan de simulatie. Maar we missen nog een mogelijkheid tot directe manipulatie van 1 of meerdere punten tijdens de simulatie en een mogelijkheid om beperkingen op te leggen aan de simulatie.

In deze sectie zullen we daarom enkele methodes geven die de gebruiker de mogelijk geeft om beperkingen op te leggen aan de simulatie. Het eerste deel zal gaan over het beperken van bepaalde particles, terwijl het tweede deel een meer algemene methode geeft om beperkingen op te leggen, namelijk beperkende krachten.

5.1.1 Beperken van particles

In het hoofdstuk over integratietechnieken hebben we gezien dat de krachten die op de particles inwerken verplaatsingen van deze particles teweeg brengen. Deze verplaatsingen zijn in een willekeurige richting en houden bijgevolg geen rekening met de beperkingen die een gebruiker wil opleggen aan de simulatie. Als we dus beperkingen willen opleggen aan een particle zullen we de snelheden en veplaatsingen van deze particle niet enkel mogen laten afhangen van de krachten die op deze particle inwerken maar ook van deze beperkingen. We zullen nu enkele mogelijke beperkingen van particles geven en een manier om deze beperking af te dwingen.

Vast punt

Een eerste beperking die we kunnen opleggen aan een particle is het volledig vastzetten van een particle, bijvoorbeeld om te simuleren dat een kledingstuk vasthangt met een wasknijper. We kunnen dit op verschillende manieren afdwingen. De meest eenvoudige methode is het negeren van het vaste punt bij het uitvoeren van de integratietechnieken op het mass-spring systeem.

Hierdoor verliezen we ook geen kostbare tijd met het berekenen van snelheden en verplaatsingen van een particle die toch niet mag bewegen. Sommige integratietechnieken laten dit echter niet toe en dan is er ook de mogelijkheid om de snelheid van de particle na elke integratiestap terug op nul te zetten of de massa van de particle op oneindig te zetten, wat eigenlijk hetzelfde resultaat heeft. [17]

Punt op een lijn

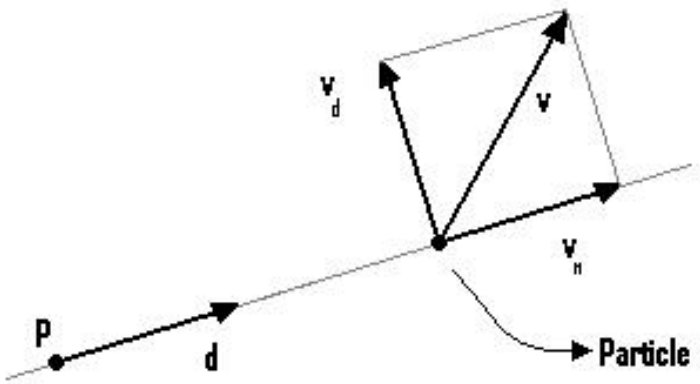
Het kan ook dat we een particle willen beperken in zijn bewegingsvrijheid tot een bepaalde lijn. Denk maar aan het hierboven gegeven voorbeeld van een gordijn. Een lijn wordt voorgesteld door een punt p een een richting d en we willen dus dat een particle op deze lijn blijft en dus bijgevolg enkel beweegt in de richting van d. Om deze beperking af te dwingen moeten we de snelheid die we verkrijgen via de integratietechnieken opsplitsen in twee aparten snelheden(Figuur 5.1:

1. De snelheid in de richting van de lijn(v_d):

$$v_d = (v \cdot d)d \tag{5.1}$$

2. De snelheid loodrecht op de lijn(v_n):

$$v_n = v_d - v \tag{5.2}$$



Figuur 5.1: Een particle beperkt op een lijn.

Om nu af te dwingen dat het punt op de lijn zal blijven moeten we de originele snelheid v vervangen door de snelheid v_d . De snelheid v_n die loodrecht staat op de lijn negeren we. Een nadeel van deze methode is dat er een deel van de krachten gewoon weg gelaten worden. Dit is in tegenstrijdigheid met de wet van behoud van energie. Maar ook in werkelijkheid zullen deze krachten niet leiden tot beweging maar eerder omgezet worden in wrijvingskrachten en bijgevolg leiden tot de productie van warmte. Aangezien dit weinig of geen belang heeft op de bewegingen van ons kledingstuk, kunnen we deze krachten dus zonder ernstige gevolgen weglaten.

Punt in een vlak

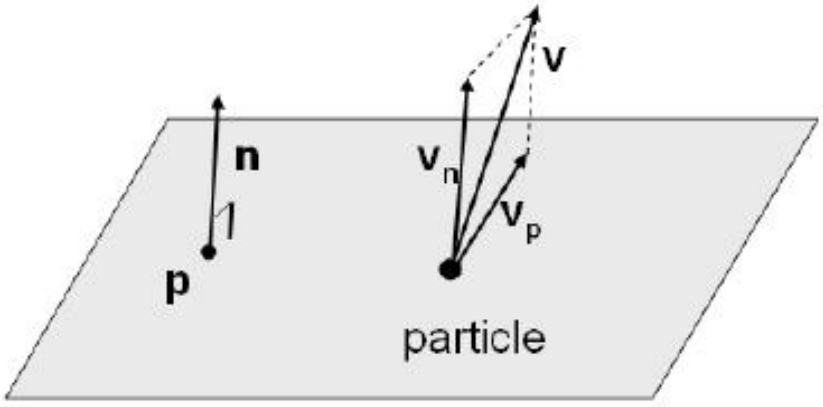
Een derde mogelijkheid is dat we een particle willen beperken tot het bewegen in een bepaald vlak, dit is eigenlijk een uitbreiding van het bewegen op een bepaalde lijn en we gebruiken dan ook ongeveer hetzelfde principe. Een vlak kunnen we definiëren door een punt p en een vlaknormaal n. Om er voor te zorgen dat een punt p in het vlak blijft moeten we nu net zoals bij een lijn de snelheidcomponenten in de richting van de vlaknormaal elimineren. We splitsen dus de snelheid v weer op in twee aparte snelheden(Figuur 5.2):

1. De snelheid parallel met het vlak(v_p):

$$v_p = (v \cdot n)n \tag{5.3}$$

2. De snelheid loodrecht op het vlak(v_n):

$$v_n = v_p - v \tag{5.4}$$



Figuur 5.2: Een particle beperkt in een vlak.

Om nu af te dwingen dat het punt in het vlak zal blijven moeten we de originele snelheid v vervangen door de snelheid v_p en de snelheid v_n die loodrecht staat op de lijn negeren.

Vast punt volgen

Het is ook mogelijk dat we een particle een bepaald traject willen laten volgen, zoals bijvoorbeeld een parabool of een cirkel. Of we willen dat een particle de muispointer gaat volgen zodat we kunnen trekken aan een kledingstuk. Er zijn twee verschillende technieken mogelijk om dit te verwezenlijken:

1. Aangezien we op elk tijdstip de positie kennen waar we de particle willen hebben, kunnen we de particle reduceren tot een vast punt zoals in het eerste onderdeel van deze sectie. Na elke tijdstap zetten zetten we de particle dus gewoon op de positie waar we hem willen hebben en negeren alle krachten die inwerken op deze particle. Deze manier kan echter onrealistische bewegingen van een particle met zich meebrengen wat dan weer als gevolg heeft dat het systeem onstabiel kan worden.

2. Een tweede manier om een particle een vast punt te laten volgen is het verbinden van deze particle met het vaste punt met behulp van een stijve veer met een rustlengte nul. Deze veer moeten we dan mee opnemen in ons mass spring systeem zodat ook zijn krachten gebruikt

worden tijdens de integratiestap. Op deze manier zal de particle steeds getrokken worden naar het vaste punt waar we hem willen hebben zonder dat hij perfect op deze plaats geforceerd wordt. Deze methode geeft meestal een realischer ogend resultaat maar een nadeel is dat de particle nooit exact op de positie zal staan waar we hem willen hebben.

5.1.2 Beperkende krachten

Een beperkende kracht in een particle systeem is een kracht die de versnellingen van de particles zo aanpast dat ons particle systeem voldoet aan de opgelegde beperkingen, zoals beschreven in [17]. Een voorbeeld van zo een beperking is bijvoorbeeld dat we twee punten in ons particle systeem altijd op een vaste afstand van elkaar willen hebben. Een ander voorbeeld is dat we een particle enkel willen laten bewegen op een bepaalde bol.

In het eerste voorbeeld zouden we deze beperking simpelweg kunnen afdwingen door een veer te gebruiken tussen deze 2 particles met als lengte de gewenste afstand, deze zal dan steeds een kracht uitoefenen die de particles op de vaste afstand probeert te houden. Ook in het tweede voorbeeld zouden we de particle met een veer kunnen verbinden aan het middelpunt van de bol, en deze veer als lengte de gewenste straal van de bol meegeven. Het probleem bij deze aanpak is echter dat de veer wel probeert om de gewenste beperking af te dwingen maar dat deze veer hierbij moet concureren tegen alle andere krachten die op de particles inwerken, zoals de zwaartekracht, andere veren en de trekkrachten van de gebruiker. De enige manier om deze veer te laten winnen is om deze veer een zeer hoge veerconstante te geven, waardoor zelfs kleine veranderingen in zijn lengte een zeer grote herstellende kracht teweeg brengen. [20] Dit is echter geen goede oplossing omdat strakke veren een onstabiel systeem met zich meebrengen.

Het grote probleem bij de hierboven vermelde oplossing is dat de herstellende krachten en de inwerkende krachten volledig onafhankelijk zijn van elkaar en dus elkaar zullen tegenwerken. De basistechniek om dit probleem te vermijden is gebaseerd op het principe dat we beter kunnen voorkomen dan genezen. In plaats van krachten toe te voegen die het systeem zullen herstellen van zodra dit niet meer voldoet aan de vooraf opgelegde beperkingen zullen we de krachten die reeds inwerken op het systeem zo aanpassen dat deze de opgelegde beperkingen niet schaden. We maken hierbij gebruik van energy functies, die we reduceren tot beperkende krachten (Deze methode wordt uitgebreid beschreven in [20]). In het hierboven gegeven voorbeeld van

een bol willen we de krachten die inwerken op deze particle dus zo aanpassen dat deze enkel versnellingen teweegbrengen in een richting op het raakvlak van deze bol. Een evenaardig probleem, waarbij we een particle op een cirkel willen houden is volledig uitgewerkt en berekend in [20]. We zullen hier in deze thesis niet verder op ingaan.

5.1.3 Conclusie

We hebben in deze sectie twee methodes gezien om beperkingen op te leggen aan ons systeem, in een eerste methode voegen we geen krachten toe maar zullen we de bewegingen en versnellingen van particles zo aanpassen dat deze voldoen aan de opgelegde beperkingen. In de tweede methode voegen we beperkende krachten toe die de inwerkende krachten op het systeem zullen verhinderen de opgelegde beperkingen te vermijden.

Beperkingen geven de gebruiker weer een heleboel nieuwe inputmogelijkheden en zijn een enorme hulp om onze simulatie een realistischer karakter te geven, maar we hebben nog een methode nodig om botsingen tussen ons kledingstuk en andere objecten te detecteren en op te lossen om een volledig bruikbaar systeem te verkrijgen. Hieraan wijden we onze laatste twee onderdelen van de real dynamics.

5.2 Collision detection

Collision detection is het detecteren van een botsing of een overlapping van twee gegeven objecten die beide bewegen door de ruimte. Het meeste onderzoek hierin is gedaan naar een botsing tussen twee vaste objecten, maar in ons geval zijn we vooral geïntresseerd in een botsing tussen een vervormbaar object, in dit geval een kledingstuk, en een vast object. Ook het detecteren van een botsing van een kledingstuk met zichzelf is van toepassing bij cloth simulation, we noemen dit self-collision.

Zonder collision detection zal het moeilijk zijn om een realistische simulatie te creëren, ons kledingstuk zou door alle objecten heen vallen en zou zelfs niet kunnen rusten op de vloer en bijgevolg oneindig blijven vallen. Dit geval van de vloer zouden we nog kunnen oplossen door op alle particles een beperking op te leggen van een minimum hoogte, maar wanneer er zich meerdere objecten in onze omgevingen zullen bevinden is het toch handig om een automatisch systeem te hebben om botsingen te detecteren en ook

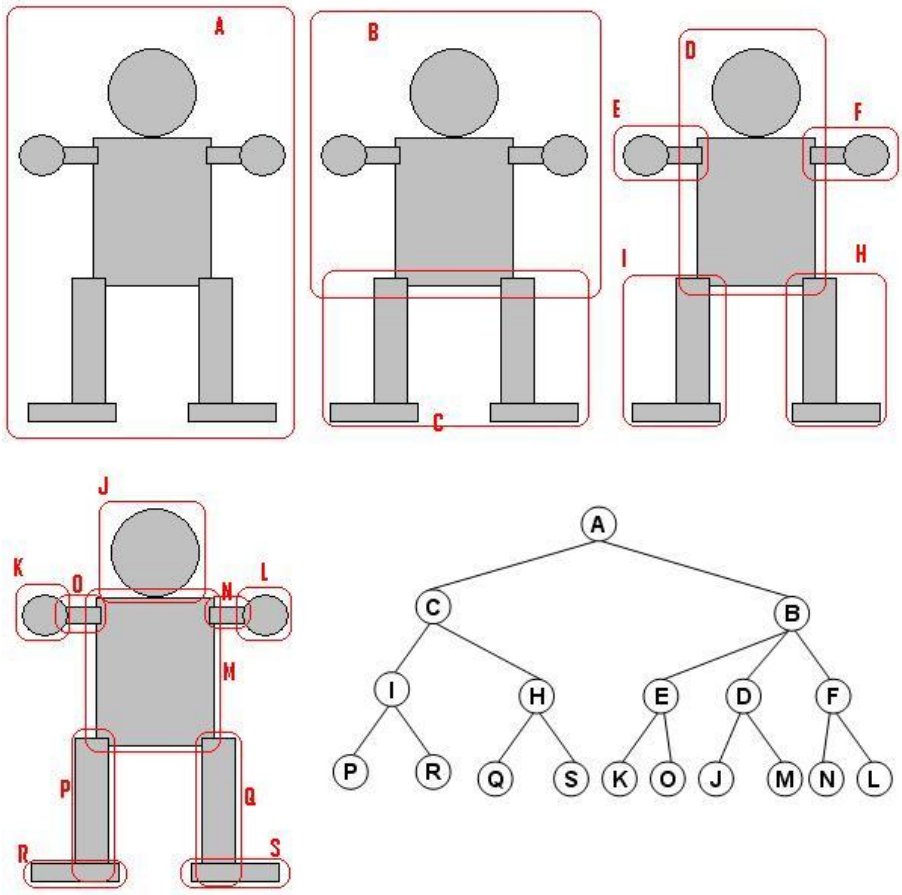
op te lossen.

Aangezien alle objecten in een computersimulatie zullen bewegen aan de hand van tijdstappen, komt het vinden van een botsing meestal neer op het detecteren van intersecties na elke tijdstap. We kunnen het vinden van deze intersecties indelen in 2 grote delen: De broad fase en de narrow fase [3, 7]. In de broad fase zullen we op een efficiente manier het aantal te berekenen botsingen beperken door alle objecten die duidelijk te ver van elkaar verwijderd zijn en dus bijgevolg niet met elkaar kunnen botsen te detecteren. Hiervoor worden vaak BSP-trees gebruikt [12] in combinatie met bounding volumes zoals bounding spheres [9] en bounding boxes [6, 16].

Zonder deze broad fase zouden we in een omgeving met N objecten, N^2 mogelijke botsingen moeten berekenen. Het is dus duidelijk dat deze broad fase vooral belangrijk is in een omgeving met zeer veel objecten, en dus niet zo belangrijk is in onze eenvoudige cloth-simulatie. Wij zijn vooral geïntresseerd in het vinden van een botsing tussen twee objecten, en dit is wat er gedaan wordt in de narrow fase. In deze fase worden de gebieden van een object gevonden die met elkaar in botsing zouden kunnen zijn en hierna worden de exacte posities van deze botsingen bepaald. Voor dit snel te laten verlopen zijn er twee veel gebruikte technieken in omgang, deze zijn Bounding Volume Hierarchies [11, 13, 22] en Spatial Subdivision. We zullen deze beide kort bespreken in de volgende secties.

5.2.1 Bounding Volume Hierarchies

Bounding Volume Hierarchies zijn een van de meeste gebruikte datastructuren bij collision detection. Ze worden vooral gebruikt bij niet vervormbare objecten maar kunnen mits enkele uitbreidingen ook gebruikt worden voor vervormbare objecten zoals kleding. Het principe werkt relatief eenvoudig: we bouwen een boomstructuur op van boundingvolumes die recursief de primitieven van een bepaald object zullen bevatten. De bovenste node zal dus een bounding volume bevatten rond het volledige object en de bladeren zullen bounding volumes zijn rond de primitieven van het object. Deze primitieven zijn de kleinste bouwstenen van het object en zullen meestal eenvoudige vormen zijn, zoals veelhoeken, kubussen of bollen. Een voorbeeld van een BVH zien we in afbeelding 5.3

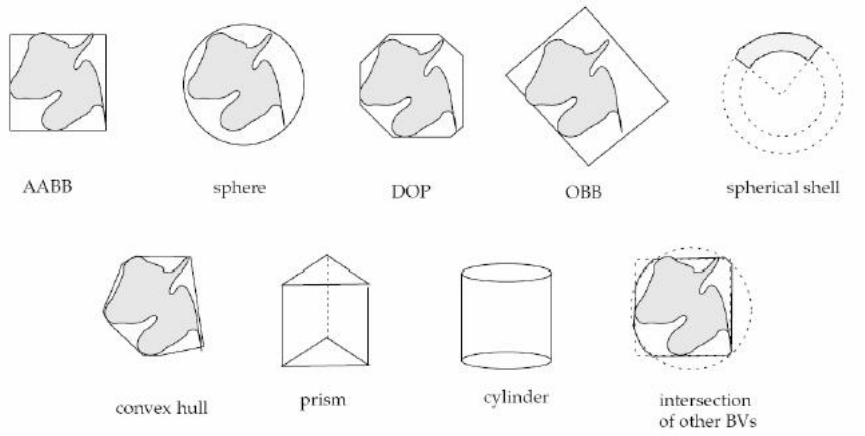


Figuur 5.3: BVH toegepast op een 2D-object met bijhorende boomstructuur.

De eigenschappen waaraan een BVH moet voldoen om gebruikt te kunnen worden voor collision detection zijn vermeld in [3] en zijn de volgende:

1. Elk niveau in de hiërarchie past strakker rond een deel van het object dan zijn ouder.
2. De kinderen van elke knoop moeten de objecten bevatten die zijn ouderknoop bevat.
3. De hiërarchie moet automatisch gecreëerd kunnen worden.
4. De bounding volumes moeten het object so nauwkeurig mogelijk omvatten.

Enkele mogelijke bounding volumes zijn vermeld in [21] en hier nog eens weergegeven in afbeelding 5.4



Figuur 5.4: De meeste gebruike bounding volumes.

Het construeren van deze Bounding Volume Hierarchies moet zoals hierboven reeds vermeld volledig automatisch kunnen gebeuren. Hiervoor zijn drie verschillende manieren in omgang, de meest populaire manier is top-down, de twee manieren die iets minder vaak worden gebruikt zijn bottom-up en insertion. We zullen in deze thesis niet dieper ingaan op de werking van deze methodes maar een uitgebreide en theoretische uitleg hierover kan teruggevonden worden in [21]. Een nadeel van deze methodes is dat het construeren van een BVH zeer lang kan duren. Bij niet vervormbare objecten is dit meestal geen knelpunt omdat deze BVH's voor het starten van de simulatie berekend kunnen worden en nadien steeds gebruikt kunnen worden. Wanneer we echter zoals in ons geval te maken krijgen met vervormbare objecten zouden we na elke tijdstap opnieuw een BVH moeten construeren. Aangezien dit vaak enkele seconden kan duren is dit geen optie in real-time simulaties en zullen we dus een andere oplossing nodig hebben. Deze bestaat ook en zal de BVH herstellen na elke tijdstap in plaats van deze volledig opnieuw te construeren.

Herstellen van een BVH

We zullen bij deze methode nog altijd beginnen met een BVH te construeren alvorens de simulatie start. Hierbij is het belangrijk dat we zorgen voor

een goede structuur van onze boom, want deze structuur zal tijdens het herstellen niet meer veranderen. En aangezien we bij het herstellen steeds gaan trachten een zo klein mogelijk deel van onze boom aan te passen zal een goede structuur waarbij de primitieven goed gebalanceerd zijn zeker een groot voordeel zijn.

Wanneer we nu enkele tijdstappen verder zijn in onze simulatie zal de BVH niet meer kloppen met ons vervormbaar object en moeten we de BVH dus gaan herstellen. De meest eenvoudige manier is bottom-up. Hierbij starten we met het zoeken naar bladeren in onze boom die niet meer correct zijn, wat dus wil zeggen dat een primitief zich niet meer volledig in zijn bounding volume bevind. We passen dan het bounding volume zo aan dat deze het primitief terug omhult waarna we de boom bottom-up doorlopen en elke parentnode aanpassen aan dit nieuwe bounding volume.

Er zijn nog vele andere manieren voor het herstellen van een BVH in omgang maar deze vallen buiten het bestek van deze thesis. We geven hier enkel een kort overzicht: Larson et al. [10] vergelijken een bottom-up techniek met een top-down techniek, en stellen aan de hand van hun bevindingen een hybride techniek voor. Brown et al. [4] stellen het gebruik van een priority queue voor als toevoeging bij de bottom-up techniek.

Met behulp van deze BVH's kunnen nu botsingen tussen twee objecten zeer snel gevonden en gelokaliseerd worden. Ook voor het detecteren van een botsing met zichzelf (self-collision) kunnen we BVH's gebruiken. Een methode die hiervoor gebruikt kan worden is beschreven in [17], hier vinden we ook de pseudocode om botsingen te vinden met behulp van deze BVH's.

5.2.2 Spatial subdivision

Bij bounding volume hierarchies werden de objecten in de omgeving hierarchies opgesplits tot hun primitieven. Bij spatial subdivision gaan we net andersom te werk. We gaan de omgeving rond het object of zelfs de volledige omgeving gebruikt bij de simulatie opdelen in kleinere hokjes waarin we dan de primitieven van de objecten op een bepaalde manier gaan plaatsen. Een voordeel van deze methode is dat objecten mogen opsplitsen of samengevoegd worden omdat de opsplitsing van de ruimte niet afhankelijk is van de lokatie en de vorm van het object.

Ganovelli et al. [5] stellen een techniek voor waarbij ze een buckettree construeren rond een object. Ze starten met een AABB (zie figuur 5.4) rond het object te plaatsen, die zal overeenkomen met de wortel van een octree.

Deze zullen ze dan opdelen in 8 octanten, die ze dan allemaal weer opslitsen in 8 octanten. Dit herhalen ze tot op een bepaald niveau. Op deze manier creëren ze een octree, de bladeren van deze octree noemen we de buckets. Nu zullen ze alle primitieven van het object in de juiste bucket steken waarna collision detection kan gebeuren zoals bij een HBV zoals hierboven beschreven. In [5] kan een meer technische uitleg gevonden worden alsook een vergelijking van deze techniek met andere technieken.

Een andere techniek, voorgesteld door Teschner [15] gebruikt spatial haching voor collision detection. Hier word de omgeving simpelweg ingedeeld in 3d-gridcellen en word gebruik gemaakt van een hashfucntie om deze gridcellen op een hashtabel af te beelden. De juiste werking van deze techniek zullen we hier niet bespreken maar kan terug gevonden worden in [15]

5.2.3 Self-collision

Self-collision is het detecteren en ook oplossen van een botsing tussen een vervormbaar object met zichzelf. Dit is een situatie die zich bij het simuleren van kleding vaak kan voordoen. Het detecteren van self-collision bij kleding is tot op heden echter nog altijd 1 van de moeilijkste onderdelen bij cloth-simulation. Een eerste reden hiervoor is dat kleding in de meeste gebruikte modellen, alsook in ons mass-spring model, oneindig dun is. Hierdoor is het vaak moeilijk te bepalen of een punt van het kledingstuk na een bepaalde tijdstap door het kledingstuk heen is gegaan of zich reeds aan die kant bevond. Om dit te vermijden moeten we onze kleding een realistische dikte geven en hier rekening mee houden tijdens de collision detection. [8]

Een tweede probleem kan ontstaan wanneer we een van de technieken willen gebruiken die in de vorige sectie besproken werden. Doordat we botsingen gaan zoeken tussen een object met zichzelf zullen we de primitieven van een object moeten testen tegen andere primitieven van datzelfde object. Hierdoor zullen we dus ook primitieven testen die eigenlijk aan elkaar vast zitten en dus constant in een soort van botsing verkeren. Dit is meestal wel te detecteren door een kleine extra test in te voeren, maar wanneer we bijvoorbeeld zoals bij de BVH's met bounding volumes gaan werken zullen deze bounding volumes van aangrenzende primitieven elkaar overlappen en bijgevolg een heel aantal overbodige tests met zich meebrengen. Volino et al. [18] stellen een methode voor om deze overbodige testen te vermijden.

Een nog groter probleem dan self-collision detecteren is het realistisch reageren wanneer er een self-collision plaats vindt. Dit is vaak zeer kostelijk

en geavanceerd en we zullen er hier dan ook niet dieper op ingaan.

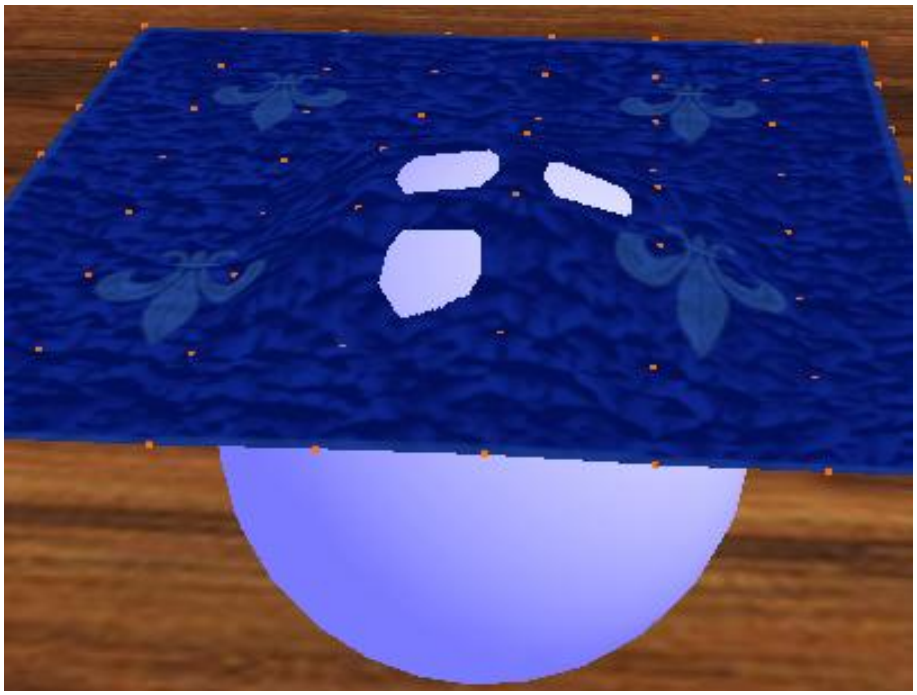
Doordat self-collision zo moeilijk en kostelijk is, word dit vaak weggelaten in real-time simulaties en proberen we dit zo veel mogelijk te vermijden door bepaalde beperkingen op te leggen aan onze cloth.

Nu we enkele methodes hebben gezien om botsingen te detecteren tussen bepaalde objecten hebben we nog een manier nodig om onze simulatie op een realistische manier te laten reageren op deze botsingen. We noemen dit collision response en zullen hier dieper op in gaan in de volgende sectie.

5.3 Collision responce

Collision response is het simuleren van hetgene wat er zal plaatsvinden wanneer we een botsing gedetecteerd hebben. Wanneer we een botsing tussen twee niet vervormbare objecten detecteren die beide een snelheid en een rotatie hebben is collision response vaak zeer complex. Deze botsing zal namelijk een invloed hebben op zowel de rotatiesnelheden als de bewegingssnelheden van deze objecten en zal afhangen van de materiaaleigenschappen en de positie van de botsing. Een blad papier dat met dezelfde snelheid als een rubberen bal op een betonnen vloer zal vallen zal een volldig ander resultaat met zich meebrengen. We kunnen dus concluderen dat er zeer veel bij komt kijken wanneer we een botsing tussen twee niet vervormbare objecten willen oplossen op een realistische manier.

Gelukkig kunnen we bij het simuleren van collision responce bij een botsing tussen een kledingstuk en een vast object enkele vereenvoudigingen aanbrengen zonder veel realisme te verliezen. We kunnen collision detection en zeker collision response zeer sterk vereenvoudigen wanneer we enkel puntmassa's gebruiken als de te controleren objecten. Puntmassa's hebben in tegenstelling tot objecten die opgebouwd zijn uit veelhoeken geen rotatie en bijgevolg ook geen rotatiesnelheid. Ook zal er geen positie van de botsing bepaald moeten worden. Wanneer we dus collision detection en response op ons mass-spring systeem willen uitvoeren kunnen we dit zeer sterk vereenvoudigen door alleen rekening te houden met de punt-massa's. Een nadeel van deze aanpak is dat we de veren en dus ook de veelhoeken die zich tussen deze punt-massa's bevinden negeren bij de collision detection en deze dus bijgevolg kunnen gepenetreerd worden door scherpe voorwerpen of zelfs een bol. Een voorbeeld hiervan is te zien op afbeelding 5.5.



Figuur 5.5: Fout bij collision detection van clothes.

Een tweede vereenvoudiging die we kunnen invoeren is een gevolg van de botskracht van kleding. Aangezien deze zeer klein is kunnen we deze zonder veel verlies van realisme negeren en er dus vanuit gaan dat wanneer een kledingstuk met een bepaalde snelheid tegen een vast object komt, het tegen dat object zal blijven zolang er geen andere krachten op inwerken.

Wanneer we nu deze twee vereenvoudigingen in beschouwing nemen zal het zeer eenvoudig zijn om collision response te berekenen bij het detecteren van een botsing tussen een kledingstuk en een niet vervormbaar voorwerp. Tijdens de collision detection houden we dus enkel rekening met de punt-massa's van ons mass-spring systeem. Om ons kledingstuk een realistische dikte te geven construeren we rond elke punt-massa een virtuele bol met als straal de helft van de gewenste dikte. We beschouwen deze bollen nu elk als een apart object en zoeken naar botsingen zoals eerder in deze sectie reeds besproken is.

Als we een botsing detecteren van zo een virtuele bol met een bepaald vlak gedefiniëerd door een punt p en een vlaknormaal n dan splitsen we de snelheid van de betreffende particle (v_p) op in de volgende twee snelheden(zie

figuur **??**:

1. De snelheid parallel met het vlak(v_p):

$$v_p = (v \cdot n)n \tag{5.5}$$

2. De snelheid loodrecht op het vlak(v_n):

$$v_n = v_p - v \tag{5.6}$$

We vervangen hierna de huidige snelheid van de particle door de snelheid parallel met het vlak. Wanneer we deze methode toepassen op elke particle van ons mass-spring systeem zal dit er voor zorgen dat een kledingstuk dat terecht komt op een vast object, langs dit object zal afglijden zonder er doorheen te vallen. En dit is voor de meeste simulaties goed genoeg omdat een realistisch uitzicht meestal belangrijker is dan een werkelijk realistische beweging.

5.4 Conclusie

We hebben in dit hoofdstuk gezien hoe we beperkingen kunnen opleggen aan een mass-spring systeem en hoe we collision detection en collision response kunnen uitvoeren in een mass-spring systeem. Dit waren de twee laatste onderdelen die nodig waren voor het visualiseren van kleding met behulp van real dynamics. We beëindigen hiermee dan ook het eerste deel van deze thesis en zullen de rest wijden aan het onderzoek naar Fake dynamics.

Deel II

Fake Dynamics

Hoofdstuk 6

Introductie

6.1 Wat zijn fake dynamics?

In het eerste grote deel van deze thesis hebben we besproken hoe kleding in het algemeen gemodeleerd wordt met behulp van mass-spring systemen. We hebben gezien hoe deze mass-spring systemen tot leven komen door integratie-technieken te gebruiken en hoe we beperkingen kunnen opleggen aan de bewegingen. We zijn geëindigd met een hoofdstuk over collision detection en collision response waarin we besproken hebben hoe we de kleding kunnen laten interageren met de omgeving. Deze methodes zijn reeds allemaal vanaf het begin van cloth simulation in gebruik en er is dan ook enorm veel onderzoek naar gedaan. Een gevolg hiervan is dat er momenteel enorm veel verbeteringen en uitbreidingen bestaan op de technieken besproken in het eerste hoofdstuk.

Wanneer we op zoek gaan op het internet naar cloth simulation zullen we duizenden artikels vinden over mass-spring systemen, integratietechnieken en collision detection. Hetgeen we in het eerste deel van deze thesis gezien hebben is dan ook slechts een zeer beknopte basis van al hetgene dat er te vinden is over cloth simulation. Maar omdat bijna alle te vinden technieken een uitbreiding zijn van hetgene wat we hier in het eerste deel besproken hebben, hebben deze technieken allemaal 1 belangrijke eigenschap gemeen: ze zijn allemaal gebaseerd op 'real dynamics'. Dit wil zeggen dat we de wetten uit de fysica, en dan vooral uit de dynamica gebruiken om een realistische simulatie te bekomen.

Bij mass-spring systemen gebruiken we veren die krachten uitoefenen op

punt-massa's, we hebben zwaartekracht, windkracht en viscous drag geïntroduceerd. Bij de integratie-technieken zullen we deze krachten omzetten naar versnellingen, snelheden en bewegingen. Bij collision detection hebben we het over botsingen gehad en reacties op deze botsingen. Het is dus heel duidelijk dat bijna heel het eerste deel en bijgevolg bijna alle technieken die gebruikt worden bij cloth simulation gebaseerd zijn op de wetten van de dynamica. Dit is eigenlijk ook de meest logische aanpak, want ook in de werkelijkheid onstaan bewegingen als een gevolg van de wetten van de dynamica.

Maar nu onstaat de vraag of het wel nodig is om een omgeving die onderhevig is aan de wetten van de dynamica ook te gaan simuleren door deze wetten te implementeren. Wanneer we met een computer animaties willen maken kan het zelfs zo zijn dat we helemaal niet willen dat onze animatie volledig correct beweegt volgens de wetten van de fysica. Dit is de vraag die we in deze thesis gaan proberen te beantwoorden. We zullen gaan onderzoeken of er geen andere methode bestaat om kleding te visualiseren, een methode waarbij we al deze wetten overboord kunnen gooien. We noemen deze aanpak 'fake dynamics', omdat we gaan proberen om een dynamisch gedrag te 'faken' zonder enige dynamica te implementeren.

6.2 Reeds bestaand onderzoek naar 'fake dynamics'

Wanneer we op zoek gaan naar 'fake dynamics' van kleding op het internet, zullen we in tegenstelling tot 'real dynamics' bijna niets vinden. Dit kan verschillende redenen hebben, zo zou het bijvoorbeeld kunnen dat nog nooit iemand op het idee gekomen is om deze aanpak uit te voeren en er dus nog nooit onderzoek naar gedaan is. Dit is echter hoogst onwaarschijnlijk in de informaticawereld, waarbijna alles bedacht en onderzcoht is. Het is ook mogelijk dat er wel onderzoek naar gedaan is maar dat er nooit bruikbare resultaten behaald zijn. Ook dit is echter zeer onwaarschijnlijk omdat er zelfs zonder bruikbare resultaten, informatie te vinden zou moeten zijn over dit onderzoek.

Een meer voor de hand liggende reden is dat het wel degelijk al vaker onderzocht en gebruikt is, maar dat dit toen niet onder de naam 'fake dynamics' viel. Een reden hiervoor zou kunnen zijn dat de term 'fake dynamics' bij het simuleren van vervormbare objecten pas geïntroduceerd is in 1997 door Ro-

nen Barzel in de paper "Faking Dynamics of Ropes and Springs". [2]. In de bibliography van deze paper staat ook een citatie naar een artikel uit 1986 van Jerry Weil [19]. Hierin wordt ondermeer besproken hoe we een doorhangend stuk stof dat opgehangen is aan enkele vaste punten kunnen benaderen door enkel wiskundige formules te gebruiken. Dit is een vorm van 'fake dynamics' en we kunnen dus concluderen dat 'fake dynamics' reeds vaker gebruikt zijn en dit met zelfs redelijk goede resultaten.

De twee papers die hierboven vermeld zijn, zijn de basis van deze thesis. De eerste paper beschrijft hoe fake dynamics succesvol zijn toegepast op touwen en veren in de film 'toy story'. Omdat dit 1 van de enige papers is die effectief over fake dynamics gaat en de manier waarop we deze kunnen gebruiken om animaties te genereren, zullen we het eerste hoofdstuk van dit deel hieraan wijden. De tweede paper is reeds uit 1986 en dus een beetje verouderd, maar omdat hierin succesvol fake dynamics gebruikt zijn voor het visualiseren van kleding zullen we enkele algoritmes uit deze paper gebruiken als basis bij de 'fake dynamics' van kleding.

6.3 Voor- en nadelen van Fake Dynamics

Aangezien er reeds enorm veel onderzoek is gedaan naar 'real dynamics' en hiermee dus enorm goede resultaten behaald kunnen worden is de nood naar een volledige andere aanpak niet altijd duidelijk. We geven daarom hier de voor- en nadelen van Fake dynamics en ook waar deze wel of niet bruikbaar zijn.

voordelen

1. Omdat we niet werken met veren en krachten, zijn er geen arbeidsintensieve integratie technieken nodig en kunnen we bijgevolg veel grotere datastructuren gebruiken.

2. Omdat we bij 'Fake dynamics' zullen werken met keyframes en omdat we geen rekening moeten houden met de wetten van de dynamica is er een directe manipulatie mogelijk. De animator kan exact bepalen op welk tijdstip hij welke situatie wil bereiken en ook de bewegingen zijn veel exacter te controleren. Bij 'Real dynamics' zullen hiervoor (vaak moeilijke) beperkingen opgelegd moeten worden.

3. In animantiefilms of cartoons worden bewegingen vaak overdreven, denk maar aan de gezichtexpressies van bepaalde karakters. Deze expressies zullen zelden volledig realistisch zijn, maar het blijft wel altijd duidelijk dat het een gezicht is en welke expressie er uitgebeeld wordt. Bij het animeren van bewegende stoffen willen we vaak hetzelfde effect: We willen de bewegingen overdrijven om meer aandacht te vestigen op deze bewegingen maar natuurlijk willen we wel een zekere graad van realisme behouden. Deze resultaten zijn zeer moeilijk te bekomen wanneer we 'real dynamics' gebruiken. Met 'fake dynamics' is het echter zeer eenvoudig om niet realistische aminaties te maken.

nadelen

1. Doordat we een dynamisch karakter willen creëren door enkel wiskundige formules te gebruiken is de implementatie van 'fake dynamics' vaak veel moeilijker dan 'real dynamics'. Bij 'real dynamics' is de implementatie meestal zeer straight-forward en bekomen we al goede resultaten door een eenvoudig mass-spring systeem te construeren.

2. Het systeem dat in deze thesis zal besproken worden geeft enkel een mogelijkheid om keyframes te construeren en tussen deze keyframes verschillende soorten overgangen te gebruiken. We krijgen dus geen real-time simulatie waarbij krachten uitgeoefend kunnen worden en waarbij collision detection gedaan kan worden.

3. Een tweede nadeel van het gebruik van keyframes is dat een mooie, realistisch ogende animatie veel bekwaamheid en ervaring van een animator zal vergen.

Deze voor- en nadelen geven duidelijk aan dat het gebruik van 'fake dynamics' niet altijd veroorloofd is. Bij real-time aminaties of computergames, waarbij een gebruiker directe input wil kunnen geven door bijvoorbeeld tegen een gordijn te lopen of aan een kledingstuk te trekken zal het gebruik van 'real dynamics' meestal een betere keuze zijn. Ook wanneer we een realistisch resultaat willen bekomen voor bijvoorbeeld een wetenschappelijk onderzoek zal het gebruik van 'real dynamics' het beste zijn.

Wanneer we echter een animatie willen maken waarbij het realisme niet zo belangrijk is en waarin we graag wat overdrijven met de bewegingen of het wapperen van een gordijn dan zullen de 'fake dynamics' handiger zijn als de

'real dynamics'. Ook wanneer er een directe manipulatie nodig is of wanneer we een bepaalde situatie moeten bereiken in een bepaalde frame van onze animatie zal het gebruik van 'fake dynamics' de beste keuze zijn.

6.4 Verdere verloop

In hoofdstuk 2 van dit deel van de thesis zullen we starten met een bespreking van de paper "Faking Dynamics of Ropes and Springs"van Ronen Barzel [2]. Deze paper gaat over het gebruik van 'fake dynamics' bij het animeren van touwen en veren. Er worden keyframes gebruikt om een animatie te genereren. Om een keyframe te construeren zullen ze eerst een algemene rustvorm bepalen waarop ze nadien een grote vervorming en een golf-vervorming uitvoeren. We zullen bespreken hoe deze verschillende vormen geconstrueerd kunnen worden, en hoe deze samen gevoegd kunnen worden om frames van een animatie te worden.

De objecten die door Ronen Barzel besproken worden zijn allemaal 2d-objecten. In hoofdstuk 3 gaan we proberen om de aanpak die Barzel heeft ingevoerd in zijn paper uit te breiden naar 3D-objecten, en meer specifiek op kleding. Dit wil zeggen dat we een methode gaan zoeken om keyframes te construeren zoals dit ook in zijn werk gedaan werd. We bepalen dus een natuurlijk rustvorm, een grote vervorming en een golf-vervorming die we hierna gaan samenvoegen om keyframes te construeren.

In hoofdstuk 4 gaan we de geconstrueerde keyframes gebruiken om een animatie te creëren, we zullen zien hoe we frames kunnen toevoegen aan een animatie, welke verschillende overgangen er mogelijk zijn tussen de frames en hoe we golven kunnen toevoegen aan de animatie en hoe we de instellingen van deze golven kunnen weergeven en aanpassen, in hoofdstuk 5 bespreken we de behaalde resultaten en geven we een conclusie.

Hoofdstuk 7

Faking dynamics of Ropes and springs

In dit hoofdstuk zullen we de aanpak bespreken die door Barzel wordt voorgesteld in de paper 'Faking dynamics of Ropes and Springs'. We starten met enkele redenen te geven die de manier van de aanpak met behulp van keyframes in sommige gevallen motiveren en verklaren waarom 'real dynamics' in deze gevallen minder bruikbaar zijn. In een volgende sectie motiveren we het gebruik van de drie vormen(rust vorm, grote vervorming, golf-vervorming) en bespreken hoe we deze kunnen construeren voor een doorhangend koord en een los koord. We zullen hierna uitgebreid ingaan op het construeren van de rust-toestand van een doorhangend touw (ook wel catenary genoemd) omdat dit de basis zal zijn voor het uitbreiden naar 3d-objecten in het volgende hoofdstuk.

Het systeem waarbij we gebruik maken van verschillende lagen of vormen die bij elkaar worden gevoegd kan ook voor zeer veel andere modellen gebruikt worden. We zullen dus een meer algemene methode geven om dit systeem toe te passen. We eindigen dit hoofdstuk met enkele resultaten van het systeem te geven en een conclusie.

7.1 Inleiding

Bij het produceren van animaties is het essentieel dat animators op een eenvoudige wijze de positie en de timing kunnen aanpassen met een accuraatheid tot op de frame om aan de wensen van een producer of klant te voldoen[2].

Animatie systemen die gebruik maken van keyframes zijn het meest geschikt voor deze taak. Om deze reden introduceert Barzel een systeem dat gebruik maakt van keyframes om een animatie van flexibele objecten te genereren. Hun doel is om om te kunnen gaan met elke graad van vrijheid die een flexibel object heeft en op deze manier een zo realistisch mogelijk resultaat te bekomen. En op hetzelfde moment willen ze een directe manipulatie mogelijk maken.

Om dit doel te bereiken stelt Barzel een methode voor waarbij de vorm van het vervormbare object bekomen wordt door drie lagen vervormingen bij elkaar op te tellen. Deze drie lagen zijn zo gekozen dat het uiteindelijke resultaat grote overeenkomsten toont met een bewegend vervormbaar object uit de realiteit. Door de parameters van de verschillende lagen aan te passen kan een animator het realisme van de beweging verhogen of net verlagen naargelang het gewenste resultaat. Natuurlijk speelt de ervaring en bekwaamheid van de animator hierbij een grote rol.

Het gebruik van van 'real dynamics' bij ditzelfde soort animaties is vaak een stuk moeilijker omdat er geen directe controle mogelijk is. Controle over de bewegingen zal zoals we in het vorige deel gezien hebben gebeuren door beperkingen op te leggen, maar deze lenen zich niet tot een snelle en directe controle. Ook het genereren van niet fysische bewegingen en het editen per frame of per positie is bijna onmogelijk met 'real dynamics'.

7.2 De 3 lagen

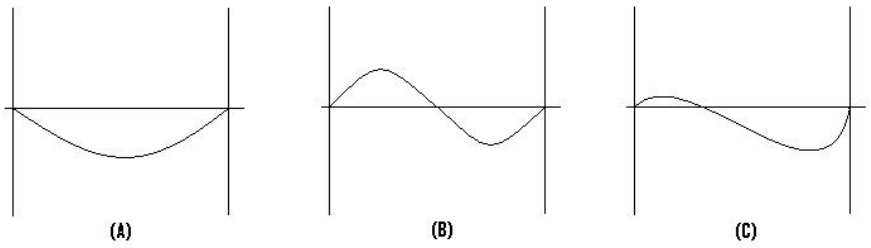
Voor het computertijdperk werden tekenfilms vaak frame per frame getekend. Toen de eerste animatiefilms ontstonden werd deze aanpak in het begin voor een deel overgenomen en er werd nog vaak frame per frame gewerkt. Om bijvoorbeeld een animatie te maken van een bewegend touw werd dit touw geconstrueerd door een aantal beweegbare controlepunten die in elke frame door een animater zo werden geplaatst dat dit uiteindelijk een realistische beweging met zich meebracht. Deze methode was logischerwijs een zeer omslachtig werk en het was een zeer moeilijke opgave om op deze manier een overtuigende beweging te creëren.

Het controleren van elk controlepunt apart is namelijk zeer vaak overbodig omdat deze punten zo goed als altijd afhankelijk van elkaar zullen bewegen. Om het werk van een animator te vereenvoudigen wordt er door Barzel een methode voorgesteld die rekening houd met deze afhankelijkheid

van de controlepunten. Hij stelt vast dat de vorm van een vervormbaar object kan beschreven worden door een vervorming van de rustpositie van dit object waarbij een golf-vervorming toegevoegd wordt. Om dus een bepaalde vorm van een object te creëren gaan we als volgt te werk:

1. Bepaal de vorm van het object in rust-toestand.
2. Voer hier een algemene vervorming op uit.
3. Voer hier een golf-vervorming op uit.

Omdat we geen perfect realistische animaties verwachten kunnen we de algemene vervorming en de golf-vervorming bekomen door eenvoudige operaties. Aan de hand van parameters zal het mogelijk zijn om deze vervormingen aan te passen zoals de animator dit wenst. Om het idee te verduidelijk geven we in afbeelding 7.1 een eenvoudig voorbeeld van een doorhangend touw. De rusttoestand van dit touw zal er uit zien zoals we zien in (a). Voor de duidelijkheid veronderstellen we dat er geen algemene vervorming is, zodat we duidelijk het effect zien van de golfvervorming die we zien in (b). Wanneer we deze golfvervorming nu uitvoeren op de rustvorm bekomen we het resultaat dat we zien in (c). Wanneer we een animatie willen creëren waarbij een golf door een doorhangend touw gaat, moeten we enkel nog de fase van onze golf in elke frame wat opschuiven. Dit voorbeeld geeft de eenvoud en toepasbaarheid aan van de aanpak in deze drie lagen.



Figuur 7.1: Een doorhangend touw waarop een golf-vervorming is uitgevoerd. (a) Een doorhangend touw (b) Een golf (c) Rustvorm gecombineerd met de golf

In de paper van Barzel worden vier modellen besproken die eenvoudig gemodeleerd kunnen worden door het systeem met de drie lagen. We zullen

de eerste drie hier beknopt herhalen, waarna we een sectie wijden aan het model van het doorhangend koord omdat dit zeer belangrijk is bij de uitbreiding naar 3D-modellen. De 3 modellen die besproken zullen worden zijn een doorhangend touw, een touw dat ophangt aan 1 punt en een gekruld touw(zoals de kabel aan een telefoontoestel). Bij elk van deze 3 modellen zullen we een methode geven voor de rust-toestand, de algemene vervorming en de golfvervorming te bepalen.

7.2.1 Doorhangend touw

Met een doorhangend touw bedoelen we een touw dat aan zijn twee punten vasthangt, zoals we zien in afbeelding 7.1(a). Het is mogelijk dat deze punten volledig vast zijn, maar dit is niet noodzakelijk. Het is ook mogelijk dat deze punten door de gebruiker gemanipuleerd kunnen worden of dat het touw met een van deze punten bevestigd is aan een ander (bewegend) object. Wel moeten we op elk moment in de simulatie de exacte positie van deze twee punten kennen. Ook de lengte van het touw moet gekend zijn.

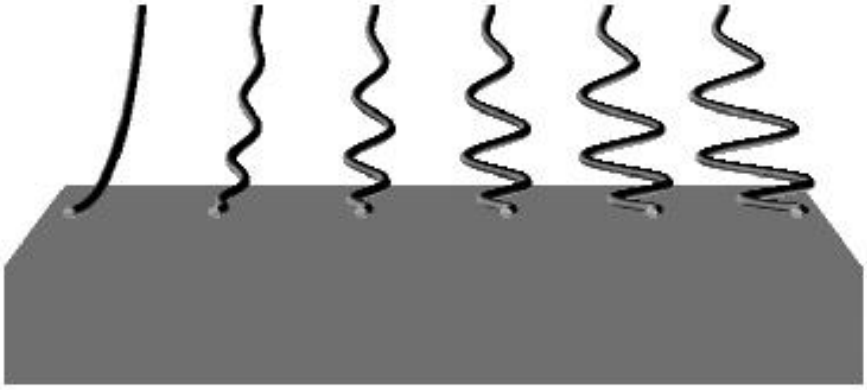
1. **Rusttoestand**: De rusttoestand van een touw dat ophangt aan twee punten en dat beïnvloed wordt door de zwaartekracht zal de vorm hebben van een catenary. Een catenary kan wiskundig bekomen worden wanneer we de twee eindpunten en de lengte van het touw kennen. De volledige wiskundige berekening van een catenary wordt gegeven in de volgende sectie.

Figuur 7.2: (a) Doorhangende touwen met gelijke lengte en verschillende eindpunten. (b) Slingeren van links naar rechts van een touw. (c) Vooruit en achteruit schommelen van een touw.

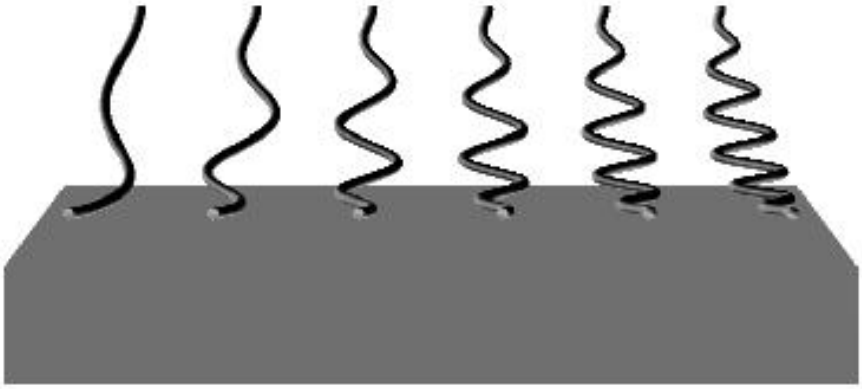
2. **Algemene vervorming**: Op een doorhangend touw zijn slechts twee grote vervormingen van toepassing: Het kan slingeren van links naar rechts en het kan vooruit en achteruit schommelen, deze twee vervormingen zijn te zien op afbeelding 7.2 (b) en (c). Omdat deze twee eigenschappen niet van toepassing zijn wanneer we het model zullen uitbreiden naar kleding zullen we niet dieper ingaan op de algoritmes achter dit slingeren en schommelen.

3. **golf-vervorming**: Een golf-vervorming kan zeer eenvoudig bekomen worden door een sinus of een cosinusfunctie. Deze functies hebben vier parameters: fase, magnitude, frequentie en azimuth, zie ook afbeeldingen 7.5 7.3 en 7.4. Om een bepaalde golf vooruit of achteriut te laten bewegen over ons koord kunnen we eenvoudigweg de fase aanpassen over de tijd. Omdat we de eindpunten van onze rusttoestand niet willen beïnvloeden is het ook nodig om onze golf te laten afzwakken naar deze eindpunten toe, hiervoor nemen we de afstand van elk punt tot het dichts bijliggende eindpunt, deze waarde delen we door de helft van de afstand tussen de twee eindpunten. Op deze manier bekomen we een waarde tussen 0 en 1 die de kracht aangeeft van de golf op die bepaalde plek. Wanneer we nu de Y-waardes van de golf vermenigvuldigen met deze waardes tussen 0 en 1 bekomen we een golf die gedempt wordt naar de eindpunten toe 7.7. Theoretisch is het mogelijk om meerdere

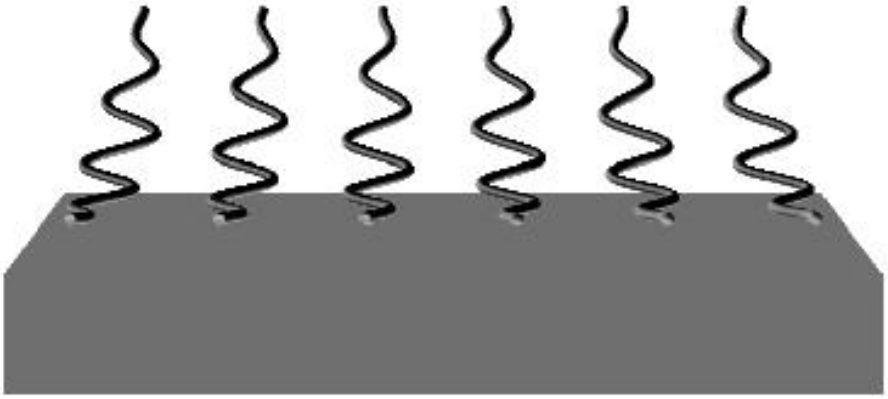
golf-vervormingen samen te gebruiken, maar dit is meestal geen meerwaarde voor de animatie.



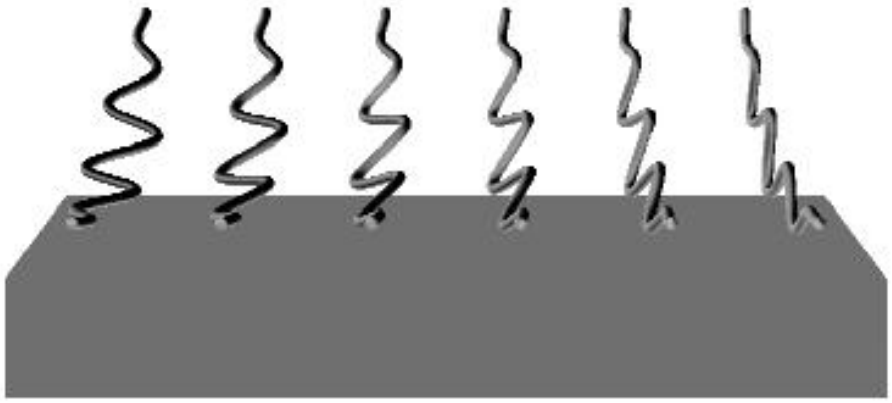
Figuur 7.3: Een golf-vervorming met verschillende magnitudes.



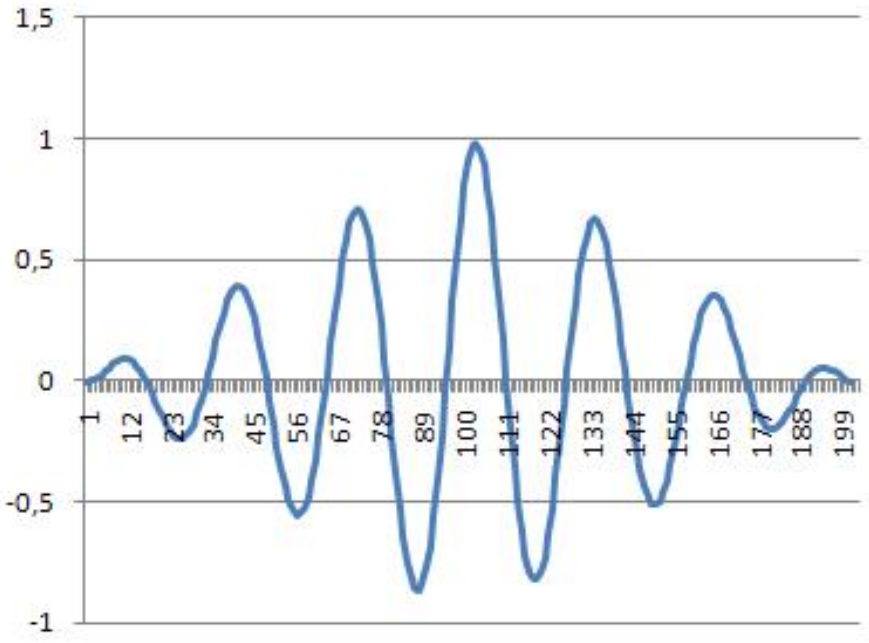
Figuur 7.4: Een golf-vervorming met verschillende frequenties.



Figuur 7.5: Een golf-vervorming met verschillende fases.



Figuur 7.6: Een golf-vervorming met een verschillende azimuth.



Figuur 7.7: Een golf die gedempt wordt naar de eindpunten toe.

7.2.2 Loshangend touw

Met loshangend touw bedoelen we een touw dat ophangt aan 1 punt terwijl het andere punt vrij kan bewegen in de ruimte. Ook hier is het zo dat het punt waar aan het touw vasthangt geen vast punt hoeft te zijn in de ruimte, we moeten enkel de positie van dit punt kennen op elk tijdstip in de simulatie. Dit punt zal ook niet beïnvloed worden door de algemene vervorming en de golfvervorming.

1. **Rusttoestand**: De rusttoestand van een loshangend touw is eenvoudigweg een rechte lijn.

2. **Algemene vervorming**: We kunnen elke mogelijke vervorming van een loshangend touw bekomen door plooiingen toe te voegen aan de rusttoestand(een rechte lijn). De algemene vervorming zal bijgevolg een methode zijn waarmee we plooiingen kunnen toevoegen aan de rusttoestand. Deze plooiingen zullen de volgende parameters meegegeven kunnen worden:

- positie: De afstand van het beginpunt van het touw waar de plooiing start.
- Lengte: De lengte van het stuk touw dat geplooid zal worden.
- Hoek: De hoek die de plooiing zal maken.
- Azitmuth: De orientatie van de plooiing.

Met behulp van deze vier parameters kan elke plooi in een loshangend touw voorgesteld worden.

3. **golf-vervorming**: De golf-vervorming zal hetzelfde zijn als bij een doorhangend touw. We zullen dus eveneens een sinus of een cosinusgolf gebruiken met de vier bijhorende parameters. Een klein verschil bij het loshangend touw is wel dat we slechts 1 vast punt hebben en we bijgevolg de golf enkel aan die kant moeten dempen. Het kan echter wel gewenst zijn dat ook het onderste punt volledig of gedeeltelijk gedempt wordt om te simuleren dat hier een zwaar object aan bevestigd is.

7.3 Catenary

Een catenary is de vorm van een hangende flexibele kabel wanneer deze enkel vasthangt aan zijn twee eindpunten en onhevig is aan de zwaartekracht [**?**]. Deze vorm zal stijler naar beneden gaan naargelang we korter bij de eindpunten komen omdat deze plaatsen meer gewicht zullen dragen dan de plaatsen in het midden. Galileo beweerde lang geleden dat een touw onderhevig aan de zwaartekracht voorgesteld kon worden door een eenvoudige parabool. In 1669 werd dit echter ontkracht door Jungius. In 1691 werd echter pas de werkelijke wiskundige formule bedacht om een catenary te genereren. We herhalen hier de formule gegeven in [19], waarna we enkele aanpassingen bespreken die voorgesteld zijn in [2] om de formule meer bruikbaar te maken in een implementatie.

Gegeven zijn de twee eindpunten en de lengte van het touw, we beschouwen de eindpunten (x_1, y_1) en (x_2, y_2) en als lengte nemen we L. De vergelijking van een catenary is de volgende:

$$y = c + acosh\left(\frac{x-b}{a}\right)$$

Om nu tot een oplossing voor a,b en c te komen starten we met de volgende drie gegevens:

$$y_2 = c + a\cosh\left(\frac{x_2 - b}{a}\right) \tag{7.1}$$

$$y_1 = c + a\cosh\left(\frac{x_1 - b}{a}\right) \tag{7.2}$$

$$L = a\cosh\left(\frac{x_2 - b}{a}\right) - a\cosh\left(\frac{x_1 - b}{a}\right) \tag{7.3}$$

Door 7.2 te verminderen met 7.1, dit te kwadrateren en hier het kwadraat van 7.3 vanaf te trekken krijgen we:

$$\frac{L^2 - (y_2 - y_1)^2}{2a^2} = \cosh\left(\frac{x_2 - x_1}{2a}\right) - 1 \tag{7.4}$$

Wanneer we hierop de formules van simpson toepassen en alles wat herschikken krijgen we:

$$\sqrt{L^2 - (y_2 - y_1)^2} = 2a\sinh\left(\frac{x_2 - x_1}{2a}\right) \tag{7.5}$$

In deze formule is a de enige onbekende en kunnen we deze bijgevolg numeriek vinden. Hoe we dit aanpakken zullen we later in deze sectie bespreken. Voor de onbekenden b en c te berekenen, kunnen we 7.3 herschrijven op de volgende manier:

$$\frac{L}{a} = \cosh\left(\frac{b}{a}\right)\left(\sinh\left(\frac{x_2}{a}\right) - \sinh\left(\frac{x_1}{a}\right)\right) - \sinh\left(\frac{b}{a}\right)\left(\cosh\left(\frac{x_2}{a}\right) - \cosh\left(\frac{x_1}{a}\right)\right) \tag{7.6}$$

Laten we dit nu eenvoudiger noteren door:

$$M = \sinh\left(\frac{x_2}{a}\right) - \sinh\left(\frac{x_1}{a}\right) \tag{7.7}$$

$$N = cosh\left(\frac{x_2}{a}\right) - cosh\left(\frac{x_1}{a}\right) \tag{7.8}$$

Zodat we 7.6 kunnen herschrijven als:

$$\frac{L}{a} = Mcosh\left(\frac{b}{a}\right) - Nsinh\left(\frac{b}{a}\right) \tag{7.9}$$

We hebben nu twee mogelijkheden:
Als N > M:

$$\mu = tanh^{-1}\left(\frac{M}{N}\right) \tag{7.10}$$

$$Q = \frac{M}{sinh(\mu)} \tag{7.11}$$

$$b = a\left(\mu - sinh^{-1}\left(\frac{L}{Qa}\right)\right) \tag{7.12}$$

Als N < M:

$$\mu = tanh^{-1}\left(\frac{N}{M}\right) \tag{7.13}$$

$$Q = \frac{M}{cosh(\mu)} \tag{7.14}$$

$$b = a\left(\mu - cosh^{-1}\left(\frac{L}{Qa}\right)\right) \tag{7.15}$$

Wanneer we zowel a als b kennen, kunnen we deze invullen in formule 7.1 om c te bekomen.

7.3.1 Numeriek vinden van a

Zoals we hierboven vermeld hebben zullen we a niet analytisch kunnen vinden en hebben we een numerieke methode nodig. We herschrijven de formule 7.16 op de volgende manier:

$$K = Asinh\left(\frac{C}{A}\right) \tag{7.16}$$

Aangezien A gelijk is aan 2a, volstaat het om in deze formule A te vinden. K en C zijn bekend. Omdat A niet op een analytische methode berekend kan worden zullen we deze numeriek gaan bepalen. We weten dat C/A een hoek is en dus gelegen is tussen 0 en 360 graden. Dit is het startinterval, we noteren het begin van een interval als i_b en het einde van een interval i_e, het midden van een interval zullen we noteren als i_m. We herhalen nu de volgende stappen recursief tot we een minimum foutmarge (e) hebben bereikt.

1. We krijgen een interval waarin de hoek C/A zich moet bevinden, en nemen als nieuwe hoek voor de test het midden van dit interval (i_m).
2. We bepalen A voor deze nieuwe hoek: $A = \frac{C}{i_m}$
3. Hierna berekenen $K - A * sinh(C/A)$ en noteren deze als P.
4. Als de absolute waarde van deze P kleiner is als onze foutmarge, hebben we A gevonden en stoppen we de recursie.
5. Als P groter is als 0 vervangen we het interval door $[i_m, i_e]$ en beginnen een nieuwe recursiestap.
6. Als P kleiner is als 0 vervangen we het interval door $[i_b, i_m]$ en beginnen een nieuwe recursiestap.

Wanneer we onze foutmarge klein genoeg nemen, zullen we op deze manier een a bekomen die het werkelijke resultaat zeer goed benaderd en dus goed genoeg is wanneer we deze formules enkel gebruiken voor het visualiseren van een catenary.

7.3.2 Aanpassingen in de formule

Wanneer we de formules die hierboven besproken zijn zullen implementeren geven deze soms foute resultaten of zelfs helemaal geen resultaat. De reden hiervoor is dat de waardes van A, M en N enorm hoog worden wanneer de afstand in x tussen de 2 eindpunten groter wordt, hierdoor kunnen deze waardes niet bevat worden door integers of zelfs doubles. Een tweede probleem in deze formules is dat deze enkel werken wanneer het tweede punt hoger gelegen is dan het eerste punt. Omwille van deze redenen introduceren we enkele aanpassingen die deze problemen voorkomen.

Wanneer we twee willekeurige eindpunten krijgen ((x_1, y_1) en (x_2, y_2)) zullen we deze eerst transformeren zodat het punt met de laagste y-waarde zich op de oorsprong bevind. Hierna zullen we het geheel scaleren met als schaal $(x_1 - x2)$, op deze manier zal het punt met de hoogste y-waarde $(1, dy)$ als nieuwe coordinaten krijgen, met $dy = (y_1 - y_2) * (x_1 - x2)$. Ook de lengte moeten we scaleren en bijgevolg zal de nieuwe lengte gelijk zijn aan $(L * (x_1 - x2))$. Voor we starten met de berekeningen van a,b en c controleren we of de lengte van het touw langer of evenlang is als de afstand tussen de twee punten, dus:

$$L > \sqrt{1 + dy^2}$$

In de formules hierboven zullen we in formule 7.16 numeriek zoeken naar A. Omdat deze A enorm groot kan worden stelt barzel voor om numeriek te zoeken naar α waarbij α gelijk is aan 1/A. Deze α zal zeer kort bij 0 liggen en dus veel eenvoudiger zijn om mee te werken de de A voorgesteld in de formules van weill. De formule 7.16 kan nu herschreven worden met deze α op de volgende manier:

$$\sqrt{L^2 + dy^2} = \frac{sinh\alpha}{\alpha}$$

M en N kunnen nu eenvoudig berekend worden op de volgende manier:

$$M = sinh(2\alpha)$$

$$N = cosh(2\alpha) - 1$$

De berekeningen van μ en Q zullen niet veranderen, b en c bekomen we op de volgende manier:

$$als N > M \qquad b = sinh^{-1}\left(\frac{2L\alpha}{Q}\right) - \mu$$

$$als N \leq M \qquad b = cosh^{-1}\left(\frac{2L\alpha}{Q}\right) - \mu$$

$$c = \frac{cosh(b)}{2\alpha}$$

Omdat we α slechts met benadering hebben kunnen bepalen compenseren we dat door de fout(e) te bepalen en deze af te trekken van het uiteindelijke resultaat.

$$e = \frac{cosh(2\alpha + b)}{2\alpha} - c - dy$$

De uiteindelijke formule van de Catenary zal er nu als volgt uitzien:

$$C(s) = \frac{cosh(2\alpha + b)}{2\alpha} - c - se \qquad s \in [0, 1]$$

7.4 Algemene modellen

Eerder in dit hoofdstuk hebben we gezien hoe we een touw dat opgehangen is aan twee punten kunnen simuleren met behulp van de drie lagen. Daarna hebben we gezien hoe we een touw dat ophangt aan 1 punt kunnen simuleren met drie lagen die bijna volledig verschillen van de drie lagen uit het eerste geval. We kunnen dus concluderen dat alhoewel het in beide gevallen gaat om een touw dat ophangt, we toch twee volledig verschillende systemen nodig hebben. Het lijkt dus op het eerste zicht dat dit systeem niet veralgemeend kan worden: elk model zal zijn eigen rusttoestand hebben, zal verschillende parameters en methodes hebben voor de algemene vervorming. Voor elk model zullen ook andere golf-vervormingen van toepassing zijn.

Een voordeel is wel dat de modellen die geconstrueerd worden dankzij dit systeem vaak zeer standaard zijn en dus vaak gecombineerd kunnen worden tot meer complexe modellen. In de paper van Barzel wordt het voorbeeld gegeven van een lasso, hierbij kunnen we het stuk touw van de hand tot de eigenlijke cirkel van de lasso voorstellen door een touw opgehangen aan twee punten, en voor het stuk dat verder loopt uit de hand gebruiken we een loshangend touw.

Een ander voordeel van de gelaagde structuur is dat het zeer eenvoudig is om aanpassingen of uitbreidingen te doen op 1 van de lagen en op deze manier variaties te construeren van de reeds bestaande modellen. Ook andere modellen van vervormbare objecten kunnen dankzij deze gelaagde aanpak eenvoudig geconstrueerd worden door de volgende 7 stappen voorgesteld door Barzel:

1. Bepaal het onderliggend model van de curve.
2. Creëer een natuurlijke rusttoestand.
3. Voer een algemene vorm verandering uit.
4. Definieër een vervormings enveloppe.
5. Voer een golf-vervorming uit.
6. Verhinder een intersectie met andere objecten.
7. Creëer het uiteindelijke object.

We kunnen dus concluderen dat we per verschillend model andere methodes nodig hebben om de lagen te construeren. Het systeem van fake dynamics dat hier is voorgesteld kan dus niet zomaar simpelweg geïmplementeerd worden en hierna losgelaten worden op alle mogelijke vervormbare objecten, zoals dit wel het geval was met een mass-spring systeem. Wat echter wel in het algemeen voor alle vervormbare objecten kan gebruikt worden is het systeem van de drie lagen voor een algemene rusttoestand, een algemene vervorming en een golfvervorming voor te stellen.

7.5 Resultaten en conclusie

Volgens de paper is de gelaagde aanpak met behulp van 'fake dynamics' zeer succesvol. Ze beweren dat de resultaten overtuigend realistisch zijn binnen het gebied van karakter animatie. Animators die ook 'real dynamics' hebben gebruikt bij het produceren van ditzelfde soort animaties hebben een grote voorkeur voor deze aanpak. Ook de mogelijk om de positie en tijd interactief te bepalen met een accuraatheid tot op de frame is zeer belangrijk wanneer we deze animaties willen infiltreren in een animatiefilm.

In dit hoofdstuk is een methode voorgesteld om vervormbare objecten te creëren en deze te kunnen animeren. Er werden geen wetten uit de dynamica gebruikt, het dynamisch gedrag van de vervormbare objecten werd enkel gesimuleerd door wiskundige formules. Het systeem werd hoofdzakelijk bedacht als hulp bij het creëren van animaties, en hier levert het dan ook goede resultaten. In het verdere verloop van deze thesis zullen we onderzoeken of deze methode uitbreidbaar is naar 2d-modellen. We zullen echter steeds in het achterhoofd houden dat het systeem vooral bedoeld is als hulp bij het creëren van animaties en niet om real-time omgevingen met vervormbare objecten te bouwen.

Hoofdstuk 8

Uitbreiding naar 2d-modellen

8.1 Inleiding

In het vorige hoofdstuk hebben we besproken hoe we 'fake dynamics' kunnen gebruiken om flexibele modellen te kunnen visualiseren. We hebben hierbij gebruik gemaakt van drie lagen die elk een vervorming van het touw voorstelden: de rustvorm, de algemene vervorming en de golf-vervorming. We hebben geconcludeerd dat dit systeem zeer goede resultaten oplevert wanneer we dit systeem gebruiken waarvoor het ontworpen is, namelijk als hulp bij het animeren van van flexibele objecten. De modellen waarop de methode werd toegepast zijn echter allemaal 1 dimensionale objecten, dit terwijl de methode op het eerste zicht nergens beperkt is om ook gebruikt te worden bij meerdere dimensionale objecten.

In zijn paper vermeld Barzel wel dat het mogelijk zou kunnen zijn om zijn methode met de drie lagen uit te breiden naar 2 dimensionale objecten. Golven in 2d zijn natuurlijk wel meer complex en zeker de controle over een algemene vervorming krijgt bij 2d-objecten een heel aantal nieuwe mogelijkheden en parameters. Maar het idee van de rusttoestand waarop een algemene vervorming en een golf-vervorming worden toegepast zou in principe zou evenzeer moeten werken.

Het hoofddoel van deze thesis is dan ook om te onderzoeken of het model van Barzel toegepast kan worden op 2D-objecten. Als model voor ons object hebben we gekozen voor een horizontale doek die ophangt aan een onbeperkt aantal punten. Dit model leunt het meeste aan bij het model dat we in het vorige hoofdstuk gezien hebben van het doorhangend touw en leek ons

daarom het meest geschikt als startobject bij onze poging om het systeem uit te breiden naar 2d-modellen. We zullen proberen om voor dit model een rusttoestand te creëren, waarna we gaan zoeken naar methodes om hier een algemene en een golf-vervorming op uit te voeren.

We zullen in dit hoofdstuk eerst de manier geven waarop we onze doek intern gaan voorstellen en hoe we deze gaan opbouwen. Hierna zullen we een methode geven die is voorgesteld door Weill [19] om een rusttoestand te construeren. We zullen enkele aanpassingen geven aan deze methodes om een resultaat te verkrijgen dat meer aan onze wensen voldoet. In de sectie die hierop volgt bespreken we waarom we de algemene vervorming momenteel achterwege hebben gelaten. We eindigen dit hoofdstuk met de bespreking van de golf-vervorming en een conclusie.

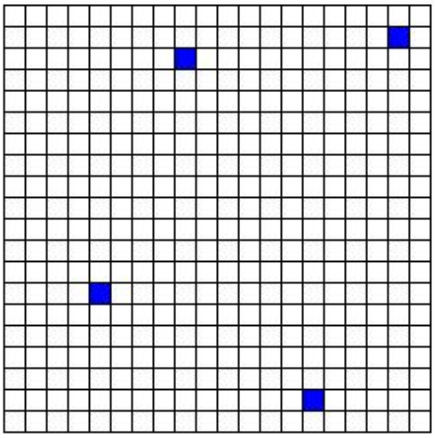
8.2 Voorstelling van de doek

Om het systeem zo eenvoudig mogelijk te houden zullen we onze doek voorstellen zoals we deze ook hebben voorgesteld bij de real dynamics, namelijk met behulp van een gridstructuur. Onze doek zal dus rechthoekig zijn en elk punt uit onze grid zal 5 parameters hebben, ten eerste zullen er een X en een Z waarde zijn die de lokatie in de grid aangeven. Daarbij zal ook elk punt een positie hebben in de 3-dimensionele ruimte, wat wil zeggen dat deze 3 waardes heeft voor de x,y en z-coordinaat. We werken dus met 2 verschillende coordinaat-systemen, het eerste is het grid coordinaat systeem en is 2-dimensionaal. Dit geeft de positie van de punten in het grid aan. Het 2de is het object coordinaat systeem en is 3-dimensionaal, deze geeft de postie aan van de punten in de ruimte.

Als voorbeeld voor het vervolg van deze sectie nemen we een vierkantig grid met een breedte en een hoogte van 20 dat opgehangen is aan de volgende vier vaste punten:

1. punt a: gridcoordinaten: (8,2) objectcoordinaten: (4,0,1)
2. punt b: gridcoordinaten: (18,1) objectcoordinaten: (9,0,1)
3. punt c: gridcoordinaten: (4,13) objectcoordinaten: (2,0,6)
4. punt d: gridcoordinaten: (14,18) objectcoordinaten: (7,0,9)

In afbeelding 8.1 zien we een voorbeeld van deze grid waarbij de blauwe vakken onze vaste punten zijn.



Figuur 8.1: Het grid met de 4 vaste punten dat we verder in dit hoofdstuk zullen gebruiken.

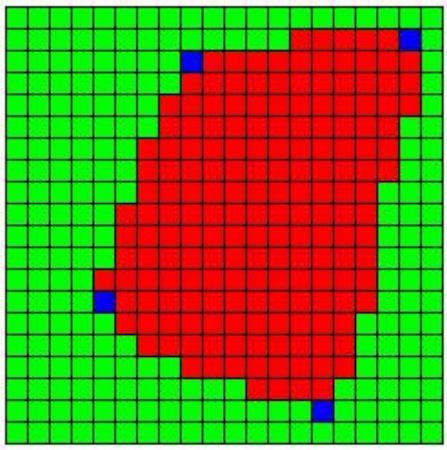
8.3 Rusttoestand

Om de rusttoestand van een vrij hangend doek te bepalen dat ophangt aan een aantal vaste punten willen we een wiskundige methode vinden om het oppervlakte dat zo een doek beschrijft te benaderen. Een dergelijke methode is voorgesteld door Weill in zijn artikel "The Synthesis of Cloth Objects". Hij beschrijft hierin een methode die bestaat uit twee stappen. Eerst zullen we het oppervlakte benaderen dat binnen de vaste punten ligt, dit kunnen we, net zoals bij het doorhangend touw volledig wiskundig doen aan de hand van catenary's. In de tweede stap zullen we op alle punten een relaxatie process uitvoeren dat zal proberen de oorspronkelijke afstanden tussen de punten in het grid te benaderen.

8.3.1 Oppervlakte benaderen binnen de vaste punten

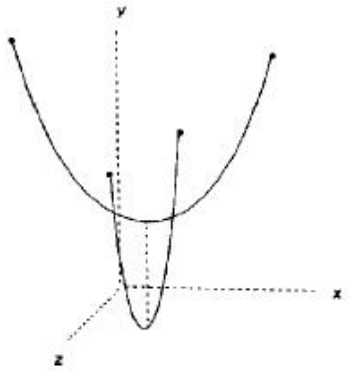
Wanneer we de punten van onze grid willen positioneren in de ruimte moeten we eerst een onderscheid maken tussen de punten binnen de convex hull van de vaste punten, we noemen deze de interne punten, en de punten die hierbuiten liggen, deze noemen we de externe punten. Dit onderscheid wordt

nog eens getoond in figuur 8.2 waar de rode punten de interne, en de groene punten de externe punten zijn. De posities van de interne punten worden bepaald door de vaste punten en kunnen bijgevolg benaderd worden met behulp van catenarys en de methode die we hiervoor kunnen gebruiken zullen we hier beschrijven.



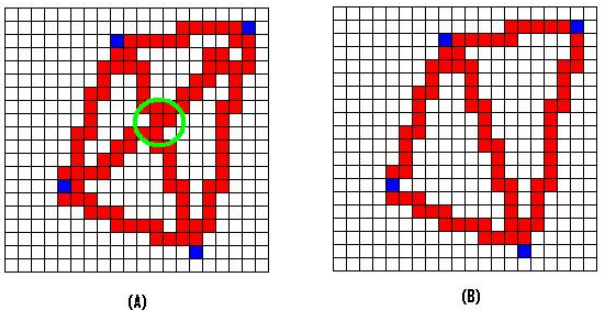
Figuur 8.2: De interne punten (rood) en de externe punten (groen).

Bij de start van dit procces kennen we enkel de posities van de vaste punten. Daarom beginnen we met het positioneren van de punten tussen paren van deze vaste punten. Omdat we de posities van deze punten kennen en ook de lengte van de stof tussen deze twee punten kunnen bekomen met behulp van de gridcoordinaten komt dit positioneren eigenlijk overeen met een positioneren van de punten bij een doorhangend touw. We kunnen dus de formule van een catenary gebruiken om deze punten te positioneren. Wanneer we nu op deze manier tussen elk paar vaste punten uit ons voorbeeld de punten zullen positioneren zijn alle punten die rood aangegeven zijn in tekening 8.4(a) gepositioneerd. We zien echter dat er twee catenarys zullen gaan door het punt in het midden van de interne punten en dit punt zou dus bijgevolg op twee verschillende manieren gepositioneerd worden. Dit geval kunnen we zien in figuur 8.3.



Figuur 8.3: Een kruising tussen twee catenaries.

In werkelijkheid zou een punt dat gepositioneerd worden door twee catenarys zich ergens in het midden van deze twee posities bevinden. Omdat het zeer veel extra rekenwerk met zich mee zou brengen om dit punt op zijn positie uit de werkelijkheid te plaatsen zullen wij eenvoudigweg 1 van de twee catenarys verwijderen. Wanneer we de krachten die de verschillende catenarys op elkaar uitoefenen negeren kunnen we besluiten dat alle punten op zo een catenary zo laag geplaatst zijn als ze in realiteit zouden vallen, aangezien dit de definitie is van een catenary. Hierdoor kan dus een punt dat gepositioneerd is door een catenary in principe niet meer lager bewegen. Omwille van deze reden zullen we de catenary die op de plaats van de kruising zich het laagst bevind verwijderen.



Figuur 8.4: De gepositioneerde punten na de eerste stap. (a) zonder detectie van kruising (b) na detectie van kruising.

Wanneer we tussen elk paar vaste punten op deze manier de punten gaan

positioneren en we verwijderen tijdens dit process de punten die geplaatst zijn met behulp van een catenary die een hoger gelegen catenary kruist zullen we uiteindelijk een structuur krijgen die is opgebouwd uit driehoeken 8.4(b). Deze driehoeken zullen we nu toevoegen aan een database die gebruikt zal worden in de volgende stap van het benaderen van de interne punten.

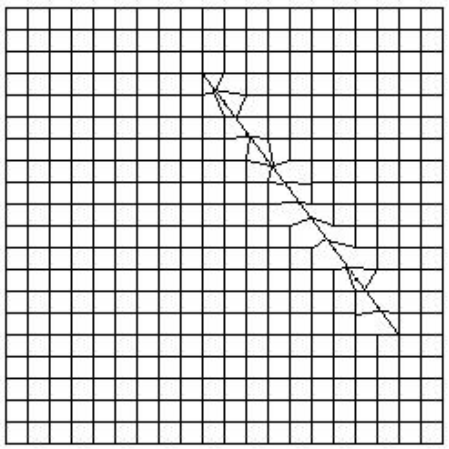
We zullen nu elke driehoek apart beschouwen en zullen de interne punten van elke driehoek proberen te positioneren op een gelijkaardige manier. Het idee is om elke driehoek herhaaldelijk op te delen in 2 kleinere driehoeken, en dit te blijven herhalen totdat elk punt binnen deze driehoek gepositioneerd is in de ruimte. De manier waarop een driehoek wordt opgedeeld wordt bepaald door drie catenaries te construeren, 1 vanuit elke hoek naar het midden van de tegenoverliggende zijde. Deze catenaries zullen alle drie kruisen op het zwaartepunt van de driehoek en zullen hier alle drie een verschillende hoogte hebben. We gebruiken de hoogst gelegen catenarie om de driehoek op splitsen in 2 kleinere driehoeken.

De methode die hierboven beschreven is, is de methode die gebruikt werd door Weill in zijn paper: "The Synthesis of Cloth Objects". De methode is zeer straight-forward en zeer eenvoudig te programmeren. Maar er zijn ook enkele nadelen verbonden aan deze methode. Alhoewel de methode uiteindelijk wel altijd alle punten zal positioneren kunnen we moeilijk inschatten hoe vaak de driehoeken moeten opgesplitst worden. Omdat binnen sommige driehoeken sneller alle punten gepositioneerd zullen zijn dan binnen andere driehoeken moeten we voordurend testen uitvoeren om te weten of we kunnen stoppen met een opsplitsing. Ook het voordurend opsplitsen van de driehoeken kan problemen met zich meebrengen, ten eerste is een opsplitsing redelijk arbeidsintensief omdat er steeds 3 catenearies en hun kruising bepaald moeten worden. Ten tweede zal het aantal driehoeken door de voordurende opsplitsing exponentieel toenemen waardoor bij een grotere grid vaak veel geheugencapaciteit nodig is.

Een tweede probleem van deze methode is dat er bij het positioneren van de punten geen rekening wordt gehouden met hun ligging binnen het grid. Doordat de catenaries alleen maar informatie geven over de hoogte van de punten zullen we de x en z coordinaten moeten bepalen met behulp van interpolatie. Dit probleem is duidelijk te zien in afbeelding 8.6 waarbij we het grid van bovenuit zien nadat de punten tussen een paar vaste punten gepositioneerd zijn. Doordat we deze fout steeds meenemen bij het opsplitsen van de driehoeken zal deze steeds groter worden. We zullen na de eerste stap ook catenaries berekenen tussen twee punten waarvan de x en z coordinaten niet

volledig correct zijn waardoor ook de y-waardes bepaald door deze catenarie foutief zullen zijn. Deze fouten zullen meestal lijden tot een resultaat waarbij we nog nauwelijks een gridstructuur kunnen herkennen 8.5. Maar het is ook mogelijk dat door deze fouten punten verder uit elkaar liggen dan ze eigenlijk uit elkaar kunnen liggen, waardoor het onmogelijk wordt om catenaries te berekenen en er bijgevolg geen resultaat kan bekomen worden.

Figuur 8.5: Het resultaat na de oppervlakte benadering met de methode van Weill.

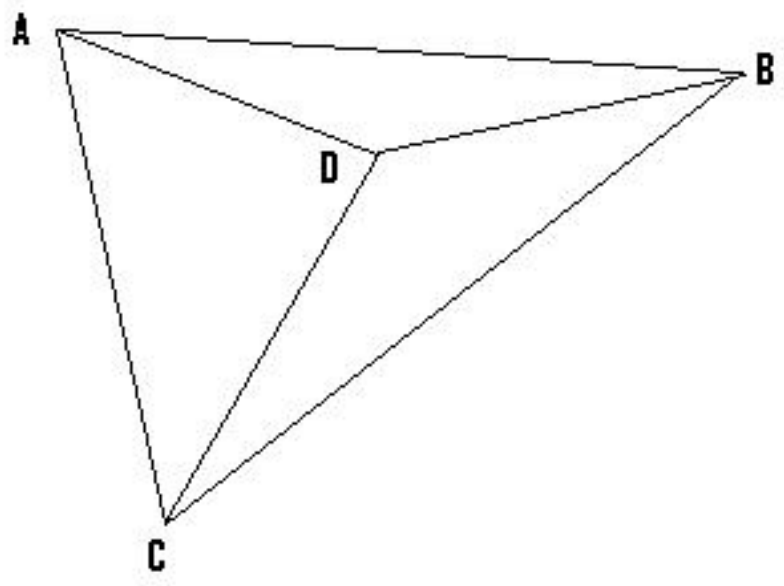


Figuur 8.6: De fout die ontstaat binnen een grid wanneer we punten positioneren volgens de methode van Weil.

Omwille van deze fouten heb ik een aanpassing bedacht van de methode van Weill waarbij we de driehoeken niet meer op gaan splitsen maar steeds de startdriehoeken gebruiken voor het positioneren van de punten. Hierbij houden we ook rekening met de gridcoordinaten van elk punt. We starten hierbij eveneens met het verbinden van paren vaste punten, tijdens dit verbinden zullen we kruisingen van twee verbindingen ook oplossen zoals bij de methode van Weill. Dit wil zeggen dat we in het geval van een kruising zullen bepalen welke catenarie het laagste ligt op het punt van de kruising en deze verwijderen. In deze fase houden we ook steeds in een boolean tabel bij welke vaste punten met elkaar verbonden zijn.

Nadat we alle verbindingen hebben gevonden zullen we ook zoals bij Weill een structuur bekomen die is opgebouwd uit driehoeken. Met behulp van de boolean tabel kunnen we deze driehoeken nu zeer snel vinden door elke drie punten te controleren op verbindingen met elkaar. We zullen deze driehoeken nu in een database steken zodat we deze kunnen gebruiken in de volgende stap, waar we de punten gaan positioneren. We gaan dus niet zoals in de methode van Weill de punten die op de verbindingen liggen reeds positioneren. Een probleem dat zich bij het detecteren van deze driehoeken kan voordoen en dat ook bij de methode van Weill een probleem kan vormen is zichtbaar in figuur 8.7. Wanneer we een dergelijke situatie krijgen na het construeren van de verbindingen zullen we uiteindelijk vier driehoeken bekomen omdat ook de driehoek ABC gevonden zal worden. Hierdoor zal

elk punt zich in twee verschillende driehoeken bevinden en kunnen we dus geen resultaat bekomen. Om dit probleem op te lossen moeten we na het vinden van een driehoek steeds controleren of er geen vast punt binnen deze driehoek ligt.

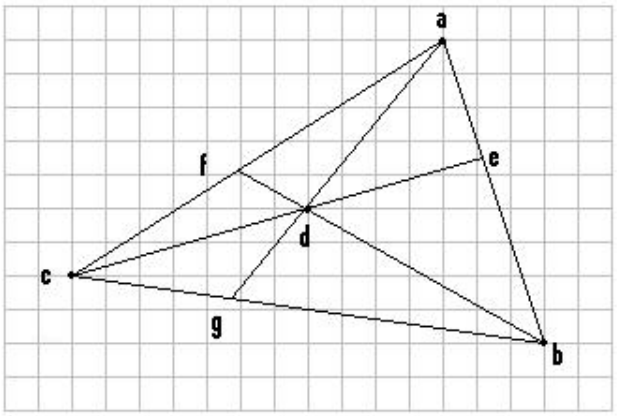


Figuur 8.7: Een probleem dat zich kan voordoen bij het construeren van driehoeken.

Nu we deze driehoeken hebben zullen we elk punt uit ons grid gaan positioneren aan de hand van deze driehoeken. We doorlopen simpelweg ons grid punt per punt en voeren hierna de volgende stappen uit:

1. We bepalen eerst in welke driehoek het punt zich bevindt.

2. We verbinden nu elk hoekpunt van de driehoek met het punt dat we willen positioneren en zoeken de snijpunten van deze lijnen met de tegenoverliggende zijde van de driehoek. We bekomen op deze manier een constructie zoals we zien in figuur 8.8, waar we de driehoek abc hebben en het punt dat we willen positioneren is d. De snijpunten zijn e,f en g. Dit alles doen we met de gridcoordinaten en dit zijn dus bijgevolg redelijk simpele bewerkingen.



Figuur 8.8: De constructie die we maken om een punt binnen een driehoek te positioneren.

3. Nu zullen we deze snijpunten gaan positioneren in de ruimte, de x en z coordinaten in de ruimte bekomen we door de x en z coordinaten van de vaste punten te interpoleren. Voor de y coordinaat contrueren we catenaries tussen de twee vaste punten waar dit snijpunt tussen ligt en met behulp van een booglengte functie op deze catenaries kunnen we deze y bepalen.

4. Nu we deze snijpunten gepositioneerd hebben kunnen we het uiteindelijke punt dat we willen positioneren op dezelfde manier behandelen als deze snijpunten. We construeren drie cateraries vanuit de snijpunten naar hun tegenoverliggend hoekpunt en kunnen met behulp van deze catenaries en hun booglengte functie drie verschillende hoogtes bekomen voor ons te positioneren punt. We zullen net zoals bij de Methode van Weill de hoogst gelegen catenerie gebruiken en de y-waarde die deze heeft op de plaats van het punt nemen. De x en z coordinaten bekomen we opnieuw door interpolatie.

Op deze manier kunnen we alle interne punten positioneren en bekomen we een resultaat dat duidelijk beter is dan de resultaten bekomen door Weill. Het algoritme zal afhankelijk van de grootte van het grid ook ongeveer tweemaal minder rekentijd nodig hebben en zal minder vaak foute resultaten geven.

In de uiteenzetting van deze methode hebben we gezien dat we vaak een manier nodig hebben om te bepalen of twee lijnen snijden, en ook om te bepalen wat dit snijpunt is. Dit kwam zowel terug in de eerste fase waar we twee snijdende verbindingen willen kunnen detecteren en oplossen. En ook bij het positioneren van de punten moeten we een mogelijkheid hebben om snijpunten te bepalen. Wat we ook nodig hadden in ons algoritme was een mogelijkheid om de hoogtewaardes van een catenarie te verkrijgen aan de hand van de booglengte. Deze twee onderdelen zullen we kort bespreken in de volgende secties.

Snijpunt bepalen tussen 2 lijnstukken

In dit hoofdstuk bespreken we hoe we bepalen of twee lijnstukken elkaar snijden en als dit zo is, hoe we dit snijpunt kunnen bepalen. We doen dit in de volgende stappen:

1. We beginnen met voor beide rechten een formule op te stellen die er uit ziet als volgt: $x = ay + b$, we kennen twee punten van elke rechte, en nemen hiervoor (x_1, y_1) en (x_2, y_2). a is de richtingcoëfficient en kan bekomen worden op de volgende manier:

$$a = \frac{y_1 - y_2}{x_1 - x_2}$$

   b kunnen we nu bekomen door de volgende formule:

$$b = z_1 - ax_1$$

2. We testen nu of de richtingscoëfficienten van de 2 rechten verschillend zijn, als dit niet zo is zijn ze evenwijdig. Anders kunnen we het snijpunt bepalen op de volgende manier:

$$x_{snijpunt} = \frac{b_{rechte1} - b_{rechte2}}{a_{rechte1} - a_{rechte2}}$$

$$y_{snijpunt} = a_{rechte1} x_{snijpunt} + b_{rechte1}$$

3. We eindigen met te testen of $x_{snijpunt}$ zich tussen x_1 en x_2 van de beide rechten bevindt. Als dit zo is snijden de twee rechten en is ($x_{snijpunt}$,$y_{snijpunt}$) het snijpunt.

Booglengte van een catenarie

De formule van de catenarie die we in het vorige hoofdstuk hebben gezien geeft ons enkel een methode om een y-waarde te geven voor een bepaalde x-waarde. Wij willen echter een methode hebben die ons een y-waarde kan geven bij een bepaalde booglengte. We zullen dit bekomen door na het opstellen van de formule van de catenarie onze curve te gaan sampelen op een aantal gelijke intervallen van x. Stel bijvoorbeeld dat x gaat van 0 tot 1 en dat we kiezen om onze curve te sampelen op 20 posities. We zullen onze curve dan sampelen op de posities 0, 0.05, ,0.10,0.15,... Bij elke sample berekenen we de geschatte booglengte en slaan deze op in een tabel. Deze booglengte berekenen we door steeds de waarde van de vorige sample op te tellen bij de afstand van het punt bekomen door de vorige sample tot het punt bekomen bij deze sample.

Op deze manier creëren we een tabel waarbij in de eerste kolom een aantal gelijk verdeelde x-waarden staan en in de tweede kolom de geschatte afstand in booglengte tot deze x waarde. Wanneer we nu een y-waarde willen bepalen voor een bepaalde booglengte, zoeken we de kortst gelegen booglengte in deze tabel op, nemen de overeenkomstige x-waarde en vullen deze in in de formule van de catenarie.

De voordelen van deze aanpak is dat het gemakkelijk te implementeren, intuïtief en snel te berekenen is. Het nadeel is echter dat we slechts een geschatte waarde krijgen en dus een kleine fout krijgen voor onze uiteindelijke y-waarde. We kunnen dit probleem reduceren door meer samples te nemen, maar door verschillende waardes uit te proberen hebben we kunnen concluderen dat voor het doel waarvoor wij dit systeem gebruiken een 100-tal samples genoeg is. We hebben geen meer nauwkeurige resultaten nodig omdat we na de oppervlakte benadering nog een relaxatie van de punten uitvoeren. Dit zal worden besproken in de volgende sectie.

8.3.2 Relaxatie

De volgende stap bij het creëren van het oppervlakte van het kledingstuk is een iteratieve relaxatie stap. Deze relaxatie van het oppervlakte wordt bekomen door alle gridpunten te verplaatsen over de oppervlakte totdat een maximale verplaatsing in 1 stap onder een zekere tolerantie valt [19]. De verplaatsing van elk punt tijdens 1 stap zal bepaald worden door de afstanden die door het grid worden opgelegd te benaderen. Op deze manier zullen we zeker geen oplossing krijgen die overeenkomt met de realiteit, maar wel een oplossing die voldoet voor het animeren van kleding.

Voor de eenvoudigheid zullen we de zwaartekracht negeren tijdens deze relaxatiestap. Voor de interne punten heeft dit geen gevolgen omdat deze gepositioneerd zijn met behulp van catenaries en er dus bijgevolg al rekening is gehouden met de zwaartekracht. Om voor de externe punten ook een hangend effect te creëren zullen we deze in het begin van de relaxatiestap op een hoogte plaatsen die zeker lager is dan hun werkelijke hoogte. Door het relaxatieproces zullen deze punten dan uiteindelijk toch omhoog gebracht worden tot de beperkingen die opgelegd zijn door de gridstructuur voldaan zijn. Wanneer het kledingstuk opgehoffen wordt vanaf een bepaald oppervlake is het ook mogelijk om deze externe punten initieel op dit oppervlak te plaatsen.

Het resultaat waar we naar streven is dat elk punt in de grid-structuur op een afstand van 1 grid eenheid van zijn vier aangesloten buren verwijderd is. Weil zal hiervoor bij elke stap voor elk punt een richtingsvector bepalen die de best mogelijke richting aangeeft waarheen we dit punt kunnen verplaatsen om op de juiste afstand van zijn buren te staan. In de paper van Weil wordt ook vermeld dat de magnitude van deze richtingsvector zeer moeilijk te bepalen is, omdat deze ook rekening moet houden met de toekomstige verplaatsingen van de andere grid-punten. Omwille van deze reden zullen we een vereenvoudiging van het systeem gebruiken waarbij we elk grid-punt bij elke stap verplaatsen op zo een manier dat de afstanden tussen zijn 4 buren gelijk is. We zullen op deze manier meer stappen nodig hebben om een goed resultaat te verkrijgen maar omwille van de eenvoud van de bewerking zal dit niet lijden tot een vertraging van de berekening.

In een relaxatieproces gebeurt eigenlijk hetzelfde als hetgeen er plaats vind bij een mass spring systeem. De punten worden namelijk stap per stap zo verplaatst dat deze op de juiste afstand van hun buren zullen belanden, dit effect wordt ook bekomen door de veren in een mass spring systeem die

steeds proberen de punten op hun juist afstand te trekken of te duwen. Het is dan ook mogelijk om de volledige benadering van het oppervlakte van onze doek te bekomen door dit relaxatieprocess, dit zou echter veel meer stappen vergen dan wanneer we eerst een benadering doen van de interne punten met behulp van catenaries.

We kunnen dus concluderen dat het bepalen van een rustvorm voor een doek opgehangen aan meerdere vaste punten zeer goed mogelijk is met behulp van deze twee stappen. We zullen nu in het vervolg van dit hoofdstuk zien of we ook voor een algemene vervorming en een golf-vervorming goede methodes kunnen vinden.

8.4 Algemene vervorming

Bij het doorhangend touw werden twee verschillende algemene vervormingen voorgesteld, de eerste liet het touw slingeren van voor naar achter, en de tweede liet het touw schommelen van links naar rechts. Omdat we werken met een 2d-object kunnen we deze schommelingen of slingeringen veralgemenen naar 1 schommelbeweging die kan gebeuren in alle richtingen. In de paper "faking dynamics of ropes and springs"werd geen methode gegeven om een algemene vervorming van een schommeling te construeren, maar na enkele dingen te proberen kwamen we tot een zeer eenvoudige oplossing. Wanneer we een touw dat ophangt aan 1 punt willen laten slingeren kunnen we dit zeer eenvoudig doen door elk punt te laten bewegen met een afstand die evenredig is met zijn afstand tot het vaste punt.

Deze methode kunnen we nu ook toepassen op ons doek opgehangen aan een aantal vaste punten. We berekenen eerst de gridafstand van elk gridpunt tot zijn kortst bijgelegen vaste punt en nemen hiervan de maximale afstand, we noemen deze d_{max}. Om nu een frame uit een schommeling te bepalen waarbij we een schommelafstand hebben van d_{sa} berekenen we voor elk gridpunt de grootte van zijn verplaatsing $d_{verplaatsing}$ op de volgende manier:

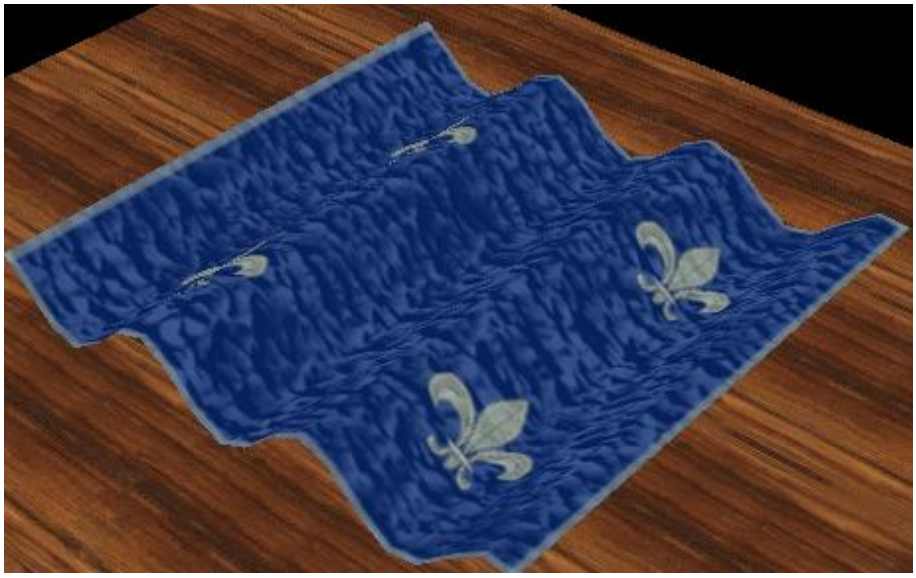
$$d_{sa} = \frac{d_p}{d_{max}} * d_{verplaatsing}$$

Hierbij is d_p de afstand van het gridpunt tot zijn dichtsbij gelegen vaste punt. Wanneer we nu $d_{verplaatsing}$ laten variëren over de tijd tussen een bepaalde maximale verplaatsing d_{max} en zijn omgekeerde $-d_{max}$ en we voeren deze verplaatsing uit in een door de gebruiker gekozen richting krijgen

we een redelijk realistisch uitziende schommeling.

Een schommeling is natuurlijk een zeer simpele vervorming van een doorhangende doek en er zijn in tegenstelling met een touw nog zeer veel andere mogelijkheden om zo een doek te vervormen. Een ander voorbeeld van een algemene vervorming zou bijvoorbeeld zijn om plooien of kreuken toe te voegen. We zouden dan als parameters het aantal, de grootte en de richting van deze plooien geven. Omwille van de complexiteit van deze vervorming is deze niet geïmplementeerd in de applicatie, maar deze vervorming zou wel een intressante toevoeging kunnen zijn in de toekomst.

8.5 Golf-vervorming



Figuur 8.9: Een golvend vlak.

In het vorige hoofdstuk hebben we gezien dat we voor de golf-vervorming van een doorhangend touw simpelweg een sinus of een cosinusgolf kunnen gebruiken. Dit kunnen we eenvoudig uitbreiden voor 2d-objecten door aan deze golf een z-coordinaat toe te voegen en op die manier een golvend vlak te construeren 8.9. Ook de parameters die we bij het doorhangend touw gebruikt hebben kunnen we hier overnemen, deze waren de frequentie(fr), de

magnitude(m) en de fase(p). Bij het doorhangend touw gebruikten we ook nog een parameter voor azimuth om de draaiing van de golf te controleren, dit zal echter lijden tot minder realistische effecten bij 2d-modellen. De y-waarden van de gridpunten voor een golvend vlak te creëren kunnen nu op de volgende manier berekend worden:

$$y = m * sin(x * fr + p)$$

We hebben nu een methode voor een golvend vlak te creëren, maar wanneer we dit vlak zullen optellen bij onze rusttoestand zal dit ook de vaste punten beïnvloeden. We hebben dus net zoals bij het touw opgehangen aan 2 punten een methode nodig om de golven te dempen naargelang deze korter bij de vaste punten komen. Hiervoor berekenen we voor elk gridpunt een dempingsfactor, we beginnen hierbij met vanaf elk gridpunt de gridafstand te berekenen naar zijn dichtst bij gelegen vaste gridpunt. We nemen van deze waarden het maximum en noemen dit d_{max}, we kunnen nu de dempingsfactor van elk gridpunt bekomen door de volgende formule:

$$dampingsfactor = \frac{d_p}{d_{max}}$$

waarbij d_p de afstand is van het gridpunt tot het dichtst bijgelegen vaste punt. Wanneer we nu de y waarde bekomen in de bovenstaande formule vermenigvuldigen met deze dempingsfactor bekomen we uiteindelijk een golvend vlak dat gedempt wordt naar de vaste punten toe.

Zoals ook vermeld bij de algemene vervorming zijn de mogelijkheden bij 2d-objecten veel uitgebreider dan bij 1d-objecten. Zo is het hier bijvoorbeeld ook mogelijk om de richting van de golf als parameter mee te geven. Ook is het mogelijk om volledig andere golven te creëren, zoals bijvoorbeeld een cirkelvormige golf die vertrekt vanuit een bepaald punt, zoals een golf op water die veroorzaakt wordt door een object dat in dit water valt. Deze golf kunnen we dan ook laten dempen naargelang deze verder verwijderd zal zijn van zijn oorsprong. Deze mogelijkheden zijn niet geïmplementeerd voor deze thesis maar ze zijn zeker mogelijk en misschien goede ideëen als uitbreiding van de toepassing.

8.6 Conclusie

In dit hoofdstuk vertrokken met het idee dat het mogelijk was om met behulp van een gelaagde structuur een dynamisch effect te faken voor 1-dimensionale objecten. We hebben verondersteld dat deze methode uit te breiden was naar 2-dimensionale objecten en hebben als voorbeeld een doek genomen die ophing aan verschillende vaste punten. Voor dit model hebben we gezien hoe we een rusttoestand, een algemene vervorming en een golf-vervorming kunnen creëren en we kunnen dus concluderen dat het wel degelijk mogelijk is om de methode die geïntroduceerd werd door Weill uit te breiden naar meerdere dimensionale objecten. Dit wil zeggen dat we ook voor meerdere dimensionale objecten algoritmes kunnen vinden voor het construeren van de 3 lagen die nodig zijn voor het dynamisch effect te creëren. Nu we deze drie lagen hebben willen we hiermee natuurlijk een animatie gaan creëren, hieraan zullen we het laatste hoofdstuk van deze thesis wijden.

Hoofdstuk 9

Animatie

In het vorige hoofdstuk hebben we gezien hoe we de rusttoestand, de algemene vervorming en de golf-vervorming van een doek die opgehangen is aan verschillende vaste punten kunnen construeren. Maar uiteindelijk zullen al deze methodes slechts losstaande frames produceren die ofwel een rusttoestand, een algemene vervorming of een golf-vervorming voorstellen. Wanneer we een animatie willen creëren zullen we ten eerste een methode nodig hebben om deze drie lagen samen te voegen en ten tweede zullen we hun parameters moeten laten variëren over de tijd om verschillende frames na elkaar te genereren die op die manier een animatie vormen.

We starten met de rusttoestand, wanneer we met de methode die hierboven beschreven is twee of meer verschillende rusttoestanden zullen genereren en deze op een bepaalde tijd in de animatie willen plaatsen hebben we een manier nodig om een overgang te maken tussen de verschillende frames. We zullen in dit hoofdstuk verschillende manieren geven om deze overgang te maken en deze manieren uitvoerig bespreken.

Na deze fase zullen we een animatie krijgen van verschillende opeenvolgende rusttoestanden die in elkaar overgaan met 1 van de besproken overgangsmanieren. Nu willen we aan deze animatie golvingen en andere algemene vervormingen, zoals schommelingen toevoegen. We willen ook kunnen bepalen hoe deze vervormingen zich gaan gedragen over de tijd, zo willen we bijvoorbeeld de snelheid van de golving of de schommeling kunnen instellen alsook het moment dat deze starten en stoppen. We zullen in dit hoofdstuk eerst bespreken hoe we deze twee vervormingen kunnen toevoegen aan de animatie en hoe we deze vervormingen kunnen herleiden tot bewegingen. We eindigen dit hoofdstuk met de userinterface te bespreken die we

gebruikt hebben om op een eenvoudige manier onze animatie te kunnen manipuleren en te kunnen overzien.

9.1 Overgang tussen rust-vormen

Zoals aangegeven in de paper van Barzel dient zijn methode enkel om keyframes te maken die zich op een bepaalde tijd in de animatie moeten voordoen. We willen dus een methode die automatisch de frames zal construeren die zich tussen deze keyframes gaan voordoen. We hebben bijvoorbeeld een doek die ophangt aant zijn vier hoekpunten en een punt in het midden. Wanneer we nu een animatie willen maken waarbij we dit middelste punt loslaten, dan willen we deze animatie niet handmatig frame per frame gaan construeren. We willen een systeem dat automatisch deze animatie zal creëren wanneer we de eerste en de laatste frame invoeren.

Wanneer we dus twee verschillende toestanden hebben van onze gridstructuur, dan willen we een methode om een eindig aantal toestanden te bepalen die samen een overgang zullen vormen tussen deze begin en deze eindtoestand. We kunnen deze toestanden eenvoudig bekomen door elk punt uit de grid te laten bewegen vanaf zijn positie in de starttoestand naar zijn positie in de eindtoestand. Om het systeem eenvoudig te houden zullen we deze punten enkel in rechte lijnen naar hun nieuwe positie laten bewegen. Het bewegen in rechte lijnen kan soms zeer rare resultaten geven (bijvoorbeeld wanneer het grid is omgedraaid) maar zal in ons systeem meestal een voldoende resultaat opleveren. Wanneer we meer gesofisticeerde bewegingen willen voorstellen volstaat het meestal om meer keyframes te maken die dan toch in rechte lijnen naar elkaar zullen overgaan.

Naargelang de wensen van de gebruiker kunnen we verschillende manieren nodig hebben om de punten van hun begin naar hun eindpositie te laten bewegen. In het voorbeeld dat we hierboven hebben gegeven van een vallende doek willen we bijvoorbeeld dat de beweging traag start maar dat deze steeds sneller zal bewegen naargelang we de eindpositie bereiken. We hebben in onze applicatie de vier meest voorkomende overgangen geïmplementeerd en we zullen deze hier bespreken. We zullen bij elke overgang ook de formule geven voor de positie van een punt (p_i) te bepalen, wanneer dit punt zal bewegen van positie p_1 naar het positie p_2, waarbij we n aantal frames willen en waarbij i aangeeft de hoeveelste frame we berekenen.

9.1.1 Lineaire overgang

Bij de lineaire overgang zal elk punt met gelijke intervallen verplaatst worden, dit zal resulteren in een overgang waarbij de snelheid steeds constant is. De formule is als volgt:

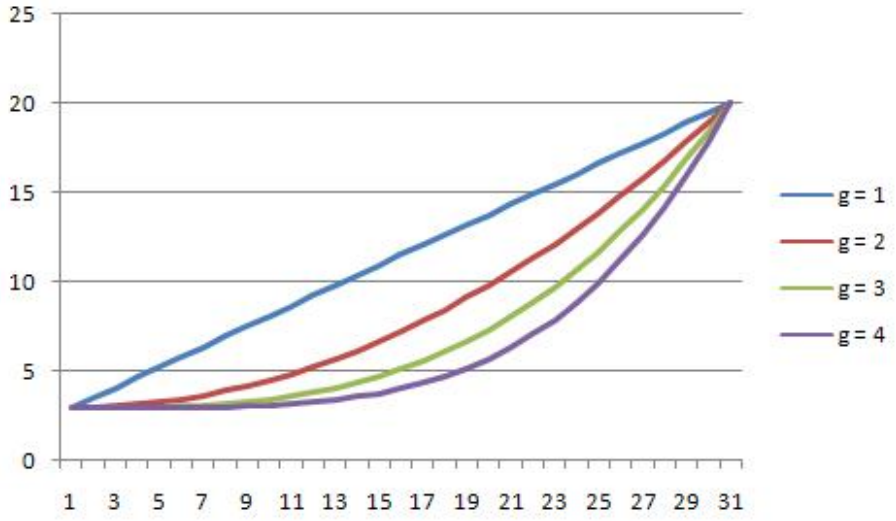
$$p_i = p_1 * (1 - \frac{i}{n}) + p_2 * \frac{i}{n}$$

9.1.2 Overgang waarbij de snelheid verhoogt

Hier zullen we het punt met steeds grotere intervallen verplaatsen, dit zal resulteren in een overgang die vertrekt met een snelheid gelijk aan nul en die steeds zal vergroten. We kunnen deze overgang gebruiken om bijvoorbeeld te simuleren dat iets valt. Hierbij kunnen we ook de graad van de versnelling aangeven, we geven deze in de formule aan met g, deze kan varieren van 1 tot oneindig maar bij een graad hoger dan 4 zullen we geen realistische bewegingen meer krijgen. De formule is als volgt:

$$p_i = p_1 * (1 - \left(\frac{i}{n}\right)^g) + p_2 * \left(\frac{i}{n}\right)^g$$

Zoals we zien in de formule zal deze hetzelfde resultaat geven als een lineaire beweging wanneer we g gelijk aan 1 nemen. Hoe hoger we de graad van de versnelling nemen hoe trager de beweging zal starten en hoe sneller deze zal eindigen. We zullen echter wel steeds dezelfde afstand afleggen binnen dezelfde tijd. Een voorbeeld van de verandering van de positie ten opzichte van de tijd bij een veranderlijke graad zien we in figuur 9.1. We hebben hier 30 frames, de startpositie is 4 en de eindpositie is 20.



Figuur 9.1: De verandering van de positie ten opzichte van de tijd bij een graad van 1,2,3 en 4.

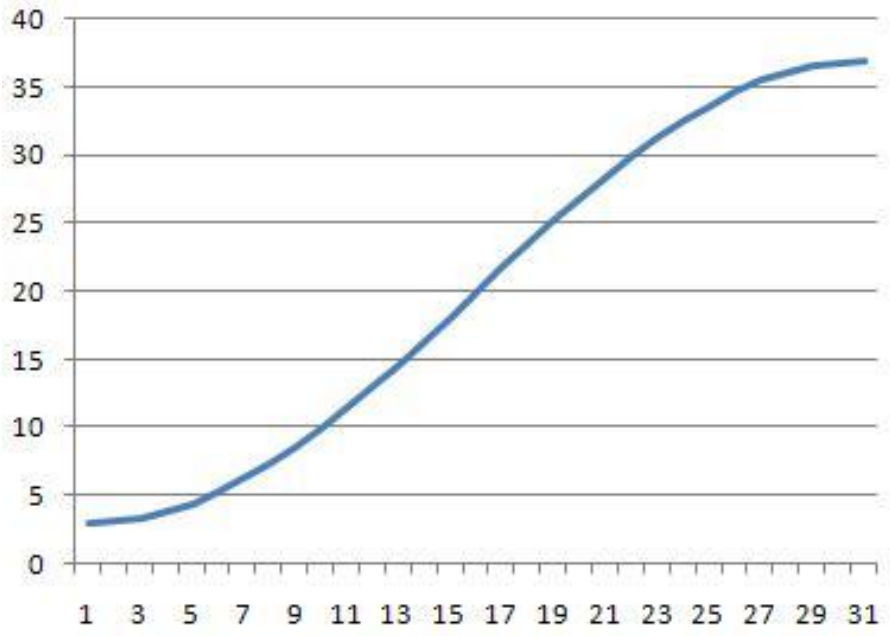
Een overgang waarbij we de snelheid willen verlagen zal juist omgekeerd werken en zullen we hier bijgevolg niet verder bespreken.

9.1.3 Overgang met ease-in, ease-out

Ease-in, ease-out wil zeggen dat de beweging start en stopt in een volledige stilstand. We willen dus geen plotse veranderingen in snelheid. We kunnen hierbij een stuk hebben waarbij we een constante snelheid hebben, dit zal afhangen van de techniek die we gebruiken. De meest eenvoudige manier om ease-in/ease-out te implementeren is om als snelheid het stuk van de sinus curve te nemen tussen $-\pi/2$ en $\pi/2$, op deze manier bekomen we de formule gebruikt in onze applicatie, deze ziet er uit als volgt:

$$p_i = p_1 * (1 - \frac{sin(\frac{i}{n} * \pi - \frac{\pi}{2}) + 1}{2}) + p_2 * (\frac{sin(\frac{i}{n} * \pi - \frac{\pi}{2}) + 1}{2})$$

We zien de beweging van een punt met behulp van ease-in/ease-out over de tijd in figuur 9.2, de startpositie is 3, de eindpositie is 37 en we gebruiken 30 frames.



Figuur 9.2: De beweging van een punt over de tijd gebruik makend van ease-in/ease-out .

Er zijn natuurlijk nog enorm veel andere mogelijkheden om overgangen te creëren tussen twee keyframes, deze kunnen al dan niet bewegingen zijn over een rechte lijn. Een voorbeeld dat bij het creëren van animaties bijvoorbeeld nuttig zou kunnen zijn is een overdrijvende overgang, waarbij de punten bijvoorbeeld in een enorme boog van hun start naar hun eindpositie bewegen. We kunnen op deze manier nog zeer veel mogelijkheden bedenken maar zullen ons in deze thesis beperken tot de 4 overgangen die hierboven reeds besproken zijn.

9.2 Golf-vervormingen en algemene vervormingen

Na het uitvoeren van de vorige sectie zullen we reeds een animatie hebben van verschillende opeenvolgende rustvormen die in elkaar overgaan met behulp van 1 van de overgangsmethode's. We hebben dus eigenlijk een array van frames met als lengte het aantal seconden dat de animatie duurt ver-

menigvuldigd met het aantal frames per seconde. Nu willen we aan deze animatie golvingen en schommelingen toevoegen. Dankzij het systeem met de drie lagen zal dit een zeer eenvoudig proces zijn.

Om een vervorming toe te passen op een bepaald moment in de animatie tellen we de punten van de gridstructuur van de vervorming samen met de punten van de gridstructuur op dat moment in de animatie. Door de parameters van deze vervormingen te laten variëren over de tijd kunnen we bijvoorbeeld een golf laten voortbewegen of een algemene vervorming een schommeling laten voorstellen. Het aanpassen van deze parameters zou handmatig kunnen gedaan worden voor elke frame, maar in de applicatie bij deze thesis zullen deze aanpassingen automatisch gebeuren.

Voor een golving toe te voegen aan de animatie zal de gebruiker het begin, het einde en de voortbewegingssnelheid van deze golving aan moeten geven. Het systeem zal dan automatisch de fase van deze golf aanpassen bij elke frame zodat deze de aangegeven voortbewegingssnelheid zal krijgen. De gebruiker heeft natuurlijk ook de mogelijkheid om de freqentie, de magnitude en de beginfase van de golf in te stellen. Ook het toevoegen van een schommeling aan onze applicatie werkt op dezelfde manier en de parameters zullen ook automatisch aangepast worden om op die manier een schommeling te creëren.

9.3 User-interface voor het manipuleren en creëren van een animatie

Het uiteindelijk doel van onze applicatie is om een animatie te produceren van een doek die aan verschillende punten is opgehangen. We willen dit doen met behulp van de methodes die we eerder in dit hoofdstuk hebben besproken, dit wil zeggen dat we in onze user-interface een mogelijkheid moeten voorzien om de volgende dingen uit te kunnen voeren:

1. Creëren van een doek (grootte en texture)
2. Keyframes construeren en deze op een bepaald tijdstip in de animatie plaatsen.
3. De duur en starttijd van deze keyframes aanpassen.
4. Overgangen tussen deze keyframes toevoegen.

5. Deze overgangen kunnen instellen (duur, type, graad).
6. Golven op een bepaald tijdstip toevoegen aan de animatie.
7. Deze golven instellen (duur, frequentie, voortbewegingssnelheid, magnitude, beginfase en starttijd)
8. Schommelingen op een bepaald tijdstip toevoegen aan de animatie.
9. Deze schommelingen instellen(starttijd, duur, richting, kracht)
10. Het totaalbeeld overzichtelijk kunnen weergeven.

We zullen deze noden opsplitsen in 3 grote onderdelen, we beginnen met de creatie van het doek, hierna wijden we een korte sectie aan het produceren van keyframes en we eindigen met l hetgene wat met de eigenlijke animatie te maken heeft.

9.3.1 Creatie van een doek

Bij het starten van onze applicatie moeten we onze doek gaan creëren, we moeten hier de lengte en de breedte van onze grid aangeven en ook een texture kiezen die gebruikt zal worden bij de animatie.

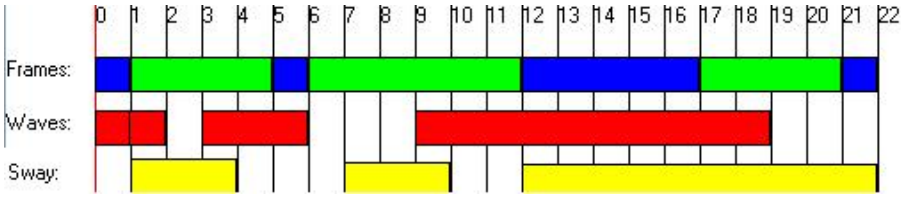
9.3.2 Produceren van een keyframe

Het creëren van een keyframe zal gebeuren in drie fases:

1. In de eerste fase krijgen we een beeld van de gridstructuur van onze doek waarop we kunnen selecteren welke punten we als vaste punten willen beschouwen.
2. In de tweede fase zien we aan de hand van catenaries tussen deze vaste punten hoe dit doek er uit zal zien waneer dit opgehangen zal zijn aan de gekozen vaste punten. We kunnen in deze fase de vaste punten manueel positioneren in de ruimte en zullen dan in real-time het resultaat zien.
3. In de laatste fase zien we de volledige grid na de oppervlakte-benadering en de relaxatie, we kunnen nu deze frame toevoegen aan onze animatie.

9.3.3 De animatie

Om de structuur van onze animatie te visualiseren hebben we gebruik gemaakt van een visualisatiesysteem dat ook gebruikt wordt bij het componeren van digitale muziek. Namelijk een tijdbalk waarin we de verschillende bouwblokken van onze animatie kunnen plaatsen. We zien een voorbeeld van zo een animatie in afbeelding 9.3. In dit voorbeeld zijn de blauwe blokken de keyframes, de groene blokken zijn de overgangen tussen deze keyframes, de rode blokken geven golvingen aan en de gele blokken tonen de schommelingen.



Figuur 9.3: Een voorbeeld van een visualisatie van de structuur van een animatie.

Deze methode leek ons het meest geschikt voor een duidelijk totaalbeeld te krijgen van het verloop van de animatie over de tijd. We zien op deze manier zeer duidelijk waar zich de keyframes bevinden en hoelang de overgangen tussen deze keyframes duren. Ook het begin en einde van golven en schommelingen is duidelijk zichtbaar. Het manipuleren van onze animatie is ook zeer eenvoudig dankzij deze structuur. Door een balk te selecteren kunnen we de parameters van deze component aanpassen en door ergens in de tijdbalk te klikken krijgen we een beeld van de frame uit de animatie op het gekozen tijdstip.

Hoofdstuk 10

Conclusie en resultaten

Het doel van deze thesis was, zoals de titel het zegt, het dynamische gedrag van kleren te faken. Meer bepaald wil dit zeggen dat we wilden proberen dit dynamische gedrag na te bootsen zonder deze dynamica werkelijk te implementeren maar eerder met behulp van wiskundige formules. Om dit te verwezenlijken hebben we ons gebaseerd op de paper "Faking dynamics of ropes and springs"van Ronen Barzel. Hierin werd een methode geïntroduceerd die succesvol het dynamische gedrag kon nabootsen van 1-dimensionale vervormbare modellen.

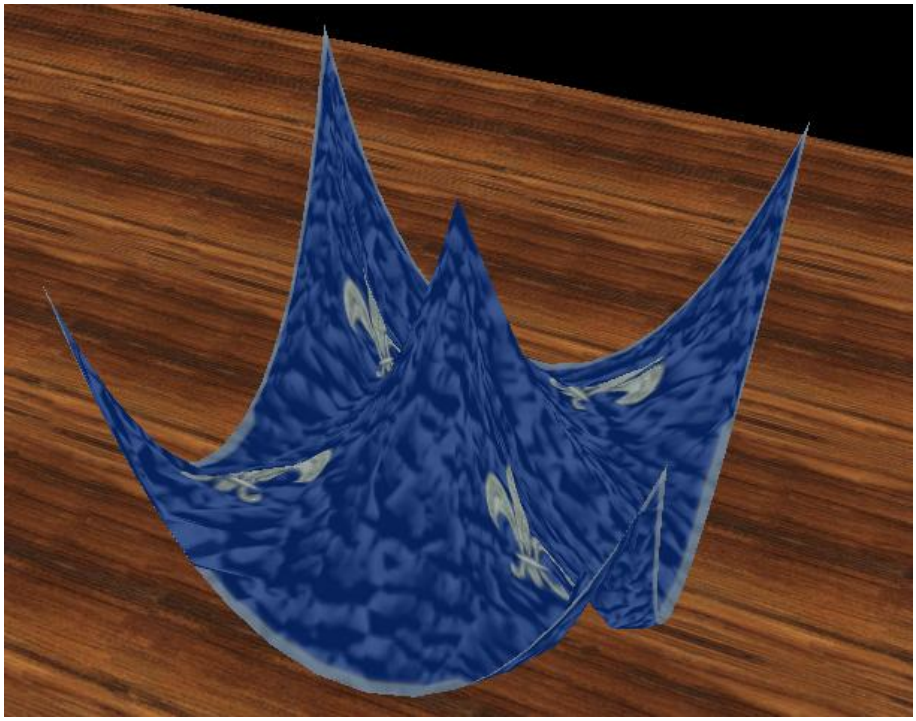
Deze methode hebben we hierna proberen toe te passen op een 2-dimensionaal object, namelijk een doorhangend doek dat opgehangen is aan verschillende punten. We hebben kunnen concluderen dat de methode van Barzel ook in dit geval gebruikt kon worden en zelfs zeer goede resultaten oplevert. Het is mogelijk om in de uiteindelijke applicatie animaties te creëren die redelijk tot zeer realistisch ogen.

Natuurlijk is, zoals ook bij de 1-dimensionale modellen , ervaring en inzicht van de animator nog altijd zeer belangrijk wanneer we echte goede animaties willen creëren met het systeem. Het systeem heeft ook maar een beperkt toepassingsgebied, buiten het creëren van animaties die niet de behoefte hebben om er compleet realistisch uit te zien, kunnen we weinig gebieden bedenken waar we het systeem kunnen toepassen. Maar binnen dit gebied echter bied het systeem wel enorme mogelijkheden en zal het vaak gemakkelijker te gebruiken zijn dan "real dynamics".

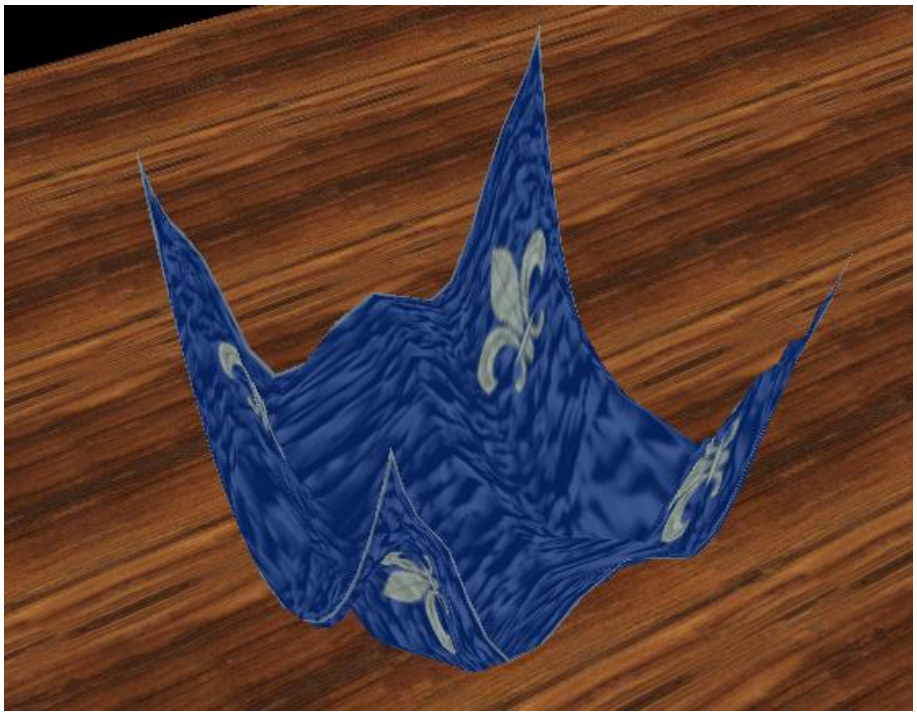
We hebben ook nog geen onderzoek gedaan naar andere modellen, denk bijvoorbeeld maar aan een vlag of een gordijn, of zelfs volledige kledingstukken. En zelfs bij het gebruikte model hebben we niet alle mogelijkheden behan-

deld, zo hebben we bijvoorbeeld enkel een zeer eenvoudige algemene vervorming geïntroduceert en ook bij de golf-vervorming hebben we enkel de eenvoudigste besproken. Ook collision detection is niet aan bod gekomen.

We kunnen dus concluderen dat we geslaagd zijn om fake dynamics te gebruiken bij het visualiseren van een doek en dus in zekere zin ook van kleding. En dit terwijl we slechts een zeer beperkte deelverzameling van alle mogelijkheden hebben geprobeerd. We besluiten dus dat Fake dynamics op kleding zeker mogelijkheden biedt naar de toekomst toe en het dus zeker de moeite loont om hier verder onderzoek naar te doen. We eindigen deze thesis met enkele screenshots uit de bijhorende applicatie.



Figuur 10.1: Een doek opgehangen aan 5 punten na de oppervlaktebenadering.



Figuur 10.2: Een doek opgehangen aan 4 punten waar een golf op inwerkt.

Bibliografie

[1] David Baraff and Andrew Witkin. Large steps in cloth simulation. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 43–54, New York, NY, USA, 1998. ACM Press.

[2] Ronen Barzel. Faking dynamics of ropes and springs. *IEEE Comput. Graph. Appl.*, 17(3):31–39, 1997.

[3] Gareth Bradshaw. Bounding volume hierarchies for level-of-detail collision handling. 2002.

[4] J. Brown, S. Sorkin, C. Bruyns, K. Latombe, and M. Stephanides. Real-time simulation of deformable objects: Tools and application, 2001.

[5] F. Ganovelli, J. Dingliana, and C. O'Sullivan. Buckettree: Improving collision detection between deformable objects.

[6] S. Gottschalk, M. C. Lin, and D. Manocha. OBBTree: A hierarchical structure for rapid interference detection. *Computer Graphics*, 30(Annual Conference Series):171–180, 1996.

[7] Hubbard. Collision detection for interactive graphics applications. 3:218230, 1995.

[8] Marco Hutter and Arnulph Fuhrmann. Optimized continuous collision detection for deformabletriangle meshes. *Science Press, Plzen, Czech Republic*, 2007.

[9] R. L. Grimsdale I. J. Palmer. Collision detection for animation using sphere-trees. 2:105116, 1995.

[10] T. Larsson and T. Akenine-Mller. Collision detection for continuously deforming bodies, 2001.

[11] Thomas Larsson and Tomas Akenine-Möller. A dynamic bounding volume hierarchy for generalized collision detection. *Computers and Graphics*, 30(3):451–460, June 2006.

[12] Rodrigo G. Luque, ao L. D. Comba Jo and Carla M. D. S. Freitas. Broad-phase collision detection using semi-adjusting bsp-trees. In *I3D '05: Proceedings of the 2005 symposium on Interactive 3D graphics and games*, pages 179–186, New York, NY, USA, 2005. ACM Press.

[13] Johannes Mezger, Stefan Kimmerle, and Olaf Etzmuß. Hierarchical Techniques in Collision Detection for Cloth Animation. *Journal of WSCG*, 11(2):322–329, 2003.

[14] Niels Soeffers. Een overzicht en vergelijking van populaire technieken bij cloth-simulation. *Masters thesis, Limburg Universitair Centrum,*, 2002.

[15] M. Teschner, B. Heidelberger, M. Mueller, D. Pomeranets, and M. Gross. Optimized spatial hashing for collision detection of deformable objects, 2003.

[16] Gino van den Bergen. Efficient collision detection of complex deformable models using AABB trees. *Journal of Graphics Tools: JGT*, 2(4):1–14, 1997.

[17] Lode Vanacken. Haptic cloth rendering. 2004-2005.

[18] Pascal Volino and Nadia Magnenat Thalmann. Collision and self-collision detection: Efficient and robust solutions for highly deformable surfaces. In Dimitri Terzopoulos and Daniel Thalmann, editors, *Computer Animation and Simulation '95*, pages 55–65. Springer-Verlag, 1995.

[19] Jerry Weil. The synthesis of cloth objects. In *SIGGRAPH*, pages 49–54, 1986.

[20] Andrew Witkin. Constrained dynamics. *Physically Based Modeling: Principles and Practice*, 1997.

[21] G. Zachmann and E. Langetepe. Geometric data structures for computer graphics, 2003.

[22] Gabriel Zachmann and Rene Weller. Kinetic bounding volume hierarchies for deformable objects. pages 189–196, 2006.

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:
Faking Dynamics of Clothes
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,

Bram Gerits

Datum: **22.08.2007**