Haptic Cloth Rendering

Thesis voorgedragen tot het behalen van de graad van licentiaat in de informatica / doctorandus in de kennistechnologie, afstudeervariant multimedia / informatica

Lode Vanacken

Promotor: Prof. dr. Karin Coninx
Co-Promotor: Prof. dr. Chris Raymaekers
Begeleider: Erwin Cuppens

Academiejaar 2004-2005

Samenvatting

In deze thesis wordt een overzicht gegeven van alle technieken die nodig zijn voor Haptic Cloth Rendering. Haptic Cloth Rendering is, zoals de naam al doet vermoeden, een combinatie van twee technieken, namelijk cloth simulation en haptic rendering. Beide technieken worden eerst apart besproken en daarna gecombineerd. Voor cloth simulation worden mass spring systems besproken, waarbij veren op een naai-patroon verbonden worden. Hierna worden integratietechnieken besproken die broodnodig zijn om tot een stabiele en realistische simulatie te komen. Om gebruik te kunnen maken van cloth simulation zijn er nog een aantal technieken nodig die helpen bij het manipuleren van een cloth zoals beperkingen van de beweegruimte of collision detection. Deze technieken worden allemaal besproken met snelheid in het achterhoofd, omdat het andere deel van deze thesis, haptics, zeer rekenintensief is, vanwege de broodnodige hoge update rate. Een introductie tot haptics wordt gegeven, waarna een aantal technieken om gevoelsterugkoppeling te realiseren worden besproken. Na de bespreking van cloth simulation en haptics gaan we verder met een bespreking van de implementatie. Deze werd gerealiseerd aan de hand van de eerder besproken technieken. We sluiten af met een conclusie.

Voorwoord

Mijn thesis zou nooit tot een goed eind gebracht kunnen worden zonder een bepaald aantal mensen. Daarom wil ik hun via deze weg bedanken.

Eerst en vooral de mensen die me begeleid hebben tijdens het maken van deze thesis, mijn promotor Prof. Dr. Karin Coninx, mijn co-promotor Prof. Dr. Chris Raymaekers en mijn begeleider Erwin Cuppens. Ze hebben me voorzien van tips en suggesties gedurende de loop van mijn thesis.

Tenslotte wil ik ook graag mijn vriendin Vicky en tweelingbroer Wim bedanken voor het nalezen van mijn tekst en hun morele steun; ze waren ook altijd bereid me te helpen indien er problemen waren.

En als allerlaatste mijn ouders, want zonder hun zou ik nooit deze opleiding hebben kunnen volgen en succesvol afronden.

Lode Vanacken

Inhoudsopgave

Lijst van figuren

Lijst van tabellen

Hoofdstuk 1

Inleiding

De titel van deze thesis is *Haptic Cloth Rendering*. Deze term kan in twee grote delen opgesplitst worden: Cloth Simulation en Haptic Rendering. Cloth simulation is het fysisch simuleren van vervormbare weefsels en wordt besproken in deel I van deze tekst. In het tweede deel van de tekst worden haptics besproken, het onderzoeksgebied waarbij onderzoek wordt gedaan naar gevoel. Zowel cloth simulation als haptics zorgen voor een verhoging van de immersiviteit van virtuele omgevingen, maar hebben ook het nadeel dat ze beide grote eisen vergen van het systeem qua rekenkost. Beide onderwerpen bestaan uit een groot aantal onderdelen van de wetenschap. Cloth simulation wordt gerealiseerd door gebruik te maken van een fysisch model dat de realiteit nabootst. In deze thesis is dit model een mass spring systeem waarbij de veren op speciale manieren verbonden worden met particles. Om dit fysisch model te doen bewegen onder krachten zijn er wiskundige technieken, integratietechnieken, nodig. Om cloth simulation te kunnen gebruiken in animaties of in virtuele omgevingen zijn er ook technieken nodig om het cloth te kunnen manipuleren of autonoom te laten interageren met de virtuele omgeving door middel van collision detection. Dit laatste onderzoeksgebied is ook van zeer groot belang bij haptics, naast het onderzoek naar hoe gevoel werkt en waarop gelet moet worden om een haptische applicatie te ontwikkelen. Al deze technieken zullen aan bod komen in de eerste twee delen van deze tekst.

Verder wordt in deze thesis gekeken naar de haalbaarheid en performantie van een realistische cloth simulation, waarbij ook haptics gebruikt worden, aan de hand van een implementatie. Dit is niet zo vanzelfsprekend aangezien beide methodes nogal een grote rekenkost vergen omwille van ingewikkelde berekening (cloth simulation) of omwille van een hoge update rate (haptics). Tijdens de literatuurstudie wordt dan ook vooral aandacht gehecht aan technieken die weinig rekenkost hebben zodat een real-time simulatie mogelijk is. Op basis van deze bespreking werd dan een implementatie gemaakt waarbij een cloth haptic gerenderd wordt, of mooier gezegd: Haptic Cloth Rendering. In het laatste gedeelte van deze tekst zal eerst een bespreking gegeven worden van de cloth simulator die ontwikkeld is, waarna de integratie in een haptics library besproken wordt. Hierna zullen we ook een evaluatie maken van de geïmplementeerde technieken, zodat we een zicht hebben op welke technieken de voorkeur krijgen bij cloth simulation en haptics. En tot slot wordt in het laatste hoofdstuk een conclusie gegeven.

Deel I

Cloth Simulation

Introductie

De literatuurstudie van deze thesis begint met de bespreking van cloth simulation. Hierbij wordt getracht om zo realistisch mogelijke doeken te creëren. Cloth simulation hoort thuis in de wereld van het fysisch modelleren. Bij het fysisch modelleren van bepaalde fenomenen kunnen allerlei aspecten van belang zijn. Bij het bouwen van een schip is de accuraatheid van het watermodel zeer belangrijk, terwijl aan de andere kant in computerspelletjes vooral snelheid en visuele kwaliteit primeren.

Er zijn twee veel gebruikte technieken om cloth simulation te realiseren, Mass Spring Systems en Finite Elements[Hau03]. In deze thesis zullen we gebruik maken van mass spring systemen. Omdat er naast cloth simulation in deze thesis ook nog haptics aan te pas komen. Het gebruik van haptics eist veel rekentijd, zoals zal blijken in deel II. Daarom verkiezen we dus een techniek met een zo laag mogelijke rekenkost en toch nog een goed visueel resultaat. Bij Finite Elements is de rekenkost zonder grove benaderingen zeer hoog. Deze techniek is echter wel nuttig bij ingenieurstoepassingen of numerieke analyse omwille van de veelzijdigheid en superieure convergentie-eigenschappen. Bij mass spring systemen is het dus mogelijk om met een lage rekenkost tot een goede cloth simulation te komen.

We beginnen dit deel met een bespreking van de basisonderdelen van een mass spring systeem: particles, veren en andere krachten (bijvoorbeeld zwaartekracht). In een volgend hoofdstuk zullen we bespreken hoe we een mass spring systeem gebruiken om een cloth te construeren. Hierbij wordt vooral aandacht besteed aan de manier om veren te verbinden met particles. Vervolgens moet het cloth nog tot leven kunnen komen, hiervoor zijn er integratietechnieken nodig. Deze technieken zorgen ervoor dat het cloth kan bewegen door de ruimte rekening houdend met de krachten die op het doek inwerken. Er bestaan veel integratietechnieken die allemaal hun voor- en nadelen hebben, de bespreking in hoofdstuk 4 zal vrij wiskundig zijn, waarbij vooral cloth simulation voor ogen wordt gehouden.

Na al deze technieken besproken te hebben kunnen we een cloth simuleren, maar er ontbreken nog een paar technieken. Stel dat we een cloth willen gebruiken in een simulatie waarbij we een vlag of een tafellaken willen voorstellen. Om een vlag voor te stellen zou het handig zijn de vlag aan een 'vlaggenstok' te kunnen vastmaken, wat meestal met twee hoeken van de vlag gebeurt. Dit kan gezien worden als de beweegruimte van een cloth beperken. Beperkingen zijn dan ook het onderwerp van hoofdstuk 5. Om een tafellaken voor te stellen kunnen beperkingen gebruikt worden, maar het zou veel makkelijker zijn om het doek te draperen over een gemodelleerde tafel zonder enige beperkingen. Om zulk een drapering mogelijk te maken moet er getest worden op een botsing tussen het tafellaken en de tafel. Het efficiënt detecteren van botsingen bij vervormbare objecten wordt besproken in hoofdstuk 6.

Hoofdstuk 2

Mass Spring Systems

Een mass spring systeem is een speciaal soort van particle systeem dat uit een aantal *particles* (ook puntmassa's genoemd) bestaat, deze zijn onderling verbonden met veren. Mass spring systemen worden veel gebruikt in physically based modelling, voor het modelleren van vervormbare voorwerpen[Soe02] (bv. touw, ballon, tafelkleed, ...). In de volgende secties zullen particles meer uitgebreid aan bod komen met de krachten die op zulk een particle kunnen inwerken.

2.1 Particles

Een particle kan gezien worden als een punt met een bepaalde massa, het is tevens het kleinste deeltje van een mass spring systeem. Elke particle bevindt zich op een bepaald tijdstip t in een bepaalde toestand. Deze toestand bevat ook meteen de karakteristieken van een particle[Lan99a]:

- positie[1]: $\mathbf{x}(t)$
- snelheid: $\mathbf{v}(t)$
- massa: m
- kracht vector: $\mathbf{f}(t)$

Een particle beweegt onder de bewegingswetten van Newton, dit wil zeggen dat een kracht die op een particle of zijn versnelling $\mathbf{a}$ (tweede afgeleide van de positie) wordt uitgeoefend gelijk is aan:

$$\mathbf{f} = m.\mathbf{a} \Rightarrow \ddot{\mathbf{x}} = \frac{\mathbf{f}}{m} \tag{2.1}$$

Vergelijking 2.1 is de eerste afgeleide van de snelheid. Wanneer we deze vergelijking integreren, krijgen we de snelheid. De eerste afgeleide van de positie is dus gelijk aan de snelheid (meer informatie over integratie en hoe dit wordt toegepast wordt later uitgelegd in hoofdstuk 4):

[1] vectoren worden voor de leesbaarheid vet gedrukt genoteerd in plaats van op de traditionele $\vec{x}$ manier.

$$\dot{\mathbf{v}} = \frac{\mathbf{f}}{m} \qquad \dot{\mathbf{x}} = \mathbf{v} \tag{2.2}$$

Nogmaals integreren geeft ons de positie van een particle onder een bepaalde kracht op een bepaald tijdstip.

2.2 Krachten uitgeoefend op een particle

Er zijn een aantal veel voorkomende, algemene krachten die op een particle kunnen inwerken. Alle krachten worden geaccumuleerd in de kracht-vector $\mathbf{f}$. We kunnen deze krachten opsplitsen in twee categorieën.

2.2.1 Unaire krachten

Dit zijn krachten die op ieder particle van het mass spring systeem apart worden uitgeoefend. Meestal heeft dit soort kracht dezelfde sterkte voor elke particle of een sterkte afhankelijk van de toestand van een bepaalde particle. Enkele voorbeelden van deze krachten zijn:

Zwaartekracht: Dit is de normale zwaartekracht zoals we deze kennen op aarde. In zijn meest algemene vorm wordt de kracht voorgesteld door een zwaartekracht-vector $\mathbf{g}$ zodat de zwaartekracht in elke richting kan wijzen. De kracht-vector wordt verhoogd met de volgende vergelijking:

$$\mathbf{f} \mathrel{+}= m\mathbf{g}$$

Viscous Drag: Het doel van deze unaire kracht is om beweging tegen te gaan, hierdoor zal de particle langzaamaan tot rust komen in de afwezigheid van andere invloeden of krachten. Dit impliceert ook dat het mass spring systeem numeriek stabieler wordt (zie ook 4.3.1). De vergelijking van deze kracht met k_d als de drag constante is:

$$\mathbf{f} \mathrel{+}= -k_d\mathbf{v}$$

Wind: Er zijn veel manieren om wind te simuleren in een fysische simulatie, we zullen simpelweg een vergelijking geven zonder er in diep detail op in te gaan. Deze vergelijking genereert windstoten:

$$\mathbf{f} \mathrel{+}= \mathbf{v}_{wind}.(\sin(t^2) + 0.5)$$

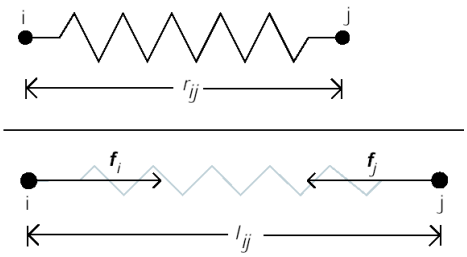
Met als $\mathbf{v}_{wind}$ de richting, of snelheid, van de wind en waarbij t lineair geïnterpoleerd wordt over de tijd. Voor een meer geavanceerde en realistische manier om wind te implementeren worden er twee technieken, die goede resultaten behalen, besproken in [KKTW04]. Deze maken gebruik van de Navier-Stokes vergelijkingen of van een particle tracing methode, hierdoor kan er ook rekening gehouden worden met andere objecten in de simulatie om windvrije plaatsen te simuleren. Het bespreken van deze twee technieken valt buiten het bestek van deze thesis.

2.2.2 N-aire Krachten

N-aire krachten werken op meer dan één particle gelijktijdig. De meest voor de hand liggende kracht, die we ook meer in detail zullen behandelen, is een veer. Maar meer ongewone krachten kunnen gebruikt worden om bijvoorbeeld met een particle systeem een vloeistof voor te stellen. Elke particle heeft dan een straal en elke particle oefent twee krachten uit, drag en afstoting/aantrekking, op alle andere particles. Deze techniek noemt men Globular Dynamics[MP89].

Lineair gedempte veren

Een veer probeert altijd zijn rustlengte te behouden of anders verwoord, de veer streeft naar minimale energie. Wanneer een veer is verbonden met twee objecten (in ons geval twee particles i en j), een binaire kracht dus, zal de veer de nodige krachten uitoefenen, rekken of samendrukken, op deze beide objecten totdat de rustlengte terug bereikt is. De kracht waarmee een veer zijn rustlengte r_{ij} probeert te bereiken is afhankelijk van de stijfheidsconstante k_s en de huidige lengte l_{ij}.



Figuur 2.1: Een lineaire veer en de krachten die ze uitoefent[Mel02]

Een lineaire veer, het simpelste type veer, verbonden tussen particle i en j kan gezien worden in figuur 2.1. De kracht van de lineaire veer met huidige lengte l_{ij} uitgeoefend op particle i kan berekend worden met behulp van vergelijking 2.3:

$$f_i = -k_s(|l_{ij}| - r_{ij})\frac{l_{ij}}{|l_{ij}|} \tag{2.3}$$

De kracht op particle j is de omgekeerde van die die wordt uitgeoefend op particle i:

$$f_j = -f_i$$

Een lineaire veer zal bijna altijd oscilleren rond zijn rustlengte. Omdat dit een ongewenst effect is introduceren we een gedempte lineaire veer. Dit kan gemodelleerd worden door een demping constante k_d die inwendige wrijving voorstelt. Deze inwendige wrijving zal de kracht van de veer reduceren afhankelijk van de relatieve snelheid van de twee objecten verbonden met de veer:

$$f_d = k_d\frac{[(v_j - v_i)\cdot l_{ij}]}{|l_{ij}|}\frac{l_{ij}}{|l_{ij}|}$$

2.3 Conclusie

In dit hoofdstuk werden alle delen van een mass spring systeem besproken. De basisdelen zijn particles, veren en andere soorten krachten die inwerken op de particles. Er zijn twee soorten krachten, unaire en n-aire. Deze krachten, zoals bijvoorbeeld zwaartekracht, zorgen ervoor dat de simulatie realistischer zal overkomen. Merk op dat een veer een binaire kracht is en dus ook bij de n-aire krachten hoort. Particles kunnen gezien worden als kleine puntjes in de ruimte met een massa die bewegen onder de wetten van Newton.

Als we deze particles verbinden met veren krijgen we een mass spring systeem. In het volgende hoofdstuk zullen we bespreken hoe we hiermee een cloth vormen.

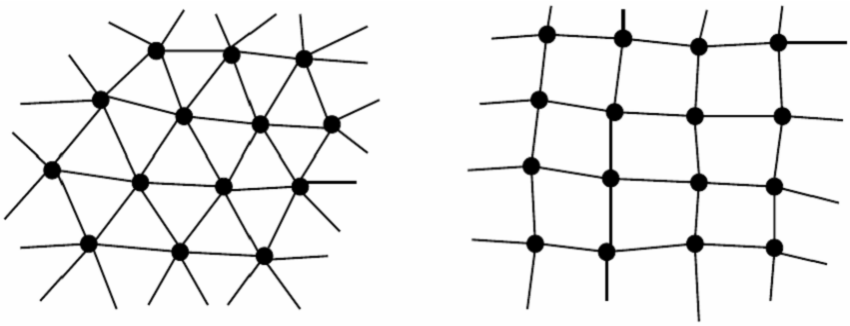
Hoofdstuk 3

Cloth Topology

Een mass spring systeem is de basis voor cloth simulation, hiermee wordt er een doek gemodelleerd. In dit hoofdstuk zullen we bespreken hoe we het oppervlak van het doek voorstellen en hoe we de veren verbinden over dit oppervlak.

3.1 Oppervlakte

Er zijn verscheidene manieren om particles te verbinden met elkaar om een oppervlakte te vormen, maar de twee meest voorkomende manieren zijn een driehoekige of vierhoekige mesh (zie figuur 3.1). Het moet echter gezegd worden dat dit niet de manier is om de veren tussen particles te plaatsen, maar dat deze de voorstelling los is van het mass spring systeem.



Figuur 3.1: Een driehoekige of vierhoekige mesh om een cloth oppervlak te modelleren[Mel02]

Er is geen echt overwegend voordeel voor één van deze twee. Een vierhoekig oppervlak lijkt meer op een weefpatroon en het zal makkelijker blijken om veren voor te stellen met als denkwijze een vierhoekig oppervlak (zie sectie 3.2). Maar aan de andere kant heeft het driehoekig oppervlak als voordeel dat het makkelijkere berekeningen heeft voor collision detection. Voor het beste van twee werelden te verkrijgen veronderstellen we dat het cloth een vierhoek heeft van omtrek en als inhoud driehoeken die het vierkant opvullen. Het duidelijke nadeel is natuurlijk dat er geen cloth gevormd kan worden met een speciale vorm. Afhankelijk van het doel van de cloth simulation, dient er een afweging gemaakt te worden.

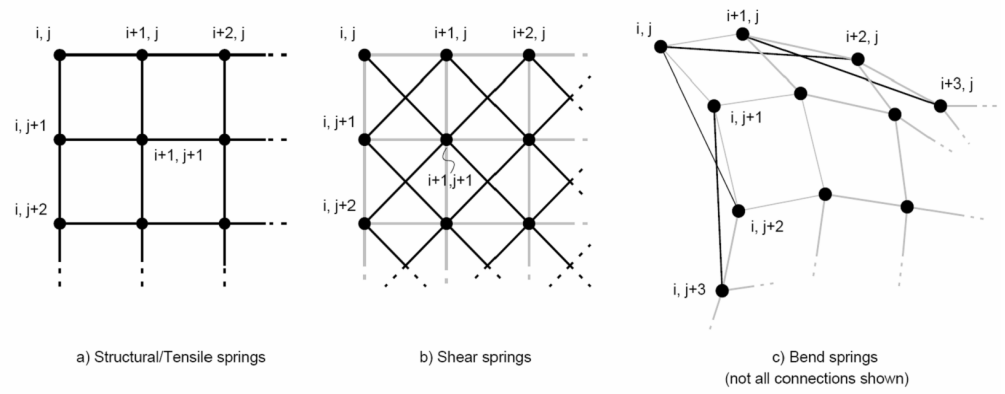
3.2 Veren

Om de drie fysische eigenschappen van cloth te simuleren: weerstand tot uitrekking, afschuiving en buiging; worden er meestal drie types veren gebruikt (de stijfheid, demping en manier om particles te verbinden verschilt)[Lan99b, Pro95, Soe02, Mel02]. Vrij recentelijk hebben Choi et al.[CK02] een methode geïntroduceerd om veren te gebruiken met twee verschillende manieren om de krachten te berekenen (interaction models genoemd in de desbetreffende paper). Deze techniek heeft als voordeel dat het *post-buckling* probleem van cloth, het probleem dat een doek in een stilstaande plooiing voor instabiliteit zorgt (meestal kan het doek gewoon niet tot stilstand worden gebracht in een plooiing vanwege dit probleem), wordt opgelost zonder een extra demping kracht in te voeren in het mass spring systeem (een voorbeeld van zulk een demping kracht is viscous drag, zie sectie 2.2.1). Beide methodes zullen worden besproken.

Als we ons alleen concentreren op hoe veren tussen particles verbonden worden is het simpeler om te denken aan de vierhoekige oppervlakte geïntroduceerd in sectie 3.1. Op deze manier kan ook elke particle op een denkbeeldige matrix gemapt worden, met als index (i, j).

3.2.1 Drie type veren

Zoals eerder vermeld worden er drie type veren gebruikt om de drie fysische eigenschappen van een cloth te modelleren:



Figuur 3.2: De 3 type veren gebruikt om een cloth the modelleren

Structurele veren: Het *structurele* type stelt de *weerstand tot uitrekking* eigenschap voor. Dit type modelleert de vezels van een cloth die in schering en inslag richtingen lopen. Een echt cloth rekt niet heel hard. Elke particle is verbonden met zijn kortste buren of anders gezegd, ze zijn verbonden met de particle boven, onder, links en rechts van zich, zie figuur 3.2a. Als we enkel dit type van veren zouden gebruiken in onze cloth simulation zou het doek zijn vorm niet behouden, het zou te hard doorzakken.

Afschuif-veren: het *afschuif*-type stelt de *afschuiving* eigenschap voor. Dit type maakt afschuiving in het vlak moeilijker omdat particles diagonaal verbonden worden, ze worden

verbonden met hun buren rechtsboven, rechtsonder, linksonder en linksboven (zie figuur 3.2b). De toevoeging van dit type samen met het structurele type zorgt ervoor dat het cloth veel realistischer wordt omdat het minder zal uitrekken en doorzakken. Zonder het afschuif-type zou het cloth, indien het opgenomen wordt met één particle, herleid worden tot een draad. De andere particles schuiven dan als het ware af. Er blijft zich nog een probleem voordoen, indien het cloth zich op de vloer bevindt zal het opproppen tot een grote massa van *springy spaghetti*[Lan99b]. Om dit te voorkomen introduceren we nog een derde en laatste type van veer.

Buig-veren: Het *buig*-type of ook flexion type genaamd[Pro95], stelt de *buiging* eigenschap voor. Zoals de naam aangeeft zorgen deze veren ervoor dat het cloth moeilijker buigt. Maar echt cloth buigt eigenlijk heel gemakkelijk, daarom zullen ook de veer constanten hier meestal beduidend minder sterk zijn dan die van de twee andere types en dient dit type meer te voorkomen dat het doek perfect opplooibaar is (vb. is wanneer we een tafel doek opvouwen in twee delen, deze twee delen kunnen nog steed van elkaar onderscheiden worden). Particles worden verbonden met de buren van hun kortste buren, je kan het bezien alsof je de veren stretcht over twee cellen langszij de structurele veren, zie figuur 3.2c.

In het implementatiegedeelte zijn figuren te vinden waarbij duidelijk zichtbaar is dat het toevoegen van afschuif- en buig-veren zorgt voor extra realisme, zie 9.2.

3.2.2 Twee verschillende veren

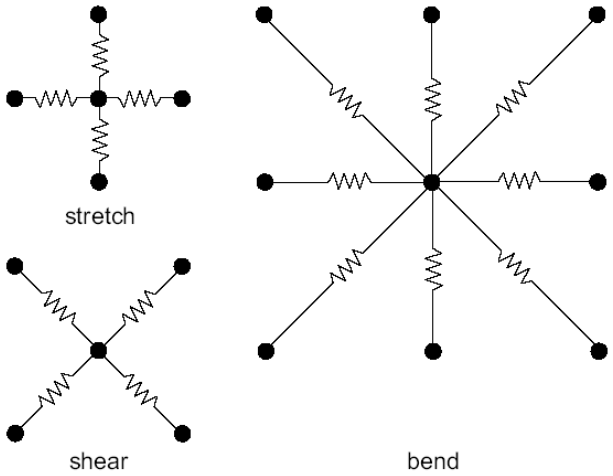
Wanneer we gebruik maken van twee verschillende veren, moet één van deze veren meer dan één fysische eigenschap van cloth modelleren. Choi et al.[CK02] stelden twee verschillende manieren voor om de kracht van een veer te berekenen. Deze manieren kregen geen naam op basis van de eigenschappen die ze voorstellen of de manier waarop ze die kracht berekenen maar op basis van de manier waarop ze particles verbinden.

Type 1 **(sequentiële verbindingen):** Dit type veer zorgt voor de rek- en afschuivingsweerstand. Particle (i, j) is verbonden met $(i\pm1, j)$, $(i, j\pm1)$ en $(i\pm1, j\pm1)$, zie de linkerkant van figuur 3.3. De manier waarop dit type zijn krachten berekent is een simpele lineaire veer.

Type 2 **(interlaced verbindingen):** Deze verbindingen zijn verantwoordelijk voor buigings- en samendrukkingsweerstand. Particle (i, j) is verbonden met $(i \pm 2, j)$, $(i, j \pm 2)$ en $(i \pm 2, j \pm 2)$, zie de rechterkant van figuur 3.3. Zoals eerder gezegd zorgt het buigen van het cloth voor een onstabiele toestand, wat als gevolg kan hebben dat het cloth divergeert. Maar eigenlijk gaat het cloth slechts heel even in deze onstabiele toestand zijn. De krachtberekening van type 2 is verantwoordelijk voor de post-buckling gecreëerd door samendrukkings- en buigkrachten. Deze berekening zal er voor zorgen dat de post-buckling positie benaderd wordt zonder door de onstabiele toestand te gaan.

Een balkstructuur benadert de regio tussen de twee particles. Voor het buigen van de balk, heeft de balk een lengte L. Wanneer de structuur buigt onder een samendrukkende kracht zal het uiteindelijk een stabiele evenwichtige structuur bereiken. Om deze

evenwichtige structuur te voorspellen wordt de evenwichtsvergelijking van de momenten onder de vastgespelde uiteinden gebruikt. De manier om dit te berekenen wordt uitvoerig besproken in [CK02] en zou ons hier te ver afleiden.



Figuur 3.3: De twee verschillende veren en hun verbindingen[BA04]

Dit is de eerste paper ooit die een praktische en stabiele oplossing brengt voor het post-buckling fenomeen zonder dat er nood is aan een lange simulatietijd of toevoeging van een dempingskracht die er voor zorgt dat er meestal ingeboet moet worden aan realisme. Voor het genereren van realistische animatiescenes met kleren voor een personage is het zeer handig dat er buigingen of plooiingen kunnen voorkomen in een kledingstuk zonder dat dit extra problemen oplevert tijdens de simulatie.

3.2.3 'Geen' Veren

Fuhrmann et al.[FGL03] kwamen op het idee om veren te gebruiken die geen krachten uitoefenen maar op een andere manier voor het nodige realisme te zorgen. Om volledig correct te zijn gebruiken ze eigenlijk geen veren. Ze maken gebruik van een driehoekige mesh waarbij de edges als *valse* veren worden beschouwd. Omdat in deze techniek er geen veren zijn die kracht uitoefenen, maar gebruik gemaakt wordt van Inverse Dynamica worden deze ook in de sectie daarover behandeld, zie 4.7.

3.3 Conclusie

Een cloth heeft een aantal eigenschappen: weerstand tot uitrekking, afschuiving en buiging. Om deze te modelleren worden de veren van een mass spring systeem gebruikt. Er zijn echter niet veel verschillende manieren in omgang om deze veren te gebruiken. De meest voorkomende methode en tevens de oudste, gebruikt drie type veren, terwijl vrij recent deze methode werd *aangepast* door twee type veren te gebruiken die op verschillende manieren hun werk doen.

In hoofdstukken 2 en 3 bespraken we uitvoerig de topologie van een cloth en de eigenschappen van een mass spring systeem. Om tot een realistische cloth simulation te komen hebben we nog nood aan een integratietechniek. Verschillende alternatieve methodes komen aan bod in het volgende hoofdstuk.

Hoofdstuk 4

Integratietechnieken

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. Integreren is een noodzaak indien we een bepaald object willen laten bewegen. In sectie 2.1 werden de eigenschappen van een particle opgesomd. Dit kan gezien worden als een zeer klein object dat beweegt. Deze eigenschappen kunnen gezien worden als de toestand van een particle op tijdstip t, om nu op één tijdstip verder te raken, $t+h$, moet deze toestand geïntegreerd worden.

In dit hoofdstuk zullen we uitleggen wat een *Ordinary Differential Equation*, kortweg ODE, is. Hiermee stellen we de bewegingseigenschappen van een particle voor. Numerieke integratietechnieken om ODE's op te lossen, met de bijkomstige problemen, worden ook besproken.

4.1 Wat is een ODE?

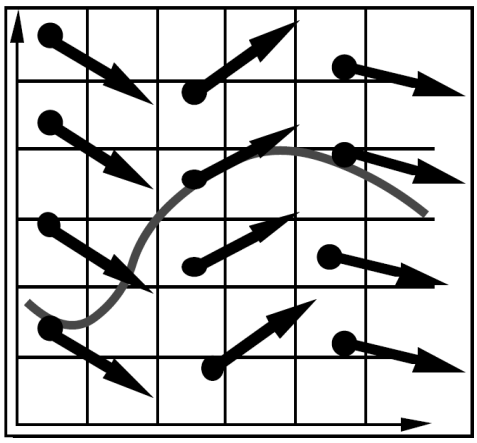
Een differentiaalvergelijking beschrijft de relatie tussen een onbekende functie en zijn afgeleiden. Als we een differentiaalvergelijking willen oplossen moeten we een functie zoeken die aan de relatie en enkele extra condities voldoet[WB01]. Een differentiaalvergelijking is van orde n als er afgeleiden tot die orde voorkomen[Mat]:

$$f(x, y, y', \ldots, y^{(n)}) = 0$$

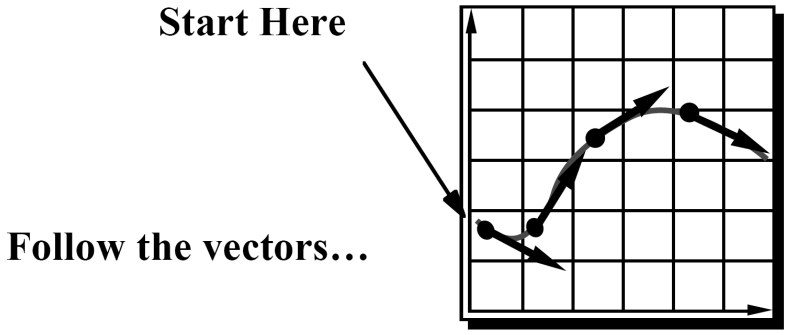
Waarbij y een functie van x is, y' de eerste afgeleide naar x en $y^{(n)}$ de n-de afgeleide naar x. Meer details over ODE's vind je op [Mat], vanaf nu concentreren we ons enkel op eerste orde ODE's. Deze kunnen gebruikt worden om *initial value* problemen op te lossen. Dit zijn problemen waarbij we in een begintoestand zitten en deze toestand willen volgen over de tijd.

Het simuleren van de beweging van een particle kan gezien worden als een initial value probleem waarbij het systeem een begintoestand $\mathbf{x}(t_0)$ of $\mathbf{x}_0$ heeft en we de toestand $\mathbf{x}$ willen volgen over de tijd. Maar we kennen enkel de afgeleide van de toestand $\dot{\mathbf{x}}$. Dit kan dus geschreven worden als een eerste orde ODE:

$$\dot{\mathbf{x}} = f(\mathbf{x}, t) \tag{4.1}$$



(a) Het vectorveld gedefinieerd door $f(\mathbf{x}, t)$



(b) Een initial value probleem waarbij de begintoestand het vectorveld volgt

Figuur 4.1: Grafische voorstelling van het oplossen van een ODE [WB01]

In vergelijking 4.1 is f een bekende functie, $\mathbf{x}$ de toestand van het systeem en $\dot{\mathbf{x}}$ de tijdsafgeleide van $\mathbf{x}$. Doordat f gekend is kunnen we dus ook $\dot{\mathbf{x}}$ berekenen voor elke ($\mathbf{x}$,t) die we meegeven.

We kunnen het oplossen van een ODE makkelijk grafisch visualiseren in $2D$. Op elke plaats $\mathbf{x}$ kan f geëvalueerd worden en is de afgeleide $\dot{\mathbf{x}}$ dus gekend. Dit kan dus voorgesteld worden als een vectorveld met als vector op plaats $\mathbf{x}$ de snelheid die een bepaald punt $\mathbf{p}$ moet hebben als $\mathbf{p}$ ooit door $\mathbf{x}$ heengaat (zie figuur 4.1(a)). $\mathbf{p}$ volgt dus het vectorveld en als $\mathbf{p}$ nu aanzien wordt als de begintoestand $\mathbf{x}_0$, kan de toestand van een bepaald moment bekomen worden door het vectorveld te volgen (zie figuur 4.1(b)).

In ons geval zijn we alleen geïnteresseerd in numerieke oplossingen—analytische oplossingen zijn zelden voor handen en het heeft dus geen zin deze proberen te bekomen—waarbij we starten met een begintoestand en discrete tijdstappen nemen. Om één stap te nemen gebruiken we f om de verandering $\triangle\mathbf{x}$ in $\mathbf{x}$ te benaderen en tellen dit op met $\mathbf{x}$:

$$\mathbf{x}(t+h) = \mathbf{x}(t) + hf(\mathbf{x}, t)$$

Tot nu toe hebben we altijd gesproken over een eerste orde ODE waarin f de verandering in $\mathbf{x}$ benaderde over de tijd. Maar bij particles en de bewegingswetten van Newton hebben we een

ODE van orde twee. Dit komt doordat we met de versnelling werken die de tweede afgeleide is van de positie (zie vergelijking 2.1). Om deze tweede orde ODE op te lossen moeten we deze converteren naar twee eerste orde ODE's. Dit kan simpelweg gedaan worden door de snelheid te gebruiken. In vergelijking 2.2 zien we de twee eerste orde ODE's die opgelost moeten worden. Ze worden voor de duidelijkheid hier even herhaald:

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

Aangezien de kracht $\mathbf{f}$ afhankelijk is van $\mathbf{x}$ en t krijgen we terug onze eerste vergelijking van een eerste orde ODE, vergelijking 4.1.

Doordat er discrete tijdstappen genomen worden en de ODE's numeriek opgelost worden is de gevonden oplossing slechts een benadering van de echte. Jammer genoeg accumuleren deze fouten doordat de nieuw berekende toestand, die een benadering is, gebruikt wordt als de nieuwe begintoestand. Dit kan er dus voor zorgen dat we heel ver van de reële oplossing afdrijven. Deze en andere problemen zullen besproken worden in de uitleg van de expliciete Euler methode, zie 4.3.1.

Er zullen drie groepen van numerieke integratietechnieken besproken worden, plus een techniek die twee groepen gebruikt. De verdeling van deze groepen komt tot stand door de manier waarop ze werken (lineair of niet-lineair) of hoe accuraat ze zijn en dus ook hun stabiliteit. Een eerste groep is die van *expliciete* integratietechnieken. Deze ligt reeds jaren al in een vaste plooi, er wordt nauwelijks tot geen onderzoek meer naar gedaan. Aan de andere kant zijn er de *impliciete* integratietechnieken, deze worden steeds meer toegepast en onderzocht. Het gebruik ervan kwam vooral in een stroomversnelling door de paper van Baraff en Witkin[BW98]. Als derde hebben we de *verlet* integratietechnieken. Deze werden eigenlijk toegepast in de moleculaire dynamica, totdat Jakobsen[Jak01] ze introduceerde bij mass spring systemen. Als laatste is er de IMEX-techniek die de impliciete en expliciete groep combineert.

Alvorens we in de volgende secties ingaan op de verschillende groepen integratietechnieken leggen we eerst nog even het verband uit tussen de toestand $\mathbf{x}$, zijn afgeleide toestand $\dot{\mathbf{x}}$ en een mass spring systeem.

4.2 De toestandsvector

De toestandsvector bevat de toestand van de simulatie [Wit01b]. Dit is een vector met lengte $6n$ lengte waarbij n het aantal particles van de simulatie voorstelt. In de toestandsvector $\mathbf{x}(t)$ op tijdstip t zitten de positie $\mathbf{x}(t)$ en de snelheid $\mathbf{v}(t)$ van alle particles van de simulatie. Merk op dat voor zowel de positie als de toestandvector dezelfde notatie gebruikt wordt, maar het zal in het vervolg altijd duidelijk zijn over welke term het nu precies gaat. Deze vector is dus een concatenatie van de toestanden van de particles. De toestand van één bepaalde particle kan dus voorgesteld worden door een 6-vector, $[x_1, x_2, x_3, v_1, v_2, v_3]$.

Voor het oplossen van een ODE is ook de afgeleide toestand nodig en dus ook de afgeleide toestandsvector $\dot{\mathbf{x}}$. Deze wordt op dezelfde manier als de toestandsvector opgebouwd, maar waarbij de afgeleide toestand van één particle ook *phase space* wordt genoemd: $[\dot{x}_1, \dot{x}_2, \dot{x}_3, \dot{v}_1, \dot{v}_2, \dot{v}_3] = [v_1, v_2, v_3, \frac{f_1}{m}, \frac{f_2}{m}, \frac{f_3}{m}]$.

Beide vectoren worden opgesteld in elke simulatiestap waarmee dan de volgende simulatiestap berekend wordt. Daarna wordt de bekomen vector terug toegekend aan de particles in kwestie.

4.3 Expliciete Integratie

Er zijn drie bekende expliciete integratietechnieken: (forward) Euler, Midpoint en Runge-Kutta. Al deze technieken bouwen op de Taylor reeks waaruit ook hun numerieke nauwkeurigheid kan bekomen worden [WB01]:

$$\mathbf{x}(t+h) = \mathbf{x}(t) + h\dot{\mathbf{x}}(t) + \frac{h^2}{2!}\ddot{\mathbf{x}}(t) + \frac{h^3}{3!}\mathbf{x}'''(t) + \ldots + \frac{h^n}{n!}\frac{d^n\mathbf{x}}{dt^n}(t) + O(h^{n+1}) \tag{4.2}$$

De numerieke nauwkeurigheid zal dus altijd van orde h^{n+1} zijn omdat er altijd met een eindig aantal termen gewerkt wordt [Soe02].

4.3.1 Euler methode

De Euler methode is een snelle maar slechts weinig accurate methode. Ze kan rechtstreeks afgeleid worden van vergelijking 4.2 door te stoppen na de tweede term:

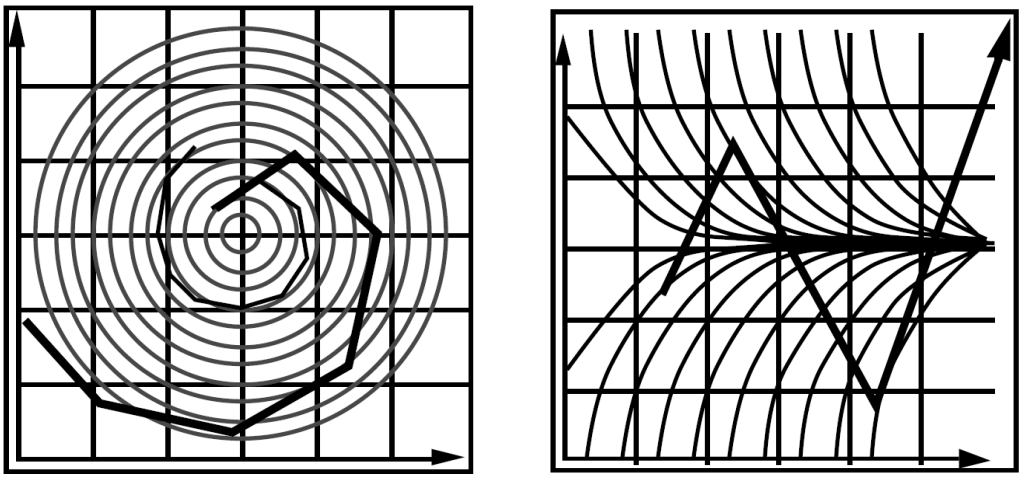
$$\mathbf{x}(t+h) = \mathbf{x}(t) + h\dot{\mathbf{x}}(t) \tag{4.3}$$

De nauwkeurigheid is $O(h^2)$. Door deze lage nauwkeurigheid is deze methode alleen nuttig voor real-time simulaties met niet stramme differentiaalvergelijkingen. Deze zeer simpele integratietechniek leent er dus ook toe om enkele problemen die numerieke integratietechnieken in het algemeen bezitten, te behandelen.

Zoals reeds eerder vermeld accumuleren fouten in de gevonden oplossing. Stel dat we een toestand willen benaderen die een cirkel voorstelt[WB01]. Dan zal dit nooit lukken met de Euler methode. Dit kan ingezien worden door naar de grafische representatie te kijken in figuur 4.2. Bij Euler zal elke stap een lijnstuk vormen, dat geschaleerd wordt aan de hand van de grootte van de tijdstap h. Indien we dus een cirkel proberen te volgen zullen we een spiraal krijgen die afdrijft van de cirkel, zie figuur 4.2(a). Het verkleinen van de tijdstappen heeft geen zin, zelfs de kleinste tijdstap zou afdrijven, alleen langzamer.

Een ander probleem dat zich kan voordoen bij numerieke integratie en in zeer sterke mate aanwezig is bij Euler is de instabiliteit van de integratie; de simulatie kan 'exploderen'. Deze zal gaan oscilleren rond de juiste oplossing en na een bepaalde tijd zeer ver verwijderd zijn van deze juiste oplossing. In figuur 4.2(b) zie je dit probleem in kwestie. De tijdstap van de simulatie is te groot waardoor duidelijk zichtbaar is dat de simulatie oscilleert rond nul maar steeds verder verwijderd raakt van nul. In dit geval is het mogelijk om uit te rekenen hoe groot de tijdstap minimaal moet zijn voor een correcte simulatie, zie [WB01] voor meer details.

Dit probleem doet zich zeer vaak voor bij mass spring systemen, indien er stijve veren gebruikt worden, veren met een hoge stijfheidsconstante k_s. Stijve veren zullen op particles grote krachten uitoefenen en dus zorgen voor grote onnauwkeurige verplaatsingen. Hierdoor zal de



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

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

Figuur 4.2: Twee veel voorkomende problemen van integratie[WB01]

simulatie dus ook ontploffen, wat een effect is dat we ten allen tijden willen vermijden. Er zijn gelukkigerwijs een aantal oplossingen:

- De massa van elke particle kan verhoogd worden zodat de stijve veer minder invloed heeft. Dit heeft natuurlijk als groot nadeel dat de massa van een particle soms enorm hoog moet worden, wat leidt tot een cloth van $100kg$ en dat is natuurlijk niet meer realistisch (zie ook de sectie over adaptieve tijdstappen 4.3.4).
- Door het toevoegen van een extra externe kracht aan het systeem, zoals Viscous Drag (zie sectie 2.2.1) wordt het systeem afgeremd en zal het oscilleren van de simulatie tegengegaan worden. Er dient opgepast te worden dat er geen te sterke externe kracht toegevoegd wordt aan de simulatie. Dit kan er voor zorgen dat de simulatie niet meer realistisch oogt.
- Een andere, meer voor de hand liggende oplossing is om een andere integratietechniek te gebruiken die f beter benadert. Het is bijvoorbeeld mogelijk om meer termen van de Taylor reeks te gebruiken, wat we in de volgende sectie ook zullen doen. Maar door het toevoegen van nieuwe termen neemt de computationele kracht toe omdat er meer termen uitgerekend moeten worden. De echte efficiëntie van een methode hangt eigenlijk af van de grootte van de tijdstappen die deze methode kan nemen terwijl de nauwkeurigheid en stabiliteit behouden blijven samen met de rekenkost per stap[Wit01b].
- Het gebruik van Inverse Dynamica zal er voor zorgen dat stijve veren niet meer of in mindere mate nodig zijn. op deze techniek wordt er dieper ingegaan in sectie 4.7.

4.3.2 Midpoint methode

Deze methode gebruikt één term meer van de Taylor reeks en zal hierdoor een nauwkeurigheid hebben van $O(h^3)$, maar met als gevolg dat er meer rekenkracht nodig is omdat we $\ddot{\mathbf{x}}$ moeten

berekenen wat veel meer rekenkracht kost:

$$\mathbf{x}(t+h) = \mathbf{x}(t) + h\dot{\mathbf{x}}(t) + \frac{h^2}{2!}\ddot{\mathbf{x}}(t) + O(h^3) \tag{4.4}$$

Het is mogelijk om de berekening van $\ddot{\mathbf{x}}$ te optimaliseren. Dit kan indien aangenomen wordt dat $\dot{\mathbf{x}} = f(\mathbf{x}, t)$ enkel indirect van de tijd afhangt[WB01]:

$$\ddot{\mathbf{x}} = \frac{\partial f}{\partial \mathbf{x}}\dot{\mathbf{x}} = f'f$$

Om te voorkomen dat we f' moeten evalueren—wat zeer veel rekenkracht zou vereisen en eigenlijk ook complex is—kunnen we de tweede orde term benaderen in termen van f en substitueren in vergelijking 4.4. Om dit te doen moeten we f ontbinden als Taylor polynoom:

$$f(\mathbf{x} + \triangle\mathbf{x}) = f(\mathbf{x}) + \triangle\mathbf{x}f'(\mathbf{x}) + O(\triangle\mathbf{x}^2) \tag{4.5}$$

We kiezen $\triangle\mathbf{x} = \frac{h}{2}f(\mathbf{x})$ opdat we $\ddot{\mathbf{x}}$ kunnen invoegen in vergelijking 4.5:

$$\begin{aligned} f(\mathbf{x} + \frac{h}{2}f(\mathbf{x})) = f(\mathbf{x}) + \frac{h}{2}f(\mathbf{x})f'(\mathbf{x}) + O(h^2) &= f(\mathbf{x}) + \frac{h}{2}\ddot{\mathbf{x}}(t) + O(h^2) \\ &\Downarrow \text{ vermenigvuldigen met } h \text{ en herschikken} \\ \frac{h^2}{2}\ddot{\mathbf{x}}(t) + O(h^3) &= h\left(f(\mathbf{x} + \frac{h}{2}f(\mathbf{x})) - f(\mathbf{x})\right) \end{aligned}$$

Het rechterlid moet nu enkel nog ingevuld worden in vergelijking 4.4 en dan bekomen we de vergelijking voor midpointintegratie met een nauwkeurigheid van $O(h^3)$:

$$\mathbf{x}(t+h) = \mathbf{x}(t) + h\left(f(\mathbf{x} + \frac{h}{2}f(\mathbf{x}))\right)$$

Deze methode noemt midpoint omdat deze eerst een Euler stap berekent en daarna de tweede afgeleide in het middelpunt van die stap gebruikt om de volgende toestand van $\mathbf{x}$ te vinden.

4.3.3 Runge-Kutta methode

Als de snelheid van de simulatie niet van belang is, is het ook mogelijk om meer en meer termen te introduceren vanuit een Taylor reeks. Hierdoor zal er meer en meer rekenwerk nodig zijn. De meest gekende Runge-Kutta is deze van orde vier, of RK4. Deze wordt ook het meest gebruikt omdat het een goede balans schept tussen benodigde rekenkracht en de verwachte nauwkeurigheid[VMT01]. Het uitrekenen van deze methode door gebruik te maken van een Taylor reeks is analoog aan de midpoint methode. We geven hier gewoon de uiteindelijke

formule:

$$\begin{aligned} k_1 &= hf(\mathbf{x}, t) \\ k_2 &= hf(\mathbf{x} + \frac{k_1}{2}, t + \frac{h}{2}) \\ k_3 &= hf(\mathbf{x} + \frac{k_2}{2}, t + \frac{h}{2}) \\ k_4 &= hf(\mathbf{x} + k_3, t + h) \\ \mathbf{x}(t+h) &= \mathbf{x} + \frac{1}{6}k_1 + \frac{1}{3}k_2 + \frac{1}{3}k_3 + \frac{1}{6}k_4 \end{aligned}$$

4.3.4 Adaptieve tijdstappen

Eén van de grootste problemen bij het numeriek oplossen van een ODE, eender welke integratiemethode gekozen is, is het kiezen van een goede tijdstap[Wit01b]. Wat we natuurlijk willen is om de tijdstap h zo groot mogelijk te kiezen zonder dat er te veel accuraatheid verloren gaat of het systeem instabiel wordt. In [Pro95] wordt uitgelegd hoe voor de Euler methode berekend kan worden hoe groot de tijdstap moet zijn opdat het systeem stabiel is. De grootte van de tijdstap moet kleiner zijn dan de natuurlijke periode T van het systeem. De natuurlijke periode T van een systeem hangt af van de stijfste veer k_{max} en de kleinste massa m_{min} aanwezig in dat systeem:

$$T = \pi\sqrt{\frac{m_{min}}{k_{max}}}$$

Dit is de theoretische benadering. Omdat er ook nog andere factoren meespelen zoals het visuele aspect is deze formule echter niet meer volledig juist. Vassilev et al.[VSC01] meldden dat ze een tijdstap $0.4T$ moesten gebruiken.

Een andere mogelijkheid is om de grootte van de stap te laten variëren. De grootte dient gevarieerd te worden gebruik makend van de fout van één simulatiestap. Als de fout klein genoeg is kunnen we de stapgrootte vergroten en andersom als de fout te groot is. De marge waarmee we de tijdstap h willen veranderen moet berekend worden. Deze berekening kan gedaan worden door $\mathbf{x_a}$ te berekenen met een tijdstap h en $\mathbf{x_b}$ door twee stappen te nemen van grootte $\frac{h}{2}$. Een geschatte grootte van de fout is de absolute waarde van het verschil van deze twee waarden:

$$e = |\mathbf{x_a} - \mathbf{x_b}|$$

De optimale tijdstap kan nu uitgerekend worden door gebruik te maken van[Soe02]:

$$\sqrt{\frac{\text{maximum toegelaten foutmarge}}{\text{huidige foutmarge}}} = \lambda \tag{4.6}$$

De bekomen λ is de factor waarmee de huidige stapgrootte moet vermenigvuldigd worden om de *optimale* stapgrootte te verkrijgen. Stel dat we per stap een maximale fout van 10^{-4}

toelaten en de huidige fout is 10^{-8}, dan zal de tijdstap moeten aangepast worden met factor $\sqrt{\frac{10^{-4}}{10^{-8}}} = 100$.

Merk op dat in vergelijking 4.6 er werd van uitgegaan dat $\mathbf{x_a}$ en $\mathbf{x_b}$ een fout hebben van $O(h^2)$ (er werd een Euler stap genomen). Indien deze met een andere foutmarge berekend werden moet in vergelijking 4.6 de n-de machtswortel getrokken worden.

4.4 Impliciete Integratie

Expliciete integratie heeft het probleem dat de simulatie nooit 100% stabiel verklaard kan worden tenzij $h \to 0$. Zelfs indien Runge-Kutta van orde vier gebruikt wordt kan het introduceren van een stramme differentiaalvergelijking (bv. een stijve veer) tot stabiliteitsproblemen leiden. Bij expliciete integratie wordt er gebruik gemaakt van de snelheid en versnelling op het huidige tijdstip t om de volgende positie en snelheid te bepalen. Impliciete integratie zal dit anders aanpakken, deze integratietechnieken zijn onvoorwaardelijk stabiel. Ze geven een stabiele oplossing voor eender welke tijdstap, maar naar gelang de tijdstap groter wordt, zal de oplossing minder accuraat zijn. Aan de hand van de Backward Euler methode, de simpelste impliciete methode, zal duidelijk worden wat de eigenschappen zijn van impliciete integratie.

Na het introduceren van de Backward Euler, of ook Impliciete Euler genoemd, zullen er enkele benaderingen die speciaal ontwikkeld zijn voor cloth simulation besproken worden.

4.4.1 Backward Euler methode

De forward Euler methode hebben we behandeld bij de expliciete integratietechnieken, zie sectie 4.3.1. Daarbij werd de snelheid en versnelling op tijdstip t gebruikt, zie vergelijking 4.3. Zoals de naam bij Backward Euler impliceert zal hier een stap terug in de tijd gedaan worden[Soe02]:

$$\begin{aligned}
\mathbf{x}(t) &= \mathbf{x}(t+h) - h\dot{\mathbf{x}}(t+h) && (4.7)\\
&\Updownarrow \text{ we zoeken } \mathbf{x}(t+h)\\
\mathbf{x}(t+h) &= \mathbf{x}(t) + h\dot{\mathbf{x}}(t+h)\\
&\Updownarrow \mathbf{x}(t+h) \text{ wordt } \mathbf{X}_{new} \text{ en } \mathbf{x}(t) \text{ wordt } \mathbf{X}_{old}\\
\mathbf{X}_{new} &= \mathbf{X}_{old} + h\dot{\mathbf{X}}_{new}\\
&\Updownarrow \dot{\mathbf{X}}_{new} \text{ is ook } f(\mathbf{X}_{new})\\
\mathbf{X}_{new} &= \mathbf{X}_{old} + hf(\mathbf{X}_{new}) && (4.8)
\end{aligned}$$

Er wordt dus de afgeleide status berekend in het punt waar we naartoe willen en niet in het punt waar we van komen. Dit kan gezien worden als het probleem omdraaien [Bar01]. Stel dat we in $\mathbf{x}(t+h)$ zitten en we nemen een stap $-h\dot{\mathbf{x}}(t+h)$, dan komen we terecht in $\mathbf{x}(t)$, dit is ook duidelijk te zien in vergelijking 4.7. Om een overvloed aan haakjes te vermijden en dus tot een duidelijkere notatie te komen wordt vanaf hier gewerkt met de notatie in vergelijking 4.8.

Doordat we $f(\mathbf{X}_{new})$ niet zomaar kunnen berekenen en dit omdat we $\mathbf{X}_{new}$ logischerwijze niet kennen, zullen we f benaderen door een veelterm bekomen van een Taylor reeks. Eerst definiëren we $\triangle\mathbf{X}$ als $\mathbf{X}_{new} - \mathbf{X}_{old}$ waardoor we vergelijking 4.8 kunnen schrijven als:

$$\begin{aligned} \mathbf{X}_{old} + \triangle\mathbf{x} &= \mathbf{X}_{old} + hf(\mathbf{X}_{old} + \triangle\mathbf{X}) \\ &\Updownarrow \\ \triangle\mathbf{X} &= hf(\mathbf{X}_{old} + \triangle\mathbf{X}) \end{aligned}$$

Nu benaderen we $f(\mathbf{X}_{old} + \triangle\mathbf{X})$ als $f(\mathbf{X}_{old}) + \triangle\mathbf{X}f'(\mathbf{X}_{old})$. Merk op dat $f(\mathbf{X}_{old})$ een vector is en zijn afgeleide $f'(\mathbf{X}_{old})$ dus een matrix vormt. De bekomen benadering voor f kan nu gebruikt worden om $\triangle\mathbf{X}$ te benaderen:

$$\begin{aligned} \triangle\mathbf{X} &= h(f(\mathbf{X}_{old}) + \triangle\mathbf{X}f'(\mathbf{X}_{old})) \\ &\Updownarrow \quad \triangle\mathbf{X}f'(\mathbf{X}_{old}) \text{ naar links brengen} \\ \triangle\mathbf{X} - h\triangle\mathbf{X}f'(\mathbf{X}_{old}) &= hf(\mathbf{X}_{old}) \\ &\Updownarrow \quad \text{delen door } h \text{ en } \mathbf{I} \text{ is de eenheidsmatrix} \\ \left(\frac{1}{h}\mathbf{I} - f'(\mathbf{X}_{old}))\right)\triangle\mathbf{X} &= f(\mathbf{X}_{old}) \\ &\Updownarrow \quad \text{oplossen naar } \triangle\mathbf{X} \\ \triangle\mathbf{X} &= \left(\frac{1}{h}\mathbf{I} - f'(\mathbf{X}_{old})\right)^{-1} f(\mathbf{X}_{old}) \end{aligned} \tag{4.9}$$

Het berekenen van $\mathbf{X}_{new} = \mathbf{X}_{old} + \triangle\mathbf{X}$ is duidelijk meer werk dan de expliciete methodes omdat er een lineair systeem moet opgelost worden in elke simulatiestap. Dit lijkt een sterk negatief punt, maar voor veel soorten problemen, zoals mass spring systemen, zal de matrix f' *sparse* zijn. Dit betekent dat de matrix heel veel nullen bevat. Door gebruik te maken van de juiste technieken kan het oplossen dan in lineaire tijd, proportioneel tot de grote van $\mathbf{X}$ [Bar01]. Later in deze sectie worden de populairste technieken opgesomd.

Deze methode is niet meer zo simpel en voor de handliggend om te implementeren als de forward Euler methode. Hierdoor kan een uitgebreid voorbeeld, om het grote verschil tussen forward en backward Euler duidelijk te maken, geen kwaad. We zullen het voorbeeld besproken in [Bar01] hier overnemen: een stramme differentiaalvergelijking wordt gegeven en opgelost met beide methodes.

Oplossen van een stramme differentiaalvergelijking

Stel dat we een particle hebben met een positie $(x(t), y(t))$ waarbij we de y-coördinaat altijd op nul willen hebben. Op de x-coördinaat is er geen restrictie en de x-coördinaat kan dus arbitrair over de x-as bewegen. Om deze omstandigheden na te bootsen nemen we de volgende differentiaalvergelijking en gaan we er ook van uit dat de particle niet start met $y_0 = 0$:

$$\dot{X}(t) = \frac{d}{dt}\begin{pmatrix} x(t) \\ y(t) \end{pmatrix} = \begin{pmatrix} -x(t) \\ -ky(t) \end{pmatrix} \tag{4.10}$$

Meer precies zal de particle sterk aangetrokken worden tot de y-as of de lijn $y = 0$, met k een grote positieve constante en minder hard naar de x-as of $x = 0$. Indien we de ODE ver genoeg in de tijd vooruit oplossen zal de positie van de particle in $(0, 0)$ terecht moeten komen en eens op die positie daar ook blijven. Laten we nu als integratietechniek eerst forward Euler nemen en daarna Backward Euler.

Met een tijdstap h krijgen we:

$$\begin{aligned}
\mathbf{X}_{new} &= \mathbf{X}_0 + h\dot{\mathbf{X}}(t_0) \\
&= \begin{pmatrix} x_0 \\ y_0 \end{pmatrix} + h \begin{pmatrix} -x_0 \\ -ky_0 \end{pmatrix} \\
&= \begin{pmatrix} x_0 - hx_0 \\ y_0 - hky_0 \end{pmatrix} \\
&= \begin{pmatrix} (1-h)x_0 \\ (1-hk)y_0 \end{pmatrix}
\end{aligned}$$

Als we kijken naar de y-coördinaat zien we dat $|1 - hk|$ zal bepalen of de y-coördinaat naar gewenste positie, $y = 0$ zal convergeren. Indien $|1 - hk| > 1$ zal de nieuwe y-coördinaat een absolute waarde hebben die groter is dan $|y_0|$. Dit betekent dus dat de forward Euler methode niet zal convergeren naar het antwoord, maar divergeren en dus onstabiel is als $|1 - hk| > 1$. Als we willen dat de simulatie zal convergeren moet $1 - hk > -1$ of $hk < 2$ wat als gevolg heeft dat $h < \frac{2}{k}$ moet zijn. Als k een groot getal is zullen er dus kleine stapjes genomen moeten worden.

Je kan ook een analogie merken met de stijve veren waarbij de stijfheidsconstante k_s een grote waarde heeft. Het nemen van deze kleine stappen zorgt ervoor dat de simulatie heel traag zal gaan. Omdat het nemen van kleine stappen niet gewenst is en het soms ook niet mogelijk is om de stramme differentiaalvergelijking zo te veranderen dat ze niet meer stram is, dient er naar impliciete integratie gegrepen te worden. We zullen nu hetzelfde voorbeeld integreren aan de hand van backward Euler wat ook meteen de werking van de techniek iets duidelijker zal maken.

We gaan nog steeds uit van de ODE in vergelijking 4.10. De eerste stap in het oplossen van een backward Euler stap is het berekenen van $f'(\mathbf{X}(t))$ (zie vergelijking 4.9):

$$f'(\mathbf{X}(t)) = \frac{\partial}{\partial \mathbf{X}} f(\mathbf{X}(t)) = \begin{pmatrix} -1 & 0 \\ 0 & -k \end{pmatrix}$$

Met $f'(\mathbf{X}(t))$ kunnen we de matrix $\frac{1}{h}\mathbf{I} - f'(\mathbf{X}(t))$ berekenen:

$$\begin{pmatrix} \frac{1}{h} + 1 & 0 \\ 0 & \frac{1}{h} + k \end{pmatrix} = \begin{pmatrix} \frac{1+h}{h} & 0 \\ 0 & \frac{1+kh}{h} \end{pmatrix}$$

Nu rest enkel nog de inverse van bovenstaande matrix te berekenen en deze te vermenigvuldigen met $f(\mathbf{X}_0)$:

$$\begin{aligned}
\triangle\mathbf{X} &= \left(\frac{1}{h}\mathbf{I} - f'(\mathbf{X}_0)\right)^{-1} f(\mathbf{X}_0) \\
&= \begin{pmatrix} \frac{1+h}{h} & 0 \\ 0 & \frac{1+kh}{h} \end{pmatrix}^{-1} \begin{pmatrix} -x_0 \\ -ky_0 \end{pmatrix} \\
&= \begin{pmatrix} \frac{h}{1+h} & 0 \\ 0 & \frac{h}{1+kh} \end{pmatrix} \begin{pmatrix} -x_0 \\ -ky_0 \end{pmatrix} \\
&= -\begin{pmatrix} \frac{h}{1+h}x_0 \\ \frac{h}{1+kh}ky_0 \end{pmatrix}
\end{aligned}$$

Uit de bekomen vector kan afgeleid worden dat de grootte van de tijdstap h niet meer van belang is. Als we deze naar oneindig laten gaan zullen we in één enkele stap de gewenste positie $(0, 0)$ bekomen:

$$\lim_{h\to\infty} \triangle\mathbf{X} = \lim_{h\to\infty} -\begin{pmatrix} \frac{h}{1+h}x_0 \\ \frac{h}{1+kh}ky_0 \end{pmatrix} = -\begin{pmatrix} x_0 \\ \frac{1}{k}ky_0 \end{pmatrix} = -\begin{pmatrix} x_0 \\ y_0 \end{pmatrix}$$

Voor de meeste stijve ODE's zal het niet mogelijk zijn een arbitraire stapgrootte te nemen, maar het zal mogelijk worden om veel grotere te stappen te nemen ten opzichte van een expliciete methode. Door deze grotere stappen is er een trade-off tussen de rekenkracht van het oplossen van een lineair systeem en de mogelijkheid tot het nemen van grotere stappen.

Het nemen van grotere stappen heeft natuurlijk ook nadelen: de nauwkeurigheid hangt nog steeds af van de tijdstap en hoe groter deze wordt, hoe minder accuraat de oplossing zal zijn[Hau03]. Voor cloth simulation specifiek kan het zijn dat een cloth niet accuraat gesimuleerd moet worden maar slechts visueel aantrekkelijk moet zijn of dat we een cloth willen draperen over een pop of op een tafel laten vallen. Volino et al.[VMT01] kwamen tot de conclusie dat impliciete integratietechnieken voor draperen zeer geschikt zijn.

Er moet nog één stap in deze bespreking genomen worden. De algemene techniek voor backward Euler toe te passen is nu reeds uitgebreid besproken met behulp van een voorbeeld. Zoals eerder gezegd leveren de bewegingswetten van Newton een twee orde ODE op die kan opgelost worden door een extra variabele te introduceren. Dit kan met de backward Euler methode ook toegepast worden, maar geeft een lineair systeem van $2n$ x $2n$ waarbij n de dimensie is van $\mathbf{x}$[Bar01]:

$$\frac{d}{dt}\begin{pmatrix} \mathbf{x}(t) \\ \mathbf{v}(t) \end{pmatrix} = \begin{pmatrix} \mathbf{v}(t) \\ f(\mathbf{x}(t), \mathbf{v}(t)) \end{pmatrix} \tag{4.11}$$

Het is mogelijk om via een simpele transformatie naar een systeem van n x n te gaan dat even sparse is als het $2n$ x $2n$ systeem en dus sneller opgelost kan worden. Om de notatie duidelijker te maken schrijven we $\mathbf{x}_t = \mathbf{x}(t)$ en $\mathbf{v}_t = \mathbf{v}(t)$. Nu moet $\triangle\mathbf{x}$ en $\triangle\mathbf{v}$ gedefinieerd worden zodat we een backward Euler stap kunnen toepassen op vergelijking 4.11: $\triangle\mathbf{x} = \mathbf{x}(t+h) - \mathbf{x}(t)$ en $\triangle\mathbf{v} = \mathbf{v}(t+h) - \mathbf{v}(t)$. De backward Euler stap geeft:

$$\begin{pmatrix} \triangle\mathbf{x} \\ \triangle\mathbf{v} \end{pmatrix} = h \begin{pmatrix} \mathbf{v}_t + \triangle\mathbf{v} \\ f(\mathbf{x}_t + \triangle\mathbf{x}, \mathbf{v}_t + \triangle\mathbf{v}) \end{pmatrix} \tag{4.12}$$

$f(\mathbf{x}_t + \triangle\mathbf{x}, \mathbf{v}_t + \triangle\mathbf{v})$ is een niet-lineaire functie en kan alleen berekend worden via iteraties. Maar het is beter om weer een eerste orde benadering te maken door een Taylor reeks te gebruiken[BW98]:

$$f(\mathbf{x}_t + \triangle\mathbf{x}, \mathbf{v}_t + \triangle\mathbf{v}) = f_t + \frac{\partial f}{\partial \mathbf{x}} \triangle \mathbf{x} + \frac{\partial f}{\partial \mathbf{v}} \triangle \mathbf{v} \tag{4.13}$$

In deze vergelijking, 4.13, worden $\frac{\partial f}{\partial \mathbf{x}}$ en $\frac{\partial f}{\partial \mathbf{v}}$ geëvalueerd voor de toestand $(\mathbf{x}_t, \mathbf{v}_t)$. Deze benadering van f stoppen we nu in vergelijking 4.12:

$$\begin{aligned}
\begin{pmatrix} \triangle\mathbf{x} \\ \triangle\mathbf{v} \end{pmatrix} &= h \begin{pmatrix} \mathbf{v}_t + \triangle\mathbf{v} \\ f_t + \frac{\partial f}{\partial \mathbf{x}} \triangle \mathbf{x} + \frac{\partial f}{\partial \mathbf{v}} \triangle \mathbf{v} \end{pmatrix} \\
&\Downarrow \quad \text{substitueer } \triangle\mathbf{x} = h(\mathbf{v}_t + \triangle\mathbf{v}) \text{ in de onderste rij} \\
\triangle\mathbf{v} &= h\left(f_t + \frac{\partial f}{\partial \mathbf{x}} h(\mathbf{v}_t + \triangle\mathbf{v}) + \frac{\partial f}{\partial \mathbf{v}} \triangle \mathbf{v}\right) \\
&\Downarrow \quad \text{hergroeperen met } \mathbf{I} \text{ de eenheidsmatrix} \\
\left(\mathbf{I} - h\frac{\partial f}{\partial \mathbf{v}} - h^2 \frac{\partial f}{\partial \mathbf{x}}\right) \triangle \mathbf{v} &= h\left(f_t + \frac{\partial f}{\partial \mathbf{x}} h\mathbf{v}_t\right)
\end{aligned}$$

Met $\triangle\mathbf{v}$ gekend kunnen we $\triangle\mathbf{x} = h(\mathbf{v}_t + \triangle\mathbf{v})$ berekenen. De bekomen formule kan nog lichtjes aangepast worden[1]. De f die we hier benaderen is de versnelling van de particle op een gegeven tijdstip. De versnelling is gedefinieerd in sectie 2.1 als de krachtvector gedeeld door de massa. Vergelijking 4.11 wordt dan:

$$\frac{d}{dt} \begin{pmatrix} \mathbf{x}(t) \\ \mathbf{v}(t) \end{pmatrix} = \begin{pmatrix} \mathbf{v}(t) \\ M^{-1} f(\mathbf{x}(t), \mathbf{v}(t)) \end{pmatrix}$$

Waarbij M de massa matrix van het systeem voorstelt. Dit is een diagonaalmatrix die als volgt ingevuld wordt: $M \in \mathbb{R}^{3nx3n} : diag(M) = (m_1, m_1, m_1, m_2, m_2, m_2, \ldots, m_n, m_n, m_n)$ waarbij n het aantal particles is en voor een particle i we de massa schrijven als m_i. De volledige afleiding, rekening houdend met M als massa matrix, is te vinden in [BW98]. We geven hier het eindresultaat:

$$\left(\mathbf{I} - hM^{-1}\frac{\partial f}{\partial \mathbf{v}} - h^2 M^{-1}\frac{\partial f}{\partial \mathbf{x}}\right) \triangle \mathbf{v} = hM^{-1}\left(f_t + \frac{\partial f}{\partial \mathbf{x}} h\mathbf{v}_t\right) \tag{4.14}$$

Om een backward Euler stap te berekenen moeten er een aantal dingen berekend worden, $\frac{\partial f}{\partial \mathbf{x}}$ en $\frac{\partial f}{\partial \mathbf{v}}$ en f_t. Van $\frac{\partial f}{\partial \mathbf{x}}$ en $\frac{\partial f}{\partial \mathbf{v}}$ weten we niet meteen hoe we deze moeten berekenen, van f_t wel. f_t is de vector die alle krachtvectoren bevat van de particles voor tijdstip t.

[1] De literatuur gebruikt meestal de volgende notatie en de benaderingen van deze methode die nog besproken zullen worden dus ook.

$\frac{\partial f}{\partial \mathbf{x}}$ en $\frac{\partial f}{\partial \mathbf{v}}$ zijn matrices, jacobianen om meer precies te zijn, met grootte $3n$ x $3n$. Deze matrices worden opgevuld met 3 x 3 matrices die op volgende manier worden gedefinieerd[Kau03]:

$$\begin{aligned}(\frac{\partial f}{\partial \mathbf{x}})_{ij} &= \frac{\partial f_i}{\partial \mathbf{x}_j} \\ (\frac{\partial f}{\partial \mathbf{v}})_{ij} &= \frac{\partial f_i}{\partial \mathbf{v}_j}\end{aligned}$$

Deze afgeleiden zijn niet nul indien particle i en particle j invloed op elkaar hebben en in dit geval zijn ze onderling verbonden door een veer. Merk op dat dit er voor zorgt dat het uiteindelijk bekomen systeem sparse is zoals eerder werd verwacht. Om deze matrices te berekenen moeten we de jacobiaan van de veren berekenen en dit voor positie $J\mathbf{x}_{ij}$ en voor snelheid $J\mathbf{v}_{ij}$. Dit is de aanpak die Choi et al.[CK02] gebruiken en deze wordt verduidelijkt in [Kau03] waarbij ook aandacht wordt besteed aan de implementatie van deze methode. Deze aanpak is net dezelfde als in [BW98] alleen wordt daar met *constraint forces* gewerkt wat minder intuïtief is. Constraint forces zullen nog besproken worden in het hoofdstuk over beperkingen, zie 5.2.

$$\begin{aligned}J\mathbf{x}_{ij} &= k_s\frac{\mathbf{x}_{ij}\otimes\mathbf{x}_{ij}}{\mathbf{x}_{ij}\cdot\mathbf{x}_{ij}} + k_s(\mathbf{I} - \frac{\mathbf{x}_{ij}\otimes\mathbf{x}_{ij}}{\mathbf{x}_{ij}\cdot\mathbf{x}_{ij}})(1-\frac{r_{ij}}{|\mathbf{x}_{ij}|}) \\ J\mathbf{v}_{ij} &= \mathbf{I} * k_d\end{aligned} \tag{4.15}$$

In vergelijkingen 4.15 zijn k_s, k_d, r_{ij} waardes van een lineair gedempte veer (zie sectie 2.2.2) en $\mathbf{x}_{ij} = \mathbf{x}_i - \mathbf{x}_j$. Hier stelt $\cdot$ het dotproduct voor en $\otimes$ het outer product. Deze jacobianen worden dus elke simulatiestap berekend en daarna gebruikt om $\frac{\partial f}{\partial \mathbf{x}}$ en $\frac{\partial f}{\partial \mathbf{v}}$ te bekomen. Ze worden gebruikt als volgt: $J\mathbf{x}_{ij}$ wordt opgeteld bij $\frac{\partial f_i}{\partial \mathbf{x}_j}$ en $\frac{\partial f_j}{\partial \mathbf{x}_i}$ en afgetrokken van $\frac{\partial f_i}{\partial \mathbf{x}_i}$ en $\frac{\partial f_j}{\partial \mathbf{x}_j}$. Voor $J\mathbf{v}_{ij}$ is dit analoog.

Alle componenten voor $\triangle\mathbf{v}$ te berekenen zijn nu gekend. We kunnen vergelijking 4.14 zien als een lineair systeem van vergelijkingen $Ax = b$ waarbij A de matrix is in de vergelijking, b de vector en x een vector die we zoeken namelijk $\triangle\mathbf{v}$. Dit kan best gezien worden als het vinden van een vector x die, vermenigvuldigd met A, b als resultaat heeft. Voor het oplossen van zulke lineaire systemen zijn er een aantal mogelijkheden. We gaan niet in detail uitleggen hoe deze opgelost kunnen worden, maar zullen populaire technieken opsommen.

- In [BW98] wordt de *Modified Conjugate Gradient Methode* gebruikt. Maar om deze te kunnen gebruiken moet de matrix van het systeem symmetrisch en positief (semi)-definiet zijn. Wat betekent dat de vergelijking 4.14 vermenigvuldigd moet worden met M. Alle details zijn te vinden in [BW98] en voor een uitleg over de conjugate gradient methode geeft Shewchuk in [She94] een duidelijke en lange uiteenzetting. Vrij recent hebben Ascher et al. [AB03] de gebruikte methode van Baraff et al. aangepast zodat deze sneller tot een oplossing komt van het lineaire systeem.

- Volino et al.[VT00] bespreken een manier om de backward Euler methode te implementeren zoals in [BW98], maar zonder expliciet de sparse matrix op te bouwen. In plaats daarvan worden de krachten zo gemodelleerd dat deze worden ingevoegd in het Conjugate Gradient algoritme. Door het meteen invoegen van deze krachten wordt de berekening van één simulatiestap sneller. Alle details zijn uitgebreid terug te vinden in [VT00].
- Hilde et al.[HMC01] gebruiken de *Broyden* methode om een lineair systeem op te lossen. Deze methode heeft als voordeel dat de matrix niet verplicht symmetrisch en positief-definiet moet zijn. Hierdoor kunnen ook andere mechanische simulaties die de bewegingswetten van Newton gebruiken opgelost worden. Het systeem dat ontwikkeld werd in hun paper dient dan ook voor allerlei soorten simulaties en is dus niet toegespitst op cloth simulation.

Het werk van Baraff et al. [BW98] zorgde voor een herintrede van impliciete integratie in de cloth simulation met het logische gevolg dat andere onderzoekers deze methode nog verder gingen versimpelen of optimaliseren. Omdat de opmars van impliciete methodes niet genegeerd kan worden bespreken we enkele benaderingen van deze methode. Ook de intrede van hogere orde impliciete methodes begint langzaamaan. Hogere orde methodes zorgen voor meer accuuraatheid in de oplossing zoals dit het geval was bij expliciete technieken. We zullen niet in detail deze methoden bespreken, maar slechts enkele bestaande technieken aanhalen en ernaar verwijzen.

4.4.2 Benaderingen van de backward Euler methode

Er zijn een aantal benaderingen van de backward Euler methode. Twee hiervan bouwen op elkaar verder en deze zullen we bespreken. De eerste benadering werd gerealiseerd door Desbrun et al.[DSB99] (later nog uitgebreid in [MDDB01]) waarna Kang et al.[KCCP00, KCCL01] deze benadering gebruikten om zelf tot een nog minder fysisch correcte benadering te komen. We zullen eerst de benadering van Desbrun et al. bespreken om dan deze van Kang et al. uit te leggen.

Er zijn te veel andere benaderingen om ze allemaal te kunnen bespreken en we houden het dus bij deze twee, omdat deze het bekendst zijn.

Benadering van Desbrun et al.

Deze benadering maakt gebruik van een op voorhand berekende krachtfiltermatrix W die in elke simulatiestap gebruikt wordt. Hierdoor dient er ook nog een correctiekracht toegevoegd te worden zodat er rekening gehouden wordt met behoud van het hoekmoment. Het is belangrijk dat in een simulatie zowel het lineaire moment als het hoekmoment bewaard blijft (het lineaire moment wordt behouden zonder een extra correctiekracht). We zullen enkel de afleiding geven tot deze krachtfiltermatrix zonder in diep detail uit te leggen waarom deze correct is en waarom er nood is aan een correctiekracht. De details zijn terug te vinden in [DSB99].

Als we uitgaan van vergelijking 4.11 en deze oplossen naar $\triangle\mathbf{v}$, krijgen we:

$$\triangle \mathbf{v} = \frac{h}{m} W \left(f_t + h \frac{\partial f}{\partial \mathbf{x}} \mathbf{v}_t \right) \qquad W = \left(\mathbf{I} - \frac{h^2}{m} \frac{\partial f}{\partial \mathbf{x}} - \frac{h}{m} \frac{\partial f}{\partial \mathbf{v}} \right)^{-1} \tag{4.16}$$

Merk op dat de massa matrix vervangen is door een constante massa m. Er wordt dan ook van uitgegaan dat alle particles een zelfde massa hebben. Als volgende stap wordt $\frac{\partial f}{\partial \mathbf{x}}$ benaderd door een constante matrix H en zal de term $\frac{\partial f}{\partial \mathbf{v}}$ genegeerd worden. Vergelijking 4.16 wordt dan:

$$\triangle \mathbf{v} = \frac{h}{m} W \left(f_t + h H \mathbf{v}_t \right) \qquad W = \left(\mathbf{I} - \frac{h^2}{m} H \right)^{-1}$$

Met als H een n x n matrix die $\frac{\partial f}{\partial \mathbf{x}}$ benadert waarbij k_{ij} de stijfheidsconstante is van de veer tussen particle i en j. Indien er geen veer is tussen i en j zal de constante nul zijn:

$$\begin{cases} H_{ij} = k_{ij} & \text{als } i \neq j \\ H_{ii} = -\sum_{j \neq i} k_{ij} \end{cases}$$

Al deze benaderingen brengen een aantal nadelen met zich mee. Een duidelijk nadeel aan het op voorhand berekenen van W is dat de massa en de veerconstanten niet kunnen veranderen tijdens de simulatie tenzij W opnieuw berekend wordt. Maar nog belangrijker, de tijdstap h kan ook niet veranderen en de kans dat deze gedurende een simulatie verandert is veel groter. Een ander nadeel is dat de tot nu toe besproken benadering niet goed overweg kan met andere externe krachten zoals wind. Een simpele oplossing voor dit probleem bestaat eruit om deze externe krachten expliciet te integreren na de impliciete stap[MDDB01]. Doordat de externe krachten normaal gezien kleiner zijn dan de interne cloth krachten levert dit geen instabiliteiten op.

Er rest nog de correctiekracht om het verlies in hoekmoment te compenseren. Wanneer de gefilterde interne krachten $F_i^{filtered}$ berekend zijn (voor de berekening zie listing 1), kan de globale torsie $\delta \mathbf{T}$ berekend worden:

$$\delta \mathbf{T} = \sum_{i=1}^{n} (\mathbf{x}_G - \mathbf{x}_i) \wedge F_i^{filtered}$$

Hier is $\wedge$ het vectorproduct en $\mathbf{x}_G$ het zwaartepunt, maar doordat de som van alle interne krachten gelijk moet zijn aan nul (wet van actie en reactie) kan ook een meer directe expressie gebruikt worden:

$$\delta \mathbf{T} = \sum_{i=1}^{n} \mathbf{x}_i \wedge F_i^{filtered}$$

Aangezien de torsie nul moet zijn, kunnen we een correctiekracht toevoegen aan elke particle om de verandering in hoeksnelheid te benaderen:

$$F_i^{correct} = h(\mathbf{x}_G - \mathbf{x}_i) \wedge \delta \mathbf{T}$$

Een laatste stap die in deze benadering wordt toegepast is Inverse Dynamica, maar dit zullen we hier niet behandelen, maar in latere sectie van dit hoofdstuk, namelijk 4.7.

Listing 1: Benadering van desbrun

```
1  Bereken op voorhand: W = (I − (h²/m)H)^−1
2
3  Voor elke tijdstap h:
4      Reset het zwaartepunt:
5          x_G = 0
6
7      Bereken de interne krachten F_i (Veren en
8      artificiële viscositeit):
9      Voor elke particle i:
10         F_i = 0
11         x_G += x_i
12             Voor elke particle j verbonden met i door een veer:
13                 F_i += k_s(|l_ij| − r_ij) x_ij/|x_ij|
14                 F_i += k_d h(v_j − v_i)
15
16     Bereken het zwaartepunt en reset de torsie:
17         x_G /= n
18         δT = 0
19
20     Benader integratie gebruik makende van W:
21     Voor elke particle i:
22         F_i^filtered = Σ_j F_j W_ij
23         δT += F_i^filtered ∧ x_i
24         v_i(t + h) = v_i(t) + ( F_i^filtered + F_i^grav ) h/m
25         x_i(t + h) = x_i(t) + h v_i(t + h)
26
27     Correctie van het hoekmoment:
28     Voor elke particle i:
29         F_i^correct = (x_G − x_i) ∧ δT
30         x_i(t + h) += (h²/m) F_i^correct
31
32     Voeg externe krachten zoals wind toe:
33         x_i(t + h) += (h²/m) F_i^wind
34
35     Voer n iteraties uit van Inverse Dynamica, zie sectie 4.7.
```

Deze benadering van de backward Euler bestaat uit nogal veel stappen en is best samengevat in pseudocode, zie listing 1. Deze pseudocode komt rechtstreeks van [MDDB01], aangepast naar de notatie hier.

Er rest nog één nadeel in deze benadering, de tijdscomplexiteit. Het berekenen van de gefilterde krachten groeit lineair met het aantal particles en dit voor alle particles (zie regel 22 van listing 1). Daardoor is de tijdscomplexiteit $O(n^2)$. Door dit nadeel en de eerder genoemde nadelen waarbij vooral het veranderen van de grootte van de tijdstap niet mogelijk is, hebben Kang et al. voor een andere benadering gekozen.

Benadering van Kang et al.

Bouwend op het werk van Desbrun et al.[DSB99], wordt er in deze benadering geprobeerd om de problemen die er waren in het werk van Desbrun et al. op te lossen. Er wordt geen gebruik gemaakt van een krachtfiltermatrix die op voorhand berekend wordt. In plaats hiervan wordt $\triangle\mathbf{v}$ benaderd door een snelle en stabiele berekening zodat de tijdscomplexiteit lineair is. We zullen de afleiding van de benadering bespreken zonder te diep in detail te gaan.

Zoals eerder gezegd wordt er verder gebouwd op de benadering van Desbrun et al., maar zonder de krachtfiltermatrix. Voor alle duidelijkheid herhalen we deze vergelijking:

$$\left(\mathbf{I} - \frac{h^2}{m}H\right) \triangle \mathbf{v} = \frac{h}{m}\left(f_t + hH\mathbf{v}_t\right) \tag{4.17}$$

In deze vergelijking 4.17 stelt $hH\mathbf{v}_t$ extra krachten voor zoals vermeld in [DSB99]. Deze extra krachten zijn viscositeitskrachten en kunnen berekend worden door:

$$(hH\mathbf{v}_t)_i = h \sum_{(i,j)\in\mathbf{E}} k_{ij}(\mathbf{v}_j(t) - \mathbf{v}_i(t)) \tag{4.18}$$

Hierbij stelt $\mathbf{E}$ de verzameling van veren voor. Gebruik makende van vergelijking 4.18 kunnen we de krachten op particle i berekenen op tijdstip t zodat we de totale kracht $\tilde{f}_i(t)$ op particle i bekomen. Vergelijking 4.17 wordt dan:

$$\left(\mathbf{I} - \frac{h^2}{m}H\right) \triangle \mathbf{v} = \frac{h}{m}\tilde{f}_t \tag{4.19}$$

De aanpassing van de snelheid van particle i, $\triangle\mathbf{v}_i$, kan berekend worden door enkel rekening te houden met de particles die met i verbonden zijn. Dit kan doordat H_{ij} nul is wanneer particle i en j niet met elkaar verbonden zijn door een veer. Hierdoor wordt vergelijking 4.19 voor particle i:

$$\left(1 - \frac{h^2}{m_i}H_{ii}\right) \triangle \mathbf{v}_i - \frac{h^2}{m_i} \sum_{(i,j)\in\mathbf{E}} (H_{ij} \triangle \mathbf{v}_j) = \frac{h}{m_i}\tilde{f}_i(t) \tag{4.20}$$

Om tot een simpelere notatie van vergelijking 4.20 te komen, wordt de notatie van de matrix H aangepast. Stel dat k de uniforme stijfheidsconstante is voor alle veren en dat n_i het aantal buren voorstelt dat verbonden is met particle i, dan kan H berekend worden door $H_{ij} = k$ en $H_{ii} = -kn_i$. Gebruik makende van deze H wordt vergelijking 4.20:

$$\begin{aligned} \frac{m_i + h^2 k n_i}{m_i} \triangle \mathbf{v}_i &= \frac{h}{m_i}\tilde{f}_i(t) + \frac{h^2 k \sum_{(i,j)\in\mathbf{E}}(\triangle \mathbf{v}_j)}{m_i} \\ &\Downarrow \text{ herschikken en breuk vermeningvuldigen met } m_i \\ \triangle \mathbf{v}_i &= \frac{h\tilde{f}_i(t) + kh^2 \sum_{(i,j)\in\mathbf{E}}(\triangle \mathbf{v}_j)}{m_i + kh^2 n_i} \end{aligned} \tag{4.21}$$

Met vergelijking 4.21 kunnen we nu $\triangle \mathbf{v}_i$ berekenen op één probleem na. In die vergelijking komen er meerdere $\triangle \mathbf{v}_j$ voor die net met deze vergelijking gaan berekend worden. Deze worden benaderd door vergelijking 4.21 te gebruiken, maar het gedeelte met de $\triangle \mathbf{v}$ te laten vallen in de breuk:

$$\triangle \mathbf{v}_j \approx \frac{h\tilde{f}_j(t)}{m_j + kh^2 n_j}$$

Deze benadering van $\triangle \mathbf{v}_j$ ingevuld in vergelijking 4.21 geeft de vergelijking waarmee we de aanpassing in snelheid kunnen berekenen:

$$\triangle \mathbf{v}_i = \frac{h\tilde{f}_i(t) + kh^2 \sum_{(i,j)\in\mathbf{E}} \left(\frac{h\tilde{f}_j(t)}{m_j + kh^2 n_j} \right)}{m_i + kh^2 n_i}$$

Het bekomen resultaat heeft geen last van parameters die vastliggen tijdens de simulatie en de tijdscomplexiteit is ook lineair, zodat alle problemen die in de benadering van Desbrun et al. aanwezig waren opgelost zijn. Net als bij Desbrun et al. wordt er ook nog een Inverse Dynamica stap toegevoegd, meer hierover in sectie 4.7.

Doordat de hier voorgestelde benaderde methode de volgende positie van een particle bepaalt door rekening te houden met de posities van het aantal verbonden buren, worden er ongewenste resultaten gegenereerd bij het gebruik van een groot aantal particles. Om dit gebrek te overkomen, wordt er een kreuk-generatie-methode gebaseerd op kubische splines geïntroduceerd. Voor meer details over deze methode verwijzen we naar [KCCL01].

4.4.3 Hogere orde impliciete integratietechnieken

De backward Euler methode kan gezien worden als een impliciete methode van eerste orde. Deze methode heeft als sterk punt de stabiliteit en dus de mogelijkheid om de tijdstap h te vergroten. De tijdstap h wordt meestal verhoogd zodat er gecompenseerd wordt voor de grotere rekenkracht die nodig is voor het oplossen van een lineair systeem. Door het vergroten van de tijdstap h wordt de oplossing minder accuraat. De enige oplossing hiervoor is om methodes van hogere orde te gebruiken die meer accuraat zijn.

Er zijn een aantal mogelijkheden om tot hogere orde methodes te komen en deze worden op exact dezelfde manier gebruikt als eerder beschreven met de backward Euler methode. Daarom sommen we slechts enkele van deze mogelijkheden op:

Impliciete Midpoint: In [VT00] wordt deze methode gebruikt. Het principe is hetzelfde als expliciete midpoint maar dan impliciet.

Meerstapmethodes: Een integratiemethode is meerstaps indien de methode gebruik maakt van vorige stappen. Indien we n-stappen terug gaan spreken we over een n-staps methode. Er zijn twee belangrijke klassen van meerstapsmethodes, de *Adams* methodes en de *BDF* methodes (Backward Differential Formulas) [Hau03]. Choi et al. gebruiken een tweede orde BDF formule (dus 2-staps), wat dus betekent dat er gebruik wordt gemaakt van 2 vorige stappen, n en $n-1$. Voor meer uitleg zie [CK02].

4.5 IMEX Integratie

Uit de naam kan afgeleid worden wat er bij deze integratietechniek gebeurt. Er wordt een combinatie van IMpliciete en EXpliciete integratie toegepast. De ODE, die dient opgelost te worden, wordt opgesplitst in een stijf gedeelte en een niet-stijf gedeelte. Hierdoor wordt alleen meer rekenkracht gebruikt waar die ook effectief nodig is. Het voordeel van deze techniek is dat deze nagenoeg de stabiliteit heeft van een impliciete integratie maar minder rekenkracht vergt. Indien in het niet-stijve deel echter iets stijf terecht komt, loopt het logischerwijze mis. Een oplossing voor dit probleem is vrij recent aan het licht gekomen door gebruik te maken van een *adaptief* IMEX schema [BA04].

We zullen ons beperken tot het bespreken van één IMEX schema geïntroduceerd door Eberhardt et al.[EEH00]. Daarna zullen we een uitbreiding van deze techniek bespreken die adaptief beslist welke krachten stijf zijn en welke niet. Daarbij wordt het cloth ook nog eens opgedeeld in meerdere 'stukken' die apart worden geïntegreerd[BA04].

4.5.1 IMEX Schema van Eberhardt et al.

Voor een IMEX schema moet het systeem dus opgesplitst worden in een stijf en niet-stijf deel; dit wordt gedaan door de ODE op te splitsen:

$$\dot{\mathbf{x}} = f(\mathbf{x}, t) + g(\mathbf{x}, t)$$

Hierbij stelt f het niet-stijve gedeelte voor en g het stijve gedeelte. Deze ODE is nog niet geschikt voor cloth simulation. Zoals in de sectie over ODE's, zie 4.1, gezegd is, dient er nog van één tweede orde naar twee eerste orde ODE's gegaan te worden. We snijden hier een paar bochten af (de volledige uitwerking is te vinden in [EEH00]) en geven meteen de vergelijking waarbij f een forward Euler stap voorstelt en g een backward Euler:

$$\mathbf{x}(t+h) = \mathbf{x}(t) + hf(\mathbf{x}(t), t) + hg(\mathbf{x}(t+h), t+h) \tag{4.22}$$

De stijfheid van de simulatie bij cloth simulation komt van de veren. We gaan even uit van het meest gebruikte model met drie type veren, zie sectie 3.2.1. Bij drie type veren zijn de structurele veren het stijfst, ze hebben de hoogste stijfheidsconstante. Deze zullen dus alleen behandeld worden door het impliciete gedeelte en de afschuif- en buigveren zullen expliciet geïntegreerd worden. De krachtberekening van een lineaire veer, zie vergelijking 2.3, kan

opgesplitst worden in een lineair en niet-lineair gedeelte waarbij k_s de stijfheidsconstante is van de veer tussen particle i en j [DSB99]:

$$\begin{aligned} \text{Lineair gedeelte} \quad & f_{il} = -k_s l_{ij} \\ \text{Niet-lineair gedeelte} \quad & f_{in} = k_s r_{ij} \frac{l_{ij}}{|l_{ij}|} \end{aligned}$$

Het lineair gedeelte wordt in g geïntegreerd omdat dit het stijve gedeelte is van de veer. Als particles zich verder van elkaar verwijderen zal deze waarde sterk stijgen. Het niet-lineaire gedeelte wordt in f geïntegreerd. Merk op dat deze splitsing alleen voor structurele veren wordt gedaan, afschuif- en buigveren worden volledig door f behandeld.

Doordat we gebruik maken van lineair gedempte veren moet er ook nog een dempingsterm k_d, die geldt voor die veer, toegevoegd worden. Deze zal ook lineair zijn en dus ook door g behandeld worden:

$$f_d = k_d(\mathbf{v}_j - \mathbf{v}_i)$$

De krachten die behandeld worden door g zijn dus:

$$f_g = -k_s l_{ij} + k_d(\mathbf{v}_j - \mathbf{v}_i)$$

In analogie met Desbrun et al. [DSB99] wordt er gebruik gemaakt van een stijfheidsmatrix H en analoog hiermee een dempingsmatrix D om de lineaire krachten voor te stellen:

$$f_g = H\mathbf{x} + D\mathbf{v}$$

Met als H:

$$\begin{cases} H_{ij} = k_{s(ij)} & \text{als } i \neq j \\ H_{ii} = -\sum_{j \neq i} k_{s(ij)} \end{cases}$$

en als D:

$$\begin{cases} D_{ij} = k_{d(ij)} & \text{als } i \neq j \\ D_{ii} = -\sum_{j \neq i} k_{d(ij)} \end{cases}$$

De volgende stap is om het $6n$-waardensysteem uit vergelijking 4.22 te herschrijven naar $3n$ (zie de sectie over de toestandsvector 4.2):

$$\mathbf{v}(t+h) = \mathbf{v}(t) + \frac{h}{m} f_a(\mathbf{x}(t), \mathbf{v}(t)) + \frac{h}{m} f_g(\mathbf{x}(t+h), \mathbf{v}(t+h)) \tag{4.23}$$

$$\mathbf{x}(t+h) = \mathbf{x}(t) + h\mathbf{v}(t+h) \tag{4.24}$$

waarbij f_a de functie van alle niet-stijve krachten voorstelt die expliciet opgelost kunnen worden. Dit zijn de afschuif- en buigveren als ook het niet-lineaire gedeelte van de structurele veren f_n. Ook zwaartekracht, wind , ... horen hierbij.

De kracht f_g wordt ingevuld in vergelijking 4.23 en $\mathbf{x}(t+h)$ wordt vervangen door vergelijking 4.24 waarna we de vergelijking oplossen naar $\mathbf{v}(t+h)$ om tot de eindvergelijking te komen:

$$\begin{aligned}
\mathbf{v}(t+h) &= \mathbf{v}(t) + \frac{h}{m} f_a(\mathbf{x}(t), \mathbf{v}(t)) + \frac{h}{m} \left(H\mathbf{x}(t+h) + D\mathbf{v}(t+h)\right) \\
&\Downarrow \quad \mathbf{x}(t+h) \text{ vervangen door 4.24} \\
\mathbf{v}(t+h) &= \mathbf{v}(t) + \frac{h}{m} f_a(\mathbf{x}(t), \mathbf{v}(t)) + \frac{h}{m} \left(H\mathbf{x}(t) + hH\mathbf{v}(t+h) + D\mathbf{v}(t+h)\right) \\
&\Downarrow \quad \text{oplossen naar lineair systeem} \\
\left(\mathbf{I} - \frac{h^2}{m}H - \frac{h}{m}D\right)\mathbf{v}(t+h) &= \mathbf{v}(t) + \frac{h}{m} f_a(\mathbf{x}(t), \mathbf{v}(t)) + \frac{h}{m} H\mathbf{x}(t) \qquad (4.25)
\end{aligned}$$

In bovenstaande afleiding hebben we de tekst gevolgd van Melis [Mel02] vanwege een aantal fouten in de tekst van [EEH00]. Vergelijking 4.25 kan nu gebruikt worden om het systeem op te lossen net als bij de backward Euler methode uit sectie 4.4.1.

4.5.2 Adaptieve IMEX

Eberhardt et al. gingen ervan uit dat alleen structurele veren stijf zijn, maar bij het modelleren van bepaalde soorten cloth kunnen de afschuifveren ook stijf zijn. Een ideale oplossing hiervoor is om tijdens de simulatie het splitsen in IMEX te doen.

Boxerman et al.[BA04] gebruiken de topologie van Choi et al.[CK02], zie sectie 3.2.2 en gebruiken volgende vergelijking als stabiliteitscriterium:

$$\kappa = \frac{h}{m}(k_s h + 2k_d) \leq \frac{1}{2} \qquad (4.26)$$

Gebruik makende van dit stabiliteitscriterium kan op elke tijdstap bepaald worden of een veer afhankelijk van de constanten van de simulatie op dat moment impliciet of expliciet geïntegreerd moet worden. Hierdoor krijgen we dus een IMEX schema dat adaptief kiest wat impliciet of expliciet behandeld wordt. Merk op dat de buigveren altijd expliciet behandeld worden en alleen voor de structurele en afschuifveren het stabiliteitscriterium gebruikt wordt. Boxerman et al. melden ook dat in vergelijking 4.26 vergeleken wordt met $\frac{1}{5}$ in plaats van met $\frac{1}{2}$.

In dit werk, [BA04], wordt nog een interessante nieuwe techniek voorgesteld. Het cloth wordt in stukken verdeeld volgens de beperkingen (zie hoofdstuk 5) die op dat moment actief zijn en rekening houdend met de matrix van het lineaire systeem. Op deze manier kan het oplossen van het systeem sneller gebeuren of door middel van kleinere deelsystemen tot een snellere oplossing komen. Het bespreken van deze techniek ligt niet in het bestek van deze thesis, alle details zijn te vinden in [BA04].

4.6 Verlet Integratie

Verlet integratie is populair in de moleculaire dynamica waar moleculen gemodelleerd worden als puntmassa's die bewegen onder de wetten van Newton. Het spreekt voor zich dat er

een verband is met particles. Het gebruik van Verlet integratie voor cloth simulation werd geïntroduceerd door Jakobsen [Jak01]. Er zijn een aantal mogelijke Verlet schema's. We zullen de meest populaire schema's bespreken: basic Verlet, velocity Verlet and leapfrog Verlet [Mor01].

4.6.1 Basic Verlet Integratie

Dit is het meest simpele en robuuste schema van de drie. Het bestaat uit de som van twee Taylor reeksen (zie vergelijking 4.2) rond de huidige tijdstap, dus op tijdstip $t - h$ en $t + h$. Deze som wordt dan herschreven zodat we het gewenste resultaat, de positie op tijdstip $t+h$, bekomen:

$$\begin{aligned}
\mathbf{x}(t+h) &= \mathbf{x}(t) + \mathbf{v}(t)h + \frac{1}{2}\frac{\mathbf{f}(t)}{m}h^2 + O(h^3) \\
\mathbf{x}(t-h) &= \mathbf{x}(t) - \mathbf{v}(t)h + \frac{1}{2}\frac{\mathbf{f}(t)}{m}h^2 + O(h^3) \\
&\Downarrow \text{ sommeren} \\
\mathbf{x}(t+h) + \mathbf{x}(t-h) &= 2\mathbf{x}(t) + \frac{\mathbf{f}(t)}{m}h^2 + O(h^4) \\
&\Downarrow \text{ herschrijven naar } \mathbf{x}(t+h) \\
\mathbf{x}(t+h) &= 2\mathbf{x}(t) - \mathbf{x}(t-h) + \frac{\mathbf{f}(t)}{m}h^2 + O(h^4)
\end{aligned}$$

De notatie die hier gebruikt wordt, vult meteen de eerste en tweede afgeleide in als de snelheid $\mathbf{v}$ en de versnelling, uitgedrukt in krachten $\frac{\mathbf{f}(t)}{m}$. Wanneer we kijken naar de accuraatheid van deze techniek kunnen we zien dat deze werkt met een fout van $O(h^4)$ wat evenveel is als Runge-Kutta van vierde orde (zie sectie 4.3.3). Maar het voordeel van de techniek hier is dat er veel minder nodige berekeningen zijn; er is slechts één evaluatie nodig van f. Aan de andere kant zijn bij RK4 vier evaluaties nodig. Merk op dat Euler integratie en Basic Verlet beiden één evaluatie van f nodig hebben en dus ongeveer evenveel rekenkracht nodig hebben.

In dit schema worden de posities van de huidige en vorige tijdstap gebruikt om de positie op het volgende tijdstip te bekomen. Hierdoor wordt er geen snelheid gebruikt, maar sommige berekeningen hebben nood aan deze snelheid. Het is mogelijk om een benadering van deze snelheid te berekenen:

$$\mathbf{v}(t) = \frac{\mathbf{x}(t+h) - \mathbf{x}(t-h)}{2h} + O(h^2)$$

Gebruik makende van deze vergelijking om de snelheid te berekenen zien we dat we de snelheid altijd één tijdstap terug kennen. Een mogelijke oplossing voor dit probleem is velocity Verlet integratie. Deze methode wordt verder uitgewerkt in de volgende sectie.

4.6.2 Velocity Verlet Integratie

Het Velocity Verlet schema[Mor01] berekent de snelheden van particles op het nieuwe tijdstip, $t + h$. Dit is een verbetering ten opzichte van het basic Verlet schema, wat als nadeel heeft dat er extra rekenkost is. Laten we de vier stappen van dit schema bespreken:

1. Berekening van de midpoint snelheid:

$$\mathbf{v}(t+\frac{h}{2}) = \mathbf{v}(t) + \frac{1}{2}\frac{\mathbf{f}(t)}{m}h + O(h^3)$$

2. Berekening van de nieuwe positie gebruik makende van de midpoint snelheid:

$$\mathbf{x}(t+h) = \mathbf{x}(t) + \mathbf{v}(t+\frac{h}{2})h + O(h^4)$$

3. Berekening van de nieuwe krachten op de particles gebruik makende van de midpoint snelheden en de nieuwe posities.

4. Berekening van de finale nieuwe snelheid:

$$\mathbf{v}(t+h) = \mathbf{v}(t+\frac{h}{2}) + \frac{1}{2}\frac{\mathbf{f}(t+h)}{m}h + O(h^3)$$

Dit schema geeft de 'beste' oplossing. De snelheid die dikwijls gebruikt wordt om krachten te berekenen is vrij goed benaderd, net als de nieuwe positie die ondanks dezelfde accuraatheid als het basic Verlet schema een meer precieze oplossing zal geven door het gebruik van de midpoint snelheid. Het is wel trager, aangezien er twee keer over alle particles geïtereerd moet worden: één keer voordat de krachten berekend worden en nog één keer erna.

4.6.3 Leapfrog Verlet Integratie

Een oplossing om niet over alle particles twee keer te itereren is het leapfrog Verlet schema. De snelheid wordt berekend zoals bij het velocity Verlet schema op het middelpunt van de tijdstap maar de snelheid wordt niet berekend op tijdstip $t + h$:

1. Berekening van de midpoint snelheid:

$$\mathbf{v}(t+\frac{h}{2}) = \mathbf{v}(t) + \frac{1}{2}\frac{\mathbf{f}(t)}{m}h + O(h^3)$$

2. Berekening van de nieuwe positie gebruik makende van de midpoint snelheid:

$$\mathbf{x}(t+h) = \mathbf{x}(t) + \mathbf{v}(t+\frac{h}{2})h + O(h^4)$$

Dit algoritme blijkt even stabiel te zijn als velocity Verlet [Mor01], maar is dus sneller omdat er slechts één keer over alle particles geïtereerd wordt.

4.7 Inverse Dynamica

In dit hoofdstuk werden er verschillende integratietechnieken besproken. Wat bij de meeste van groot belang is, is de stabiliteit. Tijdens de bespreking van de expliciete Euler wordt het probleem aangehaald dat de simulatie kan 'ontploffen' (sectie 4.3.1). Bij cloth simulation is

dit meestal een gevolg van stijve veren en dan vooral de structurele veren die ook het meest instaan voor het visuele realisme van het cloth. Een mogelijke oplossing voor dit probleem is inverse dynamica.

Het principe van inverse dynamica dient te voorkomen dat het cloth te hard uitrekt. Hiervoor bestaan er twee methodes: ofwel worden de posities van de particles aangepast ofwel passen we de snelheden aan. Provot[Pro95] kwam als eerste op het idee om de posities aan te passen. Een aantal onderzoekers hebben deze techniek gebruikt in hun systemen waaronder Desbrun et al.[DSB99] en Kang et al.[KCCL01]. Deze laatste stelt hierbij zelfs een uitbreiding voor. Vassilev et al.[VSC01] stellen voor om de snelheden van particles aan te passen. Beide technieken worden in het vervolg van deze sectie verder uitgewerkt.

Om de kracht van deze techniek te bewijzen bespreken we ook nog het werk van Fuhrmann et al.[FGL03], waarbij, zonder veren en met het gebruik van inverse dynamica, aan cloth simulation gedaan wordt.

4.7.1 Inverse Dynamica door posities aan te passen

Het doel van een veer is om altijd zijn rustlengte te behouden. Maar veren houden met een vrij lage stijfheidsconstante zelden tot nooit deze rustlengte. Het kan zelfs, dat de lengte op een bepaald tijdstip twee keer de rustlengte is; een realistisch cloth zal echter nooit zo ver uitrekbaar zijn. De oplossing zou zijn om de stijfheidsconstante te verhogen, maar dit heeft als nadeel dat er impliciete integratie gebruikt moet worden en er dus meer rekenkracht nodig is. Door gebruik te maken van inverse dynamica waarbij de posities worden aangepast is het mogelijk om met expliciete Euler integratie zeer stijve veren na te bootsen, ook al worden er geen stijve veren gebruikt.

Stel dat we willen dat het cloth 10% uitrekbaar is en een bepaalde veer twee keer zijn rustlengte is. Dan brengen we de huidige lengte terug tot op 10% langer dan de rustlengte. We sommen de stappen even op, zie ook figuur 4.3:

1. Bereken de uitrekratio s van de veer verbonden tussen particle i en j:

$$l = |\mathbf{x}_i - \mathbf{x}_j| \qquad s = \frac{l}{r_{ij}}$$

2. Als deze uitrekratio s groter is dan de toegelaten maximale uitrekratio s_{max} worden de posities van particle i en j aangepast waarbij de aangepaste lengte l_a gelijk is aan de toegelaten lengte en $\mathbf{r}$ de richting waarin de aanpassing moet gebeuren:

$$l_a = s_{max} r_{ij} \qquad \mathbf{r} = \frac{\mathbf{x}_j - \mathbf{x}_i}{l}$$

3. De volgende stap is het middelpunt $\mathbf{m}$ berekenen tussen particle i en j:

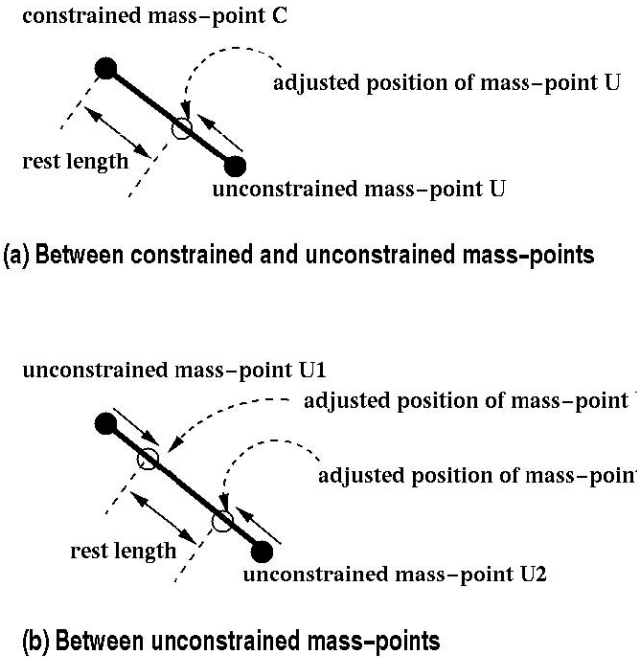
$$\mathbf{m} = \frac{1}{2}(\mathbf{x}_i + \mathbf{x}_j)$$

4. Met dit middelpunt wordt de nieuwe positie berekend:

$$\begin{aligned} \mathbf{x}_i &= \mathbf{m} - \frac{1}{2} l_a \mathbf{r} \\ \mathbf{x}_j &= \mathbf{m} + \frac{1}{2} l_a \mathbf{r} \end{aligned}$$

5. Doordat we de posities van particle i en j verplaatsen moet ook de snelheid aangepast worden:

$$\mathbf{v}_i(t+h) = \frac{\mathbf{x}_i(t+h) + \mathbf{x}_i(t)}{h}$$



Figuur 4.3: Inverse dynamica door aanpassen van de positie[KCCP00]

Er dient nog opgemerkt te worden dat, indien een particle niet vrijuit mag bewegen maar aan bepaalde beperkingen moet voldoen, hier rekening mee moet gehouden worden. In het geval dat een particle enkel mag bewegen volgens een lijn, wordt de verplaatsing hier ook gefilterd volgens de beperking. Voor meer info over beperkingen verwijzen we naar hoofdstuk 5. Indien een particle niet mag bewegen wordt $l_a\mathbf{r}$ afgetrokken of opgeteld met de particle die niet mag bewegen om de nieuwe positie van de vrije particle te bekomen.

De techniek hier besproken om te voorkomen dat veren niet te hard uitrekken wordt toegepast op de structurele en afschuifveren omdat deze het meeste invloed hebben op de vorm van het cloth[Pro95].

Provot koos er voor om geen volgorde te bepalen waarin de veren worden verkort, omdat in zijn geval alleen maar kleine lokale deformaties zouden gebeuren en er geen nood aan was. Het zou zeer ongewenst zijn om een aantal veren plots een stuk langer te maken in plaats

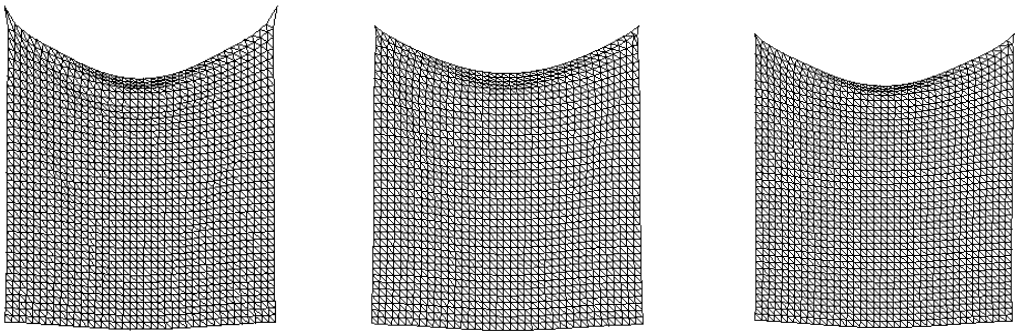
van een stuk korter doordat ze als eerste werden behandeld. Desbrun et al.[DSB99] besloot daarom meerdere iteraties te doen. Kang et al.[KCCP00] hebben een manier besproken om een volgorde te bepalen waarop de veren verkort moeten worden. Deze manier zullen we in de volgende sectie overlopen.

Het bepalen van de volgorde via bucketsort

Het gebruik van een sorteeralgoritme kan nefaste gevolgen hebben in een simulatie die liefst zo snel mogelijk gaat. De meeste sorteeralgoritmen hebben een tijdscomplexiteit van $O(n \log n)$, maar in het slechtste geval kunnen sommige sorteeralgoritmen een worst-case tijdscomplexiteit hebben die veel hoger ligt, zoals bijvoorbeeld quicksort met $O(n^2)$.

Kang et al.[KCCP00] gebruikten bucketsort om de aanpassingen van de lengte te bepalen in $O(n)$. De bepaling van de volgorde gebeurt in twee fases. Eerst wordt elke veer afgegaan om te bepalen wat de langste en kortste lengte op dat tijdstip is. Daarna worden er buckets gecreëerd. Indien er m buckets zijn wordt het interval [min,max] in m subintervallen verdeeld. In de tweede fase wordt elke veer in een bucket gestopt en er wordt geen belang gehecht aan de volgorde in een bucket zelf. Vervolgens wordt, afhankelijk van de buckets, elke veer aangepast.

In het implementatiegedeelte is er een figuur (9.4) te vinden waarbij de werking en invloed van deze techniek op een cloth duidelijk wordt. In figuur 4.4 zijn drie cloths te zien, waarbij er bij het eerste geen inverse dynamica wordt toegepast en bij de andere twee inverse dynamica met verschillende toegestane verlenging. Op beide figuren waarop inverse dynamica is toegepast, kunnen we duidelijk zien dat het cloth minder diep doorzakt, maar bij figuur 4.4(b) is de maximale toegestane verlenging misschien iets te laag. Dit hangt natuurlijk af van wat we willen modelleren, maar het tweede cloth geeft vanboven in het midden een onrealistischere indruk dan de andere twee.



(a) Cloth zonder inverse dynamica

(b) Cloth met inverse dynamica, 15% verlenging toegelaten

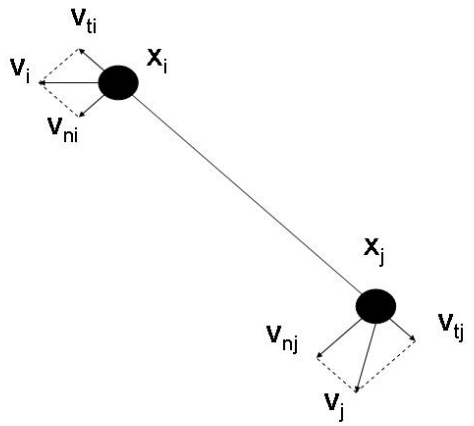
(c) Cloth met inverse dynamica, 30% verlenging toegelaten

Figuur 4.4: Cloths met 40x40 particles waarbij wel of geen inverse dynamica gebruikt werd en een verschillende maximale toegestane verlenging. Op de 750ste iteratie werd er een snapshot genomen.

4.7.2 Inverse Dynamica door snelheden aan te passen

Inverse Dynamica gebruik makende van posities kan niet verzekeren dat elke veer na één iteratie tot de juiste lengte is teruggebracht. Er wordt ook in de eerste plaats toegelaten dat de veren te lang kunnen worden. Het zou beter zijn indien er een techniek wordt gezocht waarbij er geprobeerd wordt om te lange veren te vermijden. Met dit idee zijn Vassilev et al.[VSC01] begonnen.

Het principe is zeer simpel, na elke integratiestap wordt gecontroleerd of een veer zijn uitrekratio overschrijdt net als bij Provot. Indien dit het geval is, worden de snelheden van de verbonden particles aangepast zodat verdere uitrekking niet meer toegestaan is. Stel dat $\mathbf{x}_i$ en $\mathbf{x}_j$ de posities zijn van de particles die worden verbonden door een veer die te lang uitgerokken is. Hierbij zijn $\mathbf{v}_i$ en $\mathbf{v}_j$ de bijbehorende snelheden van de particles, zie figuur 4.5. De snelheden worden op volgende manier aangepast:



Figuur 4.5: Inverse dynamica door aanpassen van snelheden

1. De snelheden worden opgesplitst in twee componenten, $\mathbf{v}_{ti}$ en $\mathbf{v}_{tj}$ langs de lijn tussen $\mathbf{x}_i$ en $\mathbf{x}_j$ en in $\mathbf{v}_{ni}$ en $\mathbf{v}_{nj}$ loodrecht op deze lijn:

$$\begin{aligned} \mathbf{v}_{ti} &= (\mathbf{v}_i \cdot (\mathbf{x}_j - \mathbf{x}_i))\,(\mathbf{x}_j - \mathbf{x}_i) \\ \mathbf{v}_{tj} &= (\mathbf{v}_j \cdot (\mathbf{x}_j - \mathbf{x}_i))\,(\mathbf{x}_j - \mathbf{x}_i) \end{aligned}$$

2. Als we naar één tijdstap kijken zien we dat de snelheidscomponenten $\mathbf{v}_{ti}$ en $\mathbf{v}_{tj}$ instaan voor het langer worden van de veer. $\mathbf{v}_{ni}$ en $\mathbf{v}_{nj}$ kunnen in meerdere tijdstappen voor een verlenging zorgen die niet negeerbaar is (de verlenging in één tijdstap is echter meestal verwaarloosbaar ten opzichte van de verlenging door $\mathbf{v}_{ti}$ en $\mathbf{v}_{tj}$). $\mathbf{v}_{ti}$ en $\mathbf{v}_{tj}$ moeten dus aangepast worden zodat de veer niet meer verder zal verlengen. Het zou mogelijk zijn om een van de snelheden gelijk te zetten aan de andere maar Vassilev et al. verkiezen een andere oplossing. Een goed resultaat wordt bekomen door de tangentiële snelheden gelijk te stellen aan de optelling van $\mathbf{v}_{ti}$ en $\mathbf{v}_{tj}$ gedeeld door twee:

$$\mathbf{v}_{ti} = \mathbf{v}_{tj} = \frac{1}{2}(\mathbf{v}_{ti} + \mathbf{v}_{tj}) \tag{4.27}$$

3. Wanneer er snel bewegende voorwerpen zijn in de simulatie zal vergelijking 4.27 zorgen voor een slecht visueel resultaat. Een oplossing hiervoor is om een directionele vector $\mathbf{v}_{dir}$ in te voeren die als volgt berekend wordt:

   $$\mathbf{v}_{dir} = \mathbf{v}_{other} + \mathbf{v}_{object}$$

   waarbij $\mathbf{v}_{other}$ snelheden zijn die invloed hebben op het cloth zoals zwaartekracht of wind. $\mathbf{v}_{object}$ is de snelheid van het object waarmee het cloth botst. Gebruik makende van deze directionele vector worden de snelheden aangepast. Laat α de hoek zijn tussen $\mathbf{x}_{ij} = \mathbf{x}_j - \mathbf{x}_i$ en $\mathbf{v}_{dir}$, dan kan de cos van α makkelijk berekend worden door het dotproduct van $\mathbf{x}_{ij}$ en $\mathbf{v}_{dir}$. Als de richting van de veer bijna loodrecht is met de directionele vector ($|\cos\alpha| < 0.3$) wordt vergelijking 4.27 gebruikt. In het andere geval, als $|\cos\alpha| > 0.3$, wordt de snelheid van de achterste particle gelijk gezet aan de snelheid van de voorste. Op deze manier kan de achterste particle de scene inhalen. Als $\cos\alpha > 0$ dan is de achterste particle $\mathbf{x}_i$, anders is deze $\mathbf{x}_j$.

Bovenstaand algoritme zal ervoor zorgen dat de veren van het mass spring systeem niet meer verder verlengen als ze over hun uitrekratio zijn gegaan. Merk op dat Vassilev et al. dit algoritme voor alle veren toepassen en niet enkel voor de structurele en afschuifveren.

Net als bij Provot wordt er geen volgorde in het behandelen van de veren gelegd, maar Dochev et al.[DV03] hebben voor deze techniek een manier bedacht om een volgorde te bepalen. Er wordt gebruik gemaakt van het feit of een particle zich in een *static* toestand bevindt, de particle is vast gemaakt aan een object (zie sectie 5.1.1) of zal met een object botsen en daar op blijven liggen. De volgorde wordt slechts één keer berekend en dit in het begin van de simulatie. De techniek gebruikt om de volgorde te bepalen is vrij lang en bevat vrij veel stages, we zullen deze hier dan ook niet bespreken.

Zowel Provot als Vassilev melden dat het bewijzen van de correctheid van deze methodes minder belangrijk is. Aangezien de bekomen resultaten zeer goed zijn en visuele weergave meestal belangrijker is dan fysische correctheid. In de volgende sectie zullen we bespreken hoe Inverse dynamica gebruikt kan worden om een cloth te creëren zonder veren te gebruiken.

4.7.3 Cloth Simulation zonder veren

Gebruik makend van enkel inverse dynamica is het mogelijk om aan cloth simulation te doen. Tijdens het soort inverse dynamica waarbij de posities van de particles worden aangepast, wordt de veer korter gemaakt als deze te lang is. Fuhrmann et al.[FGL03] passen nog een extra stap toe: als de veer te kort is wordt ze langer gemaakt.

Er wordt gebruik gemaakt van een driehoekige mesh (zie sectie 3.1) waarbij op elke edge inverse dynamica wordt toegepast. We kunnen de edges dus bezien als *veren*. Dit omdat Fuhrmann et al. er in geloven dat de uitrekkings- en buigingseigenschappen van een cloth zo goed als altijd genegeerd kunnen worden. Om deze eigenschappen toch te modelleren kan gebruik gemaakt worden van een vierhoekige mesh.

De nieuwe particle-positie wordt berekend door in een eerste stap de richting en lengte van de correctievector **u** te berekenen:

$$\mathbf{u} = \frac{\mathbf{x}_j - \mathbf{x}_i}{|\mathbf{x}_j - \mathbf{x}_i|}(|\mathbf{x}_j - \mathbf{x}_i| - s_{max} r_{ij}) \tag{4.28}$$

Dit is de correctievector voor een edge die te lang is, indien een edge te kort is moeten we s_{max} vervangen door s_{min}. De variabelen in deze vergelijking 4.28 zijn analoog met deze in sectie 4.7.1. Als het toegelaten is dat de massa van particles verschillend is, is het fout om de particles te verplaatsen met $0.5\mathbf{u}$ om de edge te verlengen of verkorten. Indien particles een verschillende massa hebben zal dit er voor zorgen dat het zwaartepunt verandert, waardoor we een gewogen correctievector moeten berekenen. De gewichten zijn:

$$\begin{aligned} m_1 &= \frac{m_i}{m_i + m_j} \\ m_2 &= \frac{m_j}{m_i + m_j} \end{aligned}$$

waardoor de posities aangepast worden als volgt:

$$\begin{aligned} \mathbf{x}_i &+= m_2\mathbf{u} \\ \mathbf{x}_j &-= m_1\mathbf{u} \end{aligned}$$

Stel dat particle j niet toegelaten is om te verplaatsen (zie sectie 5.1.1), dan moet m_1 gelijkgesteld worden aan 1 en m_2 aan 0. Het is met deze techniek nog steeds makkelijk om bepaalde beperkingen op te leggen.

4.8 Conclusie

Integratietechnieken zijn een noodzaak voor cloth simulation. Zonder integratie kan het doek niet bewegen door de tijd en is er dus geen simulatie. In dit hoofdstuk werd een algemene inleiding gegeven over integratietechnieken waarbij eerst werd uitgelegd wat een ODE is en hoe we deze kunnen gebruiken bij een mass spring systeem.

Vervolgens werden een aantal groepen van integratietechnieken besproken. De eerste en simpelste groep zijn de expliciete; deze technieken zijn reeds lang gekend en worden nog steeds veel gebruikt in simulaties. Ze zijn ofwel zeer snel, zoals expliciete Euler, ofwel heel accuraat maar traag, zoals RK4. Deze methoden zijn nooit echt stabiel tenzij met een zeer kleine tijdstap. Om dit probleem te verhelpen werden de impliciete integratietechnieken geïntroduceerd. Deze integratietechnieken nemen als het ware een stap terug in de tijd om tot de oplossing te komen. Het grote nadeel van impliciete integratietechnieken is de rekenkracht. Doordat er een stap terug in tijd genomen wordt, vormt er zich een lineair systeem dat dient opgelost te worden. Dit heeft als gevolg dat de gebruikte tijdstap bij deze integratietechnieken veel groter is om de rekenkracht die nodig is te compenseren. Hierdoor zal de simulatie minder

accuraat zijn. Het is mogelijk om impliciete integratie te gebruiken zonder een lineair systeem te moeten oplossen, hierbij wordt het lineaire systeem benaderd. Twee benaderingen werden in dit hoofdstuk besproken, namelijk de methode van Desbrun en die van Kang.

Een derde groep van integratietechnieken, IMEX, probeert de goede kwaliteiten van expliciete en impliciete integratie te combineren door de ODE op te splitsen in een stijf en niet-stijf deel. Hierbij wordt het stijve gedeelte opgelost door de impliciete integrator en het niet-stijve deel door de expliciete. Deze methodes combineren dus de stabiliteit van impliciete integratie met de snelheid van expliciete integratie.

Als vierde groep werden er nog drie Verlet schema's besproken die uit de moleculaire dynamica komen. Verlet integratie werd door Jakobsen geïntroduceerd in de cloth simulation en is tegenwoordig een van de meeste gebruikte integratietechnieken. Omdat deze schema's even stabiel zijn als RK4, maar slechts de rekenkracht van een expliciete Euler stap hebben.

Tenslote werd er nog een extra techniek besproken, inverse dynamica, die er voor zorgt dat extra realisme kan bekomen worden zonder nood aan zeer stijve veren. Doordat zeer stijve veren altijd oorzaak zijn voor onstabiele simulaties is deze techniek zeer interessant om toe te passen na een integratiestap.

In de voorgaande hoofdstukken hebben we besproken hoe een cloth gesimuleerd kan worden zonder rekening te houden met wat we willen simuleren. In het volgende hoofdstuk zullen we een aantal manieren bespreken waarop beperkingen aan het cloth kunnen opgelegd worden zodat het mogelijk is om een cloth deel te laten nemen aan een bepaalde animatiescene.

Hoofdstuk 5

Beperkingen

Beperkingen zijn zeer nuttig in cloth simulation, we willen namelijk de mogelijkheid hebben om het cloth te beperken tot een bepaalde positie of snelheid. Dit maakt het mogelijk om het cloth accurater te controleren, een animator kan bijvoorbeeld een gordijn willen creëren. Om dit makkelijker te kunnen maken zou het handig zijn als particles aan de gordijnrail beperkt kunnen worden op een bepaalde lijn.

Beperkingen worden tijdens de simulatie afgedwongen. In dit hoofdstuk zullen we dieper ingaan op de manier waarop de beweging van een particle beperkt kan worden, met name in het geval dat de particle niet mag bewegen, in een vlak moet blijven, op een lijn blijven en een punt volgen[Mel02]. Daarna bespreken we een meer algemene vorm om een particle te beperken, gebruik makende van *Constraint Forces*[Wit01a].

5.1 Beperk de beweging van een Particle

In elke simulatiestap wordt de positie en/of de snelheid van een particle veranderd. Als de beweging van een particle echter beperkt moet worden, moet de positie en/of snelheid veranderd worden volgens deze beperking en niet alleen volgens de krachten die er op inwerken. In de volgende secties zullen we meer in detail de nodige aanpassingen aan de positie en/of snelheid, die nodig zijn om een bepaalde beperking af te dwingen, bespreken.

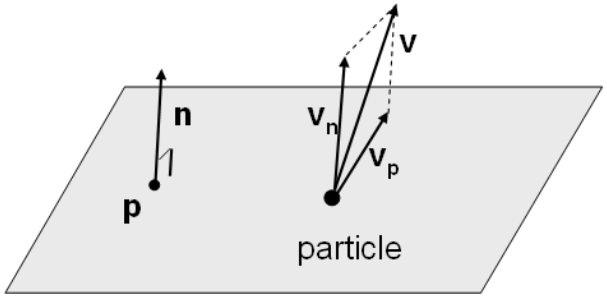
5.1.1 Fixed

Om een particle te beperken tot een bepaalde positie $\mathbf{x}_f$ moet de particle zich eerst op positie $\mathbf{x}_f$ bevinden (een verplaatsing naar $\mathbf{x}_f$ kan voor instabiliteit of verlies van immersiviteit leiden). Dit kan makkelijk bekomen worden door tijdens elke simulatie stap de snelheid nul te maken of massa van de particle aan te passen naar oneindig, wat hetzelfde impliceert als een snelheid van nul. Afhankelijk van de integratietechniek bestaat er ook de mogelijkheid om de particle uit de simulatie te halen wanneer deze in de *fixed* toestand is.

5.1.2 Particle in het vlak

Een vlak is gedefinieerd door een vlaknormaal **n** en een punt **p**. Om er voor te zorgen dat de particle in het vlak blijft moeten de snelheidscomponenten in de richting van de normaal van het vlak geëlimineerd worden. Om deze componenten te verwijderen moet de snelheid van de particle opgesplitst worden in een snelheid parallel met de normaal van het vlak $\mathbf{v}_n$ en een snelheid parallel met het vlak $\mathbf{v}_p$. Voor een grafische weergave van deze opsplitsing zie 5.1.

$$\mathbf{v}_n = (\mathbf{v} \cdot \mathbf{n})\mathbf{n}$$
$$\mathbf{v}_p = \mathbf{v}_n - \mathbf{v}$$



Figuur 5.1: Particle in het vlak

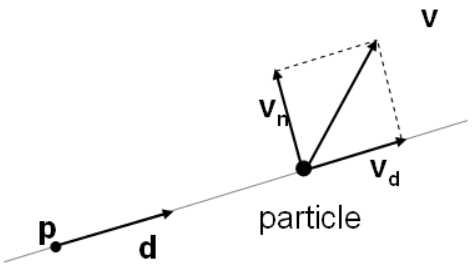
Aangezien de particle niet mag wegbewegen van het vlak en die beweging voorgesteld wordt door $\mathbf{v}_n$, wordt $\mathbf{v}_p$ als de aangepaste snelheid van de particle gebruikt.

5.1.3 Particle op een lijn

Een lijn wordt voorgesteld door een punt **p** en een richting **d**. Net als in het geval van een vlak mag de particle enkel in de richting van de lijn bewegen. Om deze beperking waar te kunnen maken moet de snelheid van de particle opgesplitst worden langs de richting van de lijn $\mathbf{v}_d$ en een component loodrecht op de richting van de lijn $\mathbf{v}_n$ (zie figuur 5.2):

$$\mathbf{v}_d = (\mathbf{v} \cdot \mathbf{d})\mathbf{d}$$
$$\mathbf{v}_n = \mathbf{v}_d - \mathbf{v}$$

Net als bij het vlak mag de particle niet bewegen volgens $\mathbf{v}_n$ en wordt de aangepaste snelheid $\mathbf{v}_d$.



Figuur 5.2: Particle op een lijn

5.1.4 Een punt volgen

Er zijn twee mogelijkheden om deze beperking af te dwingen:

Fixed volgen: Dit is de simpelste techniek om een bepaald punt te volgen. De particle wordt beperkt tot een vast punt, Fixed (zie sectie 5.1.1). Nu de particle niet meer kan bewegen wordt elke simulatie stap de positie van de particle, de positie van het punt dat we moeten volgen.

Niet stijve veer: Een andere oplossing die de beperking niet perfect afdwingt, maar wel benadert, is om een niet stijve veer met rustlengte nul tussen het punt dat we moeten volgen en de particle te plaatsen. De veer zal dan zelf de ongewenste snelheidscomponenten elimineren op de particle en geen kracht uitoefenen op het te volgen punt.

Beide technieken hebben hun voor- en nadelen, bij de eerste techniek kan de particle zover verplaatsen dat het doek uit zijn vorm getrokken wordt wat een onrealistisch effect geeft en bij de tweede techniek wordt de beperking niet perfect nageleefd, dit kan wel geminimaliseerd worden door de stijfheidsconstante van de veer te verhogen, maar dit induceert natuurlijk een meer onstabiel systeem (zie sectie 4.3.1).

5.2 Constraint Forces

Een constraint force is een kracht die de versnelling van de particle aanpast zodat deze versnelling consistent is met de opgelegde beperkingen. In het verloop van deze sectie zal het duidelijker worden wat hiermee bedoeld wordt, maar eerst zal aan de hand van een voorbeeld uitgelegd worden wat energy functies zijn, die later nodig zullen zijn om tot constraint forces te komen.

Een energy functie zorgt voor een slordig en benaderend mechanisme om beperkingen op te leggen: stel een functie $C(\mathbf{a}, \mathbf{b}) = \mathbf{a} - \mathbf{b}$; deze kan gezien worden als een functie die particle a en particle b op dezelfde plaats wil houden. Indien beide particles op dezelfde plaats zijn zal deze functie waarde nul teruggeven, dit is een *behavior* functie die waarde nul probeert te bereiken. Om het in woorden van [Wit01b] te zeggen:

> In veel gevallen is het mogelijk om de gewenste configuratie te specifiëren door een functie te geven die nul bereikt op het moment dat alles goed is.

Vanuit deze behavior functies kunnen energy functies berekend worden door de afgeleide te berekenen. In [Wit01b] is een uitgebreid voorbeeld beschreven dat uitlegt hoe men van een behavior functie tot een lineaire gedempte veer komt.

Een behavior functie beschrijft dus de manier waarop we een particle willen beperken. Met deze behavior functie worden energy functies berekend die dan gebruikt kunnen worden om een constraint force te berekenen en op die manier een particle te beperken tot die behavior functie. Het concept van de versnelling aanpassen zodat aan de beperking voldaan is, zoals eerder vermeld, wordt uitvoerig en algemeen uitgelegd in [Wit01a] als ook de manier om dit te implementeren. We zullen hier slechts een voorbeeld geven om een particle op een eenheidscirkel te houden, wat het idee weergeeft:

De behavior functie:

$$C(\mathbf{x}) = \frac{1}{2}(\mathbf{x} \cdot \mathbf{x} - 1)$$

Alle legale posities voor $\mathbf{x}$ zijn die die voldoen aan $C(\mathbf{x}) = 0$. Indien $\mathbf{x}$ een legale positie is dan zijn alle legale snelheden diegene die voldoen aan:

$$\dot{C}(\mathbf{x}) = \mathbf{x} \cdot \dot{\mathbf{x}} = 0$$

Net als alle legale versnellingen zijn:

$$\ddot{C}(\mathbf{x}) = \ddot{\mathbf{x}} \cdot \mathbf{x} + \dot{\mathbf{x}} \cdot \dot{\mathbf{x}} = 0 \tag{5.1}$$

Als we starten met een legale positie en snelheid moet enkel vergelijking 5.1 voldaan worden gedurende de simulatie. De versnelling van een particle is:

$$\ddot{\mathbf{x}} = \frac{\mathbf{f} + \mathbf{f_c}}{m}$$

Met als $\mathbf{f}$ de krachten die op de particle in werken en $\mathbf{f_c}$ de ongekende constraint force die ervoor zal zorgen dat de particle op de eenheidscirkel blijft. Als we $\ddot{\mathbf{x}}$ vervangen in 5.1 krijgen we:

$$\begin{aligned} \ddot{C}(\mathbf{x}) &= \frac{\mathbf{f} + \mathbf{f_c}}{m} \cdot \mathbf{x} + \dot{\mathbf{x}} \cdot \dot{\mathbf{x}} = 0 \\ &\Downarrow \\ \mathbf{f_c} \cdot \mathbf{x} &= -\mathbf{f} \cdot \mathbf{x} - m\dot{\mathbf{x}} \cdot \mathbf{x} \end{aligned} \tag{5.2}$$

We hebben nu één vergelijking en twee onbekenden, de twee componenten van $\mathbf{f_c}$ (cirkel is $2D$). Dit geeft als probleem dat we niet simpelweg kunnen oplossen naar de constraint force zonder een extra conditie, we voegen de extra conditie toe die zegt dat een constraint force nooit energie toevoegt of verwijdert van het mass spring systeem. De kinetische energie en zijn afgeleide van de tijd zijn:

$$\begin{aligned} T &= \frac{m}{2}\dot{\mathbf{x}} \cdot \dot{\mathbf{x}} \\ \dot{T} &= m\ddot{\mathbf{x}} \cdot \dot{\mathbf{x}} = \mathbf{f} \cdot \dot{\mathbf{x}} + \mathbf{f_c} \cdot \dot{\mathbf{x}} \end{aligned}$$

Deze laatste stelt de kracht of werk gedaan door $\mathbf{f}$ en $\mathbf{f_c}$ voor. De extra opgelegde conditie stelde dat de beperking geen extra energie mag inbrengen of verwijderen dus de laatste term van de tijdsafgeleide moet nul zijn. Hierdoor moet ook voor elke legale $\dot{\mathbf{x}}$ de $\mathbf{f_c} \cdot \dot{\mathbf{x}}$ verdwijnen en dus gelijk zijn aan nul. Hieruit volgt dat $\mathbf{f_c}$ in de richting van de positie $\mathbf{x}$ moet wijzen en de constraint force gelijk is aan:

$$\begin{aligned} \mathbf{f_c} &= \lambda \mathbf{x} \\ &\Downarrow \\ \lambda &= \frac{\mathbf{f} \cdot \mathbf{x} - m\dot{\mathbf{x}} \cdot \dot{\mathbf{x}}}{\mathbf{x} \cdot \mathbf{x}} \end{aligned} \tag{5.3}$$

Waarbij voor vergelijking 5.3 te bekomen, $\mathbf{f_c}$ vervangen wordt in vergelijking 5.2. Met lambda opgelost te hebben kunnen we nu de constraint force berekenen alsook de versnelling en gewoon verder gaan met de simulatie op een normale manier.

Als extra opmerking moeten we hier ook nog bijvoegen dat door de fouten, geïntroduceerd tijdens het integreren, we een extra feedback term moeten introduceren die ervoor zorgt dat we niet uit de cirkel zullen driften. Een voorbeeld van zo een feedback term is een gedempte veer.

Deze techniek is een krachtige manier om particles te beperken omdat het mogelijk is om een arbitraire functie te nemen als een gewenste beperking. Zoals reeds vermeld kan de algemene techniek voor het berekenen van de constraint forces gevonden worden in [Wit01a].

5.3 Conclusie

Twee technieken werden besproken om particles te beperken. De eerste techniek werkte zonder krachten toe te voegen aan het systeem, maar de verandering van positie of snelheid van de particle werd aangepast zodat deze de beperking niet breken. Een groot nadeel van deze methode is dat altijd met de hand uitgerekend moet worden welke aanpassingen nodig zijn. Met de tweede, meer algemene techniek wordt dit probleem verholpen door gebruik te maken van energy functies. Deze worden afgeleid tot constraint forces die toegevoegd worden aan het systeem. Als nadeel heeft deze techniek dat door het toevoegen van krachten en door de numerieke fouten die voorkomen tijdens het oplossen van het systeem deze beperkingen *nooit* 100% nageleefd zullen worden zonder een extra feedback term.

Met beperkingen kan het cloth op allerlei manieren gecontroleerd worden, maar er is nog één toevoeging nodig opdat we het cloth kunnen gebruiken in een animatiescene. Stel dat een gordijn gemodelleerd wordt, beperkingen zijn dan nuttig om een rij particles op de gordijnrail te doen bewegen. Maar stel dat iemand tegen het doek duwt of er aan trekt, dan zou het bijvoorbeeld tegen het raam of de muur kunnen botsen. Het efficiënt detecteren van botsingen is het onderwerp van het volgende hoofdstuk.

Hoofdstuk 6

Collision Detection

Het doel van collision detection is om automatisch te melden wanneer een geometrisch contact tussen twee objecten gaat gebeuren of feitelijk al gebeurd is. Geometrische modellen kunnen bestaan uit veelhoeken, splines of algebraïsche oppervlakken. Collision detection vindt toepassingen in veel gebieden[LG98]. In dit hoofdstuk zullen we ons vooral concentreren op het snel vinden van intersecties bij vervormbare objecten, aangezien een cloth een vervormbaar object is.

Zonder collision detection kunnen we geen realistische simulatie maken. Het cloth zou zomaar overal dooruit vallen of nergens op kunnen rusten tenzij we op de juiste plaats beperkingen leggen (hoofdstuk 5), maar dit is nogal omslachtig. Het zou veel makkelijker zijn om, indien het cloth ergens tegen aanbotst, dit automatisch op te lossen zonder dat de animator hiervoor beperkingen moet opleggen. Nadat een botsing gevonden is moet deze ook nog opgelost worden (Collision Response), hiervoor zijn er een aantal mogelijkheden. We zullen de mogelijkheden niet bespreken maar verwijzen hiervoor naar [BFA02, Soe02, VT97].

We concentreren ons vooral op het vinden van collision met andere objecten en niet met een cloth dat zichzelf intersecteert, ook wel *self-collision* genoemd. Het vinden van contact met andere objecten is meestal belangrijker in animatiescenes. Het zal later ook nog blijken dat deze methodes broodnodig zijn bij haptic rendering in hoofdstuk 8. Een kort overzicht van bestaande self-collision technieken wordt op het einde van dit hoofdstuk gegeven.

Het vinden van intersecties wordt vaak gezien als een proces van twee fases[Bra02]. Een *broad* fase en een *narrow* fase. De broad fase vermindert efficiënt het aantal objecten die misschien met elkaar botsen, door objecten die duidelijk te ver van elkaar verwijderd zijn te elimineren. Van de broad fase zullen we geen enkel algoritme bespreken. De broad fase is van belang in een simulatie met zeer veel objecten, indien er N objecten zijn moet er voor N^2 objecten getest worden of ze met elkaar botsen of niet. Maar zoals eerder gezegd zijn we vooral geïnteresseerd in het vinden van intersecties tussen twee objecten en dit is nu net de taak van de narrow fase. De narrow fase bepaalt zeer snel de gebieden van een object die met elkaar in botsing kunnen zijn en controleert deze gebieden dan exact op een botsing[1]. We zullen in dit hoofdstuk beginnen met de bespreking van twee algemene structuren die gebruikt kunnen worden in de narrow fase om sneller mogelijke botsingsgebieden te vinden, namelijk *Bounding Volume Hierarchies* (BVH's) en *Spatial Subdivision*.

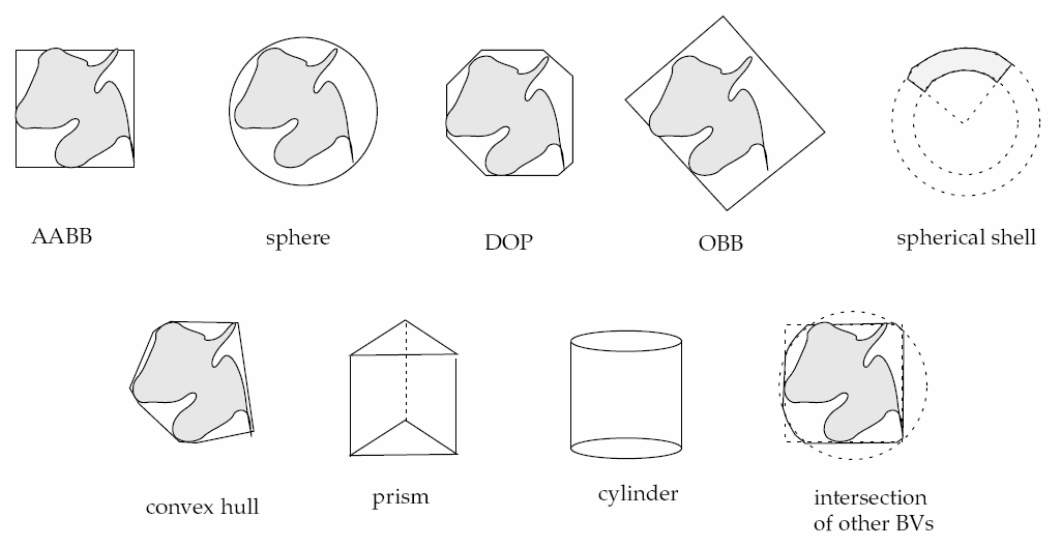
[1] Het controleren van de gevonden gebieden in de narrow fase wordt soms ook de *final* of *exact* fase genoemd.

6.1 Bounding Volume Hierarchies

Bounding Volume Hierarchies zijn een van de meest efficiënte datastructuren voor collision detection. Ze worden vooral zeer veel gebruikt bij niet-vervormbare objecten. Het idee van een BVH is een boomstructuur waarbij primitieven van een object recursief verdeeld worden totdat een blad criterium is bekomen (meestal bevat elk blad één primitief). In deze context zijn primitieven de bouwstenen van het object, dit kunnen veelhoeken zijn of andere vormen zoals NURBS patches. Een BVH moet aan een aantal eigenschappen voldoen opdat deze goed bruikbaar is voor collision detection[Bra02]:

- De hiërarchie benadert het volume van het object conservatief, elk niveau past strakker om een deel van het object dan zijn ouder (ook parent genoemd, het niveau hoger).
- De kinderen van elke knoop (node) in de hiërarchie moeten de primitieven van het object bedekken dat de ouderknoop bedekt.
- De hiërarchie moet op een automatische manier gecreëerd worden zonder enige interactie van de gebruiker.
- De *bounding volumes* in de hiërarchie moeten het origineel object zo strak mogelijk omvatten.

Meestal wordt een BVH op voorhand berekend zodat deze gewoon gebruikt kan worden tijdens de simulatie. Dit werkt alleen maar als een object niet vervormt. Indien een object wel vervormbaar is zal ofwel de BVH opnieuw opgebouwd moeten worden, ofwel hersteld worden zodat deze het object terug helemaal bevat. Het volledig opnieuw opbouwen van een BVH duurt meestal enkele seconden omdat er allerlei berekeningen nodig zijn waardoor dit veel trager is dan het herstellen van de BVH[Ber97].



Figuur 6.1: De meest gebruikte bounding volumes[ZL03]

Er zijn veel bounding volumes mogelijk, voor een overzicht van een aantal mogelijkheden zie figuur 6.1. Om een theoretische vergelijking mogelijk te maken van de verschillende bounding volumes wordt vaak volgende formule gebruikt[Ber97, Bra02, LC98]:

$$T = (N_v \times C_v) + (N_p \times C_p) + (N_u \times C_u) \tag{6.1}$$

waarbij

- T: De totale kost voor het berekenen van de intersectie tussen twee objecten voorgesteld door BVH's.
- N_v: Het aantal overlappingstesten tussen twee bounding volumes.
- C_v: De kost om twee bounding volumes te testen op overlapping.
- N_p: Het aantal primitieven dat gecontroleerd wordt op intersectie.
- C_p: De kost om twee primitieven te testen op intersectie.
- N_u: Het aantal bounding volumes dat moet ge-update worden.
- C_u: De kost om een bounding volume te updaten.

Hoe kleiner het resultaat in bovenstaande vergelijking hoe beter de BVH. Maar er zitten twee tegenstrijdige componenten in die vergelijking. De bounding volumes kunnen het origineel object zo goed mogelijk omvatten, waardoor N_v en N_p omlaag gaan, maar dit heeft meestal als gevolg dat C_v omhoog zal gaan. Een andere mogelijkheid is om ervoor te zorgen dat de overlappingstest van twee bounding volumes zo snel mogelijk gaat. Hierdoor zal C_v omlaag gaan, N_v en N_p zullen echter net omhoog gaan. Merk op dat C_p geen invloed heeft op de keuze van het bounding volume. Bijvoorbeeld bij de bol en AABB (Axis-Aligned Bounding Box) bounding volumes zal C_v laag zijn maar zijn N_v en N_p hoger. Terwijl bij een OBB (Oriented Bounding Box) het net andersom is, de overlappingstest zal duurder zijn dan bij een AABB of een bol. Het laatste deel in de vergelijking, tussen haakjes, werd aan de vergelijking toegevoegd in [LC98]. Deze toevoeging is alleen nodig indien er met vervormbare objecten gewerkt wordt. Dan is het belangrijk dat er zo snel mogelijk een bounding volume voor een primitief berekend kan worden (C_u wordt dan lager). Het berekenen van een bol of AABB gaat sneller dan bijvoorbeeld een OBB of een k-DOP (DOP staat voor Discrete Orientation Polytope en een AABB is bijvoorbeeld een 6-DOP). Rekening houdend met de applicatie waarvoor er collision detection nodig is, kan er door deze formule in acht te houden een zo goed mogelijke BVH gekozen worden.

Onafhankelijk van het bounding volume type zullen we in de volgende secties bespreken hoe we een BVH construeren en hoe we deze gebruiken voor collision detection. Na een algemene bespreking zullen we voor bepaalde bounding volumes meer uitleg verschaffen in de context van vervormbare objecten. Tenslotte zullen we ook nog bekijken hoe we zo efficiënt mogelijk een reeds gecreëerde BVH kunnen herstellen zodat deze terug past rond een vervormbaar object.

6.1.1 Construeren van een BVH

Er zijn drie gebruikte manieren om een BVH te construeren, top-down, bottom-up en insertion. De top-down techniek is de meest populaire en blijkt daarbij zeer efficiënt voor collision detection. Deze techniek is ook zeer makkelijk om te implementeren. We zullen alleen deze

techniek behandelen; voor meer informatie over andere technieken verwijzen we naar een zeer theoretische bespreking over allerlei collision detection technieken [ZL03].

Het basisidee achter de top-down techniek is om recursief een verzameling van objecten te splitsen totdat een bepaalde drempelwaarde is bereikt. Het opsplitsen wordt begeleid door specifieke criteria of heuristieken die voor een goede BVH zorgen. Een veel voorkomende en relatief simpele heuristiek is de volgende. Elke primitief wordt benaderd door zijn centrum. Vervolgens wordt voor de verzameling van deze punten de belangrijkste component berekend. Ofwel wordt de grootste component gekozen, waarna een vlak orthogonaal wordt geplaatst door deze grootste component (deze stelt een as voor) en door het barycentrum van alle punten in de verzameling. Hierdoor wordt de verzameling in twee deelverzamelingen gesplitst[Qui94]. Een andere mogelijkheid is om het splits-vlak te plaatsen door de mediaan van alle punten[Ber97]. In het laatste geval verkrijgen we een gebalanceerde boom, maar het is niet duidelijk of deze bomen een beter efficiëntie geven bij het detecteren van collision.

Nadat een verzameling in twee is gesplitst moet er rond deze twee deelverzamelingen een bounding volume berekend worden. De manier waarop hangt af van welke bounding volume gebruikt wordt en de techniek om de BVH op te bouwen. Maar een algemeen goed idee is om een minimaal volume te bekomen[BG00]. Afhankelijk van of de BVH gebruikt wordt voor niet-vervormbare objecten of voor vervormbare objecten kan gekozen worden voor verschillende technieken. In het geval van niet-vervormbare objecten kiezen we een techniek die heel veel rekenkost met zich meebrengt, in het geval van vervormbare objecten een techniek die zo snel mogelijk een goede benadering bekomt. Brown et al.[BSB$^+$01] gebruiken een BVH voor hun chirurgische simulator die als basis het algoritme van Quinlan[Qui94] gebruikt (een BVH met als bounding volume bollen), met de nodige aanpassingen voor vervormbare objecten. Deze bepaalt dan ook gewoon de straal en positie van een bol om de twee deelverzamelingen te bevatten. Hierdoor is de bekomen bol vaak iets groter maar veel sneller berekend. Merk op dat hierdoor C_u omlaag gaat.

We hebben nog niets gezegd over de drempelwaarde waarop het algoritme moet stoppen. Tegenwoordig is dit altijd totdat een bounding volume nog maar één primitief heeft. Het grote voordeel hiervan is dat bij het gebruik van vervormbare objecten de structuur van de boom behouden kan worden en alleen de bounding volumes herberekend moeten worden op elke tijdstap. Quinlan[Qui94] gebruikte ook deze drempelwaarde maar stelde voor om de primitieven te bemonsteren met bollen en zo de BVH te construeren, ook spheretree genoemd. Omdat de primitieven bemonsterd worden met bollen moet deze bemonstering elke keer als een object vervormt opnieuw gebeuren en daardoor kan de structuur van de BVH veranderen. Indien we met vervormbare objecten werken is dit een ongewenst resultaat; we willen de structuur van de BVH namelijk behouden. Merk op dat Quinlan nog helemaal niet bezig was met vervormbare objecten.

Er rest nog één eigenschap van een BVH, de opdelingsfactor. Tot nu toe verdeelden we een verzameling van primitieven altijd in twee. Bij niet-vervormbare objecten is dit de meest gebruikte manier, maar voor vervormbare objecten blijkt dat het gebruik van een quartaire of octaire boom betere resultaten oplevert[LAM01, MKE03]. Dit komt vooral omdat er minder knopen moeten ge-update worden en de totale update kost N_u hierdoor verlaagt. De diepte van de recursie tijdens de collision detection is ook lager waardoor er minder behoefte is aan geheugen op de stack. Voor het bouwen van een BVH met een hogere opdelingsfactor worden dezelfde heuristieken gebruikt, alleen wordt er dan in plaats van in twee te verdelen, in vier

of acht verdeeld.

De BVH wordt bij het begin van een applicatie opgebouwd. Hierbij worden allerlei dingen in acht genomen zodat de BVH zo optimaal mogelijk is voor de doel-applicatie. Gebruik makende van deze BVH worden er botsingen opgespoord. De manier waarop dit gebeurt zal in de volgende sectie aan bod komen.

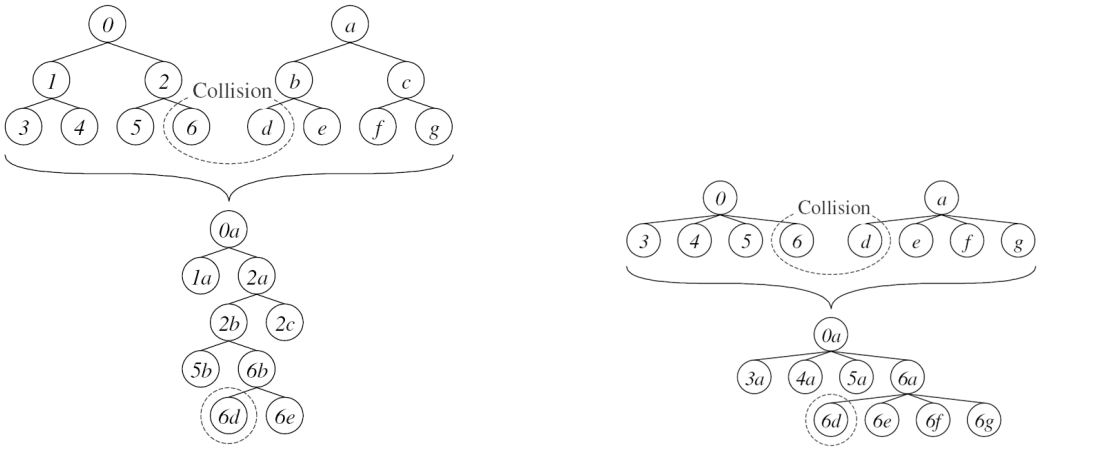
6.1.2 BVH Traversal

Voor het testen van een botsing tussen twee objecten of een self-collision van een object worden de BVH's van boven naar beneden (top-down) doorlopen en paren van knopen worden recursief getest op overlapping. Als twee knopen die elkaar overlappen bladeren zijn, worden de primitieven van de bladeren getest voor intersectie. Als één knoop een blad is en de andere een interne knoop, wordt het blad getest tegen alle kinderen van de interne knoop. Als beide knopen interne knopen zijn, wordt er geprobeerd om de kans op intersectie zo snel mogelijk te minimaliseren. Om deze reden testen Van Den Bergen[Ber97] en Quinlan[Qui94] de knoop met het kleinste volume met de kinderen van de knoop met het grootste volume. Merk op dat het ook mogelijk is om een botsing te testen tussen twee objecten waarbij één van de twee objecten geen BVH heeft. Stel dat we willen testen of een bol in botsing is met een cloth, dan kunnen we gewoon testen of deze bol overlapt met een knoop in de BVH van het cloth en indien het dan om een blad gaat testen op intersectie. Deze laatste methode zal nuttig blijken in hoofdstuk 8. Voor alle duidelijkheid sommen we het algoritme, om op een botsing tussen de BVH's A en B van twee objecten te testen, nog even op in pseudo-code:

Algorithm 1 COLLISION(A,B)

```
1: if A en B niet overlappen then
2:    return
3: end if
4: if A en B bladeren zijn then
5:    return de primitieven die moeten getest worden op intersectie
6: else
7:    if VOLUME(A) < VOLUME(B) then
8:       SWAP(A,B)
9:    end if
10:   for all kinderen van A: A[i] en B do
11:      COLLISION(A[i],B)
12:   end for
13: end if
```

In bovenstaand algoritme worden de kinderen geindexeerd met A[i] omdat het algoritme de opdelingsfactor van de BVH niet kent. In de vorige sectie werd gezegd dat de recursiediepte verlaagd werd door een BVH met hogere opdelingsfactor. Een vergelijking van deze diepte is te zien in figuur 6.2.



(a) Twee BVH's met een opdelingsfactor van twee

(b) Twee BVH's met een opdelingsfactor van vier

Figuur 6.2: Traversal van BVH's met verschillende opdelingsfactor[MKE03]

6.1.3 Updaten van een BVH

Het opbouwen van een BVH en het vinden van botsingen met behulp van een BVH hebben we reeds besproken. Maar stel dat we een dynamische simulatie hebben met vervormbare objecten. Dan zal de BVH na een paar tijdstappen niet meer geldig zijn, de primitieven van de objecten verlaten hun bounding volume. Er zijn twee mogelijkheden, ofwel bouwen we de BVH opnieuw op ofwel proberen we hem te herstellen. Het eerste geval duurt vaak enkele seconden, en in een real-time applicatie is deze manier dus niet mogelijk. Het tweede geval, het herstellen van een BVH, kan relatief snel. In deze sectie zullen we bespreken hoe we een BVH kunnen herstellen of ervoor zorgen dat het herstellen van de BVH uitgesteld kan worden. Het niet opnieuw bouwen van de BVH brengt een probleem met zich mee indien het object zich kan opsplitsen. Dit zou als gevolg hebben dat er nu twee BVH's zijn in plaats van één. Bij een applicatie waar het object dus kan splitsen moet er rekening mee gehouden worden dat op bepaalde momenten de BVH's opnieuw gebouwd moeten worden.

Indien we de BVH updaten zullen we zijn structuur niet veranderen. Om deze reden kunnen we best tijdens het opbouwen van de BVH, wat op voorhand gebeurt, veel tijd steken in het genereren van een goede structuur. Het is mogelijk om de structuur te balanceren zodat het aantal primitieven goed verdeeld is. Eerder werd gezegd dat het balanceren van de BVH geen zekerheid geeft op het sneller vinden van botsingen. Daarentegen kan wel gesteld worden dat bij vervormbare objecten, waarbij de BVH elke tijdstap hersteld wordt, het genereren van een gebalanceerde BVH wel voordelen zal bieden bij het herstellen. Indien slechts één kant vervormt zal alleen één kant van de BVH hersteld moeten worden. Bij een niet-gebalanceerde boom kan het gebeuren dat een primitief verdwaald raakte in de andere kant van de boom en dus deze kant ook hersteld dient te worden[BSB$^+$01, Ber97].

Het is mogelijk om het herstellen van een BVH een tijdje uit te stellen, als we bij het creëren van de bounding volumes de bounding volumes met een ϵ vergroten[BG00, MKE03]. Hierdoor is het mogelijk dat primitieven niet meteen uit het bounding volume gaan en dus een herstelling van de BVH niet meteen noodzakelijk is. Dit kan op een aantal manieren gebeuren

afhankelijk van het bounding volume, maar we zullen hier niet dieper op ingaan omdat het kiezen van een juiste ϵ problemen met zich meebrengt. Een te kleine ϵ heeft geen invloed en een te grote zou wel eens kunnen zorgen dat er te snel getest wordt of twee primitieven intersecteren.

De simpelste manier om een BVH te herstellen is de bladeren te zoeken die niet meer geldig zijn, waarbij de primitief in het blad dus niet meer wordt omhuld door het bounding volume. Deze bladeren worden dan hersteld door de primitief terug te omhullen en de BVH bottom-up te doorlopen (van onder naar boven) en de bounding volumes van de knopen hoger aan te passen aan de veranderingen[BG00].

Larsson et al.[LAM01] vergelijken top-down en bottom-up technieken om een BVH te herstellen. Bij top-down wordt een knoop pas hersteld indien deze wordt bezocht tijdens het zoeken van een botsing. Een bottom-up techniek werd besproken in de vorige paragraaf. Ze kwamen tot het besluit dat indien tijdens de collision detection veel diepe knopen worden bereikt de bottom-up techniek betere resultaten oplevert. Terwijl als slechts enkele diepe knopen bereikt worden een top-down methode sneller is. Rekening houdend met deze vaststellingen stellen ze een hybride techniek voor. Deze techniek herstelt de bovenste helft van de BVH bottom-up en alleen als niet herstelde knopen worden bereikt worden ze top-down hersteld. Het nadeel van deze techniek is dat alle knopen ook de informatie van de bladeren moeten kennen, maar het aantal onnodige knopen dat wordt hersteld wordt verminderd.

Brown et al.[BSB$^+$01] gebruiken een bottom-up techniek waarbij een priority queue Q wordt gebruikt. Hierbij worden de bounding volumes gesorteerd volgens hun diepte in de BVH (Brown et al. werken met bollen als bounding volumes). Q wordt geïnitialiseerd met alle bladeren die vervormde primitieven bevatten sinds de vorige update van de BVH. Daarna wordt volgend algoritme toegepast op Q:

Algorithm 2 Herstellen BVH volgens het algoritme van Brown et al.

1: **while** Q niet leeg is **do**
2: Neem het eerste bounding volume uit Q: $w \leftarrow Q$.first();
3: Update het bounding volume w
4: Voeg de ouderknoop van w in Q toe: Q.insert(w.parent());
5: **end while**

De enige bounding volumes die worden aangepast zijn deze die minstens één vervormde primitief bevatten. Elk van deze bounding volumes wordt op deze manier maar één keer aangepast, ook als het meerdere vervormde primitieven bevat. Dit algoritme werkt sneller als er lokale vervormingen zijn dan wanneer ze over het object verdeeld zijn. Stel dat een lokale vervorming k primitieven beïnvloedt waarbij k veel kleiner is dan het totaal aantal primitieven s. Dit geeft een totale hersteltijd van $O(k + \log s)$. Maar stel dat de k primitieven verspreid zijn over de bladeren, dan is de kost $O(k \log s)$. In het ergste geval, wanneer er veel primitieven vervormd zijn zal de hersteltijd nog steeds lineair, $O(s)$, zijn. Als we bovenstaand algoritme nader bekijken zullen we merken dat er iets niet klopt. Brown et al. werken met een BVH die een opdelingsfactor heeft van twee. Bij het toevoegen van de parent van een knoop in Q wordt deze dus minimaal één keer toegevoegd en maximaal twee keer. Hierdoor kan een knoop dus twee keer behandeld worden, waardoor de complexiteit $O(s)$ in het ergste geval niet gehaald wordt. Maar met een paar simpele aanpassingen van bovenstaand algoritme kan in het ergste geval toch $O(s)$ bekomen worden. Ik stel volgende aanpassingen voor:

- We geven elke knoop in de boom een unieke identifier op de volgende manier: als we de boom in breadth-first doorlopen, geven we elke knoop een oplopend nummer. Hierdoor zullen de ouders altijd een lager nummer hebben dan hun kinderen en elke knoop zal een lager nummer hebben dan zijn rechterbuur.
- In plaats van de priority queue Q op diepte te sorteren, sorteren we Q nu op de eerder bepaalde unieke identifier waarbij de hoogste identifier vanboven komt te staan. Hierdoor zullen alle bladeren boven hun linkerbuur in Q zitten.

Als we nu in de eerste iteratie de bovenste knoop in Q behandelen, houden we zijn ouder bij en voegen we deze toe in Q, omdat deze ouder sowieso nog niet in Q zat. In de volgende iteratie zal de linkerbuur van het zonet behandelde blad ge-update worden, omdat deze knoop nu bovenaan in Q zit. Na de update wordt dan gecontroleerd of de ouder van de linkerbuur al niet eerder toegevoegd is. Dit doen we door de ouder van de linkerbuur te vergelijken met de ouder die apart bijgehouden wordt. Als deze gelijk zijn, wordt de ouder van de linkerbuur niet toegevoegd aan Q. Als deze niet gelijk zijn, voegen we de ouder van de linkerbuur toe aan Q en houden we deze nu bij in plaats van de vorige bijgehouden ouder. We blijven dit verder toepassen tot het laatste element in Q. Door gebruik te maken van deze aanpassingen zal een knoop altijd maar één keer behandeld worden, wat de gewenste complexiteit geeft.

Alle algoritmes die nodig zijn voor een BVH te gebruiken in een simulatie met vervormbare (of niet-vervormbare) objecten zijn besproken. We hebben in deze bespreking zoveel mogelijk geprobeerd om nog geen bounding volume te kiezen. In de volgende sectie zullen we de populairste bounding volumes bij het gebruik van vervormbare objecten bespreken.

6.1.4 Bounding volumes voor vervormbare objecten

Om een bounding volume te kiezen kunnen we best vergelijking 6.1 gebruiken waarbij we de laatste term tussen haakjes beschouwen. Deze term is ook een van de belangrijkste, we willen niet dat het herstellen van de boom een bottleneck vormt in de simulatie. Er zijn twee zeer populaire bounding volumes, bollen en AABB's, die een zeer lage kost hebben bij het berekenen van intersecties (C_v) en het berekenen van hun grootte bij een bepaald primitief (C_u). Deze bounding volumes werden door een hele hoop onderzoekers gebruikt, onder andere in de volgende werken [Qui94, Ber97, BG00, BSB$^+$01].

Het voordeel van een bol is dat deze enkel een positie en een straal heeft, een bol is rotatie invariant en heeft dus geen oriëntatie. Het berekenen van de omvattende bol is dus snel. Om te berekenen of twee bollen elkaar overlappen, berekenen we simpelweg of de afstand tussen hun posities kleiner is dan de som van de stralen (of meer efficiënt: of het kwadraat van de afstand tussen hun posities kleiner is dan het kwadraat van de som van de stralen). Een nadeel is dat een bol vaak niet zo goed rond een bepaald object past en er dus meer intersectietesten nodig zijn. Een AABB is een doos waarbij de oriëntatie aansluit met het coördinatenstelsel. Een AABB wordt gedefinieerd door zijn centrum en zijn hoogte, breedte en diepte (of ook vaak door dit centrum, het hoekpunt linksachter beneden en het hoekpunt rechtsboven vooraan). Het berekenen van een AABB voor een primitief kan snel gedaan worden door de zes uiterste waarden (kleinste en grootste x-, y- en z-waardes) van dit primitief te berekenen. Het testen of twee AABB's elkaar overlappen kan snel gedaan worden door een separating axis test te doen langs de drie assen. Als twee AABB's niet overlappen zullen ze bij een van de drie testen

een separating axis opleveren, dit wil zeggen dat ze langs de geteste as niet overlappen. In verschil met een bol sluit een AABB meestal beter aan bij een object, tenzij dit een langwerpig schuin object is.

Als we een bol en AABB vergelijken blijkt dat deze nagenoeg gelijk zijn qua kost. Een bol omsluit een object slechter dan een AABB en kost dus meer overlappingstesten, maar daar tegenover kost een overlappingstest minder bij een bol dan bij een AABB. Afhankelijk van de vorm van de objecten kan een van deze twee bounding volumes gekozen worden. Er is nergens in de literatuur iemand die een van deze twee bounding volumes als de overwinnaar bestempeld omdat zeer veel factoren meespelen en het dus zo goed als onmogelijk is te bepalen welke nu het beste is. Het beste is dus gewoon om te kiezen naargelang de situatie.

Recent is er een derde populair bounding volume bijgekomen voor vervormbare objecten. Mezger et al.[MKE03] introduceren het gebruik van 18-DOP's (een AABB waarbij elke ribbe een nieuw vlak introduceert) om een BVH op te bouwen en ze stellen ook allerlei heuristieken voor die gebruikt worden tijdens het opbouwen en updaten van een BVH. We zullen op deze heuristieken niet dieper ingaan, meer informatie is te vinden in hun werk.

6.2 Spatial Subdivision

Er bestaan een aantal methodes die spatial subdivision voorstellen bij het gebruik van collision detection. Bij BVH's wordt rond de objecten een hiërarchie opgebouwd, bij spatial subdivision is dit net andersom. De objecten worden in de ruimte verdeeld op verschillende manieren. Het is mogelijk dat er één verdeling bestaat voor de hele ruimte van de simulatie[BT95, RSH00, THM+03] of dat de ruimte rond het object wordt verdeeld en elk object zijn eigen verdeling heeft[GDO00]. Een van de voordelen van deze techniek is dat er geen rekening gehouden wordt met de structuur of vorm van het object, het object kan zichzelf opsplitsen of objecten kunnen samengevoegd worden. Dit is mogelijk omdat het object in de ruimte verdeeld wordt en dus de vorm van het object niet bepaalt hoe de ruimte verdeeld wordt. We zullen van beide mogelijkheden één techniek bespreken die bruikbaar is bij vervormbare objecten.

6.2.1 BucketTree

Bij de BucketTree methode wordt er rond elk object een ruimte gedefinieerd waarbij het object in deze ruimte verdeeld wordt. Deze methode werd geïntroduceerd door Ganovelli et al.[GDO00] met als hoofddoel om de kost van collision detection tussen vervormbare objecten te verminderen.

In deze methode wordt een octree gebruikt. Een octree wordt recursief opgebouwd door het volume dat een object omvat op te delen in acht octanten en alleen de octanten waarbij een of meer primitieven van het object zich in dat octant bevinden behouden worden. Een octree is dus een boom waarbij elke knoop maximaal acht kinderen heeft en indien in een kind zich geen primitief van het object bevindt wordt het kind geschrapt. De recursie wordt meestal gestopt nadat zich slechts nog een bepaald aantal primitieven van het object in een octant bevindt.

Het basisidee van de techniek is om een AABB rond het object te plaatsen. Deze AABB komt overeen met de wortel van een octree. Daarna wordt een octree opgebouwd tot op

een bepaalde diepte. Deze octree bevat al zijn kinderen. Indien er dus geen primitief van een object in een blad aanwezig is, bestaat dit blad dus toch (alleen in de bladeren worden primitieven van het object bijgehouden). Vanaf nu zullen we de bladeren van de octree buckets noemen. Elke tijdstap van de simulatie worden de coördinaten van de AABB van het object aangepast en dus wordt de octree ook mee opgeschoven. Na deze opschuiving van de octree worden alle primitieven van het object in de juiste bucket geplaatst. De collision detection wordt op dezelfde manier als bij een BVH getest (sectie 6.1.2).

Stel dat we over het object een octree met l niveaus bouwen. Deze octree zal 8^l bladeren bevatten waarbij elk blad een bucket is. Als de dimensies van de AABB van het object S_x, S_y en S_z zijn, dan zijn de dimensies van een bucket: $s_x = \frac{S_x}{2^l}$, $s_y = \frac{S_y}{2^l}$ en $s_z = \frac{S_z}{2^l}$. We zullen de buckets indexeren als tripletten, (n_x, n_y, n_z) met $0 \leq n_x, n_y, n_z < 2^l$. Als nu p_x de x-positie is van een primitieve in de ruimte en B_x is de minimum x-positie van de AABB dan is de juiste bucket voor deze primitief: $n_x(p) = \dfrac{(p_x - B_x)}{s_x}$. Voor n_y en n_z is dit analoog. Er zijn twee mogelijkheden om alle primitieven in de juiste buckets te stoppen. De eerste manier is gewoon brute-force alle primitieven afgaan en ze in de juiste bucket stoppen. De tweede manier werkt met arrays voor elke as en kan gebruik maken van frame to frame coherency: de objecten verplaatsen/vervormen tegen een traag tempo waardoor ze de volgende tijdstap nog steeds op ongeveer dezelfde plaats zijn en van dit gegeven wordt dan gebruik gemaakt. Meer details van de tweede manier zijn te vinden in [GDO00].

Als we deze methode vergelijken met een BVH gebouwd van AABB's kunnen we opmerken dat bij deze techniek het vervormbaar object mag gesplitst worden, er wordt namelijk geen gebruik gemaakt van de structuur van het object. Een nadeel ten opzichte van een AABB BVH is dat de primitieven niet uniform verdeeld kunnen zijn over de bladeren van de BucketTree. Het kan zelfs zo erg zijn dat indien een object bestaat uit m primitieven, $m-1$ primitieven kort bij elkaar zijn en er 1 ver weg is. Dan kan het zijn dat deze $m-1$ primitieven in één enkele bucket zitten waardoor deze methode nutteloos wordt.

Ganovelli et al. bespreken niet voor welke soort primitieven deze methode geïmplementeerd werd. In hun werk lijkt het alsof er met punten of bollen gewerkt wordt, maar er wordt niets gezegd over een primitief dat meerdere buckets inneemt. Stel dat elk punt van dit primitief (bijvoorbeeld een veelhoek) in een bucket gestopt wordt, dan kan het nog zijn dat de primitief een bucket snijdt waar geen punt van deze primitief in voorkomt. Dit heeft als gevolg dat een primitief niet in alle buckets zit en de mogelijkheid tot het niet vinden van een botsing kan voorkomen. In de volgende sectie zullen we een methode bespreken die rekening houdt met dit probleem.

6.2.2 Optimized Spatial Hashing

Teschner et al.[THM+03] gebruiken spatial hashing voor het detecteren van collision detection. De techniek die ze gebruiken heeft een aantal gelijkenissen met deze van Ganovelli et al.[GDO00] (besproken in de vorige sectie), bij hashing echter worden waardes gehasht en in een bucket gestoken.

De techniek hier verdeelt $\mathbb{R}^3$ impliciet in kleine gridcellen in plaats van gebruik te maken van ingewikkelde 3D datastructuren zoals octrees of BSP-Trees (Binary Space Partitioning Trees). Er wordt gebruik gemaakt van een hashfunctie die 3D gridcellen op een hashtabel

afbeeldt. Deze techniek is efficiënt in geheugengebruik en zeer flexibel. In theorie kunnen oneindig grote spatial grids gebruikt worden. Er is dus ook geen informatie nodig over hoe groot de omgeving is en 3D datastructuren worden vermeden.

We zullen het algoritme bespreken voor veelhoekige modellen, voor andere soorten modellen is het principe analoog. Het algoritme verloopt in twee fases. In de eerste fase worden alle hoekpunten (vertices) geclassificeerd ten opzichte van de 3D gridcellen en dus in de hashtabel geplaatst. In de tweede fase worden ook de veelhoeken gehasht. Er wordt rekening mee gehouden dat een veelhoek met meerdere gridcellen kan snijden. Als een veelhoek met een gridcel snijdt wordt gecontroleerd of de hoekpunten die geassocieerd werden met deze gridcel in de eerste fase deze veelhoek penetreren. Aangezien alle primitieven van een object behandeld worden is het mogelijk om op botsing en self-collision te testen. Als een hoekpunt een veelhoek penetreert is er een botsing gevonden, als het hoekpunt en de veelhoek tot hetzelfde object behoren is er een self-collision gevonden. Als het hoekpunt een deel van de veelhoek is, wordt er geen intersectietest gedaan. In de volgende paragrafen zullen we beide fases uitgebreider bespreken.

Het classificeren van hoekpunten kan gezien worden als het hashen van deze punten en gebeurt op volgende wijze: de posities van de punten worden discreet gemaakt ten opzichte van de grootte van de gridcellen. De grootte van de gridcellen kan geschat worden op voorhand of door de gebruiker gespecificeerd worden. Om de juiste gridcel te bepalen wordt de positie van het punt (x, y, z) gedeeld door de grootte van de gridcel l en afgerond naar het vorige geheel getal $(i, j, k) = (\lfloor x/l \rfloor, \lfloor y/l \rfloor, \lfloor z/l \rfloor)$. Elke gridcel wordt dus net als bij de BucketTree methode van de vorige sectie aangeduid door een triplet, maar in plaats van een 3D datastructuur bij te houden wordt dit triplet omgevormd naar een 1D hashwaarde h. Daarna wordt de informatie van het hoekpunt en het object in de hashtabel op plaats h gezet. Teschner et al. stellen volgende hashfunctie voor zodat de hashwaardes uniform verdeeld zijn opdat een geschikte prestatie van het algoritme verwacht kan worden:

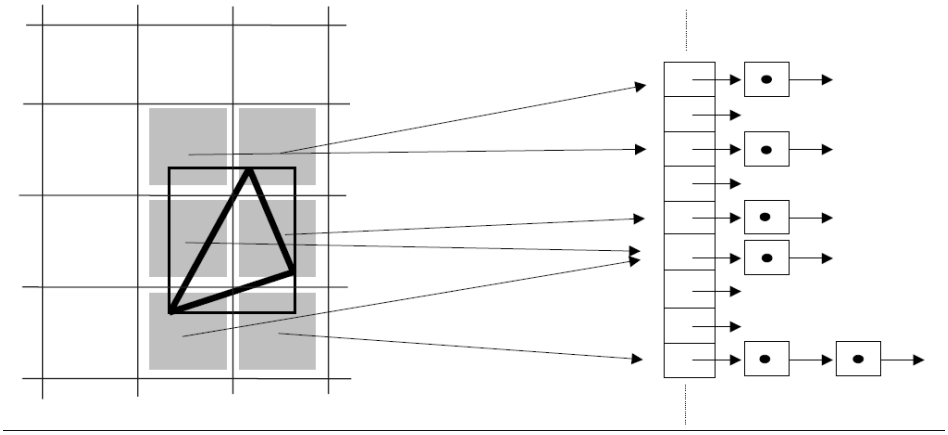
$$h = hash(i, j, k) = (i \times p_1 \textbf{ xor } j \times p_2 \textbf{ xor } k \times p_3) \textbf{ mod } n$$

waarbij p_1, p_2, p_3 grote priemgetallen zijn, respectievelijk 73856093, 19349663, 83492791, en n de grootte van de hashtabel is.

De grootte van de hashtabel is belangrijker dan een goede hashfunctie, hoe groter de tabel hoe kleiner de kans op botsingen in de hashwaardes. Een nadeel aan grotere hashtabellen is dat er meer geheugenbeheer nodig is, wat voor vertraging zorgt. De implementatie van Teschner et al. gebruikt een priemgetal voor de grootte van de hashtabel omdat dit meestal betere resultaten oplevert. Dit priemgetal wordt groter gekozen dan het aantal primitieven. In hun implementatie wordt de hashtabel ook niet leeggemaakt. Het leegmaken, of her-initialiseren, van de hashtabel in elke simulatiestap kan voor grote vertragingen zorgen. Om dit probleem op te lossen maken ze gebruik van een timestamp. In de eerste fase wordt gecontroleerd of de timestamp van de huidige simulatiestap overeenkomt met de timestamp van de hashcel waar een punt in geplaatst wordt. Indien de timestamp op de hashcel verouderd is wordt de hashcel leeggemaakt en de timestamp ge-update. Anders wordt het punt gewoon toegevoegd. Indien nu in de tweede fase veelhoeken afgebeeld worden op hashcellen wordt gecontroleerd of de hashcel dezelfde timestamp bevat, indien niet hoeft er niet gecontroleerd te worden op intersectie. Gebruik makende van deze techniek moet de hashtabel nooit leeggemaakt worden. Een laatste parameter die het algoritme beïnvloedt is de grootte van een gridcel. Hoe

groter een cel, hoe meer primitieven naar dezelfde hashwaarden worden afgebeeld. Indien de veelhoeken veel groter zijn dan de gridcellen zullen veel cellen gecontroleerd moeten worden. Het kiezen van een goede grootte voor de gridcellen is dus cruciaal. Uit testen blijkt dat een gridcel de grootte moet hebben van de gemiddelde edge lengte van alle veelhoeken om tot een optimale prestatie te komen[2]. Ook blijkt dat deze waarde de grootste invloed heeft op het algoritme.

Naast het behandelen van alle hoekpunten worden in de eerste fase ook de AABB's van alle veelhoeken berekend gebaseerd op de vervormde toestand. In de tweede fase worden dan alle veelhoeken overlopen. Eerst worden de minimum- en maximumwaardes van de AABB van een veelhoek discreet gemaakt net als in de eerste fase bij de hoekpunten. Deze waardes worden dan gebruikt om alle gridcellen te bepalen die een veelhoek snijden. Dit gebeurt door alle gridcellen te overlopen van het discreet minimum tot het discreet maximum van de AABB (figuur 6.3). Tijdens het overlopen wordt ook getest op intersectie op de hoekpunten aanwezig in deze gridcellen (of beter: aanwezig in de overeenkomstige hashwaarde in de hashtabel). De intersectietest hangt af van de veelhoek die gebruikt wordt. We zullen deze test niet bespreken, in [THM+03] wordt uitgebreid besproken hoe dit gebeurt voor tetrahedra.



Figuur 6.3: Het overlopen van gridcellen in fase twee[THM+03]

De resultaten van de techniek hier besproken zijn zeer interessant. Experimenten duiden aan dat het detecteren van botsingen en self-collisions voor vervormbare objecten tegen een snelheid van 15Hz kan uitgevoerd worden met 20000 tetrahedra en 6000 vertices op een standaard pc[THM+03].

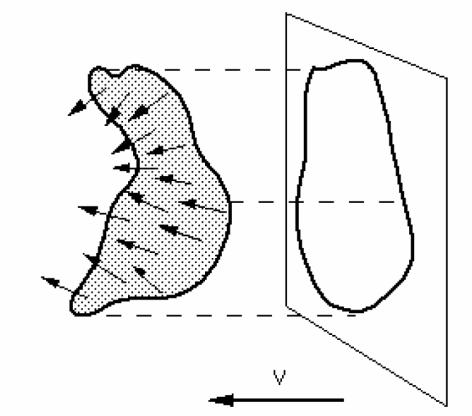
6.3 Self-Collisions

Een self-collision is een botsing waarbij een object zichzelf penetreert of snijdt. Het vinden van self-collisions is tegenwoordig nog steeds een zeer zware berekening. Het vormt een bottleneck in cloth simulation en andere simulaties van vervormbare objecten, waarbij vooral collision

[2]Ook al vervormen de objecten, bij een realistische simulatie worden objecten normaal gezien nooit twee keer zo groot als hun startgrootte.

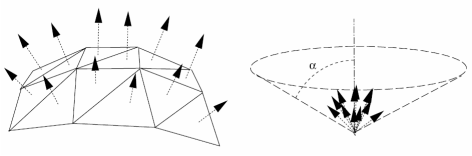
response zeer kostelijk is. We zullen geen diepgaande bespreking geven van reeds bestaande self-collision technieken, maar gewoon een overzicht.

Het vinden van self-collisions kan met de technieken besproken in vorige secties. Bij een BVH kan de BVH op zichzelf getest worden voor het vinden van een self-collision. Maar een groot probleem is dat bounding volumes van naburige gebieden van een object elkaar kunnen overlappen en dus nodeloos getest worden. Om deze gevallen efficiënt te elimineren kunnen heuristieken gebruikt worden. Volino et al.[VMT95] stellen een exacte methode voor om onnodige zelf-overlappingstesten tussen verschillende BVH's te elimineren. Hiervoor wordt een vector gezocht met een positief dotproduct met alle normalen van de primitieven in de regio. Als zo een vector bestaat en de projectie van de regio op het vlak in de richting van de vector geen zelf-overlapping geeft, dan kan de regio niet zelf-overlappen. Het idee van deze methode is terug te vinden in figuur 6.4. We kunnen dit efficiënt implementeren met behulp van BVH's, voor meer uitleg verwijzen we naar [VMT95].



Figuur 6.4: Geometrische condities waarbij geen self-collision optreedt[VMT95]

Een andere techniek werd voorgesteld door Provot[Pro97]. In deze techniek worden normaalkegels geïntroduceerd. Het idee is gebaseerd op het feit dat regio's met een lage kromming niet kunnen zelf-intersecteren, ervan uitgaande dat ze convex zijn. Voor elke regio wordt er een kegel berekend waarbij deze kegels een superset van de normaalrichtingen voorstellen, zie figuur 6.5. De kegels worden berekend gebruik makend van de BVH en ge-update tijdens het updaten van de BVH. De tophoek van een kegel stelt de kromming van de regio voor. Hierbij kunnen mogelijke zelf-intersecties voorkomen als de hoek groter is dan π. Alle details zijn te vinden in het werk van Provot[Pro97]. Deze techniek werd ook toegepast door Mezger et al.[MKE03] die als een bounding volume 18-DOP's gebruiken en ook andere optimalisaties voorstellen met betrekking tot cloth simulation.



Figuur 6.5: Een voorbeeld van een normaalkegel[Pro97]

Bij de spatial subdivision technieken kunnen we ook self-collision toepassen. Bij BucketTree worden dan de buckets getest op primitieven die elkaar snijden. Bij het algoritme van Teschner et al.[THM+03] kan in de tweede fase gecontroleerd worden of de gevonden botsing met een ander object of met zichzelf is.

Heel vaak wordt in cloth simulation het detecteren van self-collisions weggelaten vanwege het extra werk dat nodig is. Self-collisions worden wel gedetecteerd bij simulaties die fysisch correct moeten zijn en gebruikt worden voor bijvoorbeeld een film. Bij applicaties die real-time updatetijden nodig hebben zal het dus zeker weggelaten worden. Het voorkomen van bepaalde self-collisions kan op andere manieren gerealiseerd worden. Indien het cloth rond een bepaald object gemodelleerd wordt en niet vrij kan bewegen kan het opleggen van allerlei beperkingen self-collisions voorkomen.

6.4 Conclusie

In dit hoofdstuk werd de laatste toevoeging van technieken besproken. Om te zorgen dat het cloth nog meer realistisch wordt mag het niet zomaar door andere objecten in de scene vallen. In dit hoofdstuk werden dus collision detection technieken besproken waarbij vooral aandacht uitging naar de narrow fase. In deze fase wordt er zo efficiënt mogelijk de gebieden van objecten die met elkaar in botsing kunnen zijn, gevonden en gecontroleerd op een botsing.

Uit twee bekende onderzoeksgebieden van collision detection die gebruikt worden bij vervormbare objecten, zoals een cloth, werden een aantal technieken besproken. Een van deze twee gebieden is Bounding Volume Hierarchies. Er wordt een structuur rond het object opgebouwd gebruik makend van bounding volumes. Het is mogelijk om deze structuur efficiënt te herstellen voor een object dat vervormt. De structuur wordt recursief overlopen om te controleren of er een mogelijke botsing plaats heeft met een ander object.

Het andere onderzoeksgebied waaruit we technieken besproken hebben is Spatial Subdivision. In plaats van een hiërarchie rond het object op te bouwen wordt een ruimte gedefinieerd rond het object en wordt het object verdeeld over deze ruimte. Op deze manier worden gebieden van deze ruimte gecontroleerd op het bevatten van meerdere objecten, als dit zo is wordt er gecontroleerd op een botsing. Er zijn twee mogelijkheden bij spatial subdivision, ofwel wordt er een ruimte gedefinieerd die niet rond het object zit en worden alle objecten in dezelfde datastructuur gestopt, zulk een techniek is Optimized Spatial Hashing. Ofwel wordt rond elk object de ruimte verdeeld, dit werd er gedaan in de BucketTree techniek.

Beide gebieden hebben hun voor- en nadelen. Bij BVH's zijn er problemen indien een object kan opsplitsen. Bij Spatial Subdivision treden er problemen op indien de ruimte niet uniform verdeeld is. Het geeft ook problemen als er zoveel primitieven zijn die in de ruimte verdeeld worden dat het indelen in elke stap veel meer tijd zal kosten ten opzichte van het updaten van een BVH. Hierbij moet immers niet elke primitief behandeld worden indien deze niet buiten zijn bounding volume is vervormd. Als laatste werd er in dit hoofdstuk nog een overzicht gegeven van bestaande self-collision technieken die gebruik maken van de eerder besproken technieken.

De voorbije hoofdstukken gaven een introductie op cloth simulation door middel van mass spring systemen. Technieken om de simulatie een realistisch uitzicht te geven en de simulatie te controleren kwamen aan bod. In het volgende deel van deze thesis zullen we een overzicht

geven van haptics en de manier waarop haptics gerealiseerd kunnen worden in samenwerking met vervormbare objecten. Hierna bespreken we in het derde deel mijn implementatie van cloth simulation gecombineerd met haptics.

Deel II

Haptics

Introductie

In deel I werden technieken besproken om aan cloth simulation te doen. In dit deel zal het andere gedeelte van mijn literatuurstudie aan bod komen, namelijk haptics. Haptics is het onderzoeksgebied waarbij onderzoek wordt gedaan naar gevoel. In deze thesis betreft het vooral haptics in een virtuele omgeving, een 3D visualisatie van een reële wereld, waarbij de immersiviteit van deze omgeving verhoogd kan worden door het simuleren van gevoel. Bijvoorbeeld het vastnemen en verplaatsen van objecten of het voelen van de hardheid van een object (bijvoorbeeld een zachte stressbal of een harde tennisbal).

Haptics wordt onder andere toegepast in gebieden zoals medische training, chirurgische simulaties, tele-operaties, CAD design, entertainment... Deze toepassingen begeven zich stilaan vanuit het onderzoeksgebied naar meer commerciële toepassingen. Er is echter nog een lange weg te gaan eer haptics een standaard wordt in commercieel gebruik.

In het eerste hoofdstuk van dit deel, 7, wordt er een overzicht gegeven omtrent haptics. Dit overzicht gaat van een algemene bespreking van wat haptics precies is tot de bespreking van mogelijke haptic devices en applicaties.

In dit deel wordt samen met een algemeen overzicht van haptics een bespreking gegeven van de meest recente technieken omtrent haptic rendering, een techniek waardoor de gebruiker in staat is om in een virtuele omgeving te kunnen voelen. Hoofdstuk 8 zal de twee bekendste haptic rendering technieken voor veelhoekige modellen bespreken en ook een aantal technieken die gebruikt kunnen worden bij vervormbare objecten.

Hoofdstuk 7

Haptic Feedback

Haptics is het onderzoeksgebied rond gevoel. Een letterlijke vertaling van het woord haptic (haptisch) is: 'met betrekking tot het gevoel'. Haptic feedback of gevoelsterugkoppeling kan het best uitgelegd worden aan de hand van een voorbeeld. Tegenwoordig hebben moderne vliegtuigen servo-systemen om het vliegtuig te besturen. Vroeger bestond dit nog niet en kon de piloot voelen aan de kracht die de stuurknuppel uitwerkte op zijn handen of het vliegtuig in een overtrokken vlucht (*stall*, het te steil stijgen) terecht ging komen. Door de komst van de servo-systemen voelt de piloot geen krachten meer. De oplossing die hiervoor bedacht werd, was een van de eerste toepassingen van gevoelsterugkoppeling. Indien het vliegtuig in een stall kan komen (dit wordt berekend), zullen er krachten op de besturingsknuppel gezet worden zodat deze lijken op een oud systeem zonder servo [wik].

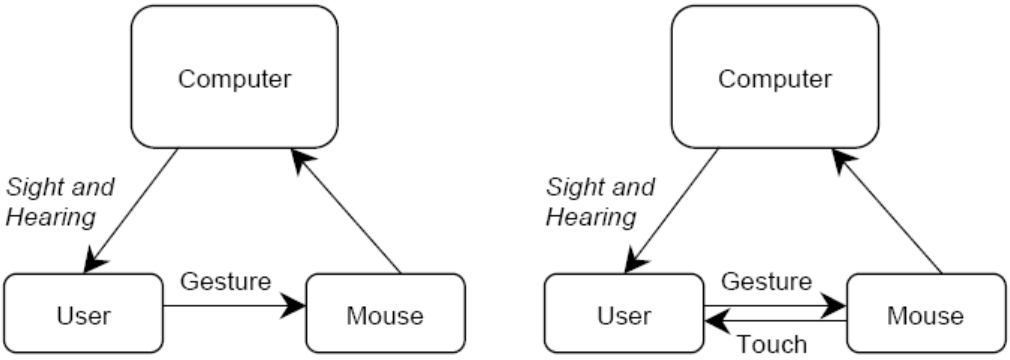
In het geval van computers bespreken we gevoelsterugkoppeling in virtuele systemen. Hierdoor wordt de mogelijkheid gecreëerd om een persoon objecten in een virtuele wereld te laten voelen. Door toevoeging van het gevoelszintuig wordt er getracht om de immersiviteit van de virtuele omgeving te verhogen.

In dit hoofdstuk zullen we aspecten bespreken die belangrijk zijn voor realistische gevoelsterugkoppeling: haptic interfaces en multi-modaliteit. Daarna overlopen we kort een aantal haptische devices en applicaties.

7.1 Haptic Interfaces

Door gebruik te maken van haptic devices (sectie 7.3), kan de gebruiker niet alleen informatie doorgeven aan de computer, maar ook informatie ontvangen in de vorm van een gevoel. Dit uitwisselen van gegevens wordt mogelijk gemaakt door een haptic interface. Er wordt dus gebruik gemaakt van bidirectionele sensoring[SIG99]. De acties van de gebruiker worden via de haptic interface doorgegeven aan de computer, die deze acties verwerkt. De gebruiker voelt op zijn beurt objecten in de virtuele wereld voorgesteld door de computer via het haptic device. Deze bidirectionele informatiestroom kan duidelijk gemaakt worden door te kijken naar het verschil tussen een normale muis en een haptische muis. In figuur 7.1 is aan de linkerzijde de normale muis te zien en aan de rechterzijde de haptische muis. Merk op dat bij een gewone muis de beweging van de muis ook gevoeld wordt, maar bij een haptische muis is

het mogelijk deze beweging te programmeren of het klikken van de knoppen anders te doen aan voelen[HACH+04].



Figuur 7.1: Voorbeeld van informatiestroom tussen een normale en haptische muis[HACH+04]

Haptic interfaces moeten rekening houden met de technische aspecten die nodig zijn om gebruik te kunnen maken van het menselijke gevoel of om de eigenschappen van het menselijke gevoel net uit te buiten. Het menselijk gevoel kan opgesplitst worden in tactiel en kinetisch gevoel, deze twee samen zorgen voor het menselijke gevoel en worden door de hersenen samengevoegd:

Tactiel: Tactiele gevoelens worden waargenomen door de ontvangers in de huid. Deze prikkels kunnen gebruikt worden om de details te voelen van een object zoals de textuur. Er kan ook zachtheid, vochtigheid, wrijving en nog vele andere eigenschappen mee worden waargenomen.

Kinetisch: Kinetische gevoelens zijn deze van de positie en beweging van de ledematen samen met de krachten overgedragen door de spieren. Het kan gezien worden als het bewustzijn van de toestand van het lichaam, ook proprioceptie genoemd.

De menselijke perceptie legt de limieten van technische aspecten vast. Dit kan best uitgelegd worden aan de hand van een voorbeeld. Wanneer we naar een TV scherm kijken zien we geen opeenvolging van stille beelden, noch zien we een matrix van gekleurde pixels. In plaats daarvan zien we bewegende beelden zoals we die zien als we rondom ons kijken. Dit is mogelijk omdat het menselijke visuele systeem de snelle opeenvolging van beelden niet kan verwerken als stille beelden en het kan ook de pixels niet onderscheiden. Dit voorbeeld geeft duidelijk aan hoe het menselijke perceptiesysteem kan uitgebuit worden om tot een realistische prikkel te komen[HACH+04].

De update rate ligt voor het voelen veel hoger dan het zien, er is nood aan minstens 1000Hz om een realistisch gevoel te simuleren. Verder zijn er nog andere drempelwaardes van het menselijke gevoelssysteem, bijvoorbeeld om het verschil tussen 2 punten op de vingertoppen te kunnen onderscheiden moeten deze minstens 1 mm van elkaar verwijderd zijn [SIG99].

7.2 Multi-modaliteit van haptic feedback

Het gebruik van haptic feedback samen met andere modaliteiten zoals visuele informatie blijkt van enorm belang te zijn voor de ervaring van de gebruiker. De combinatie van haptics met andere modaliteiten leidde tot enkele interessante resulaten[SIG99]:

- De toevoeging van visuele informatie aan het systeem zorgt voor een sterke toename van haptische perceptie. Aangezien hedendaagse systemen allemaal visuele informatie geven kan de toevoeging van haptic feedback voor extra realisme zorgen.
- Wanneer er auditieve informatie wordt toegevoegd aan het systeem zal de invloed ervan op de haptische perceptie slechts zwak zijn. De toevoeging van deze modaliteit is dus geen noodzaak om een realistische ervaring te creëren.
- Andere sub-modaliteiten van haptics kunnen nog toegevoegd worden aan het systeem zoals vibraties of het veranderen van de temperatuur. Deze zorgen voor een verhoging van immersiviteit of het aanwezigheidsgevoel. De toevoeging van deze sub-modaliteiten is echter niet zo voor de hand liggend als de toevoeging van visuele of auditieve informatie.

Het begrijpen van de invloed van verschillende modaliteiten is belangrijk voor het verdere onderzoek naar haptics en het creëren van goede applicaties.

7.3 Haptic Devices

Een haptic device is een toestel dat gevoelsterugkoppeling simuleert. Haptic devices hebben een aantal eigenschappen. Eerst zullen we deze eigenschappen bespreken, waarna we een aantal devices zullen overlopen, ingedeeld aan de hand van deze eigenschappen.

7.3.1 Eigenschappen van een haptic device

De vijf belangrijkste eigenschappen van een haptic device (zoals beschreven in [SIG99]) zijn:

Degrees of freedom (DOF): Het aantal vrijheidsgraden waarin haptic feedback kan gebeuren. Meestal is dit 1, 2, 3 of 6. Dit kan gezien worden als de richtingen waarin kan bewogen worden met het toestel en dus ook waarin gevoeld kan worden.

Actief of passief: Een haptic device is ofwel actief ofwel passief. Indien het actief is kan het device kracht teruggeven, indien het passief is niet. Passieve devices kunnen wel andere gevoelens terugkoppelen of dienen gewoon om de interactie met de virtuele omgeving realistischer te maken. Een passief device kan bijvoorbeeld gebruikt worden om de toestand van de hand van een gebruiker te weten (open of dicht).

Aarding: Indien een haptic device geaard (*grounded*) is, betekent dit dat het device niet vrij bewogen kan worden. Het is als het ware vastgemaakt aan de grond. Ongeaarde devices kunnen dus vrij bewogen worden. Hierin bestaat nog een speciale groep, exo-skeletal. Devices die exo-skeletal zijn, zijn vaak een soort van plastiek framework dat over een hand of arm word gezet zodat de gebruiker krachten kan voelen.

Sensor- en Aandrijfkwaliteit: De resolutie waarmee de sensoren en aandrijvers werken is van groot belang, net als hun dynamisch bereik.

Bandbreedte: De bandbreedte van het toestel is zeer belangrijk. Deze moet minstens gelijk zijn aan de nodige update rate. Neem als voorbeeld de bandbreedte van gevoel. Deze is 1000Hz; een haptic device moet dus 1000 keer per seconde zijn gevoelsterugkoppeling kunnen veranderen om voor realisme te zorgen.

Bij het onderzoeken naar en het maken van haptic devices moet er dus rekening gehouden worden met deze eigenschappen. Hierbij moet een afweging gemaakt worden tussen de performantie en de commercialiseerbaarheid. In de volgende secties zullen we een aantal voorbeelden geven van actieve haptic devices die we verder zullen onderverdelen volgens het aantal vrijheidsgraden.

7.3.2 Actieve haptic devices

De actieve devices zullen onderverdeeld worden volgens hun vrijheidsgraden. Deze onderverdeling geeft een goede groepering voor devices die gebruikt kunnen worden voor dezelfde doeleinden.

1 DOF

De devices met slechts 1 DOF kunnen meestal alleen maar twee soorten krachten teruggeven: ofwel genereren ze vibraties, ofwel een kracht tegengesteld aan de duw- of draairichting.

Deze devices worden veel gebruikt in entertainment applicaties: de meeste spelconsoles hebben tegenwoordig controllers die vibreren op goed gekozen momenten. Een ander voorbeeld is een stuurwiel voor het tegensturen bij het nemen van een bocht, zie figuur 7.2. Het doel van deze devices is meestal niet om een realistische terugkoppeling te geven maar eerder om de gebruiker bepaalde signalen te geven zoals bijvoorbeeld het raken van een obstakel.

Figuur 7.2: Sidewinder Force Feedback Steering Wheel (Microsoft)

2 DOF

Er zijn twee grote groepen van devices die 2 DOF ondersteunen. Een eerste groep zijn muizen en pen devices. De tweede groep is de groep van de joysticks, die als bij 1 DOF vooral gebruikt worden in de entertainment wereld. Voorbeelden van devices uit deze categorie zijn te zien in figuur 7.3.

Het gebruik van haptische muizen of pennen met 2 DOF is vooral geconcentreerd op een uitbreiding voor de 2D grafische user interface. Er kunnen krachten uitgewerkt worden zodat de foutgevoeligheid van applicaties omlaag gaat. Een persoon kan de muis niet buiten het actieve venster brengen of kan met een pen veel makkelijker de lijntjes volgen van een tekening omdat deze pen krachten uitoefent op de gebruiker aan de hand van deze lijnen.

Joysticks worden zoals eerder gezegd vooral gebruikt in de entertainment wereld, voor simulatoren zijn ze meestal niet correct genoeg. De krachten die gemodelleerd kunnen worden door een joystick kunnen vergeleken worden met deze van een 1 DOF stuurwiel; er worden krachten uitgeoefend op de joystick in de x- of y-richting.

(a) Sidewinder Force Feedback Pro (Microsoft)

(b) Logitech Wingman Force Feedback Mouse

Figuur 7.3: 2 DOF

3 DOF

Deze groep is de meest gebruikte groep in hedendaagse applicaties. Het bekendste haptic device is in deze categorie te vinden, namelijk de PHANToM. De PHANToM is een geaard toestel waarin de gebruiker zijn vinger in een vingerhoedje legt en in 3 DOF kan bewegen en krachten voelen. Wanneer aan het vingerhoedje een stylus wordt verbonden kan in 6 DOF gewerkt worden, in de meeste gevallen blijft de terugkoppeling wel beperkt tot 3 DOF.

Een ander bekend apparaat is de HapticMASTER[Hap]. Aan de kant waar kracht uitgeoefend wordt kan een andere eindeffector vastgemaakt worden zodat de HapticMASTER voor allerlei applicaties gebruikt kan worden. Een standaard eindeffector en een versnellingspook eindeffector wordt getoond in figuur 7.4. Een ander apparaat dat ook een wisselbare eindeffector heeft is de 3DOF Delta. Dit apparaat kan net als de PHANToM uitgebreid worden naar 6 DOF.

6 DOF

De haptic devices die het hoogst aantal vrijheidsgraden hebben namelijk 6, zijn het meest uitgebreid. Met 6 DOF kunnen de haptic devices zowel gebruik maken van de positie van een punt als de oriëntatie van het device of ledemaat van de gebruiker.

(a) Standaard eindeffector

(b) Versnellingspook eindeffector

Figuur 7.4: De HapticMASTER[Hap]

(a) Premium PHANToM 1.5

(b) 6 DOF Delta Haptic Device

Figuur 7.5: 6 DOF

Er bestaan nog niet zo veel 6 DOF haptic devices die effectief de markt bereikt hebben. De PHANToM premium 6DOF[1] is er één van, net als het 6DOF Delta Haptic Device (figuur 7.5. Deze devices zijn nog steeds duur en zijn dus nog altijd niet zo populair.

7.4 Haptische applicaties

In deze sectie zullen we een aantal applicaties die gebruik maken van haptics bondig bespreken. Hierbij snijden we volgende topics aan: chirurgische simulatie en medische training, teleoperaties en CAD design. Haptics kunnen op veel plaatsen gebruikt worden en zorgen voor een verbetering in het gebruiksgemak of de entertainment waarde van reeds bestaande applicaties.

[1]Niet alle premium versies zijn 6 DOF.

7.4.1 Chirurgische simulatie en medische training

Een van de eerste en populairste applicatiegebieden van haptics is in het gebied van chirurgische simulatie en medische training. Er zijn reeds talloze applicaties te vinden waarbij getracht wordt om een systeem te bouwen dat zo realistisch mogelijk een bepaalde chirurgische ingreep kan nabootsen, zodat de chirurg in kwestie kan oefenen of de moeilijkheden kan overlopen [MBB+02].

Voor medische training is het nut van haptics zeer hoog. De studenten moeten niet meer oefenen op mensen of dieren, maar kunnen gebruik maken van een computer met een haptic device. Neem als voorbeeld tandarts-studenten, deze moeten leren boren in het gebit. In plaats van te oefenen op mensen, wat sowieso al geen goed idee is, of op een kunstgebit, wat kostelijk is, kunnen de studenten oefenen met een haptic device dat er net zo uitziet als een boor. Na het boren worden zelfs de resultaten gegeven van hoe goed het geleverde werk was, die kunnen dienen als een objectieve evaluatie[Den].

7.4.2 Tele-operaties

Bij een tele-operatie wordt een bepaald systeem bestuurd vanop een afstand. Dit kan nuttig zijn voor werkzaamheden uit te voeren op de zeebodem of in de ruimte of andere gevaarlijke plaatsen zoals in een kerncentrale. Een voorbeeld hiervan is de *Canadarm*, een robotarm van de space shuttle waarmee onder andere satellieten worden losgelaten in de juiste omwenteling of satellieten die stuk zijn worden teruggehaald voor herstelling.

Het zou ook mogelijk zijn om dit principe te gebruiken in de geneeskunde om een operatie van op afstand uit te voeren. Een voorbeeld hiervan is minimaal binnendringende chirurgie [SIG99].

7.4.3 CAD design

Voor het modelleren van bepaalde objecten wordt meestal gebruik gemaakt van CAD. Het toevoegen van haptics aan CAD kan ervoor zorgen dat er veel sneller gemodelleerd kan worden en ook veel preciezer. Stel dat we complexe oppervlakken willen modelleren, dan kan het gebruik van een haptic device ten opzichte van een muis of toetsenbord zeer interessant zijn. De gebruiker kan als het ware het oppervlak vastnemen en er tegen duwen, aan duwen of uitrekken. Tijdens het manipuleren van het oppervlak voelt de gebruiker preciezer wat hij doet waardoor het gewenste resultaat sneller en gemakkelijker bekomen kan worden. Net als bij beeldhouwen is het heel belangrijk dat je voelt wat je doet, er zou een soort van beeldhouwstap kunnen komen in de modelleerfase waarbij er uitgaande van een concept een beeldhouwwerk wordt gemaakt door middel van haptics. Dit beeldhouwwerk wordt dan omgezet naar een CAD model[SIG99].

7.5 Conclusie

In dit hoofdstuk werd een introductie gegeven tot haptics. We bespraken aspecten van haptics, haptic interfaces en multi-modaliteit, waar rekening mee gehouden moet worden tijdens het ontwikkelen van applicaties.

Om gevoelsterugkoppeling te verwezenlijken is het noodzakelijk dat er haptic devices bestaan. De eigenschappen die belangrijk zijn voor een haptic device werden besproken, gevolgd door een overzicht van een aantal devices. Daarna werden er nog enkele voorbeelden van haptische applicaties gegeven.

In het volgende hoofdstuk zullen we technieken bespreken om aan gevoelsterugkoppeling te doen, waarbij we ons vooral concentreren op krachtterugkoppeling voor veelhoekige modellen.

Hoofdstuk 8

Haptic Rendering

Een actief haptic device is een device dat kracht teruggeeft, hierdoor kan de gebruiker objecten voelen, manipuleren en creëren. Haptic rendering is het proces waarbij een modelspecificatie wordt genomen en de gepaste krachten worden berekend om de illusie te geven van fysiek contact, gebruik makend van een haptic device[SIG99].

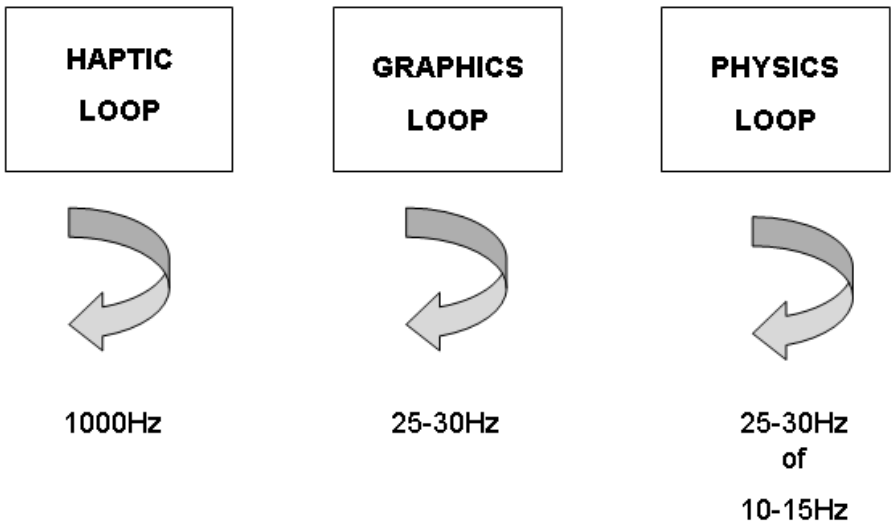
In dit hoofdstuk zullen we ons vooral concentreren op het haptic renderen van veelhoekige modellen, waarbij de besproken technieken ook toegepast kunnen worden op andere soorten modellen zoals parametrische oppervlakken. We zullen ook andere mogelijke modellen die gebruikt worden bij haptic renderen kort overlopen. Tenslotte zullen we een aantal gebruikte technieken voor het haptic renderen van vervormbare objecten bespreken. Maar eerst zullen we een overzicht geven van hoe een haptic systeem in elkaar zit zodat we een goed idee krijgen waar de haptic rendering nu net gebeuren zal.

8.1 Overzicht van een haptic systeem

Een haptic systeem is een goed voorbeeld van een real-time systeem. Dit betekent dat een aantal berekeningen of acties moeten gebeuren onder bepaalde voorwaarden. Deze voorwaarden werden behandeld in sectie 7.1, enkel de meest doorwegende en belangrijke zullen bij een haptic systeem van belang zijn. Bij haptics is de belangrijkste voorwaarde de bandbreedte van het gevoel. Deze is 1000Hz waardoor de berekeningen van de kracht die het haptic device moet teruggeven 1000 keer per seconden berekend moeten worden. Het berekenen van deze krachten met een snelheid van 1000Hz wordt de *Haptic Loop* genoemd. Deze tijdsbeperking is een groot nadeel en daarom werden er allerlei technieken ontworpen om ervoor te zorgen dat de nodige update rate behaald wordt.

In een haptic systeem is er meestal ook grafische feedback. Om tot vloeiende beelden te komen dienen deze tegen 25-30Hz gegenereerd te worden (*Graphics Loop*). Om deze reden kunnen we het haptic systeem opsplitsen in twee delen die elk apart hun ding doen. Maar stel dat we ook nog aan cloth simulation of meer algemeen, andere soorten fysische simulaties willen doen. Dan wordt het systeem meestal in drie delen gesplitst omdat de berekeningen van een fysische simulatie ook veel tijd kosten. Fysische simulaties hebben meestal een update snelheid gelijkaardig aan deze van de grafische feedback maar kunnen indien nodig ook aan een tragere rate ge-update worden (*Physics Loop*).

Het voordeel van het systeem op te splitsen, is dat elk deelsysteem alleen maar bezig is met zijn taak en er niet meer tijd aan verspilt dan nodig. Het is ook mogelijk om deze deelsystemen op een andere computer (CPU) te laten runnen en via het netwerk te laten communiceren [MRF+96]. Tegenwoordig bestaan er ook computers met hyperthreading of meerdere CPU's waardoor elk deelsysteem beter kan verdeeld worden. Een overzicht van een mogelijk haptic systeem waarbij er zowel een fysische simulatie als grafische feedback aanwezig is, is terug te vinden in figuur 8.1. In de rest van dit hoofdstuk zullen we vrij dikwijls teruggrijpen naar de termen haptic, graphics of physics loop waarbij meestal de update snelheid bedoeld wordt.



Figuur 8.1: Overzicht van een haptic systeem

8.2 Veelhoekige modellen

Het gebruik van veelhoekige modellen ligt voor de hand. In het gebied van de Computer Graphics wordt er het meest met veelhoekige modellen gewerkt die worden opgebouwd uit dezelfde primitieven. Doordat al jaren dezelfde bouwstenen gebruikt worden is de hardware voor het grafisch renderen van deze primitieven zeer sterk geoptimaliseerd voor deze primitieven. Deze primitieven zijn meestal driehoeken.

Naar het berekenen van intersecties met veelhoekige modellen is ook reeds veel onderzoek gedaan. Een overzicht van mogelijke technieken voor het berekenen van intersecties werd gegeven in hoofdstuk 6. Omdat we meestal de buitenkant van objecten voelen is deze manier van representeren ook zeer efficiënt. Over de binnenkant van het model is niets gekend en wordt ook geen informatie bijgehouden.

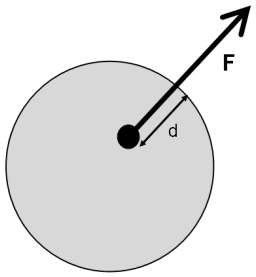
Bij het uitleggen van haptic rendering methodes zullen we de vinger van de gebruiker of het haptic device voorstellen als een bol zonder massa met een bepaalde straal. In deze bespreking zullen we deze bol zonder massa de *proxy* noemen. In sectie 8.2.2 zal blijken dat de term proxy daar vaak gebruikt wordt voor een vertegenwoordigend object, maar we zullen de term

proxy door heel de tekst blijven behouden en een andere naam voor dit vertegenwoordigend object gebruiken.

Er zijn twee populaire technieken om haptic rendering toe te passen op veelhoekige modellen. We stellen eerst de simpelste en eerste gedefinieerde techniek voor: *Penalty Based Methods*. Daarna bespreken we *Constraint Based Methods*, een techniek die werd geïntroduceerd om enkele problemen bij Penalty Based Methods op te lossen.

8.2.1 Penalty Based Methods

De eerste haptic rendering systemen gebruikten deze methode. Hierbij wordt een afstotende kracht proportioneel tot de penetratiediepte in een object gegenereerd, zie figuur 8.2. De manier waarop deze kracht berekend wordt, kan afhangen van de soort krachtterugkoppeling die gewenst is. We overlopen hier even de stappen die worden genomen in deze methode.



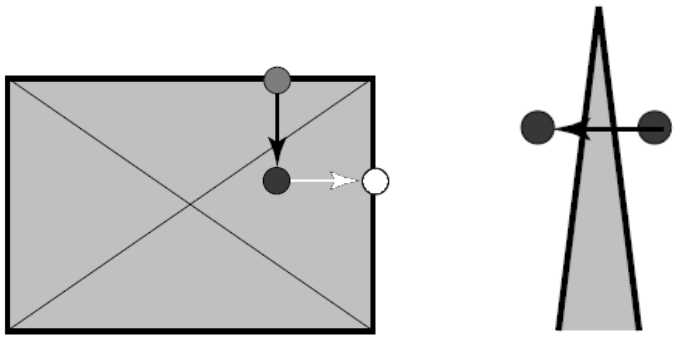
Figuur 8.2: Het principe van Penalty Based Methods

De eerste stap bestaat er in om de penetratiediepte te bepalen. Dit kan op allerlei manieren gebeuren. Voor bepaalde simpele objecten zoals een vlak of een bol kan dit zeer simpel berekend worden. Voor een veelhoekig model moet gekeken worden achter welk primitief de proxy gelegen is. Hiervoor worden meestal technieken uit de computer graphics gebruikt, een bespreking van een aantal van deze techniek is te vinden in hoofdstuk 6. Deze technieken moeten wel aangepast worden omdat er zich een aantal problemen voordoen bij deze methode; deze problemen zullen in de volgende paragrafen besproken worden. Het vinden van het contactpunt is één van de grote problemen die zich voordoen bij haptic renderen.

Gebruik makend van de penetratiediepte wordt in de volgende stap een afstotende kracht berekend waarbij meestal als richting de normaal gekozen wordt. Deze kracht wordt dan doorgegeven aan het haptic device. Deze berekeningen gebeuren in de haptic loop en moeten dus voldoen aan zijn update rate om een stabiele kracht te verkrijgen.

Deze methode is zeer efficiënt en makkelijk te implementeren maar bevat een paar nadelen. Wanneer het toegelaten is dat objecten elkaar doorsnijden kan het gebeuren dat de proxy twee objecten penetreert. Het optellen van deze krachten brengt problemen met zich mee, de krachten kunnen zo groot worden dat het haptic device of de gebruiker hier schade onder kunnen lijden. Een ander probleem doet zich voor doordat de penetratiediepte en richting niet altijd uniek gedefinieerd zijn. Wanneer een gebruiker in een object duwt, maar op een bepaald moment korter bij een ander oppervlak komt zal deze naar dit oppervlak geduwd worden, zie

figuur 8.3(a). Dit is normaal gezien niet het gewenste gedrag. Een laatste probleem doet zich voor bij dunne objecten. Het kan zijn dat deze niet dik genoeg zijn om een kracht te genereren en het haptic device zal dan gewoon erdoor geduwd worden, zie figuur 8.3(b).



(a) Krachtdiscontinuïteiten door foutieve verandering oppervlak

(b) Erdoor duwen bij te dunne objecten

Figuur 8.3: Problemen bij Penalty Based Methods[RKK97]

Voor al deze problemen is het mogelijk oplossingen te vinden die werken voor een bepaalde applicatie. Maar door deze problemen is een meer algemene methode uitgevonden die in bijna elke situatie werkt. Deze methode wordt besproken in de volgende sectie.

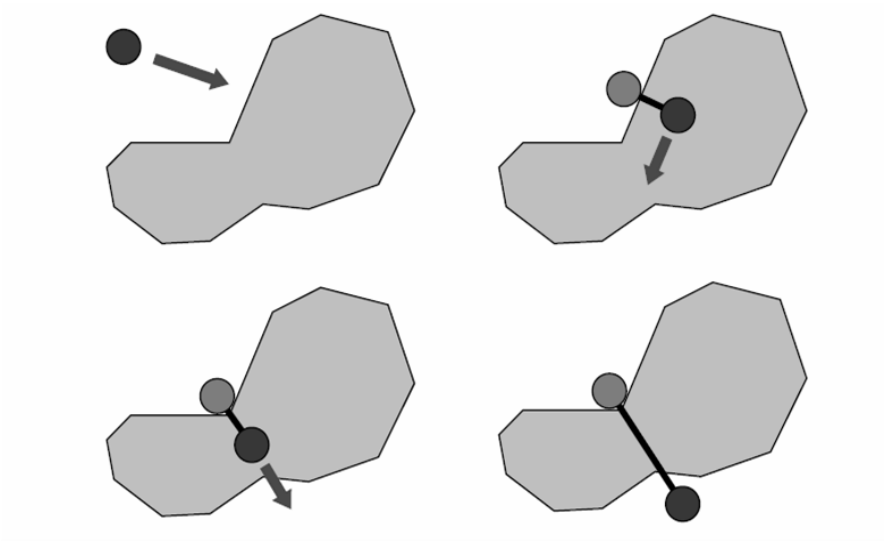
8.2.2 Constraint Based Methods

De methode van Constraint Based Methods, werd eerst voorgesteld door Zilles et al.[ZS95]. Er wordt gebruik gemaakt van een vertegenwoordigend object in de virtuele omgeving die de proxy *vervangt*. Dit vertegenwoordigend object is als het ware verbonden met de gebruiker's vinger door een stijve veer, zie figuur 8.4 (voor meer informatie over veren zie 2.2.2). Wanneer de gebruiker het haptic device beweegt en dus ook het vertegenwoordigend object, zal indien een bepaald object of obstakel geraakt wordt dit vertegenwoordigend object het obstakel niet penetreren maar aan de buitenkant stoppen. Een nieuwe positie wordt berekend voor het vertegenwoordigend object zodat de afstand van het vertegenwoordigend object tot de proxy zo minimaal mogelijk is. Nadat de positie bepaald is wordt er een kracht gegenereerd gebruik makende van de stijve veer.

Dit vertegenwoordigend object is gekend onder vele namen, GOD-Object, haptic point, ideal haptic interface point, surface contact point (SCP)[1]. In het vervolg van onze bespreking zullen we de term SCP gebruiken. Doordat het SCP een bepaald object van de simulatie niet kan penetreren, maar altijd op het oppervlak moet blijven kunnen computer graphics technieken voor collision detection gebruikt worden zonder nood aan enige aanpassingen.

Zowel de proxy als het SCP zijn massaloze bollen. Vrij recent werd voor het eerst een SCP voorgesteld die niet meer statisch is maar dynamisch. Doordat het SCP niet meer statisch is, is er meer controle over de beweging en het gedrag van het SCP. Bijvoorbeeld: wanneer

[1]Het vertegenwoordigend object kan ook proxy genoemd worden. Maar in deze bespreking gebruiken wij de term proxy al voor de echte positie van het haptic device.



Figuur 8.4: Constraint Based Method in actie: de donkere cirkel is de proxy en de lichtere cirkel is het vertegenwoordigend object[RKK97]

een statische SCP in contact komt met een object zal deze gewoon tegen het oppervlak gaan plakken, terwijl een dynamische SCP tegen dit oppervlak zou ketsen afhankelijk van de restitutie van het oppervlak en langzaamaan naar dit oppervlak terugkomen. We zullen in dit werk niet dieper ingaan op dynamische SCP's. Voor meer informatie over dynamische SCP's zie [NM04].

Er zijn veel manieren waarop deze methode geïmplementeerd kan worden, maar de meeste zijn analoog en hebben allemaal als doel om het SCP zo kort mogelijk bij de proxy te krijgen. We zullen de methode van Ruspini[RKK97, SIG99] bespreken, deze methode is zeer goed afgelijnd en duidelijk. Met deze methode is het ook mogelijk om force shading, textures en statische of kleverige en dynamische wrijving te realiseren.

8.2.3 Het updaten van het SCP

Er zijn twee mogelijkheden wanneer er bewogen wordt met een haptic device, ofwel komen we in contact met een object ofwel niet. In het geval dat we geen object raken wordt de positie van het SCP gelijkgesteld aan deze van de proxy. Wanneer we een object raken moet de afstand tussen de proxy en het SCP geminimaliseerd worden. Elke tijdstap in de simulatie veranderen we de positie van het SCP zodat deze zo kort mogelijk bij de proxy komt te liggen. In elke tijdstap wordt dit proces opgesplitst in twee stages: een *move* stage en een *update* stage.

De move stage

In de move stage wordt gecontroleerd of de lineaire verplaatsing van het SCP naar de proxy geen botsing met primitieven van de omgeving oplevert. Door de hoge update rate van de haptic loop is het onmogelijk om alle primitieven te gaan controleren, dit zou veel te traag

gaan. Daarom worden allerlei technieken toegepast die het vinden van zulk een botsing versnellen; voorbeelden van en meer informatie over deze technieken zijn terug te vinden in hoofdstuk 6.

Indien de verplaatsing van het SCP naar de proxy geen botsing met een primitief van de omgeving oplevert, wordt deze verplaatst naar de positie van de proxy. Indien er wel een botsing is, wordt het SCP in de richting van de proxy bewogen totdat het SCP contact maakt met een of meerdere primitieven. Hierdoor is het niet meer mogelijk om rechtstreeks naar de proxy te bewegen, maar het kan wel zijn dat de afstand tot de proxy nog verminderd kan worden. In de update stage zal gekeken worden of dit nog effectief mogelijk is.

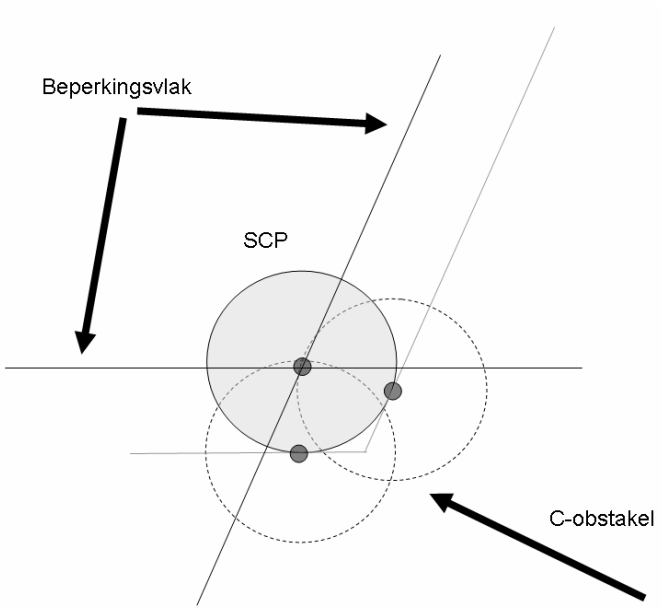
De update stage

Wanneer in de move stage geen botsing gevonden was, dient er in deze stage niets te gebeuren. Indien er wel een botsing is en het contactpunt berekend is, moet in deze stage gecontroleerd worden of het SCP nog korter bij de proxy kan komen door te bewegen over het oppervlak (of de oppervlakken) waarmee gebotst is (of zijn). Vanaf nu zal in deze bespreking het SCP bezien worden als het middelpunt van de bol, in plaats van een bol met een straal. Omdat we weten met welke oppervlakken het SCP gebotst is, kunnen we hieruit bepalen of het nog mogelijk is om korter bij de proxy te komen. De oppervlakken in contact met het SCP worden omgezet naar *configuration space obstacles* (C-obstacles of configuratieruimte obstakels). Deze bestaan uit alle punten binnen één SCP-straal van het originele oppervlak. Gebruik makend van deze informatie kan voor elke oppervlak een uniek beperkingsvlak berekend worden dat tangentieel is ten opzichte van het configuratieruimte oppervlak en dat door de positie van het SCP gaat, zie figuur 8.5. Zulk een beperkingsvlak zal de mogelijkheid van het SCP om te bewegen limiteren tot de halfruimte boven dit beperkingsvlak, aangezien de proxy altijd onder dit beperkingsvlak zal liggen.

Al deze halfruimtes samen definiëren een convexe veelhoek. Deze veelhoek stelt alle punten bereikbaar door een directe lineaire beweging vanuit de positie van het SCP voor. De nieuwe positie van het SCP is de positie in deze veelhoek die het kortst bij de proxy is. Dit kan analytisch neergeschreven worden als:

$$\begin{aligned} \text{minimaliseer } D(\mathbf{x}_{scp}) &= ||\mathbf{x}_{scp} - \mathbf{x}_p|| \text{ met} \\ \mathbf{n}_1^T \mathbf{x}_{scp} &\geqslant d_1 \\ \mathbf{n}_2^T \mathbf{x}_{scp} &\geqslant d_2 \\ &\vdots \\ \mathbf{n}_n^T \mathbf{x}_{scp} &\geqslant d_n \end{aligned}$$

waarbij $\mathbf{x}_{scp}$ de gezochte positie van het SCP is, $\mathbf{x}_p$ de positie van de proxy is en $\mathbf{n}$ de eenheidsnormaal is van een vlak. De n ongelijkheden zijn de beperkingsvlakken in contact met het SCP. Normaal gezien wordt er in 3D gewerkt. Hierdoor kan het SCP maar met maximaal drie unieke oppervlakken in botsing zijn. Bovenstaand systeem met ongelijkheden moet nu nog omgevormd worden naar een systeem met gelijkheden:



Figuur 8.5: Een voorbeeld van een SCP in botsing met twee oppervlakken

$$\begin{aligned} \text{minimaliseer } D(\mathbf{x}_{scp}) &= (\mathbf{x}_{scp} - \mathbf{x}_p)^T(\mathbf{x}_{scp} - \mathbf{x}_p) \text{ met} \\ \mathbf{N}^T\mathbf{x}_{scp} &= 0 \end{aligned}$$

Hierbij is N een matrix, $\mathrm{N} = [\mathbf{n}_1\ \mathbf{n}_2\ \mathbf{n}_3]$, die de normalen bevat van de drie beperkingsvlakken. Gebruik makende van Lagrange vermenigvuldigers $\lambda = [\lambda_1\ \lambda_2\ \lambda_3]$ wordt de Lagrangiaan:

$$L = \frac{1}{2}(\mathbf{x}_{scp} - \mathbf{x}_p)^T(\mathbf{x}_{scp} - \mathbf{x}_p) + \lambda^T(N^T\mathbf{x}_{scp})$$

Ruspini lost dit minimalisatieprobleem op in combinatie met Gilbert's algoritme dat snel de afstand tussen twee convexe polyhedra kan berekenen. Hier zullen we de meer algemene aanpak volgen die ook beschreven is in [ZS95, ML01]. We vinden de minimumafstand als de afgeleiden van L nul zijn.

$$\begin{aligned} \frac{\partial L}{\partial \mathbf{x}} &= 0 \\ \frac{\partial L}{\partial \lambda} &= 0 \end{aligned}$$

Als we nu deze afgeleiden berekenen krijgen we volgend systeem:

$$\begin{aligned} \mathbf{x}_{scp} + N\lambda &= \mathbf{x}_p \\ N^T\mathbf{x}_{scp} &= D^T \end{aligned}$$

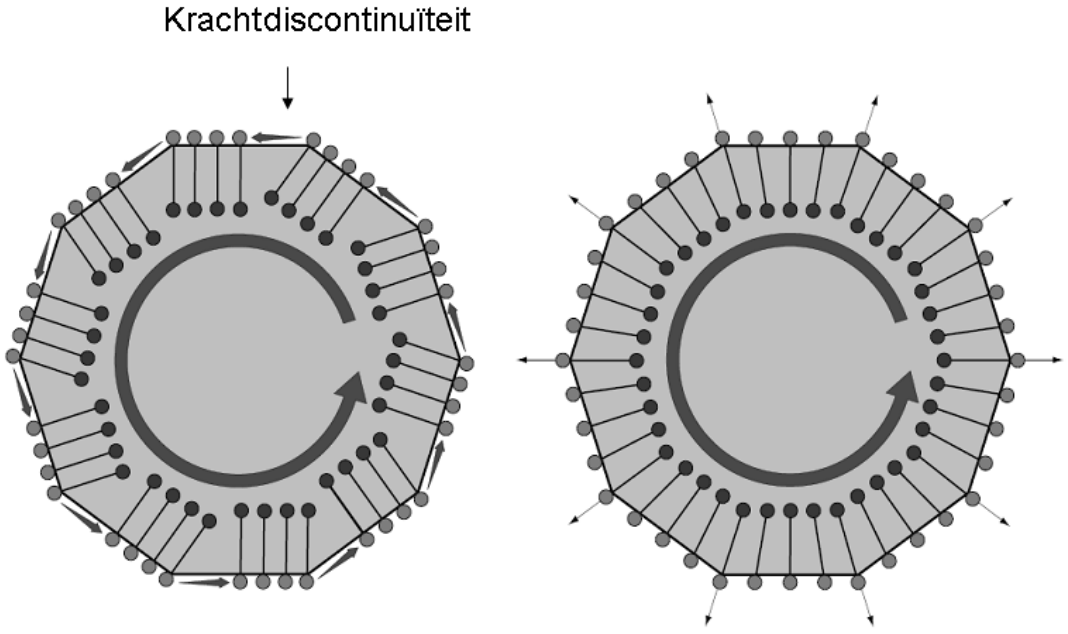
waarbij $D = -[d_1\ d_2\ d_3]$. De positie van het SCP kan nu berekend worden door een systeem van vergelijkingen op te lossen waarbij $\lambda = N^{-1}(\mathbf{x}_p - (N^T)^{-1}D^T)$:

$$\mathbf{x}_{scp} = \mathbf{x}_p - N\lambda$$

Het oplossen van dit lineair systeem kan zeer vlot aangezien N zeer makkelijk inverteerbaar is en zelfs uitrekenbaar is waardoor alleen nog maar de variabelen ingevuld moeten worden. Meer details zijn te vinden in [ZS95].

8.2.4 Force Shading

Een nadeel van veelhoekige modellen is dat deze geen continue oppervlakken hebben, waardoor tijdens het voelen een bol niet aanvoelt als bol, zie figuur 8.6. In de computer graphics wordt aan phong of gouraud shading gedaan om ervoor te zorgen dat een bol gemodelleerd door een veelhoekig model er daadwerkelijk uitziet als een bol en het dus niet mogelijk is om de primitieven te zien. Een gelijkaardige techniek kan gebruikt worden voor haptic rendering. In [MS96] worden de normalen van het veelhoekige model gebruikt om een richting te geven aan de kracht die berekend wordt. Een veelhoekig model kan een normaal per primitief of een normaal per vertex (hoekpunt van een primitief) hebben. Afhankelijk van welke normalen het veelhoekige model bezit, wordt er interpolatie gebruikt om de richting van de kracht te bepalen.



Figuur 8.6: Effect van force shading[RKK97]

Bovenstaande techniek werkt vrij goed zolang de topologie van het model bekend is. Dit is een probleem wanneer de omgeving objecten bevat die elkaar snijden, meer specifiek wordt er

geen rekening gehouden met de mogelijkheid dat er meerdere beperkingsvlakken zijn. Ruspini berekent ook de geïnterpoleerde normaal van het contactoppervlak, maar gebruikt deze om een nieuw beperkingsvlak te bepalen. Gebruik makend van dit beperkingsvlak wordt een nieuwe intermediaire positie berekend die zal dienen als een sub-proxy positie. Daarna zal de update stage uitgevoerd worden gebruik makende van de sub-proxy om de afstand te minimaliseren met het SCP. Voor meer details en problemen die deze techniek met zich meebrengt verwijzen we naar [SIG99].

8.2.5 Oppervlakte-eigenschappen van het model

Stel dat een gebruiker over een bepaald oppervlak wrijft, dan kan dit oppervlak zeer glad zijn of zeer ruw. Om deze effecten te modelleren moet de beweging van het SCP aangepast worden rekeninghoudend met de oppervlakte-eigenschappen van het model. De technieken om wrijving (statische, dynamische en kleverige) te simuleren, vallen buiten het bestek van deze thesis. Voor een mogelijke aanpak gebruik makende van constraint based methods, zie [SIG99]. Er wordt ook een aanpak voorgesteld om textures te haptic renderen, maar er moet nog veel onderzoek gedaan worden naar het haptic renderen van textures vanwege een groot aantal problemen.

8.3 Overzicht haptic rendering van andere modellen

Veelhoekige modellen hebben hun voor- en nadelen. Om sommige van deze nadelen te voorkomen, worden er ook andere modellen gebruikt voor haptic rendering. In deze sectie zullen we kort enkele andere mogelijke modellen overlopen. Voor een uitgebreide bespreking van andere soorten modellen verwijs ik de geïnteresseerde lezer naar [SIG99, Qua00]

Bij veelhoekige modellen kunnen we alleen de buitenkant van het model voelen, indien we ook de binnenkant willen kunnen voelen moeten we gebruik maken van volume-informatie. Hierbij wordt het volume beschreven als volume-elementen of voxels. Gebruik makend van deze volume-informatie kan bijvoorbeeld rekening gehouden worden met de transparantie van een object in de wereld.

Een andere vorm om objecten te representeren is door gebruik te maken van parametrische curves. Deze worden vaak gebruikt in CAD/CAM applicaties omdat parametrische curves het voordeel hebben van hoge orde continuïteit en exacte bepaling van de normalen. Het gebruik van dit soort modellen heeft minder gelijkenis met computer graphics en de haptic rendering technieken die eerder besproken werden, maar het is toch mogelijk om deze parametrische curves efficiënt te gebruiken.

Tijdens haptic rendering wordt er altijd een onderscheid gemaakt tussen objecten die vervormbaar zijn of objecten die niet vervormbaar zijn. Alle objecten die tot nu toe besproken werden, werden voorgesteld als niet-vervormbare objecten. Om vervormbare objecten haptic te renderen wordt er een physics loop geïntroduceerd in het haptic systeem waardoor er minder tijd over is voor de haptic loop. Daarom worden bij vervormbare objecten een aantal technieken toegepast die helpen bij het versnellen van de berekeningen. Voor een overzicht van een haptic systeem verwijzen we naar 8.1.

8.4 Haptic rendering voor vervormbare objecten

Bij cloth simulation hebben we te maken met een vervormbaar object, als we dit willen haptic renderen zijn er een aantal mogelijkheden. We kunnen gebruik maken van een computer graphics techniek om collision detection te doen, zie hoofdstuk 6. Gebruik makende van collision detection technieken bepalen we welke primitieven van het cloth moeten gecontroleerd worden voor een botsing met het SCP.

Er zijn ook andere mogelijkheden geïntroduceerd om haptic rendering toe te passen op een vervormbaar object. Deze mogelijkheden passen bijna allemaal de modellen aan uit de vorige secties, waarbij ervanuit werd gegaan dat de objecten niet-vervormbaar waren. Er zijn geen specifieke modellen voor vervormbare objecten. De technieken die toegepast worden op de modellen uit de vorige secties zullen in deze sectie besproken worden.

8.4.1 Intermediaire modellen

Stel dat we een object hebben van een bepaalde vorm die heel ingewikkeld is of net heel simpel door vervorming. We zouden dit object kunnen benaderen door allerlei andere simpelere modellen, zoals allemaal vlakken of bollen, zodat het vinden van een contactpunt sneller kan dan wanneer we dit proberen te vinden door gebruik te maken van het origineel object zelf. In deze sectie zullen we twee technieken geïntroduceerd door Mark et al.[MRF+96] en één techniek ontworpen door Davanne et al.[DMC02, MDH+03] bespreken. Welke van deze intermediaire modellen het beste is, hangt af van de applicatie en moet in de context daarvan dan ook gekozen worden.

Intermediaire modellen van Mark et al.

Mark et al. stellen twee intermediaire modellen voor: *plane and probe*[2] en *point-to-point springs*. De berekening van het intermediair model zal in de physics loop gebeuren en wordt dus losgekoppeld van de haptic loop.

Plane and probe: Zoals de naam al doet vermoeden wordt er in dit model gebruik gemaakt van vlakken. In de haptic loop zijn alleen vlakken gekend. Wanneer de probe zulk een vlak penetreert, wordt een kracht gegenereerd proportioneel aan de penetratiediepte, zoals bij de penalty based methods in sectie 8.2.1. Aan de hand van de positie van de probe wordt een benadering van het object berekend. Dus als de probe beweegt, wordt het vlak dat gepenetreerd wordt aangepast naar de veranderende vorm van het object dat het voorstelt. De berekening van het benaderingsvlak gebeurt tegen de update rate van de physics loop waardoor de berekeningen in de haptic loop veel meer tijd hebben en de nodige 1000Hz halen. Aangezien het vlak benaderd wordt tegen een snelheid van 20Hz kan bij een zeer scherpe kant van een object dit vlak plots helemaal omdraaien. Dit is een groot nadeel en kan alleen maar opgelost worden door damping te introduceren, waardoor de kracht geleidelijk overgaat naar het nieuwe benaderingsvlak.

Omdat het intermediaire model een vlak is, zal de interactie altijd voelen alsof we een glad oppervlak (bijvoorbeeld glas) voelen. Om dit te vermijden wordt er ook een

[2]De probe wordt in deze thesis proxy genoemd.

wrijvingsmodel voorgesteld dat statische en kinetische componenten en zelfs simpele oppervlakte textures kan voorstellen. Voor meer details zie [MRF+96].

Het gebruik van deze techniek brengt wat problemen met zich mee. Stel dat we met deze techniek de binnenkant en linkeronderhoek van een doos willen voelen, daarvoor zijn er meerdere vlakken nodig als intermediair model, hier dient dan ook rekening mee gehouden te worden. Een ander probleem, minder groot weliswaar, doet zich voor wanneer meerdere probes in de virtuele omgeving actief zijn. Een voor de hand liggende oplossing is dat elke probe zijn eigen intermediair model heeft.

Point-to-point springs: Het plane and probe model wordt gebruikt om een bepaald object te voelen. Indien we nu in een simulatie bepaalde objecten willen selecteren en verslepen wordt een ander intermediair model gebruikt, point-to-point springs. Er wordt een veer opgesteld tussen het object en het haptic device zodat er een soort van verbinding is tussen de haptic loop en de physics loop. De gebruiker zal dus tegen 1000Hz het object verslepen en bijvoorbeeld voelen hoe zwaar dit is, terwijl de physics loop op een tragere update rate merkt dat de gebruiker het object versleept. Om ook torsie te kunnen simuleren kunnen er meerdere veren gebruikt worden.

Intermediair model van Davanne et al.

Davanne et al. hebben ervoor gekozen om objecten te benaderen door bollen. Er bestaan een aantal goede algoritmes om objecten te benaderen door bollen. Bijvoorbeeld het omvormen van polyhedra in blobs, maar deze techniek kan alleen maar gebruikt worden met objecten die niet vervormen en is dus niet van toepassing hier. Een andere methode bestaat erin om de verschillende parametrisaties van bollen voor het object te gebruiken. Op deze manier kunnen vervormbare objecten ook gebruikt worden, waarbij zelfs de dichtheid van bollen bepaald kan worden om de nauwkeurigheid aan te passen[DMC02].

In het geval van een vervormbaar object met een bepaald volume, vullen we dit volume dus op met bollen. Bij een cloth is deze methode echter niet toepasbaar. Bij zulk een soort vervormbaar object wordt het oppervlak bemonsterd met een voldoende aantal bollen [MDH+03].

Om gebruik te kunnen maken van deze techniek met bollen moet er nog een extra methode komen die gebruik maakt van dit intermediair model om te kijken of het SCP er mee in contact is. Het voordeel van een bol is dat deze alleen maar een positie en straal heeft; er is helemaal geen oriëntatie en deze hoeft dus gedurende de simulatie ook nooit aangepast te worden. De techniek die wordt gebruikt is voxelisatie. De bollen worden in een voxel grid gestoken. Dit is een uniform grid waarbij cellen voxels worden genoemd. Hierbij wordt er alleen maar gecontroleerd op bollen die in dezelfde voxel liggen. Indien de bollen behoren tot een vervormbaar object wordt er met een timestamp gewerkt zodat het voxel grid niet altijd volledig leeg moet gemaakt worden. Voor niet-vervormbare objecten die niet bewegen is dit niet nodig, deze objecten zullen toch niet bewegen en blijven dus altijd op dezelfde plaats in het voxel grid zitten. Gebruik makende van het voxel grid wordt gekeken of het SCP in contact staat met het intermediaire model van een object. Omdat het berekenen van zulk een voxel grid enige tijd vergt, wordt net als bij de plane and probe methode dit voxel grid ge-update tegen de update rate van de physics loop. Hierdoor krijgen we een gelijkaardig probleem als bij de plane and probe methode. Door deze update van het voxel grid kunnen

er krachtdiscontinuïteiten veroorzaakt worden indien het object dat de gebruiker voelt zich vervormt. Een oplossing hiervoor is om de overgang tussen twee voxel grids te verbeteren. De overgang tussen twee voxel grids kan verbeterd worden door een lineaire combinatie van het huidige en vorige grid. Meer details over deze techniek worden besproken in [DMC02].

Een soort gelijkaardige techniek wordt gebruikt in [MSB+04], om gebruik makende van een handschoen met krachtterugkoppeling de gebruiker vervormbare objecten te laten voelen. Hier wordt meteen met bollen gewerkt en is er geen benadering van een bestaand object, een toevoeging hiervan zou tot een zeer realistische simulatie kunnen leiden. Dit omdat de testpersonen een verschil in elasticiteit konden voelen tussen de objecten die ze aanraakten.

8.4.2 Lokaal Model

Een intermediair model zal een bepaald object vervangen door een simpele voorstelling waar sneller mee gewerkt kan worden. Bij het gebruik van een lokaal model wordt er aan de haptic loop slechts een deel van het object bekend gemaakt, meer precies het gedeelte waar er contact mee mogelijk is [ML01]. Het lokaal model wordt als een niet-vervormbaar object gebruikt in de haptic loop. Het updaten en berekenen van dit lokaal model gebeurt in de physics loop.

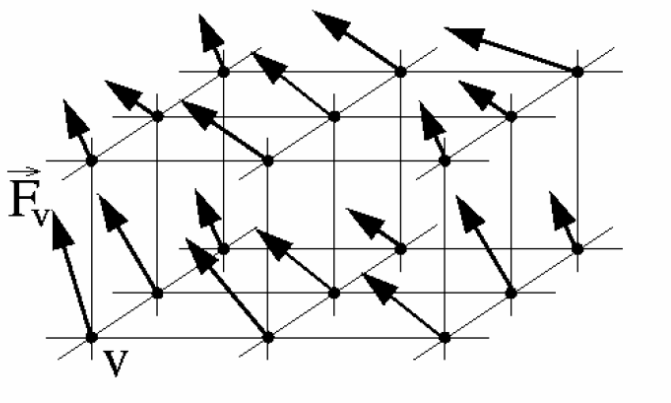
Voor het bekomen van dit lokaal model wordt er door Mendoza et al.[ML01] een virtuele balk aan de proxy vastgehecht. Met deze virtuele balk, die overigens niet zichtbaar is in de grafische simulatie, wordt dan gezocht naar het primitief dat deze balk snijdt. Deze primitief, samen met al zijn buren, vormt het lokaal model. Om te bepalen welke primitief deze virtuele balk snijdt, wordt een techniek uit OpenGL gebruikt. Het basisidee is om het *viewing volume* te gebruiken dat overeenkomt met de vorm van een virtuele balk. Daarna worden de objecten in de scene waarmee contact gemaakt kan worden gerenderd relatief tot de positie van de proxy. Deze positie van de proxy kan gezien worden als een camera die kijkt naar de scene, een viewing volume is meestal gedefinieerd ten opzichte van een camera. Als er niets zichtbaar is kan er geen botsing plaats vinden. Indien er wel iets zichtbaar is moet bepaald worden of we hiermee in botsing zijn. Voor dit basisidee te realiseren wordt het picking mechanisme van OpenGL gebruikt waarmee het mogelijk is om te zien welke primitieven zichtbaar zijn. Alle details om dit in OpenGL te implementeren zijn te vinden in [LCN99].

Doordat er gebruik gemaakt wordt van een virtuele balk kan er met het haptic device alleen bewogen worden in de richting van de balk. In het werk van Mendoza et al. wordt deze techniek dan ook gebruikt voor nabootsing van laparoscopie. In de toekomst zou deze virtuele balk vervangen worden door een bol, maar hierover is nog steeds niets te vinden in de literatuur. Het zou zeer interessant zijn moest het mogelijk zijn om deze techniek uit te breiden naar een bol. De snelheid van deze techniek is niet te onderschatten, de resultaten in [LCN99] geven op een heel oud systeem met oude grafische kaarten een berekentijd van 2.3 ms. Net als bij de vorige methodes wordt er gewerkt met een lagere update rate voor het model dat in de haptic loop gebruikt wordt. Desondanks wordt in [ML01] beweerd dat er geen krachtdiscontinuïteiten zijn, terwijl andere onderzoekers bij een gelijkaardige techniek (zoals besproken in de vorige secties) wel spreken van krachtdiscontinuïteiten.

8.4.3 Forcegrid

De forcegrid methode is een methode waarbij het forcegrid, een uniform grid, fungeert als een bufferstructuur. Deze methode kan gebruikt worden voor eender welke voorstelling van modellen, dus ook voor vervormbare objecten en werd geïntroduceerd door Mazella et al.[MML02].

In het forcegrid worden benaderende reactiekrachten opgeslagen die worden doorgegeven aan het haptic device. Deze krachten worden in het forcegrid gestopt tijdens de simulatie van de vervormbare objecten. Een forcegrid is gedefinieerd als een regulier grid waarbij elk punt van dit grid een kracht bevat, zie figuur 8.7. Als we de resolutie van het forcegrid verhogen zal de accuraatheid verhoogd worden. Gebruik makende van interpolatie wordt er dan een continu krachtveld verkregen. Het opvullen van het forcegrid kan op twee manieren. Ofwel via de silent methode die gebruikt maakt van de physics loop om de krachten te bekomen. Ofwel via de request methode die gebruik maakt van een andere processor om de kracht op een bepaalde positie te bekomen.



Figuur 8.7: Een Forcegrid[MML02]

In het begin van de simulatie bevat het forcegrid allemaal nul-krachten. Tijdens de simulatie wordt er door middel van een botsingsdetector gecontroleerd of er contact is met een object. Deze botsingsdetector geeft ook het contactpunt terug en de kracht die in het forcegrid gestopt moet worden. De botsingsdetector werkt afhankelijk van de gekozen methode tegen de snelheid van de physics loop of tegen de snelheid van de processor waar hij op draait. Hierdoor duurt het $\frac{1}{updatesnelheid}$ seconden voordat de gebruiker een kracht zal voelen indien er contact is.

Het gebruik van een forcegrid heeft gelijkaardige problemen als bij de plane and probe methode van Mark et al.[MRF+96]. Indien er meerdere haptic devices zijn moet er voor elk device een eigen forcegrid zijn. Afhankelijk van het aantal vrijheidsgraden van het haptic device moet het grid in evenveel dimensies zijn, bijvoorbeeld een 6DOF haptic device heeft een 6D grid.

8.4.4 Haptic Interpolation

De tot nu toe besproken technieken gebruiken allemaal een of ander benaderend model van het vervormbare object om het te haptic renderen. De techniek die we hier zullen bespreken kan bij al deze technieken toegevoegd worden of op zijn eentje gebruikt worden.

Zoals we al weten ondertussen moet de haptic loop ge-update worden tegen een snelheid van 1000Hz. Stel nu dat de berekeningen maar 200Hz halen, dan zouden we de resultaten van deze berekeningen kunnen interpoleren tussen t_n en t_{n+1} zodat we wel 1000Hz halen. Eender welk interpolatieschema zou voldoen, zelfs een lineair. Maar het grote probleem van deze techniek is dat we ons op tijdstip t_n bevinden en tijdstip t_{n+1} helemaal nog niet kennen. Zhuang et al.[ZC00] stellen voor om zowel de haptic loop als de graphics loop uit te stellen met één tijdstap. Hierdoor kennen we het vervormbaar object reeds op tijdstip t_{n+1} en kunnen we in de haptic loop wel interpolatie gebruiken. Het vertragen van één tijdstap zal geen probleem vormen, omdat één tijdstap slechts een aantal milliseconden duurt en deze vertraging valt in de tolerantie van de perceptie van de mens.

8.5 Conclusie

Nadat we in hoofdstuk 7 algemene aspecten van haptics besproken hebben, werd er in dit hoofdstuk besproken hoe we door gebruik te maken van een actief haptic device krachtterugkoppeling kunnen realiseren, haptic rendering genoemd. Om haptic rendering te kunnen plaatsen in de ontwikkeling van een haptic systeem werd er een overzicht gegeven van de algemene structuur van zulk een systeem.

De twee meest bekende technieken, Penalty en Constraint Based Methods, werden in detail behandeld. Deze technieken werden besproken in verband met veelhoekige modellen, maar kunnen even goed gebruikt worden bij andere soorten modellen. Een overzicht van andere mogelijke modellen werd gegeven.

Gebruik makende van allerlei modellen werd altijd verondersteld dat we werkten met objecten die niet vervormden, maar het gebruik van vervormbare objecten wordt meer en meer populair. Deze modellen voor niet-vervormbare objecten werden met bepaalde technieken uitgebreid of aangepast om deze bruikbaar te maken voor vervormbare objecten. Een aantal van deze technieken werden ook besproken.

In de voorbij twee delen van deze tekst hebben we talloze technieken besproken en vergeleken voor cloth simulation en haptics. Op basis van deze besprekingen heb ik een aantal technieken geïmplementeerd en vergeleken. Hiervan geef ik een overzicht in het volgende deel van deze tekst.

Deel III

Implementation

Hoofdstuk 9

Haptic Cloth Rendering

In de voorbije delen (deel I en II) werden technieken besproken voor cloth simulation en het creëren van een haptische applicatie. Naast een literatuurstudie was het doel om aan haptic rendering te kunnen doen van een cloth simulation of in één term: Haptic Cloth Rendering. In dit hoofdstuk zullen we de implementatie, die gemaakt is in C++ en OpenGL, bespreken. Deze maakt gebruik van de technieken eerder besproken in deze tekst. Bijna alle implementatiekeuzes werden gemaakt rekening houdend met de vereiste update rate die haptics met zich meebrengt (zie sectie 8.1).

In het eerste stuk van dit hoofdstuk zullen we de cloth simulator die geïmplementeerd is, gebruik makende van technieken uit het eerste deel van deze thesistekst, bespreken. Daarna zullen we de integratie van deze cloth simulator in HAL[1], de haptics library van het EDM[2], bespreken.

9.1 Cloth Simulator

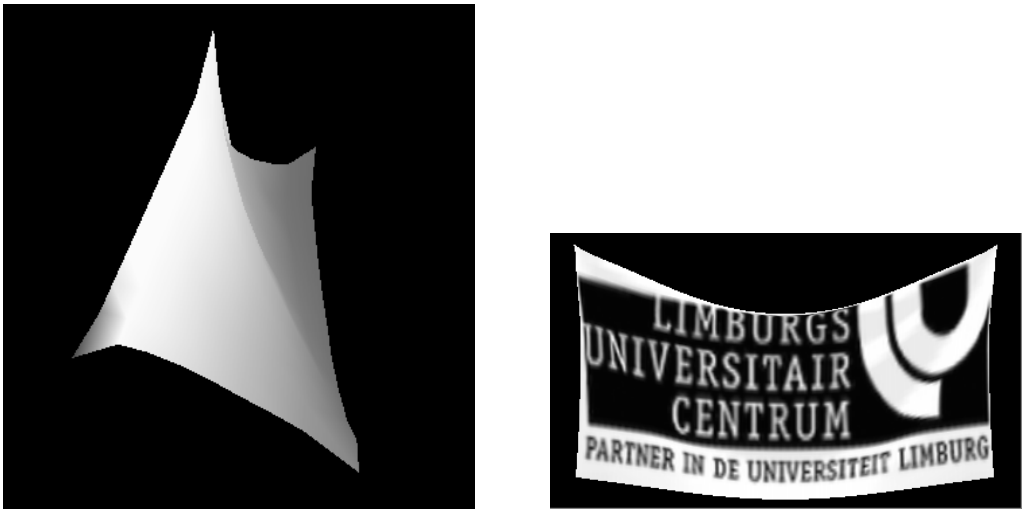
Bij het implementeren van een cloth simulator komen er een groot aantal technieken aan bod. In deze sectie zullen we gradueel de geïmplementeerde technieken overlopen.

We maken ten eerste gebruik van een mass spring systeem zoals besproken in hoofdstuk 2. De volgende krachten die op particles kunnen inwerken zijn geïmplementeerd: zwaartekracht, viscous drag en lineair gedempte veren.

Het is mogelijk om tijdens het creëren van een cloth een aantal eigenschappen te kiezen. Zo kunnen de breedte en de hoogte gekozen worden, het cloth is immers altijd een vierhoek, en de spreiding van het aantal particles over deze breedte en hoogte. Bijvoorbeeld een cloth van 4 breed en 5 hoog met 15 particles in de breedte en 20 in de hoogte. Dit geeft een cloth bestaande uit 300 particles. Om de massa van de particles te kunnen kiezen is het mogelijk om de massa van het doek te kiezen. We kunnen bijvoorbeeld 200 kiezen, dan zal bij 300 particles elke particle een massa hebben van 1.5.

[1] `http://www.edm.luc.ac.be/software/hal/`

[2] Expertise Centre for Digital Media (EDM), een onderzoeksinstituut van het Limburgs Universitair Centrum (LUC)



(a) Smooth shading van een cloth (b) Een cloth met een texture

Figuur 9.1: De twee mogelijke opties van de cloth simulator

Om tot een realistische cloth simulation te komen moeten de veren van een mass spring systeem op een bepaalde manier verbonden worden. Een ander belangrijk aspect is het grafisch weergeven van een cloth, in de volgende sectie zullen we dus de gekozen topologie bespreken.

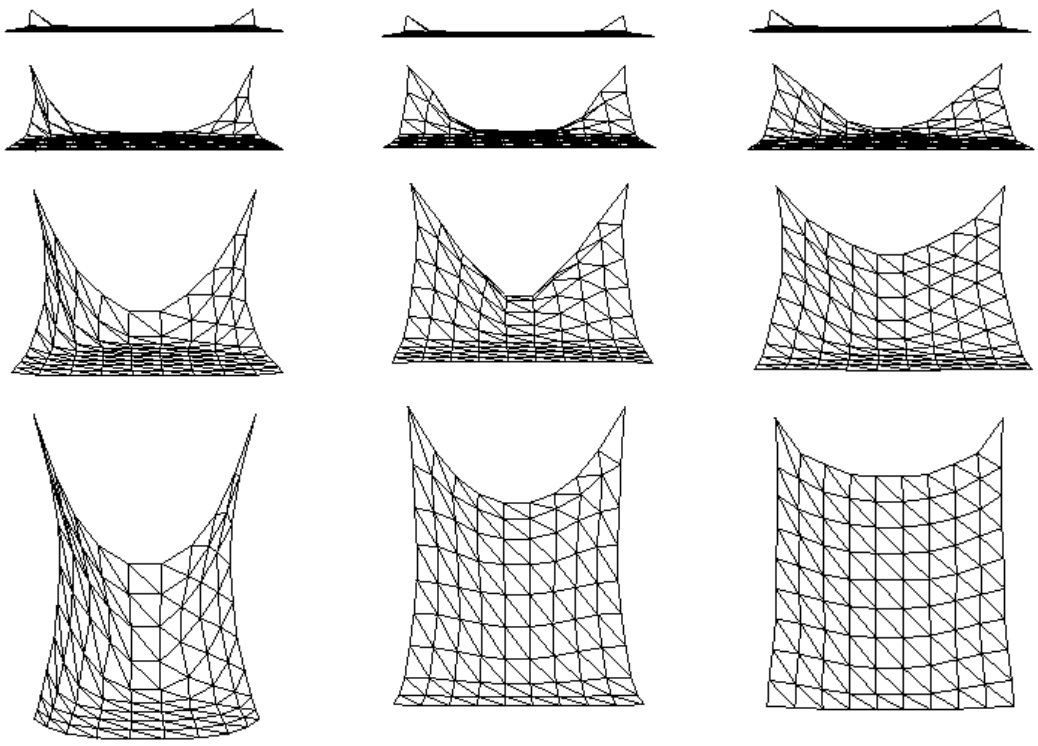
9.1.1 Topologie

Tijdens de simulatie van een cloth willen we het cloth natuurlijk kunnen zien. In mijn implementatie is het mogelijk om van het mass spring systeem dat een cloth voorstelt een driehoekige mesh te maken. Het oppervlak van een cloth heeft een vierhoek als vorm. Op deze manier is het niet mogelijk om arbitraire vormen te creëren. Maar het heeft als voordeel dat de gekozen manier om veren te verbinden voor minder problemen zorgt.

Deze driehoekige mesh wordt gerenderd met behulp van OpenGL. Tijdens elke simulatiestap worden de normalen opnieuw berekend voor elke driehoek en vertex van de mesh. Deze normalen zorgen ervoor dat er collision detection kan gedaan worden en hebben als voordeel dat er smooth shading mogelijk is. Dit zorgt ervoor dat de individuele driehoeken van de mesh niet zichtbaar zijn, waardoor het cloth het uitzicht van een glad oppervlak heeft (figuur 9.1(a)). Omdat er gewerkt wordt met een 2D driehoekige mesh is het ook makkelijk om textures te plakken op het cloth. De texturecoördinaten worden tijdens het creëren van de driehoekige mesh berekend en kunnen tijdens de simulatie gebruikt worden. Een cloth met een texture is te zien in figuur 9.1(b).

De manier waarop de veren verbonden worden is deze uit sectie 3.2.1. Hierbij wordt er gewerkt met drie type veren: structurele, afschuif- en buigveren. Deze techniek is tegenwoordig nog altijd de meest gebruikte techniek. Ze is simpeler en ook efficiënter dan de techniek geïntroduceerd door Choi et al. die werd uitgelegd in sectie 3.2.2. Efficiëntie is in mijn implementatie van zeer groot belang vanwege de beperkingen die worden opgelegd in haptische applicaties (sectie 8.1).

Deze veren worden verbonden met de particles van het doek waarbij meestal de buren of overburen verbonden worden. Wanneer het oppervlak wordt voorgesteld als een vierhoek, zijn er geen problemen, alleen de particles aan de buitenkant zijn verbonden met minder veren. Bij arbitraire vormen kunnen er allerlei uitzonderingen voorkomen waardoor de keuze voor een vierhoek voor de hand ligt. De drie type veren werden één voor één geïntroduceerd waarbij elke toevoeging van een type voor meer realisme zorgde. In figuur 9.2 zien we duidelijk dat het toevoegen van afschuif- en buigveren voor extra realisme zorgt. In deze figuur zien we het cloth ook op verschillende tijdstippen. Om te zorgen dat het cloth reageert op externe en interne krachten moet de toestand ervan geïntegreerd worden. De integratietechnieken die we gekozen hebben om te implementeren worden in de volgende sectie besproken.



(a) Structurele veren

(b) Structurele en afschuifveren

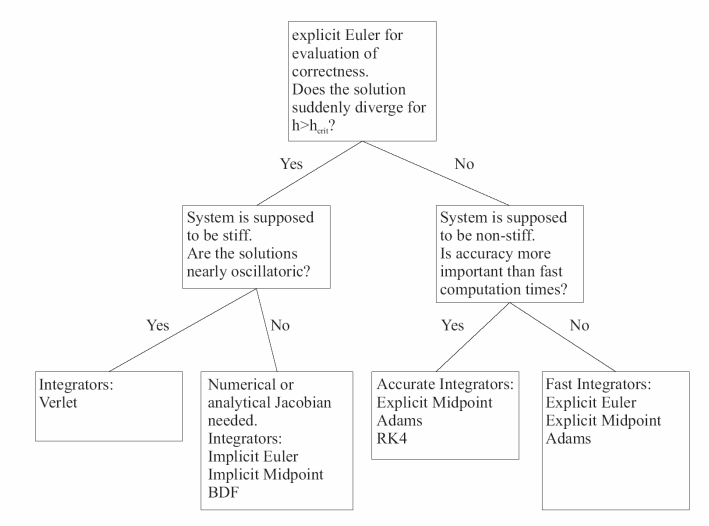
(c) Structurele, afschuif- en buigveren

Figuur 9.2: De verschillen die verkregen worden door niet alle veren te gebruiken bij een cloth. Er worden snapshots genomen op de 50ste, 100ste, 150ste en 200ste iteratie.

9.1.2 Integratie

Er bestaan een heel groot aantal integratietechnieken. Allemaal hebben ze hun voor- en nadelen, waardoor het kiezen van de juiste technieken voor de juiste applicatie dan ook niet simpel is. Meestal bestaan er ook meerdere goede keuzes. Ik heb mij laten leiden door een diagram van Hauth[Hau03] in 9.3. Rekening houdend met het feit dat we een zo snel mogelijke simulatie beogen die een realistische weergave geeft, lijkt het evident dat we kiezen voor expliciete integratietechnieken. Het nadeel van deze technieken is dat ze niet goed werken

als er een stijf systeem geïntegreerd moet worden. Om een realistische weergave te krijgen waarbij de massa van het cloth ook realistisch blijft, moeten de veerconstanten van een cloth vrij hoog zijn, wat impliceert dat we met een stijf systeem werken. In het diagram in 9.3 wordt bij een stijf systeem de vraag gesteld of de oplossingen bijna oscillatorisch zijn. Dit betekent dat de simulatie de oscillaties van het object zeer goed volgt. Bij cloth simulation kan dit een probleem geven: het kan zijn dat het cloth heel *blubberig* lijkt[Mel02].

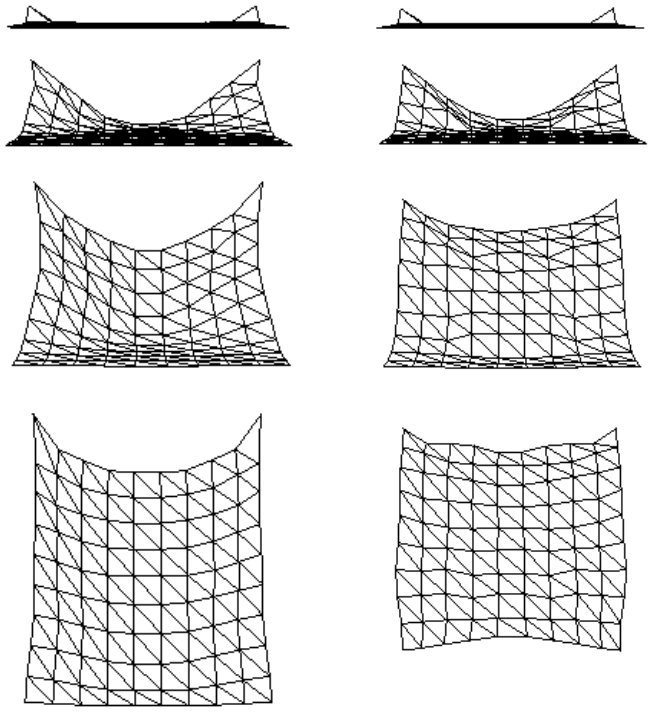


Figuur 9.3: Het kiezen van een integratietechniek[Hau03]

Met het diagram in ons achterhoofd is er gekozen voor het implementeren van de drie expliciete technieken: Euler, Midpoint en Runge-Kutta van orde 4. Om deze te implementeren werd de structuur van Witkin[Wit01b] gevolgd, die gebruik maakt van een toestandsvector. Voor een bespreking van deze drie technieken en de toestandsvector, zie secties 4.3 en 4.2. Naast deze drie expliciete technieken werden ook de drie Verlet schema's van sectie 4.6 geïmplementeerd. Het is ook mogelijk om voor deze schema's gebruik te maken van een toestandsvector; deze moet dan wel anders gedefinieerd en opgevuld worden dan bij de expliciete methodes, maar is even efficiënt. Door in mijn implementatie gebruik te maken van andere schema's dan basic Verlet, krijgen we resultaten waarbij er geen last is van het eerder vermeldde probleem, blubberigheid.

Desondanks het feit dat Verlet integratietechnieken geschikt zijn voor stijve systemen, zijn ze toch niet even stabiel als impliciete integratietechnieken. De meeste impliciete technieken zijn onvoorwaardelijk stabiel en kunnen dus nooit ontploffen of divergeren; voor een bespreking van andere mogelijk problemen zie 4.3.1. Om deze reden is het ook interessant om inverse dynamica (sectie 4.7) te implementeren en te testen. Door gebruik te maken van inverse dynamica is het mogelijk om met niet-stijve veren een simulatie te verkrijgen die toch lijkt alsof stijve veren gebruikt worden. De techniek geïntroduceerd door Provot is geïmplementeerd, hierbij worden de posities van de particles rechtstreeks aangepast na elke integratiestap. Voor een vergelijking tussen een cloth waarop inverse dynamica wordt toegepast en een cloth zonder, zie figuur 9.4. Op deze figuur is duidelijk te zien wat inverse dynamica doet met een cloth: de veren mogen maximaal 15% uitrekken en hierdoor zal het doek minder gevoelig lijken aan zwaartekracht en dus langzamer zal lijken te vallen. In figuur 9.4 bevat het doek ook

slechts weinig particles zodat duidelijk zichtbaar is hoe inverse dynamica te werk gaat. In de sectie over inverse dynamica in hoofdstuk 4 is een cloth met meer particles te zien in figuur 4.4. Bij het gebruik van een expliciete Euler integratie kan deze methode wel zijn voordeel bewijzen, maar ten opzichte van Verlet integratietechnieken waarbij stijvere veren gebruikt worden, geeft deze techniek geen voordeel meer door de extra rekenkost die inverse dynamica met zich meebrengt.



(a) Cloth zonder inverse dynamica

(b) Cloth met inverse dynamica

Figuur 9.4: Vergelijking tussen een cloth zonder en met inverse dynamica. Er worden snapshots genomen op de 50ste, 100ste, 150ste en 200ste iteratie.

De volgende technieken die werden geïntroduceerd in deel I zijn beperkingen (hoofdstuk 5) en collision detection (hoofdstuk 6). De technieken over beperkingen zullen we nog bij de cloth simulator rekenen, de technieken over collision detection zullen we bespreken in de context van haptics.

9.1.3 Beperkingen

Er zijn een heel aantal beperkingen mogelijk in cloth simulation. In hoofdstuk 5 werd één algemene techniek besproken waarbij de bewegingsvrijheden van de particles beperkt worden. Twee algoritmes van deze techniek werden geïmplementeerd in de cloth simulator:

Fixed: Het is mogelijk om een particle vast te pinnen, deze kan dan niet meer bewegen. Door de inverse massa op nul te zetten, samen met de snelheid, zullen de krachten voor deze

particle nul blijven tijdens elke integratiestap. Ook de snelheid blijft nul, waardoor de particle dus niet van zijn plaats zal bewegen. Indien een particle geselecteerd is kan deze vastgepind worden of net losgemaakt worden door op een knop op het toetsenbord te duwen.

Om bepaalde particles vast te kunnen pinnen bij het begin van de simulatie is het mogelijk om bepaalde particles te selecteren via een aantal voorgedefinieerde keuzes. De mogelijke keuzes zijn de vier hoekparticles, de randen van het cloth als volledige rij (bijvoorbeeld de bovenste rij) of alleen de hoeken van een rand (bijvoorbeeld de hoekparticles van de onderste rij).

Een punt volgen: Om enige vorm van interactie met de gebruiker te voorzien werd de mogelijkheid om een particle vast te nemen met de muis geïmplementeerd[3]. Hiervoor wordt simpelweg de particle in de toestand fixed gezet en wordt de positie aangepast aan deze van de muis.

In de figuren eerder in dit hoofdstuk is duidelijk te zien dat bepaalde hoekparticles vastgepind zijn of één van de andere mogelijkheden gebruikt werd. Op deze manier blijft het cloth op een vaste plaats in de wereld hangen zodat het mogelijk is om het cloth te gebruiken voor haptics. Alle delen die van toepassing zijn op enkel en alleen de cloth simulator werden behandeld. In de volgende sectie zullen we bespreken hoe we deze cloth simulator hebben geïntegreerd in de haptics library van het EDM, HAL.

9.2 Haptic Rendering

We zullen in deze sectie eerst een kort overzicht geven van de structuur van HAL. Daarna zullen we bespreken hoe we de cloth simulator, besproken in vorige sectie, in HAL geïntegreerd hebben. Tenslotte worden er nog technieken besproken om het cloth haptic te kunnen renderen.

9.2.1 Overzicht van HAL

HAL is een haptic library die opgebouwd is uit zes andere libraries:

- Core:
  De kerncomponenten van HAL zitten in deze library (Vector, Matrix, HapticWorld,...).
  De andere libraries zullen deze componenten gebruiken.
- Scenegraphs:
  Verschillende scenegraph implementaties worden in deze library geïmplementeerd. Op dit moment is er slechts een simpele scenegraph aanwezig die alleen maar kijkt naar het eerst geraakte object.
- Objects:
  In deze library worden haptische objecten geïmplementeerd zoals een bol of een polymesh.

[3]Deze feature zal ook nog terugkomen bij het haptics gedeelte, zie sectie 9.2.2.

- Forces:
  In de Forces library zitten allerlei forcefields, zoals een drag forcefield.
- PHANToM:
  Deze library is een driver voor het PHANToM device en is gebaseerd op de GHOST I/O driver.
- Pseudo:
  Een pseudo haptic device is geïmplementeerd in deze library en kan gebruikt worden om te testen.

Om een applicatie met HAL te bouwen moet een HapticWorld gecreëerd worden. Deze HapticWorld kan meerdere scenegraphs en devices bevatten. Slechts één scenegraph is actief, deze wordt dan ook gebruikt in de Haptic Loop. Er zijn twee mogelijkheden, ofwel heeft het haptic device zelf een haptic loop die de haptic loop van de HapticWorld oproept, bij de PHANToM is dit het geval. Ofwel wordt de HapticWorld gestart en zal de haptic loop gestart worden in een thread. HAL bevat geen mogelijkheden tot grafische feedback, er is alleen maar mogelijkheid tot haptics.

9.2.2 Integratie in HAL

Voor de integratie van de cloth simulator is er een extra library toegevoegd aan de reeds bestaande libraries. Deze heeft de naam *Physics* gekregen en implementeert alle algoritmes besproken in de vorige sectie 9.1. Het enige verschil met de vorige cloth simulator is dat er een physics loop wordt geïntroduceerd. Dit is een thread die alleen maar integratiestappen van de cloth simulation verwerkt en dan een tijdje slaapt. Eigenlijk wordt een functie opgeroepen waardoor de physics thread stopt de CPU te gebruiken en een andere thread zijn werk laat doen. De ideale update rate is 30Hz, als deze niet gehaald wordt, wordt er gewoon trager ge-update zonder enige gevolgen. De ideale update rate wordt gelijkgesteld aan de update rate van de grafische loop zodat de bewegingen van het cloth niet haperend overkomen.

Gebruik makend van deze library kan dus een cloth gecreëerd worden. Zoals reeds eerder besproken is er gekozen voor een driehoekige mesh als topologie[4]. In de objects library werd de PolyMesh omgevormd naar een klasse die bruikbaar is voor een cloth. Het grote verschil met de originele PolyMesh is dat de vertices niet gewoon een vector bevatten, maar een pointer naar de particle die ze voorstellen. Op deze manier past de driehoekige mesh zich automatisch mee aan aan de vorm van het cloth zonder opnieuw de mesh te moeten construeren. Deze nieuwe klasse die in combinatie met mass spring systemen gebruikt moet worden, heet TriangleMesh. Er zijn nog twee andere veranderingen ten opzichte van PolyMesh met betrekking tot het teruggeven van een kracht en de manier waarop de intersecties met de driehoeken gevonden worden. Bij PolyMesh is er de keuze om alle driehoeken te testen of om gebruik te maken van een octree (zie sectie 6.2.1). Beide technieken zijn niet bruikbaar in combinatie met cloth simulation. Het testen van alle driehoeken gaat veel te traag, zeker in combinatie met een physics loop. De octree voorziet geen update mechanisme en het opbouwen van een octree duurt ook enkele seconden en is daarom dus ook niet bruikbaar.

[4]Merk op dat HAL geen mogelijkheid voorziet tot grafische feedback; een externe applicatie zal HAL gebruiken en dus ook deze topologie moeten renderen. In sectie 9.2.3 zal deze externe applicatie besproken worden

Voor het vinden van de driehoeken die moeten getest worden op intersecties zijn er drie algoritmes geïmplementeerd. Al deze algoritmes zijn uitgebreid besproken in hoofdstuk 6, over collision detection. We zullen deze drie algoritmes overlopen en de designkeuzes bespreken. Hierna zullen we deze drie algoritmes vergelijken tijdens hun gebruik in haptic rendering. Als laatste zullen we bespreken hoe we het mogelijk hebben gemaakt om het doek vast te nemen met de PHANToM door gebruik te maken van de stylusbutton en de manier waarop er force shading geïmplementeerd is.

Bounding Volume Hierarchies

We hebben een bounding volume hiërarchie geïmplementeerd waarbij een update mechanisme wordt gebruikt tijdens de simulatie. In de vorige sectie werd gezegd dat er drie algoritmes geïmplementeerd zijn, maar eigenlijk zijn het twee algoritmes waarbij dit eerste twee variaties heeft. Er is één bounding volume hiërarchie geïmplementeerd, maar er is gekozen om twee bounding volumes te gebruiken: een bol en een AABB.

Bij het opbouwen van de BVH wordt de verzameling van driehoeken opgesplitst in twee deelverzamelingen. Om te bepalen welke driehoek in welke deelverzameling hoort wordt er een AABB berekend rond de verzameling driehoeken. Daarna wordt de langste as van deze AABB gekozen en gebruik makend van die as wordt door middel van de mediaan de verzameling in twee deelverzamelingen gesplitst. Hierdoor krijgen we een gebalanceerde boom wat voordelen kan geven tijdens het updaten van de BVH.

Nadat de twee deelverzamelingen bepaald zijn en recursief opgebouwd, moet het bounding volume rond deze twee deelverzamelingen nog berekend worden. Voor de AABB-methode wordt gewoon een zeer snelle benadering berekend door de linker- en rechterdeelverzameling met hun AABB te omringen. Bij de bol-methode wordt de snelle benadering van Quinlan gebruikt die besproken werd in sectie 6.1.1.

Het is ook mogelijk om te kiezen voor een opdelingsfactor van vier, waarbij de opdeling eerst gebeurt over de langste as en dan over de tweede langste as zodat we een verdeling in vier krijgen. Dit algoritme zou sneller kunnen werken bij vervormbare objecten omdat er minder recursie nodig is tijdens het updaten, het enige nadeel is dat er meer intersectietesten zijn tussen bounding volumes en de proxy.

Tijdens de cloth simulation wordt de BVH na elke integratiestap in de physics loop ge-update. Het updaten gebeurt volgens het algoritme van Brown et al. met de aanpassing die ik zelf voorgesteld heb in sectie 6.1.3. Eerst had ik het algoritme geïmplementeerd waarbij alle parents van bladeren werden toegevoegd, maar toen kwam ik tot de ontdekking dat er iets niet klopte en maakte ik dus de aanpassingen reeds eerder beschreven. Hierdoor worden dus veel minder testen gedaan; als we uitrekenen dat een binaire boom van 3042 driehoeken 6083 knopen heeft, dan weten we dat er nu maximaal 6083 knopen ge-update moeten worden terwijl dit zonder mijn aanpassingen niet gegarandeerd kon worden. In tabel 9.1 wordt een overzicht gegeven van het aantal updates die nodig zijn bij een AABB BVH en een bol BVH waarbij het updaten gebeurt met mijn aanpassingen. Uit deze tabel kunnen we afleiden dat het aantal updates bij een AABB hoger ligt dan bij een bol. Dit is in zekere zin ook logisch omdat een AABB nauwer rond een driehoek past dan een bol wat als gevolg heeft dat een driehoek vaker

uit een AABB zal vervormen dan uit een bol[5]. Bij een BVH met een grotere opdelingsfactor zou dit verschil nog groter moeten zijn en als we de waardes vergelijken zien we dit ook. Bij de bol BVH worden er 38% minder updates gedaan bij een opdelingsfactor van vier in plaats van twee en bij de AABB BVH is dit 30%. Hieruit kunnen we concluderen dat het verschil in updates van de bol BVH en de AABB BVH groter wordt als we een opdelingsfactor van vier gebruiken. Uit tabel 9.1 kunnen we ook concluderen dat een bol BVH sneller update dan een AABB BVH en dat een grotere opdelingsfactor zorgt voor een snellere update. Maar dit komt niet omdat bij een grotere opdelingsfactor er minder recursie is, maar omdat er minder updates gedaan worden. In mijn implementatie wordt er zelfs geen recursie gebruikt tijdens het updaten. De manier waarop recursie volledig vermeden wordt zal duidelijk worden in de volgende paragraaf. Deze testen zijn gedaan op een AMD 1.2 GHz met 512MB RAM, alle testen die nog zullen volgen zijn ook op deze PC gebeurd.

	binair				quartair			
	bol		AABB		bol		AABB	
#driehoeken	#u	ms	#u	ms	#u	ms	#u	ms
162	263	0,307	323	0,408	190	0,285	247	0,378
722	1191	1,759	1443	2,429	832	1,1531	1063	2,257
1682	2506	4,680	3355	7,429	1887	4,442	2675	7,216
3042	3185	7,211	5341	13,569	2162	6,360	3865	11,205
4802	6433	13,650	9401	22,686	4284	12,907	6727	20,729

Tabel 9.1: Vergelijking van updaten BVH tussen bol en AABB over 2000 iteraties. #u stelt het aantal updates van knopen voor en ms het aantal milliseconden dat één update van de BVH duurde.

Om recursie volledig te verwijderen in het update algoritme moet tijdens het bepalen van de vervormde driehoeken geen top-down techniek gebruikt worden. Deze techniek doorloopt de BVH en test of een driehoek uit het bounding volume is vervormd. In plaats daarvan wordt er een array bijgehouden van alle bladeren. Deze array wordt dan overlopen en voor elk blad wordt gecontroleerd of de driehoek uit zijn bounding volume is vervormd. Indien dat zo is, wordt deze aan de priority queue toegevoegd. Door op deze manier de bladeren toe te voegen aan de priority queue moet de BVH niet top-down doorlopen worden en wordt kostelijke recursie vermeden[Ber97]. In tabel 9.2 kunnen we duidelijk zien dat het gebruik van een array zorgt voor een versnelling. Ook kunnen we opmerken dat een BVH met een hogere opdelingsfactor sneller zal updaten vanwege minder recursie, maar zoals verwacht kan worden, is deze versnelling niet snel genoeg voor een algoritme zonder recursie.

Om een BVH te gebruiken voor haptics worden de bounding volumes getest op intersectie met de proxy in plaats van met een andere BVH. We zullen op het einde van deze sectie beide algoritmes testen gebruik makend van haptics en vergelijken met het derde algoritme wat geïmplementeerd is, wat we in de volgende sectie bespreken.

[5]Dit hangt natuurlijk af van de manier waarop het cloth vervormt, maar uitgaande van een normale vervorming is dit een logisch resultaat.

#bladeren	array	binair top-down	quartair top-down
162	0,116	0,153	0,141
722	0,652	0,922	0,799
1682	1,722	2,2277	1,990
3042	2,871	3,914	3,346
4802	4,537	5,40	5,15

Tabel 9.2: Vergelijking van top-down techniek en array techniek om recursie te vermijden tijdens het updaten van een BVH. De tijden zijn in milliseconden.

Optimized Spatial Hashing

Teschner et al.[THM+03] gebruiken spatial hashing voor collision te detecteren tussen twee vervormbare objecten. In mijn implementatie zal ik dit algoritme aanpassen zodat het gebruikt kan worden bij haptics. De parameters waarbij het algoritme het beste werkt, zoals de hashfunctie en de grootte van een gridcel zijn volledig analoog aan degene beschreven in 6.2.2. Op het einde van deze sectie zal getest worden of de manier waarop hun algoritme gebruikt wordt beter is dan een BVH.

Het origineel algoritme is zeer efficiënt in het vinden van self-collisions, maar we zijn enkel geïnteresseerd in een collision met de proxy. Om deze reden hashen we enkel de driehoeken in plaats van ook de vertices van de driehoeken te hashen. Hierdoor zijn er ook geen twee stages meer in het algoritme. Tijdens het hashen van de vertices worden de AABB's van de driehoeken berekend. Maar aangezien de vertices niet meer gehashed worden, wordt de AABB berekend en de driehoek tegelijkertijd gehashed. Tijdens elke update van de physics loop worden de driehoeken van het cloth, na de integratiestap, gehashed. In tabel 9.3 zien we hoelang het duurt om de hashtabel te creëren of te updaten voor een bepaald aantal driehoeken. Als we deze resultaten vergelijken met de update tijden voor een BVH merken we dat een BVH sneller update dan een hashtabel. Dit hoeft niet te betekenen dat dit laatste algoritme nu slechter is dan een BVH; misschien wordt de grotere update tijd gecompenseerd met de tijdsduur van het vinden van een collision. De bekomen resultaten komen overeen met deze van Teschner et al., indien we rekening houden met de aanpassingen.

#driehoeken	tijdsduur
162	1,368
722	6,071
1682	13,185
3042	23,760
4802	39,998

Tabel 9.3: De snelheid waarmee de hashtabel gecreëerd of ge-update kan worden. De tijden zijn in milliseconden.

Voor het detecteren van collisions met de proxy, wordt deze gehashed en worden de driehoeken die zich op dezelfde hashwaarde bevinden getest of hun AABB intersecteert met de proxy. Als de proxy in de AABB ligt voegen we deze toe aan de lijst van driehoeken om te testen. Deze extra test spaart veel tijd uit. Als er hash collisions zijn kan het zijn dat een aantal onnodige driehoeken wordt getest en aangezien de intersectie met een driehoek bepalen een kostelijk

proces is, is dit een goede oplossing. In de volgende sectie zullen we de techniek bespreken die gebruikt wordt voor haptic rendering, alsook de voorgestelde collision detection algoritmes evalueren.

Haptic Rendering

In de vorige secties hebben we besproken hoe we efficiënt kunnen bepalen welke driehoeken getest moeten worden op intersectie met de proxy. We hebben die algoritmes ook getest op de snelheid waarmee ze ge-update worden in de physics loop. In deze sectie zullen we kort bespreken welke techniek er wordt gebruikt voor te haptic renderen. Hierna zullen we de drie algoritmes vergelijken.

Het algoritme dat geïmplementeerd is voor haptic rendering is gebaseerd op de Constraint Based Methods besproken in 8.2.2. Het originele algoritme dat ik tijdens mijn stage geïmplementeerd heb, werkte niet onder bepaalde omstandigheden. Na deze omstandigheden te evalueren zag ik in dat er aanpassingen nodig waren; een overzicht van het algoritme dat nu gebruikt wordt, volgt in de volgende paragraaf.

In een eerste iteratie wordt er gezocht naar een driehoek waarmee gebotst wordt. Dit wordt gedaan door te kijken of de proxy en het SCP aan verschillende zijden van het vlak dat de driehoek vormt liggen. Daarna wordt gecontroleerd of het lijnstuk tussen de proxy en het SCP de driehoek in kwestie snijdt. Als we dus één of meerdere driehoeken gevonden hebben, nemen we degene waarbij het botsingspunt het kortste bij de proxy ligt. Als laatst wordt in deze iteratie de ideale nieuwe SCP positie bepaald, deze is een projectie op het vlak van de driehoek waarmee gebotst wordt. In de volgende iteratie moeten we controleren of we deze ideale SCP positie kunnen bereiken zonder tegen een andere driehoek te botsen. Dit doen we door een lijnstuk te trekken tussen het botsingspunt en de ideale SCP positie. Indien er geen botsing is, hebben we onze nieuwe SCP positie gevonden. Indien er wel een botsing is, dan moeten we een nieuwe ideale SCP positie bepalen. De nieuwe ideale SCP positie is de projectie van de proxy op de lijn gevormd door het crossproduct van de twee normalen van de driehoeken waarmee we botsen. Dit is de nieuwe ideale SCP positie, omdat deze situatie zich alleen maar voordoet wanneer het SCP zich in een concave hoek bevindt en het crossproduct van de twee normalen van de driehoeken dus de richting van die hoek vormt. Omdat we in 3D werken kunnen we nu aan de laatste iteratie beginnen. We moeten nog testen of de beweging van het tweede botsingspunt naar de nieuwe gevonden ideale SCP positie geen botsing oplevert met een derde driehoek. Als dit het geval is zal het botsingspunt de nieuwe SCP positie zijn want het SCP kan geen enkele richting meer uit bewegen. Voor bepaalde gedeeltes van dit algoritme, zoals het vinden van de tweede ideale SCP positie, heb ik me gebaseerd op de CHAI-3D library[6]. Het implementeren van deze techniek is echter niet perfect gelukt. Het SCP vindt soms geen botsing wanneer er wel één moet zijn. Hierdoor vliegt het SCP als het ware door het object heen.

Om de drie collision detection algoritmes in combinatie met haptics met elkaar te vergelijken zullen we de aanpak van De Boeck et al.[DBRC05] volgen. Er wordt een theoretisch framework voorgesteld bestaande uit een aantal definities waaruit volgt dat er oneindig aantal gevallen beschouwd moeten worden om een haptisch algoritme te testen. Maar er wordt ook een

[6] `http://www.chai3d.org`

praktische manier voorgesteld om haptic algoritmes te testen, rekening houdend met deze definities. We zullen deze dan ook volgen.

Om een nieuw algoritme te evalueren wordt er een empirische methode gecombineerd met statistische methodes. Eerst en vooral moet er een referentie-algoritme zijn waarmee vergeleken wordt, maar het is zeer moeilijk om de correctheid van een bepaald algoritme te bewijzen zonder een basis om mee te kunnen vergelijken. En stel dat we een referentie-algoritme hebben dan moet een oneindig aantal testen uitgevoerd worden om de correctheid van het nieuwe algoritme te bewijzen. Om deze redenen wordt in [DBRC05] als referentie-algoritme een algoritme gekozen dat zijn correctheid al bewezen heeft in de praktijk. Wij kiezen het algoritme eerder besproken waarbij collision detection wordt gedaan door te testen met alle driehoeken[7]. Daarna worden een aantal objecten gekozen, in ons geval nemen we een cloth met verschillende aantallen driehoeken en op verschillende tijdstippen. Met deze objecten laten we een aantal gebruikers, een aantal seconden het object voelen. Tijdens het voelen slaan we het pad van de proxy op in een log samen met andere informatie zoals het feit of er een botsing is gevonden of niet. Gebruik makend van deze log kunnen we de nieuwe algoritmes die we willen testen naspelen en de informatie van de log vergelijken zoals de executietijd, of er een botsing is gevonden, de positie van het SCP, ... Uit deze vergelijkingen kunnen we conclusies trekken, bijvoorbeeld of twee algoritmes collision equivalent zijn, wat nodig is om te bepalen of twee algoritmes render equivalent zijn. We zullen niet op de details ingaan, deze zijn te vinden in [DBRC05]. Om deze testen te kunnen doen wordt er een library toegevoegd aan HAL, *measurements*. Deze library is ook ontwikkeld door De Boeck et al. en zal gebruikt worden voor de drie nieuwe algoritmes te vergelijken.

Om de test van de drie algoritmes uit te voeren zijn er drie cloths na een aantal iteraties stopgezet en betast met de PHANToM door één persoon. Gebruik makende van deze gegevens hebben we logs gegenereerd voor de drie algoritmes. We zullen uit deze logs alleen de gemiddelde duratie berekenen van elk algoritme waarbij onderscheid wordt gemaakt of er een botsing is gevonden of niet. In tabel 9.4 zijn alle gegevens van de testen verzameld. Een eerste conclusie die we meteen kunnen zien is dat het referentie-algoritme het slechtst werkt, bij een groot aantal driehoeken zijn zelfs de noodzakelijke extra testen bij een botsing bijna verwaarloosbaar. Als we de duratietijden individueel bekijken merken we op dat bij optimized spatial hashing er nagenoeg altijd dezelfde duratietijden zijn. Dit ligt in de verwachtingen, omdat de query tijd van een hashtabel waarbij de elementen uniform verdeeld zijn constant is. Bij de BVH's merken we dat deze langzaam stijgen naar gelang het aantal driehoeken. Merk op dat uit deze tijden niet geconcludeerd kan worden welke van de drie algoritmes het snelste bepaalt welke driehoeken getest moeten worden, want we weten niet hoeveel driehoeken er gevonden werden gedurende de testen en dit is sowieso ook een belangrijke factor. We kunnen uit de tabel ook opmerken dat de intersectietesten voor een AABB BVH minder kostelijk zijn dan deze voor een bol BVH aangezien de duratietijden bijna overal lager liggen. Bij een hogere opdelingsfactor verwachten we dat de duratietijden bij geen botsing hoger liggen omdat er sowieso meer bounding volumes moeten getest worden; dit is ook terug te vinden in de resultaten. Aan de andere kant verwachten we net minder tijd bij het vinden van een botsing, omdat er minder knopen in de boom zijn; dit is ook merkbaar aan de resultaten.

Rekening houdend met de update tijden van de drie algoritmes kunnen we concluderen dat

[7]Desondanks het feit dat het eerder vermelde algoritme niet perfect werkt, kunnen we ervoor zorgen dat de problemen bij het algoritme niet voorkomen in onze testen. Hierdoor kunnen we concluderen dat dit een correct referentie-algoritme is.

	AD		OSH		bol-2	
#driehoeken	CD	N-CD	CD	N-CD	CD	N-CD
162	0,102874	0,084212	0,025004	0,009021	0,026063	0,010382
576	0,752296	0,735026	0,024336	0,012568	0,028830	0,011875
3042	2,781679	2,745635	0,025572	0,009210	0,031033	0,013224
	bol-4		AABB-2		AABB-4	
#driehoeken	CD	N-CD	CD	N-CD	CD	N-CD
162	0,025858	0,010057	0,023502	0,008524	0,023779	0,008153
576	0,028204	0,012390	0,025501	0,009450	0,024892	0,012568
3042	0,029767	0,012841	0,027431	0,010374	0,026700	0,011111

Tabel 9.4: Vergelijking van de drie collision detection algoritmes in combinatie met haptic rendering. CD is de gemiddelde tijd die nodig was om een botsing te verwerken en N-CD de gemiddelde tijd waarbij geen botsing gebeurde. AD is het referentie-algoritme waarbij alle driehoeken getest werden op intersectie, OSH is het Optimized Spatial Hashing algoritme, bol/AABB-2/4 zijn BVH's met hun respectievelijke bounding volume en opdelingsfactor. De tijden zijn in milliseconden.

een bol BVH met opdelingsfactor vier het beste presteert; de update tijd is hier duidelijk de beslissende factor. Het 'slechtste' algoritme is optimized spatial hashing en dit door de tragere update tijd, maar desondanks lijkt me dit een zeer interessant algoritme voor bijvoorbeeld statische scenes, vanwege de snelheid waarmee getest kan worden op de driehoeken die aan een intersectietest onderworpen moeten worden. Eigenlijk is het zeer moeilijk te bepalen welke van deze algoritmes nu effectief sneller is. Het gaat hier om zulke kleine verschillen in snelheden dat misschien beter gekeken kan worden naar de samenwerking van het algoritme met de rest van de simulatie, maar daar zullen we niet dieper op ingaan.

Naast het voelen van de vorm van een cloth of een object is het ook dikwijls mogelijk om een object vast te nemen in een virtuele omgeving en te verplaatsen. De manier waarop ik dit gerealiseerd heb in HAL en in combinatie met een cloth zal het onderwerp zijn van de volgende sectie.

Picking

Omdat het ook interessant is om een object te kunnen vastnemen en verslepen in een virtuele omgeving heb ik door middel van een paar aanpassingen deze functionaliteit in HAL voorzien.

Wanneer de gebruiker de stylusbutton van de PHANToM induwt zal in plaats van de haptic rendering functie, de picking functie opgeroepen worden. Deze maakt gebruik van één van de drie collision detection algoritmes om te bepalen welke particle de dichtstbijzijnde is. Eerst wordt gecontroleerd of de proxy zich in het vlak van een driehoek bevindt. Als dit het geval is, wordt de dichtstbijzijnde particle gekozen als picking-punt. Door deze particle de positie van de proxy te laten volgen kunnen we het cloth verslepen. Tijdens het verslepen worden er krachten uitgeoefend op de gebruiker. De kracht wordt berekend door een veer tussen de proxy en de vastgenomen particle te plaatsen in combinatie met de zwaartekracht van de simulatie en de massa van het cloth.

Een laatste aanpassing aan HAL is de manier waarop krachten worden teruggegeven tijdens

het voelen van een object. Er is enkel een eenvoudige force shading techniek geïmplementeerd. Deze zal besproken worden in de volgende sectie.

Force Shading

De manier waarop krachten worden teruggegeven in HAL is aangepast, zodat een simpele vorm van force shading gebeurt. De normalen van de vertices van de driehoek waarmee het SCP in contact is, worden geïnterpoleerd om de richting van de kracht weer te geven. Deze techniek wordt ook besproken in 8.2.4. De lengte van de krachtvector wordt bepaald door een veer waarbij de gekozen veerconstanten gelijk zijn aan de veerconstanten van de structurele veren van het cloth, vermenigvuldigd met een constante.

Het gebruik van deze simpele force shading heeft reeds een zeer goed effect, het verschil is duidelijk merkbaar wanneer het cloth zich in een bolle positie bevindt en er dus geen grote verschillen tussen de normalen van de driehoeken zijn. Indien dit het geval is, voel je dankzij deze techniek een glad oppervlak terwijl je anders de kleine verschillen wel zou voelen.

Tot slot van deze sectie zullen we huidige problemen van de implementatie overlopen.

Huidige Problemen

In mijn implementatie resten er nog een aantal problemen, die wegens tijdsgebrek niet meer opgelost raakt zijn.

Tijdens de haptic rendering kan het contact met het cloth soms niet meer gevonden worden en schieten we door het cloth heen. De fout hier is niet meteen duidelijk; dieper onderzoek in het algoritme dat ik geïmplementeerd heb is nodig. Ofwel is er een probleem met afrondingsfouten, waardoor de proxy plots aan de verkeerde kant van het cloth ligt en het contact dus verloren raakt. Een andere mogelijke oorzaak van het probleem kan liggen bij het vinden van de eerste contactdriehoek. Wederom door afrondingsfouten of door foute afstandsbepaling kan de verkeerde driehoek eerst gevonden worden en zal de tweede iteratie van het algoritme geen goed resultaat opleveren. Het vinden van deze problemen en het oplossen ervan ligt niet voor de hand en is een tijdrovend proces.

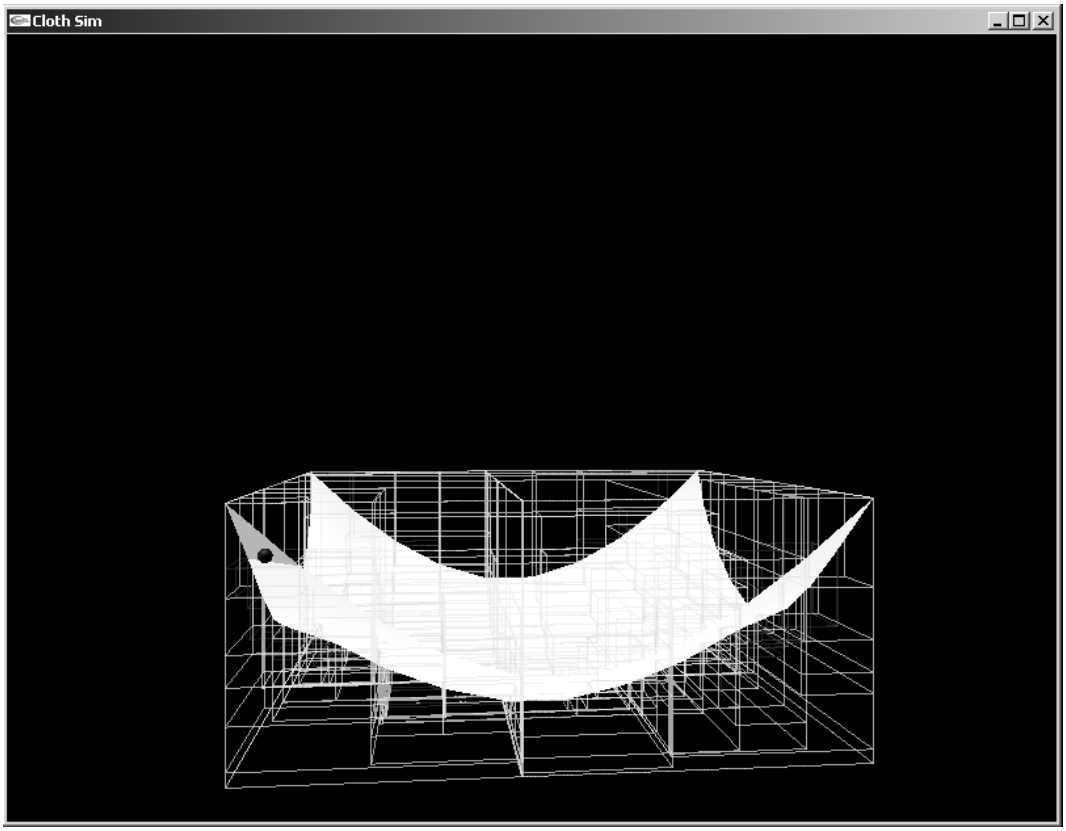
Een ander probleem, weliswaar minder groot, is het teruggeven van de krachten. Doordat de kracht die de gebruiker voelt ook op het cloth wordt gezet, worden er trillingen geïntroduceerd in de krachtsterugkoppeling. Enkele simpele interpolatiepogingen om de krachten uit te effenen boden geen perfecte oplossing. Een andere oplossing is om het cloth twee keer bij te houden waarbij het cloth in de vorige en de huidige tijdstap gekend is. Dan is het mogelijk om de krachten beter te interpoleren.

9.2.3 Externe Applicatie gebruik makend van HAL

In de voorbije secties werd besproken hoe ik de cloth simulator in HAL geïntegreerd heb en op welke manier het mogelijk is gemaakt om ervoor te zorgen dat we op een snelle en efficiënte manier aan haptic rendering kunnen doen. Op deze manier wordt de nodige update rate van de haptic loop zonder problemen gehaald. Aangezien HAL enkel haptics voorziet is er ook

nog een externe applicatie nodig die HAL gebruikt om tot een volledige applicatie te komen die ook grafische feedback voorziet.

Om tot een externe applicatie te komen—die gebruikt kan worden door allerlei toolkits die een GUI ondersteunen in samenwerking met OpenGL—is er een *Simulation* klasse gemaakt die functies bevat waarnaar de GUI events gestuurd moeten worden. Deze simulation klasse maakt dan gebruik van een OpenGLRenderer om het cloth grafisch weer te geven. Deze kan het cloth renderen op de manieren besproken in sectie 9.1.1. In figuur 9.5 is de uiteindelijke applicatie te zien in combinatie met HAL waarbij glut[8] gebruikt wordt om tot een GUI te komen.



Figuur 9.5: De uiteindelijke applicatie: aan de linkerzijde is de proxy te zien en de driehoek die getest wordt op intersectie is ook duidelijk te onderscheiden aan de donkergrijze kleur. Ook is er een AABB BVH te zien, maar de grafische weergave van een BVH is meestal onduidelijk.

9.3 Conclusie

In dit hoofdstuk hebben we de implementatie besproken die ik gemaak heb op basis van de literatuurstudie. We hebben de volgorde van de tekst aangehouden en zijn dus begonnen met het bespreken van de cloth simulator.

[8] `http://www.xmission.com/~nate/glut.html`

Om een cloth te simuleren wordt een mass spring systeem gebruikt, waarbij de veren worden verbonden volgens de meeste gebruikte klassieke manier. Als integratietechnieken zijn de drie meest gebruikte expliciete technieken en de drie bekendste Verlet schema's geïmplementeerd. Deze Verlet schema's zijn duidelijk veel stabieler dan de expliciete technieken. In samenwerking met inverse dynamica kunnen door gebruik te maken van expliciete technieken toch nog goede resultaten bekomen worden. In combinatie met verlet technieken weegt de extra rekenkost niet op tegen de extra visuele verbetering. Om voor extra realisme te zorgen is het ook mogelijk gemaakt om bepaalde particles vast te pinnen in de virtuele omgeving zodat het doek in de lucht kan opgehangen worden.

Hierna werd besproken hoe deze cloth simulator in HAL geïntegreerd werd. Dit gebeurde door een library toe te voegen aan HAL. Om haptic rendering te kunnen realiseren wordt dan een driehoekige mesh gemaakt van het cloth die er dynamisch mee verbonden is. Om snel en efficiënt een botsing tussen de proxy en het cloth te kunnen detecteren werden drie collision detection algoritmes geïmplementeerd en uitvoerig getest. Deze algoritmes—een bol BVH, een AABB BVH en Optimized Spatial Hashing—zijn alle drie duidelijk aan elkaar gewaagd. Bij het updaten van de BVH, waarbij recursie volledig vermeden wordt, merkten we dat een bol BVH minder updates nodig heeft en dus sneller ge-update kan worden dan een AABB BVH. In vergelijking met Optimized Spatial Hashing zijn beide BVH structuren sneller, maar de snelheid van de update van het eerste blijft nog steeds snel genoeg voor de simulatie. De testen van de drie algoritmes in combinatie met haptics gaven als resultaat dat geen een van de drie algoritmes superieur is, waardoor we in ons geval best kijken naar de update tijd om het beste algoritme te bepalen. We kunnen dus concluderen dat de bol BVH in gebruik met een cloth het efficiëntste voor de dag kwam, maar in een andere simulatie of in combinatie met andere objecten kan één van de andere algoritmes efficiënter zijn. Het algoritme voor haptic rendering is gebaseerd op de Constraint Based Methods, maar heeft echter nog een paar problemen waardoor de gebruiker door het object heen kan voelen.

Aangezien het ook interessant is om een object te kunnen vastnemen en verplaatsen in een virtuele omgeving, is het mogelijk gemaakt om het doek vast te nemen bij een particle en op deze manier te verplaatsen. Tijdens het verplaatsen van het doek worden ook krachten teruggegeven die rekening houden met de zwaartekracht van de simulatie en de massa van het cloth. Als laatste aanpassing aan HAL werd nog besproken op welke manier simpele force shading werd toegevoegd.

De laatste sectie in dit hoofdstuk bespreekt hoe we HAL en de aanpassingen die gebeurd zijn gebruiken om tot een applicatie te komen waarbij ook grafische feedback aanwezig is. We kunnen concluderen dat het mogelijk is om mass spring systemen te combineren met haptics waarbij een mass spring systeem van 1600 particles nog steeds zonder problemen gebruikt kan worden.

Hoofdstuk 10

Conclusie

Door de sterke toename van de rekenkracht van de huidige computers beginnen meer en meer toepassingen realistischer te worden. Cloth simulation is een van deze technieken die zorgt voor extra realisme. Cloth simulation kan gebruikt worden in combinatie met films of virtuele omgevingen of misschien zelfs ooit als marketing. Naast cloth simulation is er ook nog een nieuw opkomend domein, haptics. Haptics kunnen de immersiviteit en doeltreffendheid van een applicatie in een virtuele omgeving verhogen. Het doel van deze thesis was dan ook om cloth simulation en haptics te combineren. In de eerste twee delen is een uitgebreide literatuurstudie gebeurd waarbij veel aandacht werd besteed aan technieken die een zo snel mogelijke uitvoertijd hebben.

In het eerste deel werden technieken besproken om cloth simulation te realiseren door middel van mass spring systemen. In het tweede hoofdstuk werden de basisdelen van een mass spring systeem besproken. Hierna werd een overzicht gegeven van de meest gebruikte technieken om via een mass spring systeem tot een realistische cloth simulation te komen. Het is zelfs zo ver gekomen dat men mass spring systemen gaat gebruiken met veren die geen krachten uitoefenen. Bij deze techniek gebruikt men dezelfde verbindingstechnieken, maar worden de krachten van de verschillende types veren vervangen door een speciaal soort inverse dynamica. Niettegenstaande dat er dus nieuwe methodes ontstaan, is de meest populaire techniek om mass spring systemen te implementeren nog steeds degene waarbij drie verschillende soorten veren de drie fysische eigenschappen van een cloth voorstellen. Als volgende werd een uitgebreid overzicht gegeven van de populairste integratietechnieken die gebruikt worden om een cloth te simuleren. Al deze technieken hebben hun specifieke eigenschappen waardoor grondig gekeken moet worden naar de toepassing waarbij ze gebruikt worden. Recent werden de Verlet schema's geïntroduceerd in de cloth simulation en deze worden nu meer en meer gebruikt vanwege hun lage rekenkost en relatief hoge stabiliteit.

Vervolgens werden er twee technieken besproken om extra realisme toe te voegen aan de simulatie van een cloth. In het vijfde hoofdstuk werden technieken besproken om ervoor te zorgen dat het cloth niet vrijuit kan bewegen. Er worden allerlei beperkingen opgelegd aan de beweegruimte. Deze beperkingen kunnen opgelegd worden door twee technieken te gebruiken. De eerste van deze technieken leeft deze beperkingen perfect na, maar is niet flexibel en kan dus maar altijd op dezelfde manier gebruikt worden. De tweede techniek geeft hier een oplossing voor door gebruik te maken van functies die ervoor zorgen dat de beperkingen nageleefd worden. Maar door afrondingsfouten en integratietechnieken die altijd een benadering zijn

heeft deze techniek als nadeel dat de beperkingen niet gegarandeerd nageleefd zullen worden.

Een tweede techniek die voor meer realisme zorgt is collision detection, wat het onderwerp is van hoofdstuk 6. Door gebruik te maken van collision detection kunnen we een cloth vrij door een virtuele omgeving laten bewegen, waarbij het cloth dan tegen objecten zal botsen of erop zal blijven liggen wat logischerwijze voor extra realisme zorgt. Omdat een cloth een vervormbaar object is wordt er alleen maar aandacht besteed aan technieken die efficiënt zijn bij vervormbare objecten. Een cloth heeft een groot probleem bij collision detection, het kan zichzelf intersecteren en het oplossen van deze self-collisions is zeer kostelijk. Hierdoor wordt in real-time simulaties dit probleem niet opgelost en genegeerd.

In het tweede deel van deze tekst kwam haptics aan bod. In het zevende hoofdstuk werd een introductie gegeven tot het onderzoeksdomein rond haptics waarbij aandacht werd besteed aan haptic interfaces en aan de multi-modaliteit van haptic feedback. Daarna werd een overzicht gegeven van haptic devices en applicaties. In een volgend hoofstuk werden technieken besproken om haptic rendering te realiseren. Eerst werd een algemeen overzicht van een haptisch systeem gegeven. Daarna werden de twee meest gebruikte technieken besproken voor veelhoekige modellen. Hierna volgde nog een kort overzicht voor andere modellen en ook enkele technieken die gebruikt kunnen worden in samenwerking met vervormbare objecten. Omdat deze technieken vaak benaderingen zijn van de vervormbare objecten hebben we gekozen om gebruik te maken van collision detection gecombineerd met de constraint based methods voor haptic rendering. Deze laatste is tegenwoordig de meest gebruikte methode.

In het laatste deel werd de implementatie die ik gemaakt heb besproken. Er werd besproken welke technieken gekozen werden om een cloth simulator te realiseren. Daarna werd uitgelegd hoe deze cloth simulator in HAL, een haptische library, geïntegreerd werd en welke aanpassingen aan HAL gedaan zijn. Er is een mogelijkheid toegevoegd om een cloth dynamisch te verbinden met een driehoekige mesh en de techniek gebruikt bij haptic rendering is verbeterd maar bevat nog enkele problemen. Daarna is er nog een simpele manier van force shading toegevoegd en een manier om het cloth vast te nemen en te verslepen. Er zijn ook een aantal testen uitgevoerd op de algoritmes die gebruikt werden tijdens de haptic rendering. Hieruit kunnen we concluderen dat voor de situatie hier, een bol BVH met opdelingsfactor vier het beste algoritme is. Maar de drie geïmplementeerde algoritmes zijn erg aan elkaar gewaagd en in andere situaties met andere soorten vervormbare objecten of in situaties met andere specificaties of doeleinden zou een ander algoritme beter kunnen presteren. We kunnen concluderen dat haptic rendering van een cloth met een vrij groot aantal particles zonder snelheidsproblemen mogelijk is.

Maar cloth simulation in een virtuele omgeving en in combinatie met haptics is nog niet meteen voor morgen. Indien we een aantal cloths in een simulatie willen gebruiken zullen deze ofwel uit weinig particles moeten bestaan ofwel zal de simulatie op zeer krachtige computers moeten lopen. Onlangs is de eerste *Physics Processing Unit* (PPU)[1] aangekondigd, een nieuw soort processor. Deze nieuwe soort van processor kan dan gebruikt worden voor allerlei fysische simulaties. Het eerste doel van deze PPU is het gebruik bij games, maar deze zou bijvoorbeeld ook in combinatie met haptics en fysische simulaties gebruikt kunnen worden. Het gebruik van haptics en fysische simulaties heeft zeker nog een interessante toekomst, bijvoorbeeld in de entertainment sector, maar er is echter nog een lange weg te gaan voordat de gewone gebruikers op zo een vlak ervan zullen kunnen meegenieten.

[1] `http://www.ageia.com/technology.html`

Bibliografie

[AB03] U. Ascher and E. Boxerman. On the modified conjugate gradient method in cloth simulation. *The Visual Computer*, 19(7–8):523–531, 2003.

[BA04] Eddy Boxerman and Uri Ascher. Decomposing cloth. In Ronan Boulic Dinesh K. Pai, editor, *Eurographics/ACM SIGGRAPH Symposium on Computer Animation*, pages 153–161, Grenoble, France, 2004. Eurographics Association.

[Bar01] David Baraff. Implicit methods for differential equations. In *Physically based Modelling*, Siggraph 2001 Course Notes. Pixar Animation Studios, 2001.

[Ber97] Gino Van Den Bergen. Efficient collision detection of complex deformable models using aabb trees. *J. Graph. Tools*, 2(4):1–13, 1997.

[BFA02] Robert Bridson, Ronald Fedkiw, and John Anderson. Robust treatment of collisions, contact and friction for cloth animation. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 594–603, New York, NY, USA, 2002. ACM Press.

[BG00] Daniel Bielser and Markus H. Gross. Interactive simulation of surgical cuts. In *Pacific Conference on Computer Graphics and Applications*, pages 116–125, Los Alamitos, CA, 2000. IEEE.

[Bra02] Gareth Bradshaw. *Bounding Volume Hierarchies for Level-of-Detail Collision Handling*. PhD thesis, Trinity College, Dublin, Ireland, 2002.

[BSB+01] Joel Brown, Stephen Sorkin, Cynthia Bruyns, Jean-Claude Latombe, Kevin Montgomery, and Michael Stephanides. Real-time simulation of deformable objects tools and application. In *Computer Animation 2001*, pages 228–236, Nov 2001.

[BT95] Srikanth Bandi and Daniel Thalmann. An adaptive spatial subdivision of the object space for fast collision detection of animating rigid bodies. *Computer Graphics Forum*, 14(3):259–270, Aug 1995.

[BW98] David Baraff and Andrew Witkin. Large steps in cloth simulation. In *Proceedings of ACM SIGGRAPH 98*, pages 43–54, New York, NY, USA, 1998. ACM Press.

[CK02] Kwang-Jin Choi and Hyeong-Seok Ko. Stable but responsive cloth. *ACM Transactions on Graphics (ACM SIGGRAPH 2002)*, 21(3):604–611, July 2002.

[DBRC05] Joan De Boeck, Chris Raymaekers, and Karin Coninx. A method for the verification of haptic algorithms. Accepted for 12th International Workshop on Design, Specification and Verification of Interactive Systems (DSVIS'05), Newcastle upon Tyne, UK, July 13–15 2005.

[Den] Denx - advanced dental systems: Dentsim. `http://www.denx.com/dentsim_system_desc.html`.

[DMC02] Jérôme Davanne, Philippe Meseure, and Christophe Chaillou. Stable haptic interaction in a dynamic virtual environment. In *2002 IEEE/RSJ Intl. Conference on Intelligent Robots and Systems EPFL*, pages 2881–2886, Laussane, Zwitserland, Oct 2002.

[DSB99] M. Desbrun, P. Schröder, and A. Barr. Interactive animation of structured deformable objects. In *Proceedings of Graphics Interface (GI 1999)*, pages 1–8, San Francisco, CA, USA, 1999. Canadian Computer-Human Communications Society.

[DV03] Vladimir Dochev and Tzvetomir Vassilev. Efficient super-elasticity handling in mass-spring systems. In *CompSysTech '03: Proceedings of the 4th international conference conference on Computer systems and technologies*, pages 483–490, New York, NY, USA, 2003. Canadian Computer-Human Communications Society.

[EEH00] B. Eberhardt, O. Etzmuß, and M. Hauth. Implicit-explicit schemes for fast animation with particle systems. In *Proceedings of the Eurographics Workshop on Computer Animation and Simulation 2000 (CAS 2000)*. Springer-Verlag, 2000.

[FGL03] Arnulph Fuhrmann, Clemens Groß, and Volker Luckas. Interactive animation of cloth including self collision detection. In *Proceedings of Winter School of Computer Graphics (WSCG 2003)*, volume 11, pages 203–208, Plzen, Czech Republic, February 2003. UNION Agency - Science Press.

[GDO00] Fabio Ganovelli, John Dingliana, and Carol O'Sullivan. Buckettree: Improving collision detection between deformable objects. In *Spring Conference on Computer Graphics SCCG ?00*, pages 156–163, Bratislava (SK), May 2000. Budmerice Castle.

[HACH+04] Vincent Hayward, Oliver R. Astley, Manuel Cruz-Hernandez, Danny Grant, and Gabriel Robles-De-La-Torre. Haptic interfaces and devices. *Sensor Review*, 24(1):16–29, Feb 2004.

[Hap] Hapticmaster. `http://www.est-kl.com/hardware/haptic/fcs/hapticmaster.html`.

[Hau03] Michael Hauth. Numerical techniques for cloth simulation. In *Clothing Simulation and Animation*, Siggraph 2003 Course #29, 2003.

[HMC01] Laurent Hilde, Philippe Meseure, and Christophe Chaillou. A fast implicit integration method for solving dynamic equations of movement. In *VRST '01: Proceedings of the ACM symposium on Virtual reality software and technology*, pages 71–76, New York, NY, USA, 2001. ACM Press.

[Jak01] Thomas Jakobsen. Advanced character physics. In *In Proceedings of GDCONF 2001*, Game Developer's Conference, 2001.

[Kau03] Mikko Kauppila. Implementing the implicit euler method for mass-spring systems. 2003.

[KCCL01] Young-Min Kang, Jeong-Hyeon Choi, Hwan-Gue Cho, and Do-Hoon Lee. An efficient animation of wrinkled cloth with approximate implicit integration. *The Visual Computer*, 17(3):147–157, MAY 2001.

[KCCP00] Young-Min Kang, Jeong-Hyeon Choi, Hwan-Gue Cho, and Chan-Jong Park. Fast and stable animation of cloth with an approximated implicit method. In *Proceedings of Computer Graphics International (CGI 2000)*, pages 247–256, Los Alamitos, CA, Jun 2000. IEEE Computer Society.

[KKTW04] M. Keckeisen, S. Kimmerle, B. Thomaszewski, and M. Wacker. Modelling effects of wind fields in cloth animations. In V. Skala, editor, *Proceedings of Winter School of Computer Graphics (WSCG 2004)*, volume 12, Plzen, Czech Republic, Feb 2004. UNION Agency - Science Press.

[LAM01] Thomas Larsson and Tomas Akenine-Möller. Collision detection for continuously deforming bodies. In *Eurographics 2001, Short Presentations*, pages 325–333. Eurographics Association, Sep 2001.

[Lan99a] Jeff Lander. Collision response: Bouncy, trouncy, fun. In *Game Developer Magazine*. March 1999.

[Lan99b] Jeff Lander. Devil in the blue faceted dress: Real-time cloth animation. In *Game Developer Magazine*. May 1999.

[LC98] Tsai-Yen Li and Jin-Shin Chen. Incremental 3d collision detection with hierarchical data structures. In *VRST '98: Proceedings of the ACM symposium on Virtual reality software and technology*, pages 139–144, New York, NY, USA, 1998. ACM Press.

[LCN99] Jean-Christophe Lombardo, Marie-Paule Cani, and Fabrice Neyret. Real-time collision detection for virtual surgery. In *CA '99: Proceedings of the Computer Animation*, pages 82–90, Washington, DC, USA, 1999. IEEE Computer Society.

[LG98] M. C. Lin and S. Gottschalk. Collision detection between geometric models: a survey. In Robert Cripps, editor, *Proceedings of the 8th IMA Conference on the Mathematics of Surfaces (IMA-98)*, volume 8 of *Mathematics of Surfaces*, pages 37–56, Winchester, UK, Sep 1998. Information Geometers.

[Mat] Mathworld. `http://www.mathworld.com`.

[MBB+02] Kevin Montgomery, Cynthia Bruyns, Joel Brown, Stephen Sorkin, Frederic Mazzella, Guillaume Thonier, Arnaud Tellier, Benjamin Lerman, and Anil Menon. Spring a general framework for collaborative, real-time surgical simulation. In *Medicine meets virtual reality*, pages 296–303, Newport Beach, CA, 2002. Amsterdam: IOS Press.

[MDDB01] Mark Meyer, Gilles Debunne, Mathieu Desbrun, and Alan H. Barr. Interactive animation of cloth-like objects in virtual reality. *Journal of Visualization and Computer Animation*, 12(1):1–12, Feb 2001.

[MDH+03] Philippe Meseure, Jérôme Davanne, Laurent Hilde, Julien Lenoir, Laure France, Frédéric Triquet, and Christophe Chaillou. A physically-based virtual environment dedicated to surgical simulation. In *IS4TH*, pages 38–47, Jan 2003.

[Mel02] Paul E.C. Melis. Real-time cloth simulation in a 3d virtual environment. Master's thesis, Universiteit Twente, 2002.

[MKE03] J. Mezger, S. Kimmerle, and O. Etzmuß. Hierarchical techniques in collision detection for cloth animation. *Journal of WSCG*, 11(2):322–329, 2003.

[ML01] Cesar A. Mendoza and Christian Laugier. Realistic haptic rendering for highly deformable virtual objects. In *Virtual Reality*, pages 264–270. IEEE, Mar 2001.

[MML02] Frédéric Mazzella, Kevin Montgomery, and Jean-Claude Latombe. The forcegrid - a buffer structure for haptic interaction with virtual elastic objects. In *Proceedings of the IEEE International Conference on Robotics and Automation.* IEEE, Jun 2002.

[Mor01] Andrew B. Mor. *Progressive Cutting with Minimal New Element Creation of Soft Tissue Models for Interactive Surgical Simulation.* PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, 2001.

[MP89] Gavin Miller and Andrew Pearce. Globular dynamics: A connected particle system for animating viscous fluids. *j-COMPUTERS-AND-GRAPHICS*, 13(3):305–309, 1989.

[MRF+96] William R. Mark, Scott C. Randolph, Mark Finch, James M. Van Verth, and Russell M. Taylor II. Adding force feedback to graphics systems - issues and solutions. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 447–452, New York, NY, USA, 1996. ACM Press.

[MS96] H. B. Morgenbesser and M. A. Srinivasan. Force shading for shape perception in haptic virtual environments. In *Touch Lab Report 4. RLE TR-606.*, MIT. Cambridge, 1996.

[MSB+04] Anderson Maciel, Sofiane Sarni, Oliver BuchWalder, Ronan Boulic, and Daniel Thalmann. Multi-finger haptic rendering of deformable objects. In *Eurographics Symposium on Virtual Environments (2004)*, pages 105–111, Grenoble, France, 2004.

[NM04] Günter Niemeyer and Probal Mitra. Dynamic proxies and haptic constraints. In Federico Barbagli, Domenico Prattichizzo, and Kenneth Salisbury, editors, *Workshop on Multi-point Interaction in Robotics and Virtual Reality*, New York, 2004. Springer Verlag.

[Pro95] Xavier Provot. Deformation constraints in a mass-spring model to describe rigid cloth behavior. In *Graphics Interface '95*, pages 147–154. Canadian Information Processing Society, Canadian Human-Computer Communications Society, May 1995.

[Pro97] Xavier Provot. Collision and self-collision handling in cloth model dedicated to design garments. In *In Computer Animation and Simulation '97*, pages 177–189. Canadian Information Processing Society, Canadian Human-Computer Communications Society, May 1997.

[Qua00] Peter Quax. The use of haptic feedback in physically based modelling systems. Master's thesis, Limburg Universitair Centrum, 2000.

[Qui94] Sean Quinlan. Efficient distance computation between non-convex objects. In *Proceedings of the IEEE International Conference On Robotics and Automation*, pages 3324–3329, Los Alamitos, CA, USA, May 1994. IEEE Computer Society Press.

[RKK97] Diego C. Ruspini, Krasimir Kolarov, and Oussama Khatib. The haptic display of complex graphical environments. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 345–352, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[RSH00] Erik Reinhard, Brian Smits, and Charles Hansen. Dynamic acceleration structures for interactive ray tracing. In *Proceedings of the Eurographics Workshop on Rendering Techniques 2000*, pages 299–306, London, UK, Jun 2000. Springer-Verlag.

[She94] Jonathan Richard Shewchuk. An introduction to the conjugate gradient method without the agonizing pain (edition 1 1/4). Technical report, Pittsburgh, PA, USA, 1994.

[SIG99] *Haptics: From Basic Principles to Advanced Applications*. Number 38 in Course Notes for SIGGRAPH '99. SIGGRAPH-ACM publication, August 8–13 1999.

[Soe02] Niels Soeffers. Een overzicht en vergelijking van populaire technieken bij cloth simulation. Master's thesis, Limburg Universitair Centrum, 2002.

[THM+03] Matthias Teschner, Bruno Heidelberger, Matthias Müller, Danat Pomeranets, and Markus Gross. Optimized spatial hashing for collision detection of deformable objects. In T. Ertl, B. Girod, G. Greiner, H. Niemann, H.-P. Seidel, E. Steinbach, and R. Westermann, editors, *Proceedings of the Conference on Vision, Modeling and Visualization 2003 (VMV-03)*, pages 47–54, Berlin, GER, Nov 2003. Aka GmbH.

[VMT95] Pascal Volino and Nadia Magnenat-Thalmann. Collision and self-collision detection: Efficient and robust solutions for highly deformable surfaces. In D. Terzopoulos and D. Thalmann, editors, *Proceedings of the 6th Eurographics Workshop on Computer Animation and Simulation*, pages 55–65. Springer-Verlag, 1995.

[VMT01] Pascal VOLINO and Nadia MAGNENAT-THALMANN. Comparing efficiency of integration methods for cloth simulation. In Horace H. S. Ip, Nadia Magnenat-Thalmann, and Tat-Seng Chua Rynson W. H. Lau, editors, *Proceedings of the 19th Computer Graphics International Conference (CGI-01)*, pages 265–274, Los Alamitos, Jul 2001. IEEE Computer Society.

[VSC01] T. Vassilev, B. Spanlang, and Y. Chrysanthou. Fast cloth animation on walking avatars. *Computer Graphics Forum*, 20(3):260–267, SEP 2001.

[VT97] Pascal Volino and Nadia Magnenat Thalmann. *Interactive Cloth Simulation Problems and Solutions*. MIRALab, Geneva, Switzerland, 1997.

[VT00] Pascal Volino and Nadia Magnenat Thalmann. Implementing fast cloth simulation with collision response. In *Proceedings of Computer Graphics International 2000 (CGI 2000)*, pages 257–268, Los Alamitos, CA, Jun 2000. IEEE Computer Society.

[WB01] Andrew Witkin and David Baraff. Differential equation basics. In *Physically based Modelling*, Siggraph 2001 Course Notes. Pixar Animation Studios, 2001.

[wik] Wikipedia - haptic. `http://en.wikipedia.org/wiki/Haptic`.

[Wit01a] Andrew Witkin. Constrained dynamics. In *Physically based Modelling*, Siggraph 2001 Course Notes. Pixar Animation Studios, 2001.

[Wit01b] Andrew Witkin. Particle system dynamics. In *Physically based Modelling*, Siggraph 2001 Course Notes. Pixar Animation Studios, 2001.

[ZC00] Yan Zhuang and John Canny. Haptic interaction with global deformations. In *The International Conference on Robotics and Automations*, pages 2428–2433, San Francisco, California, 2000. IEEE.

[ZL03] Gabriel Zachmann and Elmar Langetepe. Geometric data structures for computer graphics. In *Proc. of ACM SIGGRAPH*. ACM Transactions of Graphics, Jul 2003.

[ZS95] C.B. Zilles and J.K. Salisbury. A constrained-based god-object method for haptic display. In *IROS '95: Proceedings of the International Conference on Intelligent Robots and Systems-Volume 3*, Washington, DC, USA, 1995. IEEE Computer Society.