Kerbal Physics Program

Het simuleren van algemene natuurkundige effecten

gebaseerd op klassieke mechanica

03-03-2015

Koen de Mare, 120001950

Carolus Borromeus college, 6Va

Mr. Dams

*What I cannot create, I do not understand*.

- Richard Feynman

Door op wetenschappelijke wijze met natuurkunde bezig te zijn heb ik tijdens het proces van het maken van dit profielwerkstuk veel kennis verworven over de natuurkunde die ik gesimuleerd heb. Om deze wetten goed toe te passen was het vereist om niet alleen te weten wat de wetten doen, maar ook waarom ze dat doen en wat dat betekende. Bovendien is dit het eerste grote programma dat ik heb geprogrammeerd. Gelukkig is het programma veel beter geworden dan ik had mogen hopen. Ik ben dan ook heel tevreden met mijn uitwerking van het project.

Inhoud:

Inleiding P. 4

Doel van het project P. 5

Belang van het onderzoek P. 5

Deelvragen: P. 6

Wat betekent 'bewegen'? P. 6

Welk gedeelte van bestaande natuurkunde kan gebruikt worden

om bewegingen te simuleren? P. 6

Hoe hebben anderen het simuleren van bewegingen aangepakt? P. 8

Algemene methoden P. 8

Advanced Character Physics P. 9

Position Based Dynamics P. 12

Unified Particle Physics for Real-Time Applications P. 13

Het simuleren van bewegingen van objecten: P. 13

Abstractie voor objecten P.13

Scheiding der krachten P. 14

Niet-contactkrachten P. 15

Contactkrachten P. 16

Bewegingsformules P. 17

Resultaten: P. 18

Natuurkundige precisie & aannames P. 18

Stabiliteit P. 19

Snelheid P. 19

Verder onderzoek: P. 19

Natuurkundig P. 19

Optimalisaties P. 20

Appendix: P. 20

Berekeningen voor afstandsconstraint P. 20

Broncode voor het testen van de resultaten P. 27

Referenties P. 27

Inleiding:

Ik ben al een lange tijd geïnteresseerd door natuurkunde, wiskunde en programmeren. Ook voordat ik dit profielwerkstuk maakte had ik dan ook een groot aantal artikelen over het programmeren van programma’s met uiteenlopende functies, waaronder het simuleren van natuurkunde voor gebruik in spellen.

Het simuleren van natuurkunde in spellen is een van de moeilijkste onderwerpen die ik tegengekomen ben. Zo is het gebruikelijk voor simulatoren (of ‘physics engine’) om objecten die over drie dimensies kunnen verplaatsen en over drie dimensies kunnen roteren, te gebruiken in combinatie met een collision-detection systeem(die dus ook rotaties moet ondersteunen), en om dat enig nut te laten hebben moeten impulsen toegepast kunnen worden die een natuurkundig correcte beweging gaf over alle zes dimensies die de simulator ondersteunt.

Het is niet nodig om te begrijpen wat dit precies betekent, maar bespaar jezelf de moeite en neem van mij aan dat dit betekent dat veel zware en intimiderende wiskunde vereist. Bovendien leek het niet mogelijk te zijn om een van de onderdelen weg te halen zodat zelfs de simpelste mogelijke simulator nog steeds ongelofelijk moeilijk is. Fenomenen zoals vloeistoffen worden standaard niet eens ondersteunt en vereisen nog meer zware wiskunde en moeten ook nog eens gekoppeld worden aan de rest van de simulatie, en om af te sluiten zijn zelfs simulatoren van grote bedrijven berucht om het geven van slechte resultaten. Het simuleren van natuurkunde leek onmogelijk.

Op een gegeven moment kwam ik echter een artikel tegen, ‘Unified Particle Physics for Real-Time Applications’, welke de grootste problemen in één klap oploste; rotaties waren verdwenen, collision detection was een eitje geworden en naast vaste objecten kon je naar hartenlust vloeistoffen, doek, touw vervormbare objecten en gassen simuleren zonder dat dit veel extra moeite zou kosten. Helaas komt alles met een prijs want alle wiskunde die verdwenen was werd vervangen met andere complexe wiskunde.

Toch bleef ik het achterliggende idee ongelofelijk interessant vinden. Bij het doorkijken van de bronnen was ik op 'Advanced Character Physics' gestuit, wat het idee introduceerde dat de complexe wiskunde uit ‘Unified Particle Physics for Real-Time Applications’ probeerde te bereiken. Het bleek dat het achterliggende idee verassent eenvoudig was. Ik bedacht me dat het mogelijk was om het moeilijke gedeelte van ‘Unified Particle Physics for Real-Time Applications’ te vervangen door een makkelijkere versie die bijna hetzelfde zou bereiken. Dit programma zou eenvoudig genoeg zijn dat ik het zelf zou kunnen maken. Het zou zelfs mogelijk zijn om natuurkundige wetten toe te passen om het geheel natuurkundig correcter te maken.

Ik was vastberaden om een simulator te gaan maken, dat ik het voor mijn profielwerkstuk kon gebruiken was bijzaak.

Doel van het project:

Het opstellen en testen van een systeem dat de bewegingen van objecten simuleert.

Belang van het onderzoek:

Soms is het nuttig om een simulatie van de natuurwetten te kunnen maken. Bijvoorbeeld om voorspellingen te maken over hoe een constructie zich zal gedragen onder bepaalde omstandigheden, voor films om de mogelijkheden te vergroten of juist de kosten te drukken, of bij spellen om de geloofwaardigheid van de spelwereld te vergroten.

Helaas bestaat er nog geen simulatiemethode die perfect werkt voor al deze toepassingen in alle situaties, meestal hebben systemen zelfs maar één specialiteit: of ze zijn in staat de krachten en bewegingen in constructies door te rekenen, of ze kunnen de stroming van vloeistoffen goed benaderen, of ze zijn snel genoeg om in spellen gebruikt te worden. Als een boot ontworpen word kan dit betekenen dat er twee verschillende simulaties nodig zijn: een voor de constructie van de boot zelf, en een andere voor de stroming van het water over de romp. Hoewel het soms mogelijk is om de resultaten van de simulatoren met elkaar uit te wisselen is het resultaat vaak ver van perfect of zelfs onbruikbaar.

Door onderzoek te doen naar nieuwe simulatietechnieken worden hopelijk nieuwe systemen met nieuwe, betere eigenschappen ontdekt. In dit onderzoek heb ik geprobeerd een systeem te ontwikkelen dat in staat is om een grote verscheidenheid aan stoffen met onderlinge interactie te simuleren, zonder aan de natuurkundige correctheid van het geheel af te doen.

Deelvragen:

Wat betekent 'bewegen'?

Bewegen is het veranderen van de posities en rotaties van objecten over tijd.

Welk gedeelte van bestaande natuurkunde kan gebruikt worden om bewegingen te simuleren?

Aangezien het bestuderen van bewegingen een belangrijk onderdeel is van natuurkunde kunnen we een aantal inzichten overnemen voor gebruik in de simulator. Het veld van de natuurkunde dat over bewegingen van objecten (onder normale omstandigheden) gaat heet klassieke mechanica.

De belangrijkste grootheden, eenheden een formules worden hier kort behandeld:

De volgende grootheden en bijbehorende eenheden zijn overgenomen uit het SI:

|  |  |
| --- | --- |
| Grootheid (afkorting) | Eenheid (afkorting) |
| Tijd (t) | seconde (s) |
| Positie\* (S) | meter (m) |
| Snelheid\* (V) | meter / seconde (m/s) |
| Acceleratie\* (a) | meter / seconde2 (m/s2) |
| Massa (m) | kilogram (kg) |
| Kracht\* (F) | Newton (n) |

\*vectorgrootheid

Bovendien biedt de klassieke mechanica de volgende formules:

De tweede wet van Newton:

Ft = m \* a

Ft: totale kracht (in N)

m: massa (in kg)

a: acceleratie (in m/s2)

De derde wet van Newton stelt dat bij elke kracht een andere kracht bestaat welke tegenovergesteld is in richting en gelijk in magnitude. Dit kan als volgt genoteerd word:

F1 = - F2

F1: kracht 1 (in N)

F2: kracht 2 (in N)

Klassieke mechanica bestudeert de fenomenen die krachten veroorzaken. Hierdoor zijn voor alle bekende soorten krachten duidelijke definities opgesteld, en bij een groot aantal is ook een formule gevonden om de kracht te kunnen berekenen. Zo geldt bijvoorbeeld bij zwaartekracht, veerkracht en dempkracht dat:

Hij homogene g:

Fz = M \* g

Fz = zwaartekracht (in N)

M: massa (in kilogram)

g = gravitatie (in m/(s^2))

Fv = k \* (l - lr)

Fv = veerkracht (in N)

k = krachtconstante (in N/m)

l = lengte van de veer (in m)

lr = lengte van de veer bij rust (in m)

Fd = -c \* V

Fd = dempkracht (in N)

c = dempingconstante (in Ns/m)

V: snelheid (in m/s)

Hoe hebben anderen het simuleren van bewegingen aangepakt?

Het simuleren van bewegingen is een veel bestudeerd gebied. Er zijn dan ook veel verschillende methoden om natuurkunde te simuleren ontwikkeld met uiteenlopende eigenschappen. 'Real Time Physics Class Notes' geeft een kleine greep uit de methodes voor vaste stoffen die computationeel snel genoeg zijn om in 'real-time' gebruikt te kunnen worden.

Algemene methoden:

Enkele onderdelen zijn vrijwel gelijk bij alle simulatiemethoden. Om tijdsverloop te kunnen simuleren word de tijd opgedeeld in kleine tijdsintervallen. Bij een bekende staat van de te simuleren objecten word een tijdsinterval bepaald welke gesimuleerd moet worden, waarna de simulator de staat van de objecten na dit tijdsinterval kan benaderen. Deze nieuwe staat kan nu weer gebruikt worden als beginstaat voor een nieuw tijdsinterval.

Een groot aantal simulatoren maakt gebruik van krachten aangezien veel over krachten bekend is, en krachten voor een grote verscheidenheid van situaties gebruikt kunnen worden. Om de krachten te koppelen aan de posities word de tweede wet van Newton gebruikt in de volgende vorm:

a = Ft / M

a: versnelling (in m/(s2))

Ft: totale kracht (in Newton of N)

M: massa (in kilogram)

De totale kracht word berekent door de volgende formule:

Bij n krachten:

Ft = F1 + F2 + ... + F(n-1) + Fn

Ft: totale kracht (in N)

Fn: kracht van kracht n (in N)

Individuele krachten worden berekend met formules voor het betreffende type kracht.

De meeste methoden hebben helaas een paar grote problemen: aangezien veel methoden voor elk soort stof (vloeistoffen, gassen, vaste objecten en draden/doeken) een andere techniek gebruiken speciaal voor die soort stof is het zeer moeilijk om onderling gedrag tussen verschillende stoffen die in andere categorieën vallen te simuleren. Bovendien geeft deze koppeling over het algemeen een grote natuurkundige fout. Een ander groot probleem is de normaalkracht, aangezien er geen concrete formule is om normaalkrachten uit te kunnen rekenen.

Advanced Character Physics:

Deze problemen zijn echter aangepakt door een paar simulatietechnieken die radicaal anders werken dan de meeste bekende technieken. Het probleem om normaalkrachten goed te simuleren is geadresseerd door Thomas Jakobsen in 'Advanced Character Physics'. Om normaalkrachten en andere zeer stijve krachten te simuleren wordt gekozen om niet te proberen de kracht zelf te benaderen, maar juist het resultaat dat de kracht op de posities heeft. De reden hiervoor was dat het resultaat van de simulatie zo veel stabieler wordt, niet om natuurkundige correctheid te bereiken.

Toch valt er natuurkundig iets te zeggen voor het idee. Met enkele aanpassingen blijkt het zelfs zeer geschikt te zijn om op natuurkundig accurate wijze normaalkrachten te simuleren. In het verslag zal toelichting geven worden over de natuurkundige achtergrond.

Om contactkrachten direct op posities uit te oefenen wordt in 'Advanced Character Physics' voorgesteld een Verlet-integratieschema in combinatie met constraints te gebruiken. Verlet-integratie betekent dat de huidige positie, de vorige positie en de tijd tussen deze posities opgeslagen worden (het is gangbaar om de huidige positie en huidige snelheid op te slaan). De snelheid van de particles is toch impliciet opgeslagen aangezien we deze kunnen berekenen met de volgende formule:

Bij constante V:

V = (S1 - S0) / t0

V: snelheid (in m/s)

S1: huidige positie (in m)

S0: vorige positie (in m)

t0: tijdsinterval tussen S0 en S1 (in s)

Om de nieuwe (gespeculeerde) positie uit te rekenen word deze formule samengenomen met de volgende formule:

Bij constante a:

S2 = S1 + V \* t1 + ½ \* a \* t12

S2: gespeculeerde volgende positie (in m)

S1: huidige positie (in m)

V: snelheid (in m/s)

t1: tijdsinterval tussen S1 en S2 (in s)

a: acceleratie (in m/s2)

Samengenomen krijgen we de volgende formule:

S2 = S1 + (S1 - S0) / t0 \* t1 + ½ \* a \* t12

S2: gespeculeerde volgende positie (in m)

S1: huidige positie (in m)

S0: vorige positie (in m)

t0: tijdsinterval tussen S0 en S1 (in s)

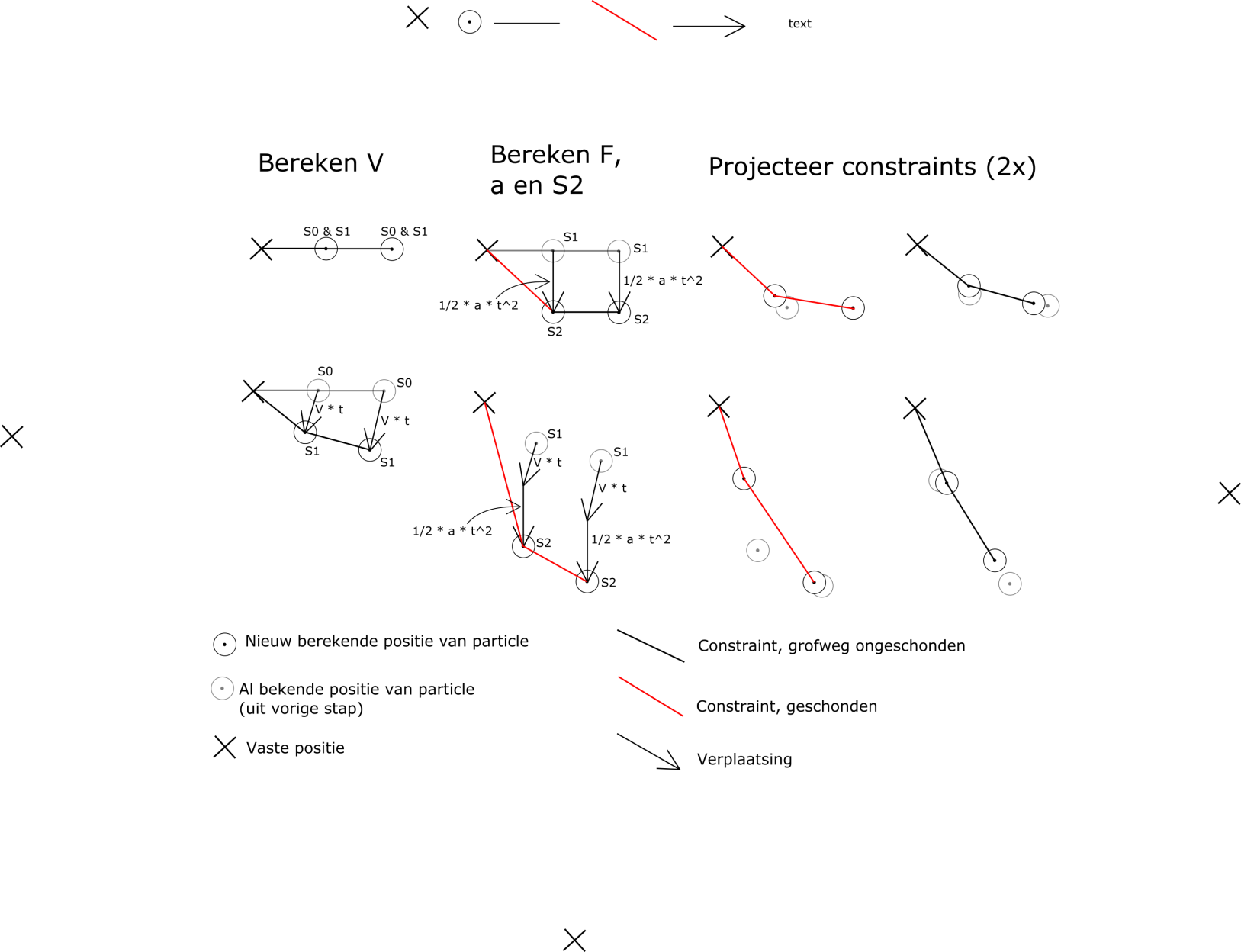
t1: tijdsinterval tussen S1 en S2 (in s)

a: acceleratie (in m/s2)

Dit verlet-integratieschema is interessant omdat de snelheid impliciet word opgeslagen in de posities. Dit betekent dat we de gespeculeerde volgende posities (S2) aan kunnen passen terwijl de snelheden voor de volgende tijdstap automatisch bijgewerkt worden. Wat belangrijk is om op te merken is dat bij deze methode aangenomen word dat de snelheid constant is, wat niet klopt aangezien acceleraties gebruikt worden. De methode is wiskundig gezien dus niet correct. Later in het verslag zal ik met een oplossing voor dit probleem komen (aan de hand van een aanpassing voorgesteld door andere onderzoekers).

Nu de gespeculeerde volgende positie bekend is kunnen de contactkrachten hierop werken. Omdat we nu niet geïnteresseerd zijn in acceleraties maar in posities kunnen contactkrachten niet berekend worden zoals krachten normaal berekend worden. Aangezien contactkrachten ontstaan door beperkingen aan de posities van objecten is het het meest voor de hand liggend om normaalkrachten te representeren als voorwaarden aan de posities van objecten. Deze voorwaarden worden 'constraints' genoemd.

Om contactkrachten uit te oefenen kijken we welke constraints geschonden zijn, en passen de posities van de objecten simpelweg aan zodat de constraint niet meer geschonden word (dit met respect tot natuurkundige wetten uiteraard). Dit word constraint projection of constraint relaxation genoemd. In de praktijk komen hierbij een aantal moeilijkheden voor, zo kan het verplaatsen van objecten ervoor zorgen dat een andere constraint juist geschonden word. Ook kan het dat twee constraints een object grofweg dezelfde richting in duwen, wat ervoor kan zorgen dat het object te ver weggeduwd word. De details en oplossingen van deze problemen worden later behandeld. Tot dat punt mag de lezer er van uit gaan dat alle constraints tijdens deze stap op natuurkundig correcte wijze opgelost worden.



Figuur - grafisch voorbeeld voor het simuleren van een korte slinger met verletintegratie en constraint projection

Position Based Dynamics:

In 'Position Based Dynamics' wordt de verletintegratie overgenomen uit 'Advanced Character Physics', hoewel hier gekozen word om de snelheden van de particles na elke tijdstap uit te rekenen en ze expliciet op te slaan. De resulterende posities zijn identiek aan de resultaten van normale Verlet-integratie, wat ook betekent dat de wiskundig fout nog steeds aanwezig is. Er is gekozen voor het expliciet opslaan van de snelheden omdat dit het mogelijk maakt de snelheden aan te passen zonder de posities te hoeven veranderen. Voor mijn toepassing heeft deze aanpassing nog een voordeel; hierdoor is het namelijk mogelijk om de formules te veranderen zodat het niet meer nodig is om een constante snelheid aan te nemen, wat er voor zorgt dat de eerder beschreven wiskundige fout niet meer aanwezig is.

Unified Particle Physics for Real-Time Applications:

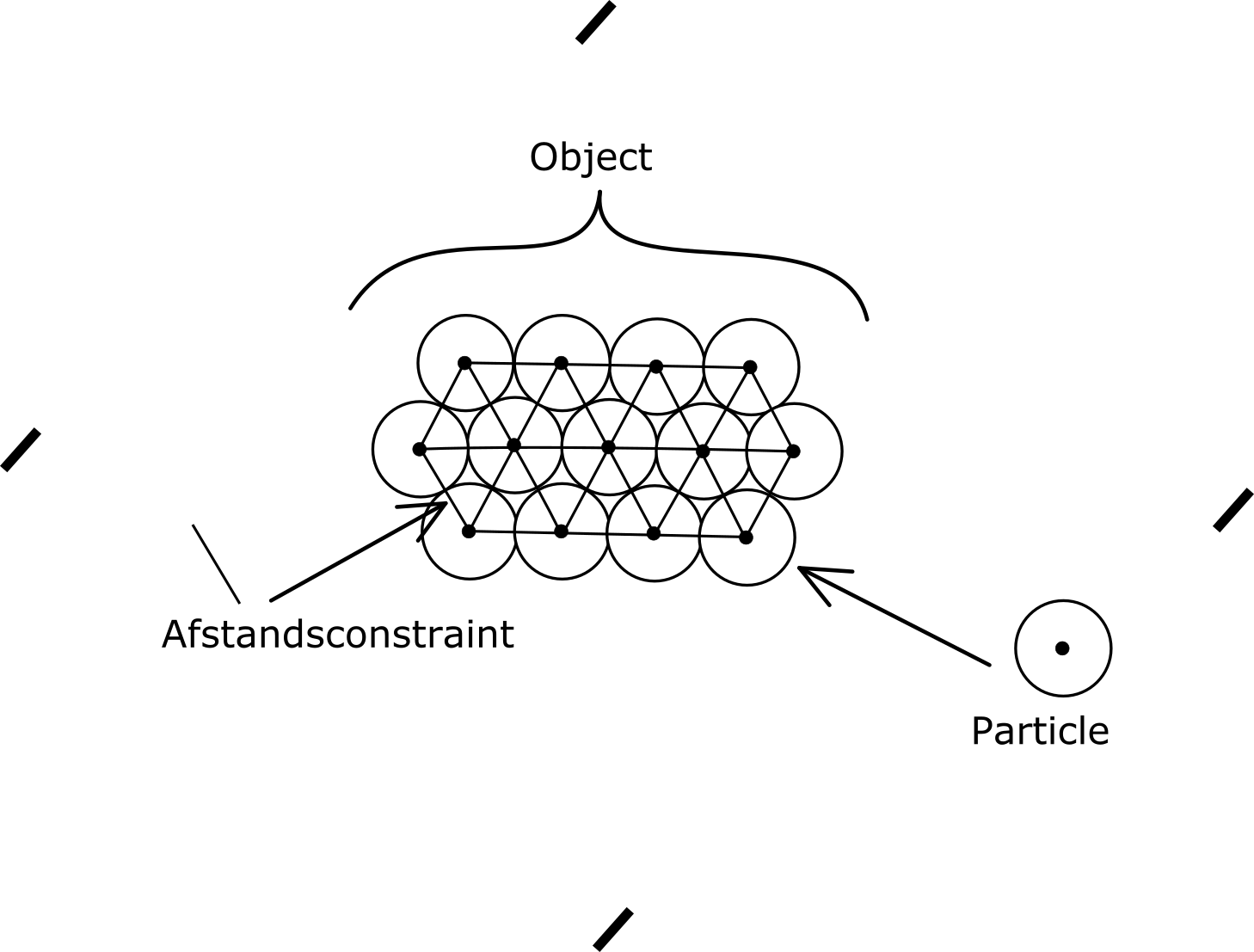
In 'Unified Particle Physics for Real-Time Applications' wordt het systeem uit 'Position Based Dynamics' aangepast met als hoofddoel te zorgen dat het systeem toegepast kan worden op verschillende typen stoffen, zonder dat de simulator voor de verschillende stoffen verschillende subsystemen nodig heeft. Om dit te bereiken worden objecten voorgesteld als 'particles' of puntmassa's met onderlinge relaties (constraints bijvoorbeeld). Zo kunnen zowel vaste stoffen, elastische/vervormbare stoffen, vloeistoffen, gassen en doek op dezelfde wijze gesimuleerd worden. Bovendien word de wiskunde vereenvoudigt door het feit dat particles individueel geen rotatie hebben, terwijl rotaties van objecten die bestaan uit onderling verbonden particles nog steeds op natuurkundig correcte manieren gesimuleerd kunnen worden.

Het simuleren van bewegingen van objecten:

Abstractie voor objecten:

Om iets te hebben om mee te rekenen moet allereerst een abstractie voor objecten gevonden worden. 'Unified Particle Physics for Real-Time Applications' stelt voor om particles te gebruiken, meerdere particles kunnen met elkaar verbonden worden om objecten met complexe vormen voor te stellen. Dit maakt de simulatie wiskundig eenvoudig en bovendien zeer flexibel, om deze redenen heb ik deze methode overgenomen.

De particles krijgen bij onze implementatie de volgende attributen: positie, snelheid, acceleratie en massa. Enkele andere goede eigenschappen van dit systeem zijn dat objecten op natuurkundig accurate wijze kunnen roteren terwijl de complexe wiskunde en de praktische problemen die normaal met rotaties gepaard gaan niet voorkomen. Bovendien kunnen interne krachten (binnen het object) met deze methode goed gesimuleerd worden, zo wordt het bijvoorbeeld mogelijk dat objecten op accurate wijze buigen of breken.



Figuur – de opbouw van een vast object benadert door een groep particles met onderlinge constraints

Scheiding der krachten:

Omdat verschillende krachten op verschillende manieren ontstaan hebben we alle krachten opgedeeld in twee groepen, de contactkrachten en de niet-contactkrachten. De definitie van contactkrachten is als volgt:

*Contactkrachten zijn krachten die alleen ontstaan door ruimtelijke beperkingen aan de posities van objecten.*

Alle andere krachten zijn niet-contactkrachten. De enige contactkracht is de normaalkracht, welke ontstaat om te zorgen dat twee objecten niet dezelfde ruimte innemen (ruimtelijke beperking). Omdat de ruimtelijke beperkingen gehandhaafd moeten blijven blijkt uit de definitie dat contactkrachten een direct effect hebben op de posities van objecten, terwijl alle andere krachten invloed hebben op de snelheden van objecten. Voor alle niet-contactkrachten bestaan concrete formules om de kracht uit te rekenen.

Een uitzondering op deze regel is wrijvingskracht, aangezien dit geen fundamentele kracht is, maar een gevolg is van meerdere soorten krachten op kleine tot zeer kleine schaal. Een van deze krachten is de normaalkracht. Aangezien de formule die normaal gebruikt word om wrijving te benaderen gebruik maakt van de normaalkracht maar de wrijvingskracht zelf een niet-contactkracht is kan deze niet goed geïmplementeerd worden in deze simulator. De simulator is wel in staat de krachten die bij elkaar 'wrijvingskracht' genoemd worden te simuleren, maar door de kleine schaal waarop deze krachten werken zal zeer veel computationeel kracht vereist zijn om goede resultaten te krijgen.

Andere bijzondere gevallen zijn zeer stijve relaties, zoals trekkrachten in vaste objecten. Zeer stijve relaties geven over het algemeen namelijk zeer instabiele resultaten als de simulator bij lage precisie werkt. Als te weinig rekenkracht voor handen is kan het nuttig zijn om de relaties als perfect stijf te beschouwen. In dat geval kunnen de relaties namelijk als contactkracht beschouwd kunnen worden, wat in veel stabielere resultaten resulteert.

Niet-contactkrachten:

Omdat contactkrachten en niet-contactkrachten op een structureel andere manier werken is het nodig om beide typen krachten te simuleren op een manier die recht doet aan de wijze waarop de kracht werkt. Niet-contactkrachten kunnen gebruikt worden zoals krachten over het algemeen gebruikt worden; aan de hand van de formules van de niet-contactkrachten en de staat van alle objecten (positie, snelheid en massa) worden de krachten uitgerekend. Dan wordt voor elke particle de totale niet-contactkracht uitgerekend als som van alle individuele niet-contactkrachten. Dit kan als volgt genoteerd worden:

Bij n krachten:

Ft = F1 + F2 + ... + F(n-1) + Fn

Ft: totale kracht (in N)

Fn: kracht van kracht n (in N)

Waarna de totale niet-contactkracht in de tweede wet van Newton wordt ingevuld om de acceleratie uit te rekenen:

a = Ft / M

a: versnelling (in m/s2)

Ft: totale kracht (in Newton of N)

M: massa (in kilogram)

Contactkrachten:

'Advanced Character Physics' stelt voor om contactkrachten voor te stellen als beperkingen aan de posities van objecten, welke 'constraints' genoemd worden. Om contactkrachten toe te passen worden de posities van de particles aangepast zodat de constraint gehandhaafd is. De enige (echte) contactkracht is de normaalkracht, hierom zal ik alleen algoritmes geven om de normaalkracht als constraint te implementeren. De naam van de constraint die normaalkrachten voortelt is de afstandsconstraint. Afstandsconstraints definiëren de minimale afstand tussen twee particles om niet met elkaar te overlappen.

Als we constraints oplossen is het eerst handig om te kunnen meten in welke mate constraints geschonden zijn. De grootheid die we hiervoor introduceren is 'fout' met als afkorting E (naar error). De eenheid van de fout is kg \* m. Voor elke constraint kan berekend worden hoe de individuele particles verplaatst moeten worden om de constraint te handhaven. Aan de hand van deze verplaatsing kan de fout als volgt berekend worden:

Bij n particles:

E = P[1]m \* |P[1]dS| + P[2]m \* |P[2]dS| + ... + P[n-1]m \* |P[n-1]dS| + P[n]m \* |P[n]dS|

E: fout van de constraint (in kg\*m)

P[n]m: massa van particle n (in kg)

P[n]dS: benodigde verplaatsing van particle n om de constraint te handhaven (in m)

Bij afstandsconstraints gebruiken we het volgende algoritme om de fout te berekenen:

if (|P[1]S - P[2]S| > d) {

E = 0

} else {

E = 2 \* P[1]m \* |d - |P[2]S – P[1]s|| / (P[1]m / P[2]m + 1)

}

E: fout (in m)

d: afstand die de particles minimaal tot elkaar horen te hebben (in m)

P[n]S: positie van particle n (in m)

P[n]m: massa van particle n (in kg)

In de appendix wordt toegelicht hoe deze formule opgesteld is.

Een normale formule volstaat niet om E uit te rekenen, want eigenlijk zijn er twee verschillende formules om E uit te rekenen; als de onderlinge afstand tussen de particles groter is dan d is de constraint gehandhaafd, en dus is E = 0. Als de onderlinge afstand kleiner is dan d is er wel sprake van een fout, welke berekent wordt met E = 2 \* ||P[1]S - P[2]S| - d| \* P[1]m / (1 + P[1]m / P[2]m). Merk ook op dat de waarde van E altijd voldoet aan E >= 0.

Nu gaan we proberen te zorgen dat alle constraints gehandhaafd worden. Het is echter belangrijk dat constraints niet in één keer gehandhaafd worden, anders zouden we er namelijk impliciet van uitgegaan worden dat de invloed van constraints op elkaar verwaarloosd kan worden. Vaak is het echter zo dat meerdere particles een constraint delen, en dat beide constraints hierdoor veel invloed op elkaar hebben.

Om dit probleem op te lossen worden alle constraints meerdere keren gedeeltelijk gehandhaafd. Door het gedeeltelijk handhaven te herhalen wordt de fout steeds verminderd tot deze 0 m \* kg benadert. Hoe veel een constraint gehandhaafd moet worden wordt gegeven met de volgende formule:

Er = E \* f

Er: hoeveel de fout van de constraint verminderd moet worden (in m\*kg)

E: fout van de constraint (in m\*kg)

f: factor (zonder eenheid) die voldoet aan 0 < f <= 1

Nu kan de verplaatsing van de particles berekend worden met:

relativePosition = P[2]S - P[1]S;

relativeDirection = relativePosition / |relativePosition|

P[1]dS = -0.5 \* relativeDirection \* Er / P[1]m

P[2]dS = +0.5 \* relativeDirection \* Er / P[2]m

relativePosition: positie van particle 2 ten opzichte van particle 1 (in m)

relativeDirection: de richting van particle1 naar particle2 (in m)

P[n]S: positie van particle n (in m)

Er: hoeveel de fout van de constraint verminderd moet worden (in m\*kg)

P[n]m: de massa van particle n (in kg)

In de appendix wordt toegelicht hoe deze formule opgesteld is.

Bewegingsformules:

De normaalkrachten hebben nu alleen de posities beïnvloed, aangezien normaalkrachten ook invloed hebben op de snelheden moeten deze nog aangepast worden. Om dit te bereiken gebruikt 'Advanced Character Physics' een Verlet-integgraschema. Wat inhoudt dat de volgende positie van objecten wordt uitgerekend met de volgende formule:

S2 = S1 + (S1 - S0) / t0 \* t1 + ½ \* a \* t12

S2: gespeculeerde volgende positie (in m)

S1: huidige positie (in m)

S0: vorige positie (in m)

t0: tijdsinterval tussen S0 en S1 (in s)

t1: tijdsinterval tussen S1 en S2 (in s)

a: acceleratie (in m/s2)

In deze formule staat de acceleratie voor de acceleratie die veroorzaakt wordt door de niet-contactkrachten. Door Verletintegratie te gebruiken kan is het niet nodig om een snelheid uit te rekenen. Door de gespeculeerde volgende posities uit te rekenen met de bovenstaande formule, en de constraints daarna op deze gespeculeerde posities te projecteren word de resulterende volgende positie uitgerekend. De huidige posities worden in de volgende simulatiestap gebruikt als vorige positie, en de resulterende volgende positie wordt gebruikt als huidige positie. In een aantal gevallen is het echter nuttig om direct over snelheden te kunnen beschikken.

In 'Position Based Dynamics' wordt voorgesteld de snelheid expliciet op te slaan. De volgende formules kunnen daarvoor gebruikt worden:

Bij constante V:

V1 = (S1 - S0) / t0

V: huidige snelheid (in m/s)

S1: huidige positie (in m)

S0: vorige positie (in m)

t0: tijdsinterval tussen S0 en S1 (in s)

En de gespeculeerde volgende positie wordt berekent met de volgende bewegingsformule:

Bij constante a:

S2 = S1 + V1 \* t1 + ½ \* a \* t12

S2: gespeculeerde volgende positie (in m)

S1: huidige positie (in m)

V1: huidige snelheid (in m/s)

t1: tijdsinterval tussen S1 en S2 (in s)

a: acceleratie (in m/s2)

Deze twee formules geven exact dezelfde gespeculeerde volgende positie als de formule van het Verletintegratie geeft. Wat hier snel duidelijk wordt is dat er een wiskundige fout in het spel is, aangezien een constante snelheid aangenomen wordt, maar de aanwezigheid van een acceleratie aangeeft dat dit niet altijd het geval is. De formule om de huidige snelheid uit te rekenen kan omgeschreven worden zodat deze niet meer uit hoeft te gaan van een constante snelheid, dit leidt tot de volgende formule:

Bij constante a:

V1 = 2 \* (S1 - S0) / t0 - V0

V1: huidige snelheid (in m/s)

S1: huidige positie (in m)

S0: vorige positie (in m)

t0: tijdsinterval tussen S0 en S1 (in s)

V0: vorige snelheid (in m/s)

De formule voor de gespeculeerde volgende positie blijft hetzelfde.

Bij het gebruiken van al deze bewegingsformules wordt aangenomen dat de acceleratie over de tijdstap constant is. De acceleratie verandert in de meeste omstandigheden echter continu. Helaas is er geen manier om de verandering van de acceleratie uit te rekenen zonder aannames te doen die eventueel een veel grotere fout kunnen introduceren. De enige correcte manier om de verandering van de acceleratie zo goed mogelijk te benaderen is door de tijdstap zo klein mogelijk te maken. Door de tijdstap te verkleinen stijgt de vereiste rekenkracht echter. Er moet dus een afwisseling gemaakt worden tussen precisie en snelheid van het simuleren.

Resultaten:

Natuurkundige precisie & aannames:

Er zijn een groot aantal factoren die de precisie van de simulatie aan kunnen passen. Ten eerste de precisie van de computer; computers hebben beperkte ruimte om de decimalen van de getallen op te slaan. Dit limiteert de precisie waarmee objecten gesimuleerd kunnen worden. De fout die hierdoor wordt veroorzaakt is echter zeer klein, en valt in het niets bij de andere fouten.

Een veel groter probleem is dat natuurkunde in het echt een continu proces is, terwijl een simulatie door de beperkte rekenkracht van computers verplicht is dit proces in kleine, simuleerbare stukjes te hakken. In drie onderdelen van deze simulator komt dit terug. De eerste en meest belangrijke hiervan is tijd; de simulator kan pas perfect zijn als lim t -> 0 (door de verplichte aanname van constante acceleratie), wat oneindige rekenkracht zou vereisen. Verder moet de resterende fout bij constraint-projection (E) gelijk zijn aan 0 om de simulatie perfect te maken, er is echte geen enkele garantie dat dit ooit behaalt wordt. De derde fout komt ook uit de constraint-projection, deze komt door de factor f waarmee constraints gedeeltelijk opgelost worden; bij f > 0 wordt aangenomen dat de situatie waarop de constraint wordt geprojecteerd niet verandert terwijl dit heel goed wel zou kunnen. Wat een belangrijke eigenschap is van deze fouten is dat ze afhankelijk zijn van de beschikbare rekenkracht; als minder rekenkracht beschikbaar is kunnen deze resultaten minder goed benaderd worden, maar als de rekenkracht enorm is zou het goed kunnen dat de fout bijna te verwaarlozen is.

Een goede eigenschap van de simulator is dat deze geen aannames doet waar niet aan voldaan kan worden (wat bijvoorbeeld wel aan de hand was bij verletintegratie), wat betekent dat de precisie door het toevoegen van rekenkracht steeds dichter bij perfect komt.

Stabiliteit:

Door gebruik te maken van constraints is de simulator redelijk stabiel, zelfs als stijve verbindingen en een groot tijdsinterval gecombineerd worden. Veren en dempers kunnen onder dergelijke omstandigheden echter wel ‘exploderen’.

Snelheid:

Enkele eenvoudige testen konden zelfs op zeer zware instellingen gedraaid worden terwijl de simulator op interactieve snelheid resultaten kon blijven bieden. Bovendien waren de omstandigheden zeer slecht om goed te presteren (relatief zwakke en verouderde hardware). Ondanks dat geen testen op grotere schaal gedaan werden lijkt het plausibel te zijn dat de simulator op interactieve snelheid goede resultaten in complexe scenario’s kan bieden, mits de hardware krachtiger is en de implementatie van het programma nog een aantal optimalisaties ondergaat.

Verder onderzoek:

Natuurkundig:

Aangezien de simulator in theorie alle soorten krachten ondersteunt, maar nog niet op alles getest is (adhesie, cohesie en elektromagnetische krachten bijvoorbeeld) kan het interessant zijn om de simulator te testen of aan te passen op dergelijke omstandigheden.

Optimalisaties:

Zoals al vermeld bij de snelheid van de simulatie is er nog veel ruimte om te zorgen dat de simulator beter presteert. Dit kan bijvoorbeeld gebeuren door te zorgen dat de bestaande logica sneller uitgevoerd wordt, of door te zorgen dat enkele onderdelen van de simulatie parallel berekend worden. ‘Unified Particle Physics for Real-Time Applications’ gebruikt bijvoorbeeld extreme parallellisatie en rekenkracht die door videokaarten geboden kunnen worden om interactieve prestaties te behalen. In de huidige implementatie is geen rekening gehouden met de prestaties van de simulatie, wat betekent dat er waarschijnlijk veel te winnen is.

Appendix:

Berekeningen bij afstandsconstraint:

Omdat een aantal grootheden hier in vrijwel alle formules gebruikt worden, wordt de toelichting van de grootheden en eenheden voor alle volgende formules van te voren gegeven:

P[n]S: huidige positie van particle n (in m)

P[n]dS: benodigde verplaatsing van particle n om de constraint te handhaven (in m)

P[n]m: massa van particle n (in kg)

P[n]Fn: normaalkracht op particle n (in N)

d: afstand die de particles minimaal tot elkaar horen te hebben (in m)

E: fout van de constraint (in m\*kg)

Er: hoeveel de fout van de constraint verminderd moet worden (in m\*kg)

a: acceleratie (in m/s2)

t: tijdsinterval (in s)

Sr: relatieve positie van particle 2 ten opzichte van particle1 (in m)

dSr: relatieve verplaatsing die particle 2 ten opzichte van particle 1 moet ondervinden zodat de constraint gehandhaafd blijft (in m)

Uit de definitie van Sr en dSr zijn de volgende formules op te stellen:

Sr = P[2]S – P[1]s

dSr = P[2]dS – P[1]dS

Aangezien normaalkrachten werken op de normaal van de contact moet de richting van de verplaatsing van particle 2 gelijk zijn aan de richting van de relatieve positie van particle 2 ten opzichte van particle 1, wat leid tot de volgende formule:

Sr / |Sr| = P[2]dS / |P[2]dS|

De derde wet van Newton stelt dat de normaalkracht die uitwerkt op het andere object de in de omgekeerde richting moet staan. Dit geeft:

Sr / |Sr| = P[2]dS / |P[2]dS| = - P[1]dS / |P[1]dS|

Dit zorgt er voor dat de richting van dSr gelijk is aan de richting van Sr:

dSr = Sr / |Sr|

De magnitude van dSr kan bepaalt worden met het volgende algoritme:

If (d > |Sr|) {

|dSr| = d - |Sr|

} else {

|dSr| = 0

}

Gecombineerd tot één algoritme geeft dit:

dSr = Sr / |Sr|

If (d > |Sr|) {

dSr = dSr \* (d - |Sr|)

} else {

dSr = dSr \* 0

}

De de volgende formules beschrijven de tweede wet van Newton, de derde wet van Newton en een de invloed van acceleratie op positie:

P[1]Fn = - P[2]Fn

F = m \* a

dS = ½ \* a \* t^2

Deze formules kunnen gebruikt worden om een algemenen formule te vinden die handig blijkt te zijn bij het projecteren van normaalkrachten. Deze wordt gevonden door de volgende berekening:

P[1]Fn = - P[2]Fn ^ F = m \* a

P[1]m \* P[1]a = - P[2]m \* P[2]a

P[1]m \* P[1]a = - P[2]m \* P[2]a ^ a = 2 \* dS / t^2

P[1]m \* 2 \* P[1]dS / t^2 = - P[2]m \* 2 \* P[2]dS / t^2

P[1]m \* P[1]dS = - P[2]m \* P[2]dS

Deze formule kan in combinatie met de formules die dSr beschrijven gebruikt worden om P[n]dS uit te rekenen:

P[1]m \* P[1]dS = - P[2]m \* P[2]dS

P[2]dS = - P[1]m \* P[1]dS / P[2]m

dSr = P[2]dS – P[1]dS ^ P[2]dS = - P[1]m \* P[1]dS / P[2]m

dSr = - P[1]m \* P[1]dS / P[2]m - P[1]dS

dSr = - P[1]dS \* (P[1]m / P[2]m + 1)

P[1]dS = - dSr / (P[1]m / P[2]m + 1)

Het uitrekenen van de fout van een afstandsconstraint is nu triviaal:

E = P[1]m \* |P[1]dS| + P[2]m \* |P[2]dS| ^ P[1]m \* |P[1]dS| = P[2]m \* |P[2]dS|

E = 2 \* P[1]m \* |P[1]dS|

E = 2 \* P[1]m \* |P[1]dS| ^ dSr = Sr / |Sr| \* (d - |Sr|)

E = 2 \* P[1]m \* |- dSr / (P[1]m / P[2]m + 1)|

E = 2 \* P[1]m \* |dSr| / (P[1]m / P[2]m + 1)

Bij deze formule wordt de magnitude van dSr berekent door een eerder gegeven algoritme.

De simulator moet de constraint gedeeltelijk op kunnen lossen gebaseerd op een bepaalde fout als invoer.

Er = P[1]m \* |P[1]dS| + P[2]m \* |P[2]dS| ^ P[1]m \* P[1]dS = - P[2]m \* P[2]dS

Er = 2 \* P[1]m \* |P[1]dS|

|P[1]dS| = ½ \* Er / P[1]m

|P[1]dS| = ½ \* Er / P[1]m ^ Sr / |Sr| = - P[1]dS / |P[1]dS|

Sr / |Sr| = - P[1]dS / (½ \* Er / P[1]m)

P[1]dS = - Sr / |Sr| \*(½ \* Er / P[1]m)

P[1]dS = -½ \* Er \* Sr / |Sr| / P[1]m

Op dezelfde manier kan de volgende formule berekent worden:

P[2]dS = ½ \* Er \* Sr / |Sr| / P[2]m

Broncode voor de testimplementatie van de simulator:

https://github.com/Failsnail/PositionBasedPhysics

Referenties:

Jakobsen T. 2001, 'Advanced Character Physics', www.gamasutra.com

http://www.cs.cmu.edu/afs/cs/academic/class/15462-s13/www/lec\_slides/Jakobsen.pdf

Müller M., Heidelberger B., Hennix M., Ratcliff J., 2006, 'Position Based Dynamics', Proceedings of Virtual Reality Interactions and Physical Simulations (VRIPhys), pp 71-80, Madrid, November 6-7

http://matthias-mueller-fischer.ch/publications/posBasedDyn.pdf

Macklin M., Müller M., Chentanez N., Kim T., 2014, 'Unified Particle Physics for Real-Time Applications', ACM Transactions on Graphics (SIGGRAPH 2014), 33(4)

http://mmacklin.com/uppfrta\_preprint.pdf

Müller M., Stam J., James D., Thürey N., 2008, 'Real Time Physics Class Notes', ACM SIGGRAPH 2008 classes Article No. 88

http://matthias-mueller-fischer.ch/realtimephysics/coursenotes.pdf/hpbd.pdf