Simuleren van objecten

Voorlopige versie

02-02-2015

Koen de Mare

Carolus Borromeus college, 6Va

Mr. Dams

Inhoud:

Doel van het project 3

Belang van het onderzoek 3

Het simuleren van bewegingen van objecten 4

Object representatie 4

Grondformules 5

Krachten 6

Normaalkrachten en constraints 7

Globale werking 10

Bronnen

Doel van het project:

Het samenstellen, implementeren, en testen van een systeem dat de bewegingen van objecten simuleert.

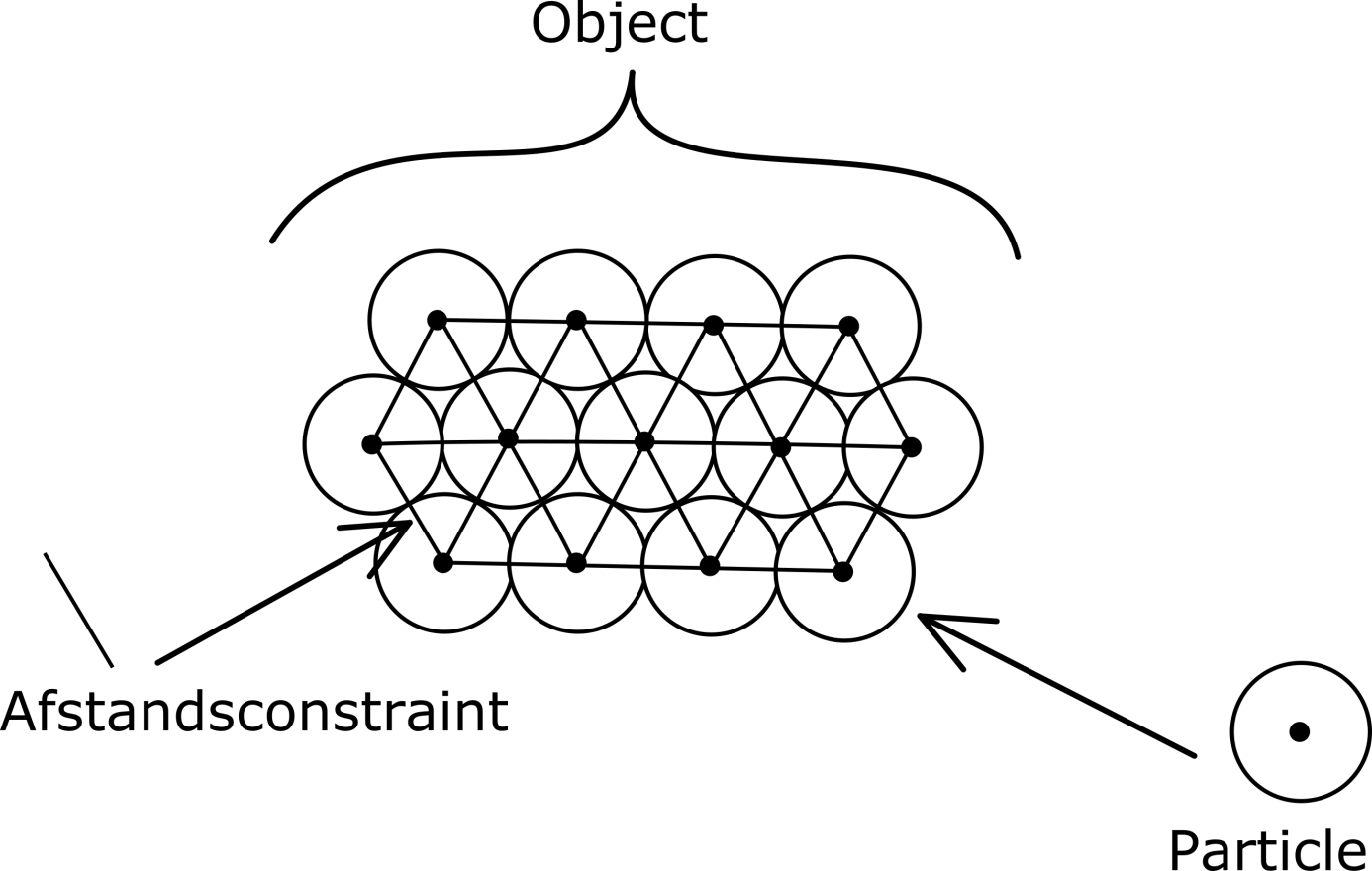
Belang van het onderzoek:

In toepassingen als werktuigbouwkunde, constructie, cinematografie en videospellen is het soms gewenst om een simulatie van de natuurwetten te kunnen gebruiken om bijvoorbeeld voorspellingen te kunnen maken over hoe een constructie zich zal gedragen. Bij film en spellen zijn simulaties een uitkomst omdat deze goedkoop zijn en veel vrijheid geven. Er zit echter een duidelijk verschil in de gevraagde eigenschappen van de simulatie voor deze velden; waar de werktuigbouw graag heeft dat simulaties natuurkundig accuraat zijn, maar de computationele kosten niet van groot belang. Cinematografie, en zeker spellen hechten niet zo veel waarde aan precisie van de simulatie, zo lang deze realistisch overkomt. De computatione kosten zijn echter van groot belang bij spellen omdat computers simpelweg beperkte middelen hebben, welke ook nog verdeeld moeten worden met andere systemen. Onderzoek doen naar methoden om dergelijke natuurkunde te simuleren kan leiden tot nieuwe methoden met nieuwe eigenschappen. Moderne simulatie methoden gekoppeld met krachtige computers kunnen al indrukwekkende resultaten geven, maar er is nog veel te winnen.

Het simuleren van bewegingen van objecten:

Object representatie:

Om iets te hebben om mee te rekenen moet allereerst een abstractie voor objecten gevonden worden. Om de simulatie flexibel en wiskundig eenvoudig te houden worden objecten opgeslagen als punt met als attributen: een positie, snelheid en massa, deze punten worden 'particle', meervoud 'particles' genoemd. Om toch objecten met vormen moeilijkere dan punten te kunnen gebruiken kunnen we meerdere punten met elkaar verbinden (later in het verslag wordt hier verder op ingegaan). Een paar handige eigenschappen van dit systeem is dat objecten op natuurkundig accurate wijze kunnen roteren zonder dat alle wiskunde die hier normaal aan verbonden zijn vereist is. Bovendien kunnen interne krachten (binnen het object) goed gesimuleerd worden, zo wordt het bijvoorbeeld mogelijk dat objecten buigen of breken.



Figuur

Afstandsconstraints worden later besproken. Ze worden gebruikt om relaties tussen particles op te slaan en te handhaven.

Grondformules:

Beweging staat voor de verandering van de positie van een object. Het ultieme doel van de simulator is het vinden van nieuwe posities na een bepaald tijdsinterval (t). Om de verandering in positie te kunnen beschrijven zien we de positie (S) als een functie van tijd (t). De afgeleide van deze functie is de snelheid (V), en de dubbele afgeleide is de acceleratie (a). Merk op dat de omschrijving van snelheid en acceleratie puur wiskundig is. Met deze theorie zijn onder andere de volgende formules te vormen:

Bij constante acceleratie:

S1 = S0 + V0 \* t + 1/2 \* a \* t^2

S1: huidige positie (in m)

S0: vorige positie (in m)

V0: vorige snelheid (in m/s)

a: acceleratie (in m/s^2)

t: tijdsinterval (in s)

Bij constante acceleratie:

V1 = V0 + a \* t

V1: huidige snelheid (in m/s)

Met deze formules kunnen we een nieuwe positie en snelheid uitrekenen. De nieuwe snelheid word in de particles opgeslagen, en word de keer daarop gebruikt als vorige positie en vorige snelheid. De twee onbekenden die overblijven zijn a en t. t staat voor de tijd die is verstreken over het interval dat we willen simuleren. t kan handmatig op een bepaalde waarde vastgesteld worden (bijvoorbeeld bij het simuleren van constructies of filmscènes) omdat de simulator simpelweg een bepaald interval als invoer nodig heeft, of deze kan dynamisch berekend worden met de volgende formule:

t = t1 - t0

t1: huidige tijd (in s)

t0: vorige tijd (in s)

Deze formule word gebruikt bij toepassingen waar het belangrijk is dat de tijd binnen de simulatie gekoppeld word met de tijd van de echte wereld, bijvoorbeeld bij spellen.

De andere onbekende, a, kan berekend worden met de tweede wet van newton:

a = Ft / M

a: versnelling (in m/(s^2))

Ft: totale kracht (in Newton of N)

M: massa (in Kilogram)

Deze wet verbind alle krachten en massa's in ons systeem met de acceleratie, zo is deze wet van onschatbare waarde. Er is hiermee echter een belangrijk probleem: de krachten, massa's en dus ook de acceleratie zijn momentopnamen, maar de acceleratie moet later gebruikt worden op een tijdsinterval. Om te zorgen dat a niet 'veroudert' over t mag t niet groter zijn dan 0, maar t mag ook niet gelijk zijn aan 0, want als dat het gaval zou zijn zou geen tijd verstrijken. Deze twee vereisten gecombineerd betekenen dat de simulatie perfect zal zijn bij lim -> t = 0. Helaas is de rekenkracht van computers beperkt, en kan de wiskunde hier geen uitweg bieden. Dit betekent dat de resultaten van de simulatie logischerwijs niet perfect kunnen zijn (dit komt overigens ook door de beperkte precisie van getallen die door computers gebruikt worden). De beste (en waarschijnlijk) enige oplossing om natuurlijke correctheid te bewaren is om lim t -> 0 te benaderen door t zo klein mogelijk te maken als mogelijk. Gelukkig is de fout die hierdoor geïntroduceerd word klein genoeg om in veel gevallen te verwaarlozen. Voor de rest van het verslag wordt a om deze reden voor constant aangenomen.

Om de acceleratie uit te rekenen hebben we dus de totale kracht die op de particle staat nodig, en de massa van de particle. Net zoals de positie en snelheid is de massa een van de attributen van de particle, welke dus al opgeslagen is en alleen nog maar uitgelezen hoeft te worden. Het berekenen van de totale kracht op een particle is simpelweg de som van alle individuele krachten die op de particle werken, opgeteld:

bij n krachten:

Ft = F1 + F2 + F3 + ... + F(n-1) + Fn

Ft: totale kracht (in N)

Fn: kracht van kracht n (in N)

Krachten:

Resterend is het berekenen van deze individuele krachten. Er zijn verschillende fenomenen die krachten opwekken, bijvoorbeeld:

zwaartekracht, veerkracht, dempkracht, hydraulische en pneumatische druk, wrijving, normaalkracht (de kracht die objecten op elkaar uitoefenen als ze tegen elkaar 'gedrukt' worden), adhesie, cohesie, magnetisme, drijfkracht etc. etc.

Al deze krachten ontstaan op andere manier, hoewel enkele nauw verwant aan elkaar zijn. Voor een aantal van deze krachten zijn formules opgesteld waarmee ze berekend kunnen worden. Zo geldt bij zwaartekracht, veerkracht en dempkracht bijvoorbeeld dat:

Bij constante g:

Fz = M \* g

Fz = zwaartekracht (in N)

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

Bij constante l:

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)

Bij constante V:

Fd = -c \* V

Fd = dempkracht (in N)

c = dempingconstante (in Ns/m)

Om de kracht die deze soorten krachten genereren uit te rekenen zijn maar enkele redelijk eenvoudig te vinden parameters nodig: bij zwaartekracht word de massa van de particle gebruikt, en een constante gravitatie. Voor de veerkracht wordt de veerkrachtconstante, een getal dat de rustlengte van de veer bepaalt en de lengte van de veer (die weer met pythagoras en de posities berekent kan worden) gebruikt. Krachten zoals deze zijn exact uit te rekenen voor elk moment. Omdat deze krachten (binnen de limieten van computers) exact uitgerekend kunnen worden met alleen bekende variabelen zullen deze krachten voor de rest van dit artikel 'deterministische krachten' genoemd worden.

Normaalkrachten en constraints:

Normaalkracht en verwante krachten (zoals hydraulische druk en drijfkracht) zijn echter niet deterministisch. Deze soort krachten ontstaan namelijk om een correcte staat te handhaven (meerdere objecten mogen niet de zelfde ruimte innemen bijvoorbeeld). Dit betekent dat de kracht niet direct gevonden kan worden: eerst moeten alle andere krachten bekend zijn, waarna berekent kan worden welke kracht daar tegenover gezet moet worden om een correcte staat te handhaven. Als een object echter onderhevig is het exact uitrekenen van de krachten logischerwijs niet mogelijk aangezien beide krachten kennis over de andere nodig hebben om een volledig resultaat te kunnen leveren.

Omdat al bleek dat de krachten het gevolg zijn van het handhaven van een correcte staat kunnen we het probleem ook anders aanpakken; Eerst verplaatsten we de particles door alle 'deterministische krachten' en de beginsnelheid toe te passen (dit heet projecteren). Daarna benaderen we een correcte staat zonder de normaalkrachten uit te rekenen. De posities die de particles na deze operatie hebben is resultaat van zowel de deterministische krachten als van de normaalkrachten, dus gelijk aan de eindpositie. De eindsnelheden van de particles na t zijn te berekenen met de vorige positie, vorige snelheid en de tussenliggende tijd t. Omdat we uitgaan van een constante a kunnen we V1 berekenen met de volgende formules:

Bij constante a:

S1 = S0 + V0 \* t + 1/2 \* a \* t^2

Bij constante a:

V1 = V0 + a \* t

V: snelheid (in m/s)

a: acceleratie

We schrijven de eerste formule om naar een formule die a als resultaat geeft:

bij constante a:

a = 2 \* (S1 - S0 - V0 \* t) / t^2

Nu kunnen we deze substitueren in de tweede formule, na vereenvoudiging geeft dit:

bij constante a:

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

Dit betekent dat we S1 en V1 uit kunnen rekenen zonder individuele normaalkrachten te hoeven onderzoeken.

Om de normaalkrachten te kunnen handhaven hebben we eerst een systeem nodig om duidelijk te kunnen bepalen of een staat 'correct' is (dit systeem is gescheiden van het systeem dat de deterministische krachten beschrijft). Dit doen we door relaties tussen particles op te stellen die we 'constraints' of 'beperkingen' noemen. Een constraint omschrijft een eigenschap van de staat van de particles die gehandhaafd moet blijven. Een voorbeeld van een constraint is een afstandsconstraint: om te zorgen dat twee particles niet te dicht bij elkaar komen kunnen we bepalen dat de afstand tussen twee particles niet kleiner mag worden dan een bepaalde waarde die voor de constraint bepaald is. Als de afstand kleiner wordt dan deze waarde beschouwen we de constraint als geschonden. Afstandsconstraints kunnen ook voorkomen in andere vormen, bijvoorbeeld een afstandsconstraint waar de afstand gelijk moet zijn aan een bepaalde waarde. Deze worden gebruikt om de relaties tussen particles in een object vast te stellen, aangezien deze samengedrukt mogen worden, maar ook niet uit elkaar getrokken mogen worden.

Om dit probleem goed op te lossen is het praktisch om uit te kunnen rekenen hoe 'incorrect' een bepaalde staat is. Als we de constraints op willen lossen proberen we deze waarde dus te verlagen. De fout van één constraint is de minimale som van alle afstanden die door de particles afgelegd moeten worden om de constraint te handhaven. Omdat er veel verschillende soorten constraints mogelijk zijn is het helaas niet mogelijk om een alomvattende algoritme (formules volstaan niet om deze waarden uit te rekenen) te geven die de fout van een constraint levert. Ter illustratie zal ik een voorbeeld geven; een algoritme die de fout geeft van een afstandsconstraint waar de afstand tussen twee particles groter moet zijn dan de waarde d:

if (d < |Sparticle1 - Sparticle2|) {

//als de waarde van d groter is dan de afstand tussen particle1 en particle2 wordt deze code uitgevoerd:

E = 0

} else {

//als de waarde van d niet groter is dan de afstand tussen particle1 en particle2 wordt deze code uitgevoerd:

E = d - |Sparticle1 - Sparticle2|

}

E: fout (in m)

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

|Sparticle1 - Sparticle2|: de afstand tussen particle1 en particle2 (in m)

Aan dit voorbeeld is goed te zien waarom een normale formule niet volstaat om E uit te rekenen; zolang de onderlinge afstand tussen de particles groter is dan d is de constraint gehandhaafd, en is het resultaat 0 (want er is geen fout). Als de onderlinge afstand echter kleiner is dan d is er wel sprake van een fout, namelijk het verschil tussen d en de onderlinge afstand. bedenk dat E in geen geval een negatieve waarde aan kan nemen.

We gaan nu proberen te zorgen dat alle constraints gehandhaafd worden. Het is echter belangrijk dat constraints niet en één keer gehandhaafd worden. Als constraints in één keer opgelost zouden worden zou er impliciet van uitgegaan worden dat andere constraints geen invloed hebben op de constraint in kwestie. Vaak is het echter zo dat meerdere particles een constraint delen, en dat beide constraints veel invloed op elkaar hebben. Om te zorgen dat constraints wel zo veel mogelijk gehandhaafd worden word het handhaven van alle constraints enkele keren herhaald. Om te zorgen dat alle constraints evenveel prioriteit krijgen word de fout van een constraint vermenigvuldigd met een factor (f) die voor alle constraints gelijk is, en waarvan de waarde tussen de 0 en 1 ligt. Alweer zal ik het voorbeeld geven van de afstandsconstraint om te demonstreren hoe dit in zijn werking kan gaan:

relativePosition = getPosition(particle1) - getPosition(particle2)

//vector die van particle1 naar particle2 wijst

displacementLength = relativePosition.getLength() - length

//de afstand waarmee beide particles van elkaar af verplaatst moeten worden

displacementLength = displacementLength \* f

vector displacementVector = relativePosition;

//vector representing the displacement of particle2 relative to particle1

displacementVector = displacementVector / |displacementVector|

displacementVector = displacementVector \* displacementLength

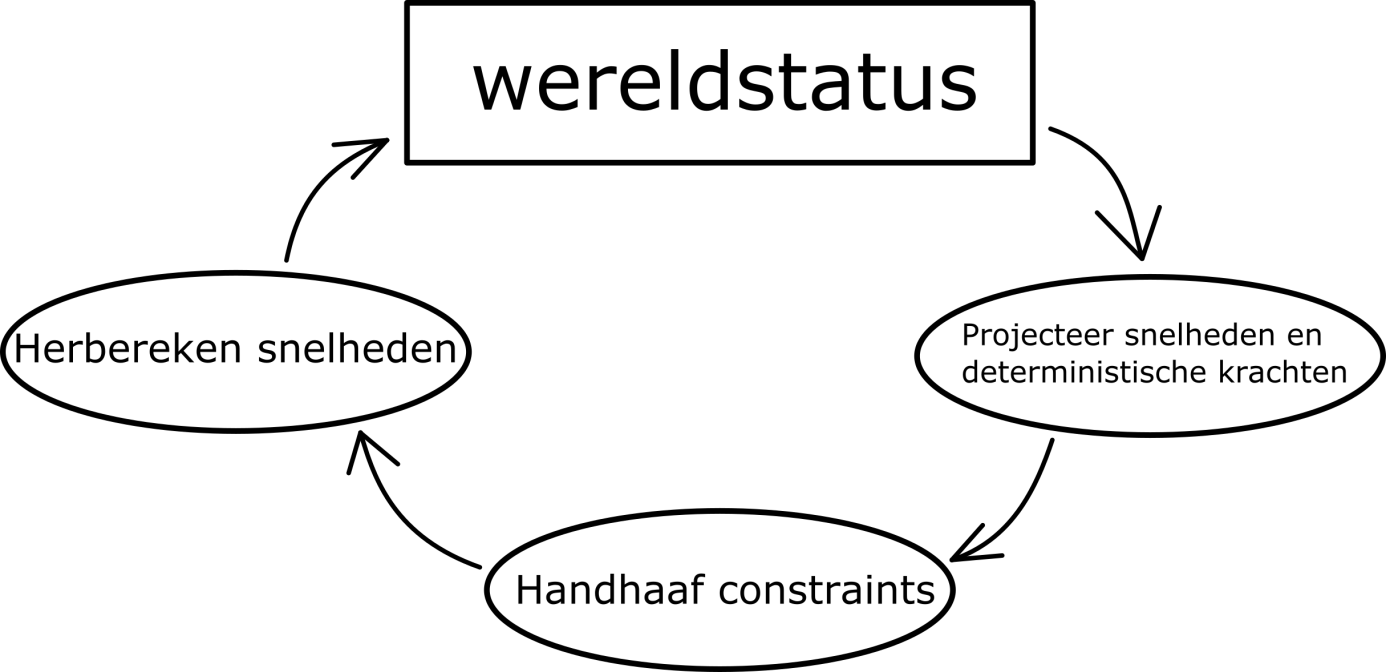
displace(particle1, displacementVector \* -1 / (1 + getParticle(particle1).getMass() / getParticle(particle2).getMass()))

displace(particle2, displacementVector \* 1 / (1 + getParticle(particle2).getMass() / getParticle(particle1).getMass()))

Merk dat bij het opstellen van dit algoritme gebruik gemaakt is van de wetten F = M \* a ^ F1 = -F2 ^ Sdelta = 1/2 \* a \* t^2. De constraint gedraagt zich dus correct naar deze wetten. Zo weten we ook hoe we constraints kunnen hanteren.

Globale werking:

Onderweg zijn een paar gedeeltes veranderd, dus ik zal een keer de uiteindelijke, globale werking uitleggen.



Naast de oorspronkelijke snelheid en positie van de particles hebben we de acceleratie van de particles nodig om iets over de eindpositie te kunnen zeggen. De acceleratie word bepaalt door de totale kracht op de particles, wat op zijn beurt de som is van alle individuele krachten die op de particle werken. Al deze krachten moeten dus benaderd worden. Omdat er een groot verschil zit tussen twee groepen krachten (deterministische krachten & constriants) moet het benaderen van de nieuwe positie in twee stappen verdeeld worden; eerst worden de deterministische krachten bepaalt, en worden toegepast op de posities van de particles samen met de beginsnelheid. Daarna worden de constraints toegepast. Dit gebeurt door alle constriants voor een bepaald gedeelte op te lossen, en dit te herhalen. Bij elke herhaling daalt fout in de constriants. We weten nu de positie van de particles nadat de oorspronkelijke snelheid, de deterministische krachten, en de constraints toegepast zijn. Dit is onze eindpositie. Voor de volgende simulatie moet echter een beginsnelheid vastgesteld worden, deze kan berekend worden aan de hand van de beginsnelheid, beginpositie, en de eindsnelheid.

Bronnen:

Thomas Jakobsen - Advanced Character Physics:

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

Matthias Müller - Position Based Dynamics:

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

Matthias Müller - Hierarchical Position Based Dynamics

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