Fysikalisk Simulering av Elastiska Material

Kalle Bladin
Erik Broberg
Emma Forsling Parborg
Martin Gråd

TNM085 - Modelleringsprojekt
Linköpings Universitet

14 mars 2014

Sammanfattning

Fysikbaserad animering av elastiska material kan sägas bestå av två skilda delar: fysikalisk simulering och grafikrendering. Materialen kan fysikaliskt beskrivas på flera olika sätt, men denna rapport är begränsad till att behandla endast en modell. Realismen i den simulering som presenteras är av stor vikt, varför fysikalisk korrekthet eftersträvas, och all beräkning grundar sig i vedertagna fysikaliska samband. För simuleringen undersöks två olika integreringsmetoder, Eulers stegmetod och Runge-Kuttas metod, och deras respektive för- och nackdelar med hänsyn till snabbhet och stabilitet.

De förstudier som presenteras härleder den ekvation som simuleringen bygger på. Denna grundar sig i sin tur på de fysikaliska samband som presenteras i Bakgrund. Förstudierna tar också upp modelleringen av det system som simuleras, och även de numeriska metoder som är implementerade. Resultaten från förstudierna valideras och ligger som grund för den slutgiltiga implementeringen.

De slutliga simuleringar som presenteras visar att den valda modellen kan efterlikna ett antal olika material och deras egenskaper. Utritningen sker med en bestämd frekvens, och noggrannheten i integreringarna bestäms utifrån denna. Undersökningen av integreringsmetoderna visar att Eulers stegmetod är att föredra för simuleringar av detta slag.

Innehåll

Figurer

Tabeller

1 Inledning

Simulering av elastiska material används flitigt i moderna fysikmotorer. Dessa kan användas både i datorspel och animeringsmjukvara där realism är en viktig faktor. Med hjälp av simulering behöver inte en animatör explicit hantera vissa rörelser hos objekten.

1.1 Bakgrund

Grundidén till systemet ligger i tidigare presenterade studier kring liknande ämnen. Partikelsystem i sin helhet kan anta många olika former. Elastiska material täcker flera sådana då det går att göra en väldigt generell modell för att simulera många olika material. Bristen med många fysikaliska simuleringar är att modellen endast täcker ett väldigt specifikt område som till exempel tygsimulering eller endast hårda material.

1.2 Syfte

Denna rapport syftar till att undersöka hur verklighetstrogna simuleringar av elastiska material kan göras i tre dimensioner med hjälp av mass-fjäder-dämparsystem. Simuleringarna som demonstreras är menade att kunna göras i realtid, och erbjuda en användare viss interaktion. Det är även viktigt att modellen ska kunna täcka många olika delar av Newtonsk fysik; allt från elastiskt tyg till tärningskast med hårda tärningar.

2 Metod

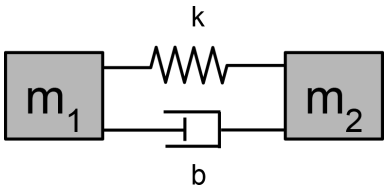
För utvecklingen av simuleringarna delades projektet upp i tre delar. Dessa var förstudier, implementering och stabilitets- och prestandatest.

2.1 Förstudier

I förstudierna ingick dels framtagandet av en kraftekvation, dels utvecklingen av MSD-system i en och två dimensioner. Detta grundar sig i den fysikaliska modelleringen samt numerisk lösning av differentialekvationer. Som verktyg användes MATLAB för att få en visuell återkoppling på kraftekvationen, med hjälp av programmets inbyggda renderingsfunktion.

2.1.1 Fysikalisk modellering

En modell som kan användas för simulering av elastiska material är ett så kallat ”mass spring damper system” (Gelenbe, Lent & Stakellari 2012) eller MSD-system; en modell som beskrivs av partiklar sammankopplade med fjädrar och dämpare.



Figur 1: Två partiklar med massorna m_1 och m_2, sammankopplade med en fjäder med fjäderkonstanten k och en dämpare med dämparkonstanten b.

Det enkla systemet i Figur 1 utgör grunden för den struktur som används i detta projekt. Genom att utöka systemet med partiklar och samma typ av kopplingar, kan många olika material och former beskrivas, exempelvis tyg, gelé, skumgummi eller liknande. Beroende på parametrarna fjäderkonstanter, dämparkonstanter, massor och fjäderlängder kan realistiska simuleringar av dessa material uppnås.

Som bakgrund till den fysikaliska modellen låg de grundläggande rörelse- och kraftlagarna som beskrevs av Newton (Halliday 2007). Newtons andra lag anges i (1),

$$\mathbf{F}(t) = m\mathbf{x}''(t) \tag{1}$$

där $F(t)$ är den resulterande kraften på en partikel, $x''(t)$ är partikelns acceleration, m är partikelns massa och t är tidsvariabeln. Newtons tredje lag summeras i ett citat:

> *”If A puts a force on B, then B puts a force on A, and the two forces are equal in magnitude and have opposite direction.”* (Halliday 2007)

Fjädrarna modellerades med Hookes lag som anges i (2),

$$F_k(t) = -k(l(t) - l_o) \tag{2}$$

där $F_k(t)$ är kraften som fjädern uträttar, k är fjäderkonstanten, $l(t)$ är fjäderns utsträckning, och l_0 är fjäderns vilolängd.

Dämparna modellerades utifrån (3),

$$F_b(t) = -bl'(t) \tag{3}$$

där $F_b(t)$ är kraften som dämparen uträttar, b är dämparkonstanten och $l(t)$ är fjäderns utsträckning.

2.1.2 Numerisk lösning av differentialekvationer

Som bakgrund till simuleringen och därmed numerisk integrering låg Eulers stegmetod och Runge-Kuttametoden (Ljung & Glad 2003). Nedanstående ekvationer beskriver numerisk integrering som krävs för lösning av första ordningens differentialekvationer, se (4).

$$\mathbf{x}'(t) = \mathbf{f}(t, \mathbf{x}(t)) \tag{4}$$

Här är $x(t)$ den sökta funktionen, och dess derivata är en funktion av tiden t samt funktionen $x(t)$ själv.

2.1.2.1 Euler

Eulers stegmetod för att lösa första ordningens differentialekvationer beskrivs i (5),

$$\mathbf{x}(t+h) = \mathbf{x}(t) + \mathbf{f}(t, \mathbf{x}(t))h \tag{5}$$

där h är tidssteget, och $\mathbf{f}$ fås från differentialekvationen. För att hitta funktionsvärdet vid tidpunkten $t + h$, alltså h tidsenheter efter det föregående, används funktionen $\mathbf{f}$ från differentialekvationen samt funktionsvärdet i det nuvarande steget, $x(t)$. Efter ett givet startvärde kan en numerisk lösning tas fram genom stegning med jämna intervall.

2.1.2.2 Runge-Kutta

Att lösa differentialekvationen med Runge-Kuttametoden innebär att flera derivator beräknas vid olika tidssteg. Detta ger en mer korrekt integrering jämfört med Eulers stegmetod (Jönsson 2010). Detta innebär att funktionen $\mathbf{f}$ har olika argument för att beräkna förändringshastigheterna $k1, ..., k4$. Funktionsvärdet i tidpunkten $t + h$ baseras på en viktad summa av förändringshastigheterna enligt (6) nedan.

$$\begin{aligned} \mathbf{x}(t+h) &= \mathbf{x}(t) + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \\ &\quad varest \\ \mathbf{k}_1 &= h\mathbf{f}(t, \mathbf{x}(t)) \\ \mathbf{k}_2 &= h\mathbf{f}(t + h/2, \mathbf{x}(t) + \mathbf{k}_1/2) \\ \mathbf{k}_3 &= h\mathbf{f}(t + h/2, \mathbf{x}(t) + \mathbf{k}_2/2) \\ \mathbf{k}_4 &= h\mathbf{f}(t + h, \mathbf{x}(t) + \mathbf{k}_3) \end{aligned} \tag{6}$$

Funktionen $\mathbf{f}$ har här givits olika argument som leder till de olika förändringshastigheterna $k1, \ldots, k4$. Med vikterna satta enlig (6) ovan ges i detta fall Runges metod (Jönsson 2010).

2.1.3 Kraftekvation

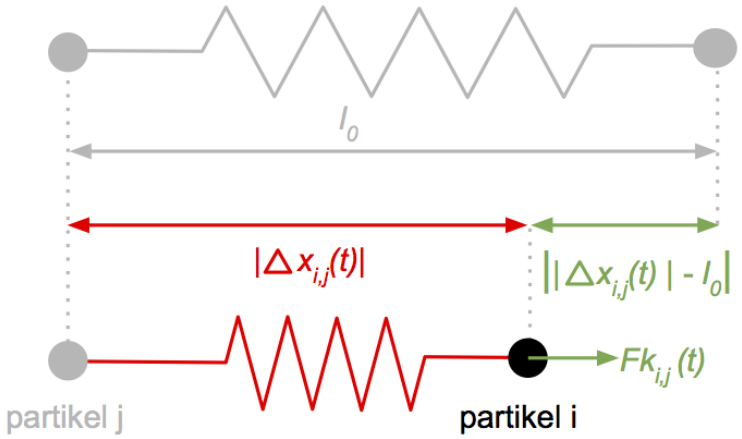
För att kunna beskriva hur kraften på partiklarna i ett MSD-system påverkar varandra härleddes en kraftekvation, se (7).

$$\mathbf{F}_i(t) = \sum_{j \epsilon A_i} (F_{k_{i,j}}(t) + F_{b_{i,j}}(t)) \Delta \hat{\mathbf{x}_{i,j}}(t)$$

$$varest \tag{7}$$

$$F_{k_{i,j}}(t) = -k_{i,j}(|\Delta \mathbf{x}_{i,j}(t)| - l_{0_{i,j}})$$
$$F_{b_{i,j}}(t) = -b_{i,j} \Delta \mathbf{x}'_{i,j}(t) \cdot \Delta \hat{\mathbf{x}_{i,j}}(t)$$
$$\Delta \mathbf{x}_{i,j}(t) = \mathbf{x}_i(t) - \mathbf{x}_j(t)$$

Ekvationen bestämmer kraften som påverkar partikel i. Ekvationen grundar sig i (1), (2) och (3) och beskriver hur den totala kraftvektorn för partikel i ges av summan av kraftpåverkan från mängden av alla sammankopplade partiklar A_i. Varje sammankopplad partikel ger en kraftkomponent som beräknas utifrån skillnad i position och hastighet samt med hjälp av kopplingarnas egenskaper. Absolutbeloppet $F_{k_{i,j}}$ av fjäderkraften togs fram med Hookes lag enligt (2). Fjäderns längd är här absolutbeloppet av skillnaderna mellan partiklarnas positioner, d.v.s. $|\Delta \mathbf{x_{i,j}}|$, se Figur 2. Absolutbeloppet $F_{b_{i,j}}$ av dämparkraften är proportionell mot hastighetsskillnaden enligt (3). Hastighetsskillnaden är den som sker i fjäderns riktning, varför hastighetsvektorn ortogonalprojiceras på $\Delta \mathbf{x_{i,j}}$, se Figur 3. Den resulterande storheten vektoriseras, varvid den ges riktningen $\Delta \hat{\mathbf{x}_{i,j}}$ mellan de två sammankopplade partiklarna, d.v.s. all kraft sker i fjäderns riktning.



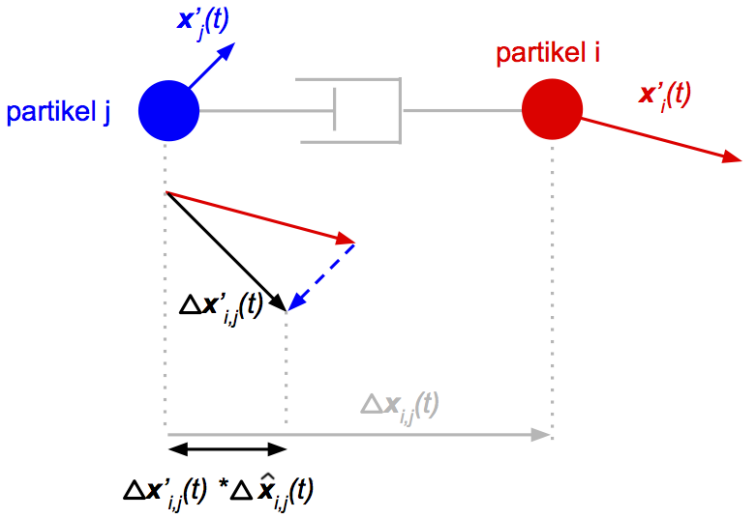
Figur 2: Fjäderkraften på partikel i från kopplingen till j.

Utifrån kraftekvationen kunde en differentialekvation för systemet ställas upp, se (8).

$$\mathbf{X}''(t) = \mathbf{G}(\mathbf{X}(t), \mathbf{X}'(t))$$

$$varest \tag{8}$$

$$\mathbf{X}''(t) = \begin{pmatrix} \mathbf{x}''_1(t) \\ \mathbf{x}''_2(t) \\ \vdots \\ \mathbf{x}''_N(t) \end{pmatrix}, \mathbf{G}(\mathbf{X}(t), \mathbf{X}'(t)) = \begin{pmatrix} \frac{\mathbf{F}_1(t)}{m_1} \\ \frac{\mathbf{F}_2(t)}{m_2} \\ \vdots \\ \frac{\mathbf{F}_N(t)}{m_N} \end{pmatrix}$$



Figur 3: Dämparkraften på partikel i från kopplingen till j

Detta är en andra ordningens differentialekvation. Accelerationen för systemets samtliga partiklar ges av $\mathbf{X}''(t)$ och (7) används för uttrycken $\mathbf{F}_1(t), \ldots, \mathbf{F}_N(t)$. Massan för partikel i ges av m_i. För att lösa ekvationssystemet numeriskt med metoderna som har introducerats (2.1.2.1 och 2.1.2.2) skrivs ekvationssystemet om till ett system av två första ordningens differentialekvationer, se (9).

$$\begin{cases} Y_1'(t) = H(Y_2(t)) \\ Y_2'(t) = G(Y_1(t), Y_2(t)) \end{cases}$$

$$varest$$

$$\begin{cases} Y_1(t) = X(t) \\ Y_2(t) = X'(t) \end{cases} \tag{9}$$

$$h(Y_2(t)) = Y_2(t)$$

Här introduceras tillståndsvariablerna $\mathbf{Y}_1$ och $\mathbf{Y}_2$ samt funktionen $\mathbf{H}$. För att applicera (5) eller (6) på detta system läggs ett extra argument till funktionen $\mathbf{f}$ för att funktionen $\mathbf{G}$ ska kunna användas. Argumentet t kan ignoreras då funktionerna inte beror på tiden separat.

2.1.4 Utveckling av MSD-system

För att skapa ett MSD-system i tre dimensioner, lades först en grund genom simulering och verifiering i en respektive två dimensioner.

2.1.4.1 En dimension

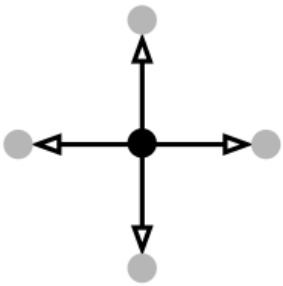
Då det enkla systemet med två sammankopplade partiklar skulle utökas med fler partiklar och kopplingar, krävdes en mer överskådlig notation för att beskriva systemet. I Figur 4 nedan introduceras den nya notationen av ett system med två sammankopplade massor.
I Matlab utvecklades först ett MSD-system i en dimension, där partiklar sammankopplades i serie enligt Figur 5 nedan.
Genom att simulera detta system med Euler-metoden samt undersöka hur partiklarnas positioner förändrades över tid, kunde implementeringen av den framtagna kraftekvationen enkelt verifieras.

Figur 4: Två sammankopplade partiklar

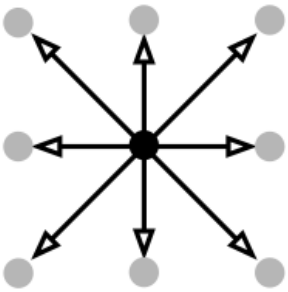
Figur 5: Fyra sammankopplade partiklar i serie

2.1.4.2 Två dimensioner

Nästa steg i utvecklingen av MSD-systemet var att implementera det i två dimensioner. För att kunna generera godtyckligt stora MSD-system krävdes en regel för hur partiklarna skulle kopplas ihop med varandra. Två exempel på en partikels möjliga kopplingar visas i Figur 6 nedan.



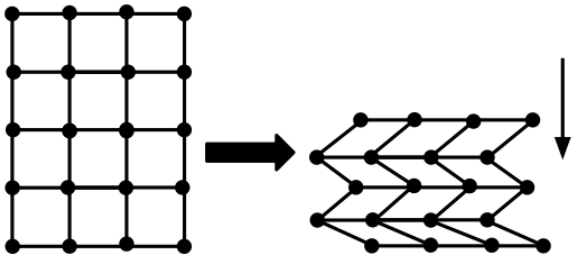
(a) Partikel med fyra kopplingar



(b) Partikel med åtta kopplingar

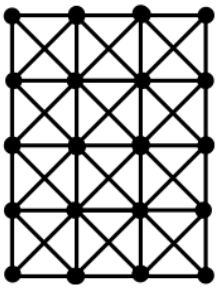
Figur 6: Två olika sätt att koppla samman partiklar

Kopplingsregeln i Figur 6a ger upphov till en struktur som är instabil. Detta beror på att kopplingarnas ändpunkter endast har en position; ingen riktning eller orientering. Detta medför att strukturen kollapsar, vilket visas i Figur 7.



Figur 7: Instabil struktur ges av kopplingsregeln i Figur 6a

Med regeln enligt Figur 6b erhålls däremot en struktur som är mer stabil, men fortfarande inte alltför komplex. Att materialet inte skulle kollapsa var en förutsättning för att elastiska material skulle kunna modelleras på ett trovärdigt sätt. Detta motiverade valet att använda kopplingar enligt Figur 6b. Strukturen som ges med denna regel redovisas i Figur 8 nedan.



Figur 8: Stabil struktur ges av kopplingsregeln i Figur 6

2.1.5 Indexering av partiklar och kopplingar

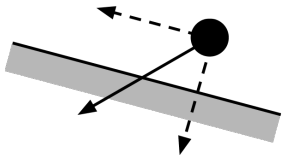
Ett stort problem som kopplingsmetoden i ovanstående stycke medförde var att ta reda på vilka partiklar som skulle höra till varje koppling. För att detta skulle kunna lösas för godtyckligt stora objekt, behövdes ett systematiskt sätt att tilldela kopplingarna deras partiklar. Detta löstes med en funktion som utifrån index till en viss koppling gav indexar till de två partiklar som den kopplade samman. Genom att låta funktionen genomlöpa alla kopplingar innan simuleringen genomfördes kunde indexeringen användas för att finna de rätta konstanterna (massorna) och variablerna (positionerna och hastigheterna) för de sammankopplade partiklarna. En liknande funktion implementerades för att finna triangelindexar som användes för renderingen.

2.1.6 Interaktion med omgivning

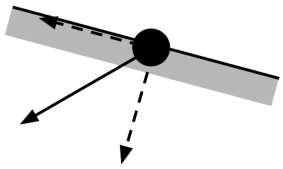
Gravitation simulerades genom att den globala tyngdaccelerationen på $9.82m/s^2$ adderades på samtliga partiklars acceleration i samband med beräkningen av krafterna. Tillsammans med detta implementerades hantering av kollision med ett markplan för att materialets beteende skulle kunna studeras under bekanta förhållanden (se Figur 9). Kollisionen implementerades genom en enkel algoritm.

Algoritm 1: Kollision

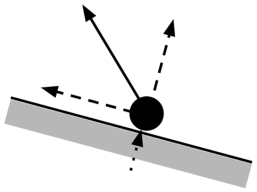
for *varje partikel* i **do**
  if *partikeln befinner sig under kollisionsplanet:* **then**
    - Spegla i:s hastighetsvektor i kollisionsplanet, se Figur 9c.
    - Ortogonalprojicera i:s positionsvektor upp på kollisionsplanet, se Figur 9c.
    - Multiplicera den komponent av i:s hastighetsvektor som är parallell med kollisionsplanet med en friktionskoefficient, se Figur 9d
    - Multiplicera den komponent av i:s hastighetsvektor, som är ortogonal med kollisionsplanet, med en elasticitetskoefficient, se Figur 9d.



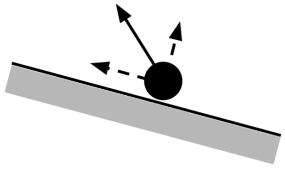
(a) Partikeln rör sig mot kollisionsplanet.



(b) Partikeln har kolliderat med planet.



(c) Partikelns position och hastighetsvektor ändras.



(d) Komponenterna multipliceras med deras respektive koefficient.

Figur 9: Två olika sätt att koppla samman partiklar

2.1.7 Implementering av kraftekvationen

Att implementera kraftekvationen för varje partikel i ett MSD-system enligt (7), innebär att kraftvektorn till varje partikel i beräknas. Med denna kraftvektor kan i sin tur accelerationen beräknas för varje partikel enligt (8). I detta projekt undersöktes två algoritmer för detta ändamål.

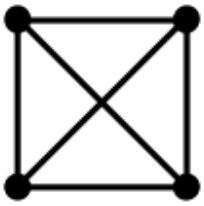
Algoritm 2: Algoritm baserad på partiklar

```
for varje partikel i do
    for varje partikel j, som är kopplad till i do
        f = kraften på i från kopplingen mellan i och j;
        F[i] += f;
    beräkna accelerationen genom att dividera F[i] med i:s massa;
    nollställ F[i];
```

Algoritm 3: Algoritm baserad på kopplingarna mellan partiklar

```
for varje koppling c (där c är kopplingen mellan partikel i och j) : do
    f = kraften på i från koppling c;
    F[i] += f;
    F[j] -= f;
for varje partikel i do
    beräkna accelerationen genom att dividera F[i] med i:s massa;
    nollställ F[i];
```

I Algoritm 3 utnyttjas Newtons andra lag, dvs det faktum att en koppling påverkar dess två ändpunkter med lika stora krafter, med enda skillnaden att de är motriktade varandra. På så sätt reduceras antalet beräkningar av kraftekvationen till hälften, vilket motiverade beslutet att Algoritm 3 användes i detta projekt. Läsaren kan övertyga sig själv om att antalet beräkningar av kraftekvationen halveras genom att betrakta följande enkla MSD-system.



Figur 10: Enkelt system med fyra hopkopplade partiklar

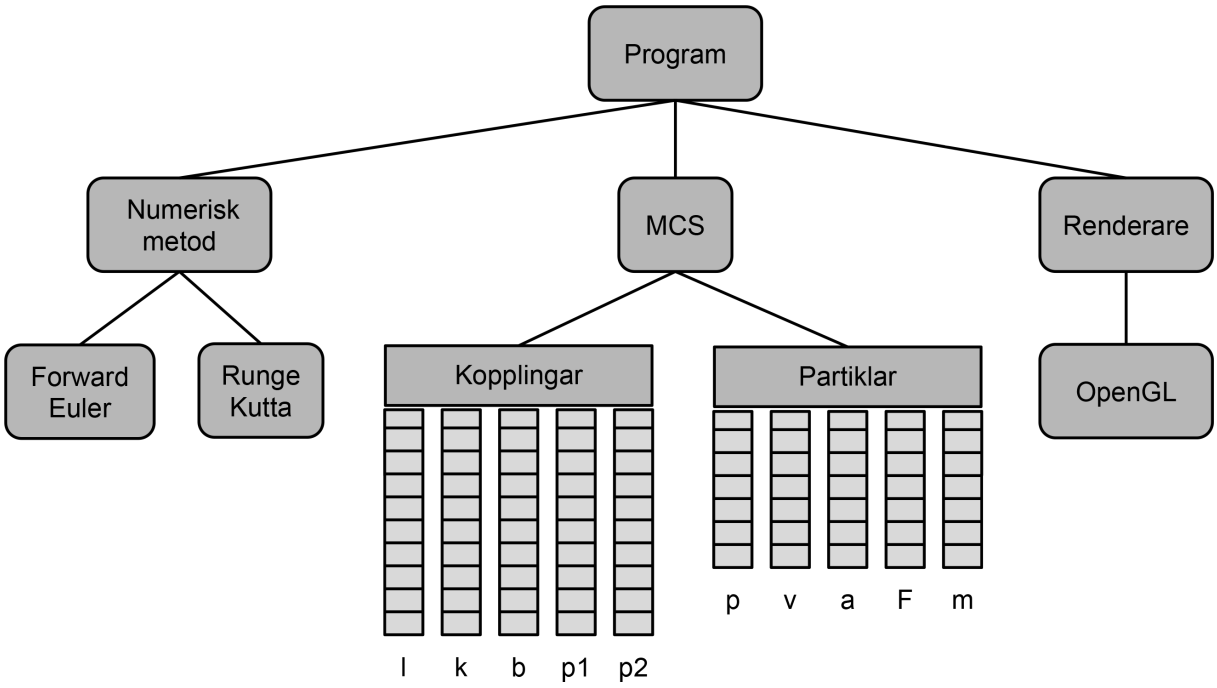
I Figur 10 har varje partikel tre hopkopplade grannar. Det totala antalet grannar blir således tolv, vilket är dubbla antalet kopplingar, som är sex.

2.2 Implementering

Programkoden från MATLAB porterades till C++ där systemet utvecklades vidare. Fördelen med C++ är att det är snabbt, objektorienterat (Skansholm 2012), samt att en utvecklare får mer kontroll, inte minst över grafikprogrammeringen. Grafikprogrammeringen gjordes med OpenGL som är plattformsoberoende och ger utvecklaren möjlighet att använda hårdvaruaccelererad grafik (Gortler 2012).

2.2.1 Grundläggande systemarkitektur

Att övergå till ett objektorienterat system innebar att en systemarkitektur behövde tas fram. För att ge en grundläggande förståelse av systemarkitekturen beskrivs denna med ett blockdiagram där de primära modulerna är separerade från varandra. Se Figur 11.



Figur 11: Övergripande systemarkitektur

Klassen MCS (Mass Connection System) är den klass som lagrar all huvudsaklig data. Den innehåller samtliga partiklar och kopplingar för ett system vilka definieras av information enligt Tabell 1 nedan.

Tabell 1: Lagrad data i kopplingar och partiklar

Typ	Namn	Förklaring	Typ	Namn	Förklaring
float	l	vilolängd	vec3	p	position
float	k	fjäderkonstant	vec3	v	hastighet
float	b	dämparkonstant	vec3	a	acceleration
int	p1	index till den ena partikeln	vec3	F	total momentan kraftpåverkan
int	p2	index till den andra partikel	float	m	massa

Vad som också framgår i Figur 11 är att all numerisk integrering beräknas i en separat modul vilket gjorde det enkelt att ändra integreringsmetod vid behov, då den var oberoende av resten av systemet.

2.2.2 Implementering av numeriska integreringsmetoder

Efter att accelerationen för varje partikel hade beräknats utifrån (8), approximerades deras positioner med en numerisk stegmetod. Den numeriska integreringen gjordes två gånger per tidssteg, då det var en andra ordningens differentialekvation som skulle lösas. Se ekvation (8). Nedan beskrivs hur de två olika integreringsmetoderna i systemet implementerades.

2.2.2.1 Eulers stegmetod

Med ett givet tidssteg beräknades den nya hastigheten utifrån accelerationen. Hastigheten användes i sin tur för att beräkna den nya positionen enligt (10).

$$\begin{aligned} v &= v_0 + a * h \\ p &= p_0 + v * h \end{aligned} \tag{10}$$

där a, v och p ges enligt Tabell 1. Variablerna v_0 och p_0 är hastigheten respektive positionen i föregående tidssteg. Steglängden är h.

2.2.2.2 Runge-Kutta

En algoritm för att simulera MSD-systemet med Runge-Kutta implementerades. Genom att kalla på kraftekvationen med olika steglängder kunde värdena $k1, ..., k4$ beräknas. SKRIVA NÅ MER HÄR?

2.2.3 Rendering

För att rendera MSD-systemet skickades hörnpunkter, en triangellista, en lista med färger för varje hörnpunkt samt en lista med normaler till grafikkortet via OpenGL. På grafikkortet kunde det elastiska materialet färgas. Dessutom kunde Phong shading implementeras för att simulera att modellen belyses från en viss riktning. Denna rendering gjordes 60 gånger per sekund.

2.3 Stabilitets- och prestandatest

För att jämföra stabiliteten i de två numeriska metoderna gjordes en enkel undersökning. För ett givet MSD-system och en viss steglängd hittades metodernas stabilitetsgränser genom en stegvis ökning av fjäderkonstanterna.

När stabilitetsgränser för de numeriska metoderna hade bestämts, behövde en avvägning mellan fysikalisk noggrannhet och renderingsfrekvens göras. För att simuleringen skulle kunna erbjuda responsiv interaktion och jämn animering togs beslutet att antalet tidssteg i simuleringen skulle anpassas för att

uppnå en bestämd utritningsfrekvens.

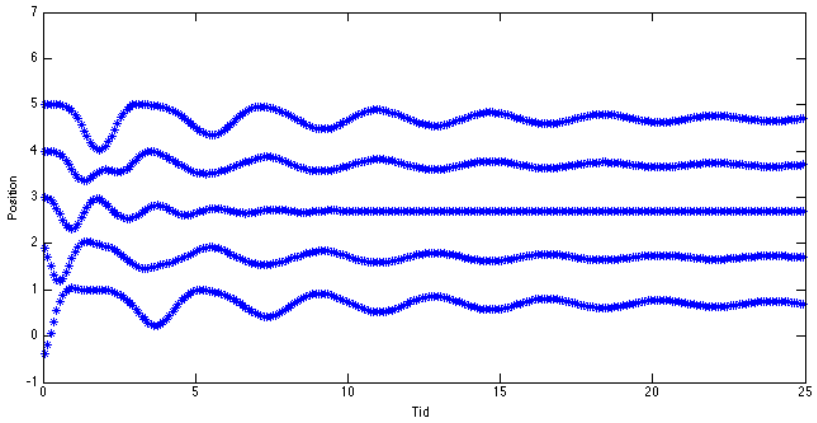
För att jämföra de två olika numeriska integreringsmetoderna som implementerades utfördes ett stabiltets- och prestandatest. Först användes Runge-Kuttametoden med ett visst antal tidssteg per utritning så att en jämförbar utritningsfrekvens erhölls. Därefter undersöktes hur många tidssteg per utritning som krävdes för att Eulers metod skulle ge samma utritningsfrekvens. Simuleringarna tog då lika lång tid att genomföra och stabilitetsgränsen kunde kontrolleras genom att öka fjäderkonstanterna eller dämparkonstanterna oberoende av varandra.

3 Resultat

Resultaten från förstudier och implementering presenteras nedan.

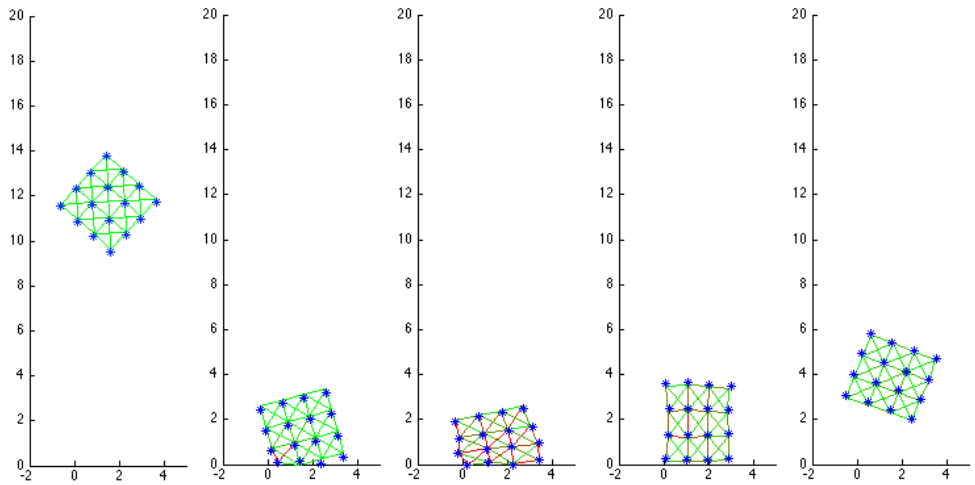
3.1 Resultat från förstudier

Resultatet från förstudierna i en dimension presenteras i Figur 12. Här illustreras fem sammankopplade partiklars samverkan efter att en av dem har givits en förskjuten startposition. Den kraft som bildas av denna förskjutning kan ses påverka de andra partiklarna och fortplantas genom systemet över tid. Krafterna ses också dämpas och tillslut avta, vilket stabiliserar systemet.



Figur 12: Endimensionellt MSD-system i MATLAB

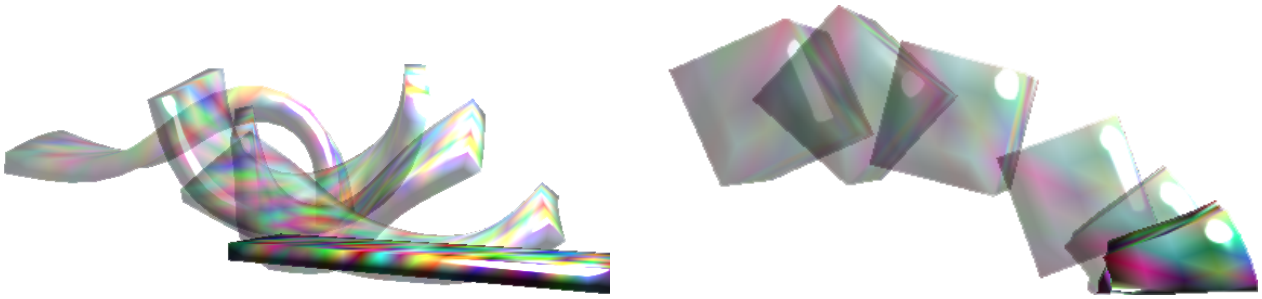
I Figur 13 åskådliggörs resultatet av förstudierna i två dimensioner. Här illustreras ett 4x4 MSD-system som faller från sin startposition och därefter kolliderar med ett markplan. Fjädrarnas inverkande krafter visualiseras genom att låta färgen bero på kopplingens längd. Kopplingen är grön i sitt viloläge och färgen går mot rött ju mer längden avviker från denna.



Figur 13: Tvådimensionellt 4x4 MSD-system i MATLAB

3.2 Resultat från implementering

I Figur 14a och 14b visas bilder från simuleringar av två olika material. Materialen skiljer sig åt både i form och parametervärden. Det första materialet har relativt höga fjäderkonstanter och låga dämparkonstanter, medan det andra materialets egenskaper är de motsatta.



(a) (b)

Figur 14: Resultat från implementering

3.3 Resultat från stabilitets- och prestandatest

Testet av stabilitetsgränser för de numeriska metoderna presenteras i Tabell 2 , och testet av stabilitet utifrån önskad prestanda presenteras i Tabell 3.

Tabell 2: Erhållna resultat för en bestämd steglängd.

	Simuleringar per utrotning	**Fjäderkonstant (för marginell stabilitet)**
Runge-Kutta:	4	10600
Euler:	4	5550

Tabell 3: Erhållna resultat för ett bestämt antal utritningar per sekund.

	Simuleringar per utritning	**Fjäderkonstant(för marginell stabilitet)**
Runge-Kutta:	7	40 000
Euler:	44	2150000

4 Diskussion

Den största anledningen till att Runge-Kuttametoden var mer beräkningstung var att mycket data behövde kopieras under simuleringen. Runge-Kuttametoden gav visserligen en mer korrekt simulering men eftersom det var kritiskt att med hög prestanda erhålla ett relativt stabilt system gjordes valet att använda Eulers metod i slutprodukten.

Valet av algoritm för kraftekvationen grundade sig i antalet gånger kraftekvationen behövde beräknas. Med Algoritm 3 behövde beräkningen göras hälften så många gånger. Dock krävde algoritmen att partiklarna loopades igenom en gång efter det att krafterna hade beräknats för att applicera accelerationen på dem. Trots detta hölls antalet beräkningar på en lägre nivå för Algoritm 3 än för Algoritm 2, varför Algoritm 3 valdes för slutimplementeringen.

Vid skapande av MSD-systemet gjordes begränsningar i val av dämpare med enbart dynamisk friktion. Att införa andra ekvationer för hur dämparna skulle modelleras skulle kunna vara en vidareutveckling av systemet. En möjlighet är att införa statisk friktion i dämparna för att simulera deformation i materialet så att det inte återfår sin ursprungliga form.

Vid utveckling av MSD-systemet underlättade MATLABs inbyggda renderingsfunktion. Med hjälp av den kunde kraftekvationen valideras utan att tid behövde spenderas på implementering av en egen renderingsfunktion. Med den tidiga visuella återkopplingen var felsökning lättare, vilket gav upphov till ett smidigare arbetsflöde och fokus kunde läggas på rätt del i början av arbetet.

En annan del av systemet som tål att diskuteras är hur andra former än rätblock skulle kunna ställas upp. Indexeringsfunktionen förutsätter att modellen är uppbyggd i rätblocksformation medan simuleringen i sig inte förutsätter något sådant. Fördelen är alltså att man kan implementera andra indexeringsfunktioner för att bygga upp modeller av andra former, som sfärer eller koner.

Indexeringsfunktionen i sig kunde också ha implementerats annorlunda för att göra arbetet med den lättare. I stället för en funktion som utifrån fjäderindex ger två partikelindexar kunde listan med partiklar loopas igenom vartefter en koppling kunde läggas till för partikelns alla grannar. Denna metod hade varit lättare att implementera men saknar fördelen att listan med kopplingar blir ordnad på ett systemetiskt sätt. Fördelen med den första metoden var att kopplingarna ordnades efter längderna, vilket underlättade proceduren att ändra dessa under körning. Möjligheten att veta exakt vilken koppling som tillhörde vilket index ger möjlighet till större kontroll.

Fortsatt utveckling av grafiken skulle främst vara införandet av texturer. Detta är en naturlig vidareutveckling då det skulle innebära införandet av en lista med UV-koordinater i stället för listan med färger som användes i renderingen. För detta skulle en ny indexeringsfunktion krävas för att ta fram vilken hörnpunkt som tillhör en viss UV-koordinat. Med texturer ges möjligheten att rendera mer verklighetstrogna och intressanta material. Rendering av andra element såsom markplan skulle också tillföra ett mervärde genom att förtydliga kollisionen med detta. Skuggkastning är också något grafiskt som skulle förtydliga interaktionen mellan materialet och markplan genom att ge en känsla för avstånd.

5 Referenser

Fausett, L.V , 2008, *Applied Numerical Analysis Using Matlab*, Pearson Prentice Hall.

Gelenbe, E, Lent, R, Stakellari, G, 2012, *Computer and Information Sciences II: 26th International Symposium on Computer and Information Sciences*, Springer-Verlag London Limited.

Gortler, S.J, 2012, *Foundations of 3D Computer Graphics*, The MIT Press.

Halliday, D, 2007, *Fundamentals of Physics*, 8th Rev edn. John Wiley Sons.

Jönsson, P, 2010, *Matlab beräkningar inom teknik och naturvetenskap*, Studentlitteratur AB

Ljung, L & Glad, T, 2004, *Modellbygge och simulering*, Andra upplagan, Studentlitteratur AB.

Skansholm, J, 2012, *C++ direkt*, Studentlitteratur AB.

A Beskrivning av systemet

A.1 Systemkrav

För att köra programmet krävs särskild hårdvara och mjukvara. Programmet är utvecklat på Mac OSX men bör teoretiskt kunna köras även på Windows- eller Linuxsystem.

A.1.1 Hårdvara

Programmet kräver ett grafikkort som stödjer OpenGL 3.3+.

A.1.2 Mjukvara

Utöver ramverket OpenGL (3.3+) kräver programmet vissa externa C++-bibliotek för att kompileras och köras. De bibliotek som krävs är GLEW och GLFW3.

A.2 Köra programmet

Programmet startas i terminalen/konsolen. När programmet körs kan användaren göra olika val i scenen med hjälp av knapptryck. Dessa listas här nedan:

A.2.1 Välja material/objekt:

1 - madrass
2 - tärning
3 - mask
4 - tyg
5 - glass

A.2.2 Ändra parametrar:

Klicka någon av kommandona nedan, och ange sedan värdet i terminalen/konsolen.
m - ändra massa
s - ändra fjäderkonstanter
l - ändra fjäderlängder
d - ändra dämparkonstanter
f - sätt kopplingarnas vilolängder till deras nuvarande längder

A.2.3 Styrning:

scroll/pekplatta - uppåt/nedåt - zooma in/ut
scroll/pekplatta - vänster/höger - vrida kameran
mellanslag - pausa simulering
mellanslag följt av höger- eller vänsterpil - rendera i lägre hastighet