Linköpings universitet

Modelleringsprojekt TNM085
Simulering av tyg

Johan Beck-Norén, johbe559
Andreas Valter, andva287
Axel Kinner, axeki412
Rasmus Karlsson, raska293
Hampus Axelsson, hamax109

2012-03-12

Sammanfattning

De metoder som är presenterade i denna rapport beskriver den bakomliggande teorin och den metodik som krävs för simulering av tyg, både för visning i realtid samt för exportering till andra renderingsprogram.

Den fysikaliska modell som implementationen grundar sig i, bygger på en partikelbeskrivning ordnat i ett rutnät där kraften som påverkar varje partikel fås av summan av alla verkande krafter på partikeln. Krafter som har implementerats är gravitations-, vind-, fjäder- och dämpningskraft. En enkel kollisionsmodell för kollisioner mot stationära objekt har även implementerats. Genom att introducera en maximal kraft som varje fjäder tål och sedan bryta av de fjädrar som påverkas av större krafter har det skapas flera möjligheter för att förstöra tyget.

Modellen har implementerats med hjälp av Verlet-integration method som är en robust approximationsmetod som är relativt enkel att implementera. På grund av antaganden gjorda i härledningen till Verlet har tidssteget valts att vara konstant. Även kraften som påverkar varje partikel antas ha en så liten förändring över tidssteget att den anses vara konstant.

I realtid har olika sätt att interagera med tyget tagits fram med en implementation i C++ med biblioteken GLUT och GLFW som bygger på OpenGL. Genom att projicera mot kameraplanet går det att ta tag i den partikel som är närmast muspekaren. Med detta verktyg har funktioner för att dra och skära i tyget implementerats.

Ett antal olika renderingsmetoder har även skapats för att på olika sätt visa hur systemet är uppbyggt och hur det uppför sig under simuleringen. Dessa är bland annat en utritning av alla partiklar, en generering av ytor mellan partiklarna samt speciella renderingsmetoder som är anpassade för att ge ett bra visuellt resultat när tyget går sönder.

Det finns även möjlighet att exportera trianglar, hörnpunkter och normaler och på så sätt kunna rendera tyget med hjälp av annan renderingsmjukvara. Med detta kan en avancerad rendering göras, som utnyttjar ray-tracing och avancerade material för att göra en ännu mer visuellt tilltalande rendering av tyget.

Innehållsförteckning

Figurer

Tabeller

Kapitel 1

Inledning

1.1 Inledning

Det finns väldigt många implementeringar av tygsimulering inom datorgrafik, både i spel och i filmer. Alla dessa implementeringar bygger i grunden på en av två olika metoder för att representera tyget. Det går antingen att beskriva tyget med hjälp av partiklar, sammankopplade med fjädrar, eller med en differentiell ytekvation. Bland implementationerna är det partikelrepresentationen som är den vanligaste. Med de algoritmer som skapas genom modellrepresentationen uppnås ett bra visuellt utseende, både för enkla modeller och för avancerade modeller. Inom realtidsrendering används ofta en enkel modell som bidrar till den visuella känslan, utan att ta speciellt mycket kraft från andra delar som har större vikt i renderingen av scenen. För tyngre implementationer där delar som kollisioner implementeras finns många vetenskapliga artiklar som beskriver metodiken, samt visar hur saker som GPGPU (general-purpose computing on graphics processing units) kan implementeras för att utnyttja att problemet är parallelliserbart och därför öka hastigheten på simulationen. Alla variabler som refereras till och används i ekvationer finns betecknade i Bilaga B.

1.2 Syfte

I kursen TNM085 - Modelleringsprojekt skall en dynamisk modell för ett fysikaliskt system skapas, med utgångspunkt i en enkel modell. Genom att sedan implementera en dynamisk visualisering av systemet skall systemets egenskaper studeras. Den modell som skall tas fram i denna rapport är en realtidssimulering av tyg, implementerad i C++ och renderad, både i realtid med hjälp av OpenGL eller offline med till exempel Maya. Tyget påverkas av yttre krafter som vind, gravitation och interaktioner från användaren, samt ett antal inre krafter mellan de partiklar som bygger upp tyget.

Kapitel 2

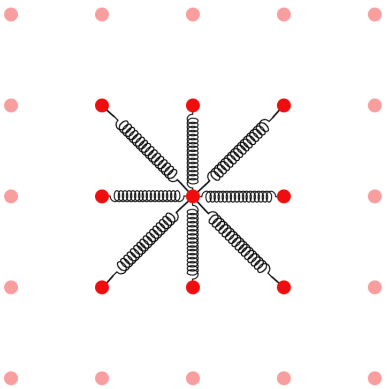
Simulering av tyg

2.1 Fysikalisk modell

Den fysikaliska metod som har skapats grundar sig i en partikelbeskrivning. Tyget kan ses som ett antal partiklar med massa, ordnade i ett rutnät (figur 2.1) med fjädrar kopplade mellan partiklarna. Modellen som använts är ett mekaniskt problem där kraften som påverkar varje partikel är en summa av alla krafter som påverkar partikeln. Tack vare den enkla modell som användes var det enkelt att bygga vidare och utöka antalet verkande krafter på tyget (2.1) (2.2).

$$\ddot{x} = \frac{1}{m}(F_{ext} + F_{int}) \tag{2.1}$$

$$F = Fs + Fb + Fd + Fg \Leftrightarrow F = F_{int} + F_{ext} \tag{2.2}$$



Figur 2.1: Fjäderkopplingar

2.1.1 Fjäderkrafter

Tyget består av två olika slags fjädrar, töjningsfjädrar och böjningsfjädrar som tillsammans motverkar yttre krafter och gör att tyget behåller sin area. Fjäderkrafterna är modifierade versioner av Hookes lag (2.3) och på grund av Newton III kommer två partiklar med en töjningskoppling eller böjningskoppling påverkas av samma kraft men med olika riktningar.

$$F = -k(x - x_0) \tag{2.3}$$

2.1.2 Töjningsfjädrar

Töjningsfjäderns uppgift är att motverka töjningar i tyget. Kraften som ges av töjningsfjädern är i grunden Hookes lag (2.3) med skillnaden att den endast påverkar simuleringen då tyget är längre än sin vilolängd. k_s i (2.5) bestämmer styvheten i tyget. Ett tyg med ett lågt värde på k_s ger ett väldigt elastiskt tyg, medan ett tyg med högt k_s-värde är väldigt styvt. Ett styvare tyg ger upphov till en mer instabil simulering eftersom felet mellan varje tidsteg blir större, då kraftförändringen mellan det nya och det gamla steget beror på k_s.

$$x_{ij} = x_j - x_i \tag{2.4}$$

$$F_s = k_s * (|x_{ij}| - L) * \frac{x_{ij}}{|x_{ij}|}, |x_{ij}| \geq L \tag{2.5}$$

2.1.3 Dämpningskraft

Dämpningskraften är en intern resistans som motverkar tygets rörelse. Den verkar endast på varje par av töjningsfjäder och beror på deras tillhörade partiklars relativa hastighet mot varandra (2.6). Att det inte finns någon dämpning mellan varje par av böjningsfjädrar är endast av kosmetiska skäl. Med hjälp av dämpningskraften kan tygets egenskaper ändras för att efterlikna allt från en tunn gummimatta till ett jeanstyg.

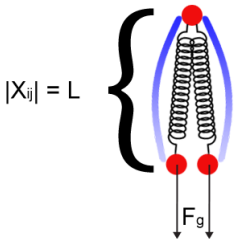
$$F_d = -k_d * (v_i - v_j) \tag{2.6}$$

2.1.4 Böjningsfjädrar

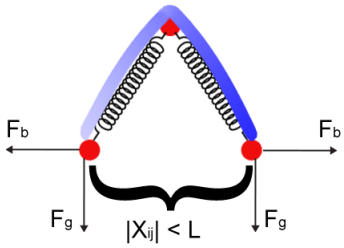
Böjningsfjädrar har introducerats för att motverka böjningar i tyget. När två partiklar närmar sig varandra uppstår en böjning av fjädern mellan dem, för att motverka denna böjning får varje partikel en kraft riktad bort från varandra. Ett fenomen som uppstår vid böjningar är när tyget viker sig. Där har böjningsfjädrarna i uppgift att motverka denna vikning (figur 2.2) (figur 2.3). För att böjningsfjädrarna skall kunna motverka vikningen måste fjädrarna vara kopplade med en partikels mellanrum (figur 2.4). Kraften är även här i grunden Hookes lag (2.3) med skillnaden att den endast påverkar simuleringen då sträckan mellan två partiklar är kortare än dess ursprungliga avstånd (2.7) (2.8).

$$F_b = k_b * (|x_{ij}| - L) * \frac{x_{ij}}{|x_{ij}|}, |x_{ij}| < L \tag{2.7}$$

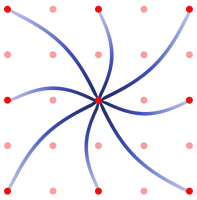
$$F_b = 0, |x_{ij}| > L \tag{2.8}$$



Figur 2.2: Vikt tyg utan böjningsfjädrar att motverka vikningen



Figur 2.3: Vikt tyg böjs tillbaka av böjningsfjäder



Figur 2.4: Böjningsfjädrar

En mer korrekt beskrivning av böjningskraften vore att använda samma typ av kraft som uppstår i bågen när en pilbåge böjer sig. Dock så ger en enkel fjädermodell som motverkar kompression tillräckligt bra resultat.

2.1.5 Kollisioner

För att beräkna ny hastighet och position när en kollision skett med statiska objekt så används en relativt enkel metod som beskrivs i *Physics for Game Development* (se Bilaga A). Metoden för varje partikel är som följer:

1. Räkna ut partikens nya position och hastighet med Verlet- och Euler- approximationsmetoderna.
2. Kontrollera om partikelns nya position är i ett statisk objekt. Om inte, gå till punkt 5.
3. Flytta partikeln så den hamnar utanför kollisionsobjektet.
4. Räkna ut partikelns korrigerade hastighet, $\dot{x}[n\text{-}1]$, och använd denna för räkna ut partikelns nya position.
5. Ge partikeln de nya värdena för hastighet och position.

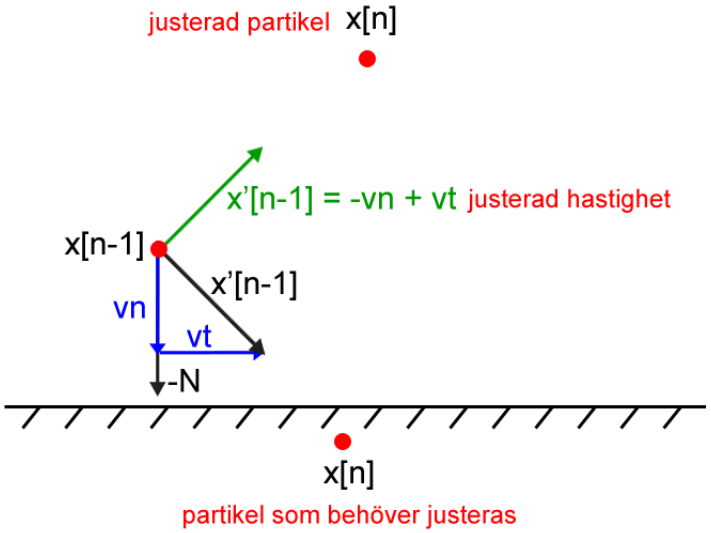
I steg 2 skall man undersöka om partikeln har hamnat i ett objekt. Denna kontroll är beroende på vad för objekt den hamnat i. Om fallet är en sfär så är det väldigt enkelt och beskrivs enligt (2.9).

$$(x[n] - c)(x[n] - c) < r^2 \tag{2.9}$$

Steg 3 är den mest komplicerade delen av ekvationen. För att få en korrekt beskrivning så skall kollisionspunkten mellan partikel och sfär beräknas, samt tiden *t* för partikeln att röra sig

från den gamla positionen. Därefter sätts *dt* till *dt - t* och gamla positionen *x[n-1]* sätts till kollitionspunktens position för att använda dessa till beräkningarna i steg 4. Dock implementerades inte steget som beskrevs ovan, istället har en mycket förenklad version använts som fungerar bra för små tidssteg. Denna metod går tillbaka till den gamla positionen och beräknar normalen till ytan utifrån den. På så vis kan beräkning av kollisionspunkten undvikas.

I steg 4 räknas normalen till ytan utifrån *x[n-1]*. Med hjälp av normalen går det sedan att räkna den korrigerade hastigheten och till slut den nya positionen *x[n]* (figur 2.5).



Figur 2.5: Resulterande kraftriktning för en triangel påverkad av vind

2.2 Approximationsmetoder

Under projektets gång har ett antal olika approximationsmetoder jämförts. Metoden måste vara både enkel att implementera, stabil och utan krävande uträkningar. Detta eftersom den modell som implementerats för tyget är väldigt känslig för illa valda konstanter och förändringar i tidssteg. För att med säkerhet hela tiden ha ett tidssteg som fortfarande håller approximationen stabil används en fix steglängd i simuleringen.

Den första och enklaste metoden som implementerades var Eulers stegmetod (2.10). Eulers stegmetod använder sig av tangenten till positionskurvan för att hitta nästföljande position. Dock gav metoden ett system som inte var tillräckligt stabilt och valdes därför bort. Försök gjordes även med en förbättrad Euler som är en Taylor-utveckling av ordning två (2.11).

$$\begin{cases} x[n+1] = x[n] + dt * \dot{x}[n] \\ \dot{x}[n+1] = \dot{x}[n] + dt * \ddot{x}[n] \\ \ddot{x}[n] = \frac{1}{m}(F_{int} + F_{ext}) \end{cases} \tag{2.10}$$

$$\begin{cases} x[n+1] = x[n] + dt * \dot{x}[n] + \frac{dt^2 * \ddot{x}[n]}{2} \\ \dot{x}[n+1] = \dot{x}[n] + dt * \ddot{x}[n] \\ \ddot{x}[n] = \frac{1}{m}(F_{int} + F_{ext}) \end{cases} \tag{2.11}$$

Den metod som sedan implementerades var en positionsberoende Verlet-approximation. Verlet-metoden utgår från en exakt uträkning av var en partikel kommer befinna sig, givet att tidigare position, hastighet och acceleration är kända (2.12). Med antagandet att accelerationen är konstant under varje tidssteg kan hastigheten approximeras med hjälp av Euler (2.13). Insättning i (2.12) ger (2.14). Genom att anta att tidssteg och acceleration är konstant kan (2.14) skrivas om och delar kan substitueras mot den vänstra termen i (2.12). Detta resulterar i en positionsberoende Verlet-approximation (2.15).

$$x[n+1] = x[n] + \dot{x}[n] * dt[n] + 0.5\ddot{x}[n] * dt^2[n] \tag{2.12}$$

$$\dot{x}[n] = \dot{x}[n-1] + \ddot{x}[n-1] * dt \tag{2.13}$$

$$x[n] - x[n-1] = \dot{x}[n] * dt[n-1] - 0.5 * \ddot{x}[n-1] * dt^2[n-1] \tag{2.14}$$

$$x[n+1] = x[n] + (x[n] - x[n-1]) + \ddot{x} * dt^2 \tag{2.15}$$

2.3 Implementering

Den ursprungliga modellen med approximation gjordes i två dimensioner i MATLAB. Efter att ha erhållit en numeriskt stabil modell fortsatte arbetet med att implementera simuleringen i C++. Simuleringen har i huvudsak renderats i realtid med OpenGL. Valet att rendera i realtid gjordes för att simpel interaktion med simuleringen skulle vara möjlig. Funktionalitet finns även att exportera sekvenser av bildrutor till OBJ-filer för importering till annan renderingsmjukvara, till exempel 3DS Max eller Maya.

2.3.1 OpenCL

För problem där väldigt många uträkningar görs samtidigt är CPU:n (Central Processing Unit på moderkortet) inte optimal. Den är skapad för att göra stora och tunga uträkningar och kan bara räkna på en eller ett par operationer samtidigt, beroende på hur många kärnor den har. För sådana operationer är det bättre att använda sig av GPU:n (Graphics Processing Unit, grafikkortets processor) som är optimerad för att göra samma operation på väldigt många komponenter samtidigt.

Den fysikaliska modell som skapats utför samma kraftberäkningar för varje partikel. Detta är ett typiskt exempel på en algoritm som lämpar sig väl att utföra på GPU:n istället för på CPU:n. De nackdelar som finns med att utföra dessa operationer är att all data som skall användas måste flyttas från CPU:n till GPU:n. Detta är en tung uppgift som ofta tar längre tid att utföra än att utföra beräkningarna på CPU:n. Det finns alltså inget att tjäna med en implementation på GPU:n om det inte finns tillräckligt mycket data som skall räknas ut. I de fall där resultatet av de uträkningar som görs skall visas med hjälp av grafik behöver grafikkortet ändå den uträknade datan i renderingsprocessen. I dessa fall är det möjligt att flytta över all information till grafikkortet en gång och sedan utföra både rendering och uträkning med grafikkortet, utan kostsamma förflyttningar av data.

2.3.2 Slitning

Modellen som skapats stödjer att tyget på olika sätt slits sönder. Detta genomförs genom att kraften mellan två partiklar endast räknas ut då fjädern mellan dem är hel. Med denna modell

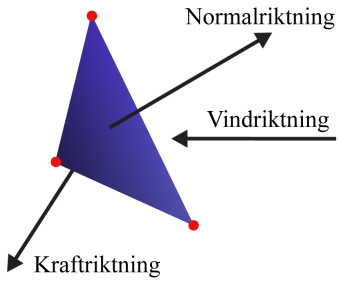
har ett antal olika metoder för att slita söner tyget introduceras, som lösningar på verkliga begränsningar som finns hos tyg.

En verklig fjäder håller inte för krafter med för stor amplitud. Därför introducerades en maximal kraft som varje fjäder kan utstå. När kraften överstiger den maximala kraften släpps fjäderns kopplingar. För att skapa en mer verklighetstrogen effekt vid sträckning genom dragning introducerades en begränsning där varken den dragna partikeln eller dess åtta närmaste grannars fjädrar kan gå av. Detta resulterar i att tyget slits av på ett naturligt sätt. Även stationära partiklar och dess åtta närmaste grannar har begränsningarna som gör att de inte kan gå av. Detta beror på att det är dessa partiklar som oftast utsätts för störst påfrestning och därmed som lättast går av.

2.3.3 Global vind

En global vind som verkar på tyget över tiden har implementerats i simuleringen. Vinden har en kraft samt en riktning i tre dimensioner. Vindberäkningarna sker på normalriktningarna för tygets alla trianglar. Det uppstår problem med hur tygets rörelse ska påverkas av vinden eftersom en triangel inte har en absolut position i världen. Istället beräknas vindens kraftpåverkan för en triangel enligt (2.16) (figur 2.6). Denna kraft fördelas sedan likformigt över de partiklar som bygger upp triangeln. Beräkningarna genomförs för alla trianglar i varje tidssteg av simuleringen.

$$Vindkraft = Normalriktning * (Normalriktning \cdot Vindriktning) \tag{2.16}$$



Figur 2.6: Resulterande kraftriktning för en triangel påverkad av vind

2.3.4 Interaktioner

Ett antal olika interaktioner med simuleringen har implementerats. Med hjälp av muspekaren och höger musknapp kan användaren greppa enskilda partiklar och förflytta dem. Tyget följer med i rörelsen på ett realistiskt vis. Om en partikel dras på ett sådant sätt att tyget spänns hårt brister tygets olika fjäderkopplingar, vilket skapar intrycket av att tyget rivs av. Vänster musknapp tillåter användaren att ta bort den greppade partikelns kopplingar till partiklar ovanför i tyget. Detta resulterar i att användaren kan skära i tyget.

Transformationsfunktion

För att implementera dessa interaktioner krävs en funktion för att transformera musens position från det två-dimensionella visningsfönstret till OpenGLs världskoordinater i tre dimensioner. För att åstadkomma detta görs ortogonal-projektion på kameraplanet vid de tillfällen denna transformation behövs. Funktionen beräknar vilken partikel i tyget som är närmast muspekarens koordinater genom olika vektorberäkningar och sätter denna partikels status till greppad. Denna transformationsfunktion anropas endast då höger musknapp trycks ned för att förflytta en partikel, eller varje tidssteg då vänster musknapp trycks ned för att skära i tyget.

Förflyttning av partiklar

Funktionen för att förflytta en given partikel anropas varje tidssteg då höger musknapp hålls nedtryckt. Om partikeln har status greppad sätts partikelns kraft till noll och partikelns position till muspekarens position. Partikeln får status ogreppad och släpps om kraften den utsätts för överstiger ett visst värde eller om användaren släpper upp höger musknapp. Detta för att behålla stabilitet i systemet.

Skära i tyget

Funktionen för att skära i tyget anropas varje tidssteg då vänster musknapp hålls nedtryckt. Även funktionen för att greppa partiklar körs här varje tidssteg, den anropas innan funktionen för att skära i tyget. För varje partikel som greppas anropas en funktion som tar bort partikelns kopplingar uppåt i tyget.

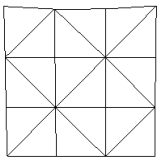
2.4 Rendering

För att kunna rendera och göra beräkningar på tyget lagras en mängd olika data. För varje partikel i tyget lagras partikelns massa, position, postionen ett tidsteg tidigare, hastighet, kraft och normalriktning. Data om hur trianglar är uppbyggda mellan partiklarna tillsammans med trianglarnas normalriktningar lagras i matriser med hjälp av pekare. Dessutom lagras ett antal booleska data för varje partikel, om partikeln är stationär, om partikeln har en fjäderanslutning i given riktning och om partikeln är greppad av muspekaren.

OpenGL används för att hantera grafiken i programmet, ihop med biblioteken GLFW och GLUT, eftersom OpenGL i sig inte har några funktioner för att skapa fönster och hantera diverse plattformsberoende anrop. För att göra ett enkelt användargränssnitt så används GLUT för att rita ut text i fönstret. Tyget kan renderas ut på fem olika vis och varje metod beskrivs här kortfattat.

2.4.1 Trianglar

För att rita ut tyget som en solid yta med textur så används trianglar. Varje triangel är uppbyggd av tre partiklar och varje partikel kan finnas med i upp till åtta trianglar. För att se slitningar på tyget ritas inte de trianglar som ligger i samma plan som en trasig koppling. För att det skall se bra ut har tyget en speciell uppsättning av trianglar i ett diamantformat mönster (figur 2.7).



Figur 2.7: Trianglar ordnade i diamantmönster

2.4.2 Kopplingar

För att bättre visualisera tygets slitning skapades en renderingsmetod som ritar ut alla hela kopplingar. Varje koppling ritas ut som en linje mellan de två partiklar den tillhör. Om en koppling har gått sönder ritas den inte ut. Denna metod visar tydligt hur slitningar påverkar tygets struktur. Man kan även se varje koppling som en söm i tyget, denna söm försvinner när kopplingen går sönder.

2.4.3 Trianglar och kopplingar

Denna renderingsmetod använder egenskaper från de två metoderna beskriva i 2.10.1 och 2.10.2. Renderingsmetoden ritar tyget med trianglar för varje triangel som är hel och de trianglar som är trasiga ritas istället ut med kopplingar. Detta skapar effekten att trådar ser ut att hänga mellan hålen i tyget.

2.4.4 Övriga renderingsmetoder

För att se hur varje triangel är uppbyggd finns det ett läge som visar de yttre linjerna för varje triangel. Ibland är det även intressant att se varje partikel i varje tidssteg och därför finns även en renderingsmetod för detta. Denna ritar ut en punkt för varje partikelposition i 3D-rymden.

2.4.5 Rendering offline

För att få en visuellt snyggare rendering än den som gjordes i OpenGL skrevs en objektexporterare. För varje tidssteg sparades positioner för tygets alla hörnpunkter, trianglar och normaler till en OBJ-fil. Denna sekvens av OBJ-filer importerades sedan till renderingsmjukvaran Maya. I Maya renderades tyget med renderaren Mental Ray och en global HDR-belysning(High Dynamic Range) användes (figur 2.8).

Figur 2.8: Tyget renderat offline i Maya med mentalRay

Kapitel 3

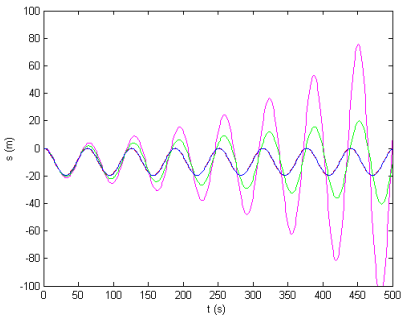
Resultat

3.1 Resultat

Projektet har resulterat i en modell som simulerar tyg och dess rörelse med påverkan av inre och yttre krafter. Med implementationen i C++ har ett gränssnitt skapats, vilket har resulterat i en interaktiv simulering där tyget kan interagera med enkla objekt, vind och slitning (figur 3.1).

En implementation av de tre approximationsmetoder som undersökts i rapporten gjordes i MATLAB med ett massa-fjädersystem med en vikt (figur 3.2).

Figur 3.1: Tyget i vila, upplyft och sönderskuret. Renderat i realtid i OpenGL.



Figur 3.2: Magenta är Euler, grön är förbättrad Euler, blå är Verlet och numeriska lösningen är röd.

3.2 Diskussion

3.2.1 Approximationsmetoder

Bland de approximationsmetoder som undersöktes och jämfördes var det endast Verlet-metoden som var stabil och därför den enda rimliga metoden för den implementering som gjordes. Även om metoden med förbättrad Euler (2.10) gav en bättre approximation så ansågs den inte vara tillräckligt robust. De begränsningar som sattes när Verlet definerades behövdes dock även undersökas för att se om metoden var rimlig för att simulera modellen.

De största nackdelarna med Verlet var att både kraften och tidssteget antogs vara kontant. I det system som togs fram är krafterna inte konstanta mellan tidsstegen, alltså kan inte metoden användas utan att ytterligare antaganden görs. Detta resulterade i att kraften fick ses som konstant mellan varje tidssteg och om skillnaden i kraft är tillräckligt kan skillnaden försummas. Tidssteget går, till skillnad från kraften, att styra och den kan därför tvingas vara konstant. I det fall att de antaganden som är gjorda är godtagbara är Verlet-metoden en robust approximationsmetod som är väldigt enkel att implementera.

3.2.2 Implementering

Försök gjordes att utnyttja parallelliserbarheten i problemet och flytta uträkningarna till grafikkortet med hjälp av OpenCL. På grund av problem i interaktionen mellan OpenCL och OpenGL blev det ingen prestandaökning i den implementation som gjordes, främst på grund av att data skickades från grafikkortets arbetsminne till moderkortets RAM-minnen och tillbaka för varje bildruta. För att få en förbättring i prestanda krävdes en buffert för de datat som skall skickas mellan OpenCL och OpenGL. Denna buffert lyckades inte gruppen skapa.

3.2.3 Rendering

Att trianglar försvinner helt när tyget slits är inte en önskvärd metod men väldigt enkel. En mer korrekt metod skulle vara om partiklarna delade på sig men saknade kopplingar mellan varandra. Att skapa ytterligare partiklar skulle dock medföra stora förändringar och prioriterades därför bort.

Bilaga A

Referenslista

- Bourg David M., *Physics for Game Development*, ISBN 0-596-00006-5
- Shreiner Dave, *OpenGL Programming Guide*, ISBN 0-321-55262-8
- Munshi Aaftab, Gaster Benedict, Mattson Timothy G., *OpenCL Programming Guide*, ISBN 0-321-74964-2

Bilaga B

Beteckningstabell

Tabell B.1: Beteckningar

Beteckning	Storhet	Enhet
F	Total kraft	N
Fs	Töjningskraft	N
Fb	Böjningskraft	N
Fd	Dämpningskraft	N
Fg	Gravitationskraft	N
F_int	Interna krafter	N
F_extt	Externa krafter	N
k	Fjäderkonstant	
k_d	Fjäderkonstant dämpning	
k_s	Fjäderkonstant töjning	
k_b	Fjäderkonstant böjning	
x	Position	m
x_0	Ursprungsposition	m
$\dot{x}$	Hastighet	m/s
$\ddot{x}$	Acceleration	m/s^2
dt	Tidssteg	s
L	Fjäderns vilolängd	m
c	Centrumpunkt för sfär	m
r	Radie	m