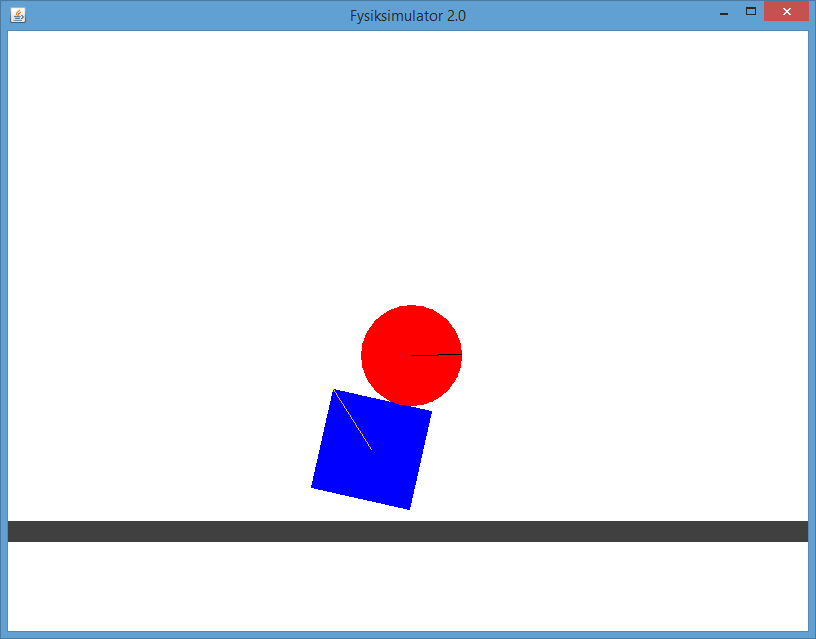
 GYMNASIEARBETSRAPPORT

Läsåret 2014/2015

DATASIMULERING UTAV FYSIK   
  
  
  
 

Gruppmedlemmar: Björn Detterfelt, NA12C   
Ludvig Siwe, NA12C   
Dario Rostirolla, NA12C

Lärare: Anders Sjöstrand  
Medbedömare: Christoffer Johnsson

**Abstract**Computers are something that everyone comes in contact with everyday and the following study aims to research what simplification have to be made to allow a modern day computer to simulate the physics of a two dimensional world in real time. This was achieved by designing and creating a program that simulates the physics of rigid bodies in a two-dimensional and drag free environment.   
The simulator is impulse based and it handles multiple collision detection and solution. Some simplifications were made in order for the simulator to run smoothly. The factor that influences the quality of the simulation the most is the calculating power of the machine running the simulator and this research found that modern day computers can simulate physics in real time effectively without the need of too many major simplifications.

**Keywords:** Physics; Simulation; Programming; Computers;

Innehållsförteckning

[Inledning 1](#_Toc406488251)

[Syfte och frågeställningar 1](#_Toc406488252)

[Teoretisk bakgrund 1](#_Toc406488253)

[Metod 3](#_Toc406488254)

[Att bestämma strukturen 4](#_Toc406488255)

[Att utveckla fysiksimulatorn och anpassa strukturen 4](#_Toc406488255)

[Att utverdera arbetet 4](#_Toc406488255)

[Resultat 4](#_Toc406488256)

[Diskussion 4](#_Toc406488258)

[Underrubriker anpassas efter vilka resultat som ska redovisas i t.ex. enkät, experiment/intervju/provtagning etc. 4](#_Toc406488259)

[Källförteckning 4](#_Toc406488260)

Bilagor  
Bilaga 1: Fysiksimulatorn

# Inledning

Dagens datorer skiljer sig väldigt mycket ifrån de stora högen av kretskort och halvledare från bara 30 år sedan, först och främst i sin beräkningskraft. Dagens datorer kan utföra enorma beräkningar på ett fåtal millisekunder och därför undrade vi hur svårt det kunde vara för en modern dator att utföra en realistisk simulering av fysik i real tid samt vilka förenklingar detta krävde.

## Syfte och frågeställningar

Syftet är att försöka avbilda en förenklad tvådimensionell simulation av verkligheten i realtid på en vanlig dator genom att tillämpa våra kunskaper inom fysik, matematik och programmering samt att undersöka svårigheter med att simulera verkligheten.

Frågeställningarna är:

* Vilka förenklingar behöver göras?
  + Friktion
  + Luftmotstånd
  + Aggregationstillstånd
* Vilka tekniska begränsningar finns och hur hanteras dem?
  + Noggrannhet på tal
  + Beräkningar per sekund

## Teoretisk bakgrund

Skalärprodukt mellan vektorerna a och b räknas ut med:

där θ är den minsta vinkeln mellan dem. Svaret blir en skalär, alltså ett reellt tal. Efter lite härledning hittas följande formel:

där X är koordinatmatrisen för Y är koordinatmatrisen för och n är vektorernas dimension.

Vektorprodukt mellan vektorerna a och b räknas ut med:

där θ är den minsta vinkeln mellan dem. Svaret blir en vektor som har en rät vinkel mot de båda originalvektorerna. Vektorprodukten är antikommutativ och är egentligen bara definierad tre- och sju dimensioner. Då den används i två dimensioner fås en vektor som bara existerar i den tredje dimensionen, som inte existerar, och därför används den nya vektorns längd.

På samma sätt används också vektorprodukten mellan en vektor och en skalär, där skalären kan beskrivas som vektorn i den tredje dimensionen, som inte existerar, och då blir svaret en vektor. Ekvation för blir:

där X är koordinatmatrisen för Y är koordinatmatrisen för och ekvationen för :

där X är koordinatmatrisen för Y är koordinatmatrisen för den nya vektorn. (Gaul, 2013)

Fysiksimulatorn använder följande formler och storheter (Gaul, 2013):

En punkt mellan variabler symboliserar skalärprodukt, ett kryss symboliserar vektorprodukt och multiplikationen symboliseras inte.

Hastigheten i en punkt *p* i en roterande stel kropp räknas ut med:

där är vektorn mellan mittpunkten och punkten *p*, är vinkelhastigheten [rad/s], är kroppens hastighet [m/s] och är hastigheten i *p* [m/s].

Hastighetsdifferensvektorn hos de kollisionspunkter, 1 och 2, hos två kolliderande objekt räknas med:

där är motsvarande punkts hastighet [m/s], är motsvarande punkts vinkelhastighet [rad/s] och är vektorn från motsvarande objekts mittpunkt till motsvarande kollisionspunkt.

Kontakthastigheten i kollisionspunkten ges av följande skalärprodukt:

där är hastighetsdifferensen[m/s] och är kollisionsnormalen.

Impulsen som appliceras på de två kolliderande objekten ges av följande formel:

Där täljaren bestämmer storleken på impulsen beroende på restitutionskoefficienten och hastighetsdifferensvektorn och nämnaren tar hänsyn till massan och tröghetsmomentet hos objekten.

Hastighetsförändringen för ett objekt betecknas enligt:

där är impulsen[kgm/s] och är massan[kg].

Vinkelhastighetsförändringen för ett objekt i en punkt p beräknas med:

där vektorn mellan punkten p och objektets mittpunkt, är impulsen[kgm/s], och är tröghetsmomentet [kgm²].

Vid beräkning av hastighets- och vinkelhastighetsförändringen på det andra   
objektet inverteras impulsen:

Datorer använder sig av floating point för att kunna spara och räkna extremt stora och extremt små tal på ett effektiv sätt. Problemet med floating points är dock att de inte kan hantera bråkdelar särskilt bra. I bas 10 har vi tiondelar, hundradelar, tusendelar osv. men i bas 2, som är basen datorer använder, används det halvor, fjärdedelar, åttondelar, sextondelar osv.

Detta gör att ett tal som 0.1 i bas 10 tecknas som 0.0001100110011... i bas 2, där "0011" fortsätter till oändligheten. Datorer kan bara spara ett visst antal siffror ( 23 på 32-bitars datorer, 56 på 64-bitars) vilket gör att det oändliga talet "kapas".

Allt detta gör att summan av till exempel 0.1 och 0.2 blir 0.30000000000000004, alltså inte 0.3 och denna effekt kallas ”floating point error”. (Floating Point Numbers – Computerphile, 2014)

# Metod

Arbetet strukturerades upp enligt följande:

1. Struktur bestäms och nödvändiga delar identifieras.
2. Fysiksimulator utvecklas enligt den förbestämda strukturen.
3. Strukturen anpassas dynamiskt och problem diskuteras, lösningar söks med hjälp av relevant källmaterial.
4. Arbetet utvärderas och repeteras från steg 2 tills tillfredställande resultat uppnåtts.

***Att bestämma strukturen***

De tre mest betydelsefulla valen som påverkar strukturen angår antal beräkningstrådar och hur fysiken hanteras. Antal trådar bestämmer hur många beräkningar programmet kan utföra samtidigt.

Flera trådar ger en möjligt noggrannare och mer optimal simulering, däremot är simuleringar i en tråd enklare och tar potentiellt mindre tid att utveckla. Med tanke på tidsbegränsningen bestämdes det att fysiksimulatorn skulle använda sig av en enda tråd.

Fysiken kan hanteras på flera olika sätt. Oftast används dessa tre:

* Beräkningar som baseras på att alla kroppar består av noder som sitter ihop med elastiska fjädrar som följer Hooks Lag. Används för mjukkroppsberäkningar.
* Beräkningar som baseras på fysiklagar med begränsningar.
* Impulsbaserade beräkningar.

På grund av tidsbegränsningen valdes det att inte använda mjuka kroppar. Sedan valdes impulsbaserade beräkningar för att det kändes enklare.

***Att utveckla fysiksimulatorn och anpassa strukturen***

Strukturen bryts ned till logiska delar för att kunna implementeras i programmeringsspråket Java. Koden testkörs och felsöks. Strukturen och formlerna anpassas till de fel och problem som uppstår. För att veta hur strukturer och formler borde anpassas används både förkunskaper och Gauls arbete.

***Att utvärdera arbetet***

Utvärderar genom jämförelse med andra etablerade fysik simulatorer t.ex. Algodoo av Algoryx Simulation AB eller med uppskattningar och beräkningar avgörs hur nära verkligheten simulationen är med hjälp av förkunskaper.

# 

# Resultat

I fysiksimulatorn är det möjligt att lägga till flera objekt som plan, rektanglar och cirklar.  
Objekten har massa, masscentrum, hastighet, acceleration, vinkelhastighet, vinkelacceleration och tröghetsmoment.  
Objekten kan bestå av olika material som trä, betong, gummi, is, glas och stål.   
De olika materialen har olika studs- och friktionskoefficienter.  
Vi har även lagt till en experimentell material som vi har döpt till "bounce" som återger fullständigt elastiska stöt.  
  
Fysiksimulatorn hanterar både statisk- och dynamiskfriktion samt som den kan hantera kollision mellan flera objekt och flera plan.

Simuleringen sker genom att simulera relativt stora tidssteg åt gången vilket ger en bra approximering av vad som borde hända men med mycket mindre beräkningar för att tillåta simulatorn att köras i realtid. Storleken på tidsstegen kallas dt(delta tid) och kan justeras efter hur noggrann man vill att simulationen ska vara. Ett tidssteg betäcknas som en uppdatering.

Under en uppdatering räknar simulatorn ut nya positioner, hastigheter och rotationer enligt följande formler:

        nästaPosition = position+hastighet\*dt+(gravitation^2 \*dt)/2

        nästaHastighet = hastighet+gravitation\*dt

        nästaRotation = rotation+rotationsHastighet\*dt

Efter det räknar den ut om det nu sker några kollisioner mellan objekt. Om den hittar en kollision bedömer den först om objekten är på väg in i varandara genom att titta på deras relativa hastighet och sedan räknar den ut en kollisionsnormal, kollisionspunkt och penetrationsdjup. Kollisionsnormalen bestämmer riktningen på den impuls som appliceras på objekten. Storleken på impulsen bestäms utifrån objektens hastigheter, rotationshastigheter, massor och kollisionspunkten. Efter att den applicerat impulsen använder den penetrationsdjupet för att flytta isär objekten lite i förhållande till deras massa för att motverka floating point errors.

Kollisionsdelen av en uppdatering upprepas flera gånger beroende på hur många rörliga objekt det finns för att få en bättre noggrannhet på simulationen.

# Diskussion

I de kommande styckena analyseras de olika förenklingar som har utförts.

## Friktion

Fysiksimulatorn hanterar friktion genom att ha satt upp olika friktinoskoefficienter mellan alla materialkombinationer vi kan hantera. Trä mot trä har t.ex. den statiska friktionskoefficienten 0,375. Detta är det mest verklighetstrogna sättet att hantera friktionen, men det finns två väldigt stora problem med den och dessa gör det svårt att veta om friktionen verkligen stämmer.

Det första problemet är att friktion beror på mer än bara materialen som är i kontakt med varandra t.ex. ojämnheter utöver det vanliga, fuktighet som kan fungera som smörjmedel och temperatur. Detta gör att när man räknar ut friktionskoefficienter så blir de alltid lite olika och därför finns det inget riktigt rätt svar. En följd av detta är att den data som finns skiljer sig åt.

Det andra problemet är att friktion och friktionskoefficienter inte är väldigt användbart i stora delar av fysiken. Det finns därför ingen data alls om friktionskoefficienter mellan många materialkombinationer. Fysiksimulatorn hanterar st kombinationer och både statiska- och dynamiska koefficienter för dessa, men friktionskoefficienter kunde bara hittas för ungefär hälften av dem och de andra kombinationerna behövde uppskattas.

## Luftmotstånd

Beräkning av luftmotstånd är väldigt komplicerad och kräver att det tas hänsyn till sådant som luftens viskositet, fuktighet, temperatur. Faktorer som ett objekts densitet och utseende är också avgörande för hur beräkningen görs.

Det blir helt enkelt väldigt stora och komplicerade differentialekvationer. Med avseende på hur mycket beräkningskraft det skulle krävas för att lösa dessa ekvationer samt faktumet att vi själva saknar verktygen för att hantera dem så bestämde vi att utesluta luftmotstånd fråm simulatorn.

## Aggregationstillstånd

Hanteringen av olika aggregationstillstånd kräver att det tas hänsyn till flera olika variabler som viskositet, fuktighet, temperatur och även andra egenskaper, beroende på aggregationstillståndet.

Olika aggregationstillstånd hanteras på olika sätt vilket gör att det egentligen skulle behövas flera fysiksimulatorer i en för att kunna hantera olika aggregationstillstånd, i och med att de beter sig så olika från varandra.

På grund av detta, med avseende på våra tidsbegränsningar, valde vi att låta bli allt annat förutom fasta kroppar eftersom de är både enklare att hantera och är enklare att relatera till verkligheten.

## Beräkningar per sekund

Fysiksimulatorns beräkningskraft begränsas utav den dator symulatorn körs på.

Simulatorn beräknar och ritar resultatet med jämna mellanrum varje sekund. Hur stora dessa mellanrum är går att bestämma men om för stora eller för små tidsteg kan ha stor påverkan på simulationen.

Det kan till exempel hända att ett föremål rör sig snabbare än vad simulatorn kan beräkna kollision, vilket gör att föremålet kan röra sig genom andra fasta kroppar utan att påverkas, dvs utan att simulatorn märker av det. Detta sker då tidstegen är för få. Om man gör som så att tidstegen är fler kan detta leda till att simulatorn kraschar eftersom delar av simuleringen hinner inte beräknas i tid. Dessa fel i simuleringen skulle kunna åtgärdas genom att använda en dator med bättre beräkningskraft eller genom att inte simulera i realtid.

## Noggrannhet på tal

Floating point errors är något som påverkar alla beräkningar i fysiksimulatorn, speciellt när det kommer till kollisionen. Då felen bygger upp över tid måste vi försöka korrigera dem. Det gör vi med en positionskorrigering:

Om penetrationsdjupet i en kollision är väldigt litet antas det vara en floating point error och föremålen flyttas isär lite längs med kollisionsnormalen med hänsyn till deras massor.  
Utan denna korrigering skulle objekten sakta men säkert sjunka igenom varandra på grund av gravitationen.

Vi använder också datatypen Double för att lagra tal istället för datatypen Float då Double är 64bitar jämfört med Float som är 32bitar vilket innebär att talen blir mycket mer noggranna och floating point errors påverkar mindre. Ett exempel på skillnaden i noggrannhet och storleken på floating point errors hos Float och Double är summan av 0.1 och 0.2 vilket i Float blir 0.30000001192092896 och i Double blir summan 0.30000000000000004.

## Slutsats

Det finns ett del förenklingar och uppskattningar som behöver göras för att fysiken ska kunna simuleras i realtid, men de flesta av de har inte något stort inflytande över det faktiska resultatet, vilket gör att verklighetstrogen fysik kan simuleras i realtid på en vanlig dator så länge man anpassar vissa variabler enligt förhållandena i simulationen.

# Källförteckning

**Elektroniska källor**

Gaul, R. 2013. *How to Create a Custom Physics Engine.* <http://gamedevelopment.tutsplus.com/series/how-to-create-a-custom-physics-engine--gamedev-12715> [2015-03-19]

*Floating Point Numbers – Computerphile.* <https://www.youtube.com/watch?v=PZRI1IfStY0> [2015-03-19]

**Fysiksimulatorn**

Ladda ner den på:

<https://dariorostirolla.se/ga>