

En interaktiv simulering av biljard



Ivar Gorenko
Casper Larsson
Viktor Larsson
Antonio Tällberg Ilestad

15 mars 2022

Abstract

The aim of this project was to implement a physical model of a two-dimensional particle system and converting the differential equations to a discrete method using numerical method. Using the discrete method a visual and interactive implementation of a pooltable game was made based on the physics of the particle system. The particle system simulates translation, rotation, friction and collision of the particles within a contained area. The visual and interactive implementation was made in both MATLAB and WebGL with some limitations on the collision and ball properties.

These methods of converting a physical system to a mathematical model are the foundation of all graphical visualisations and this shows one way of implementing it.

Innehåll

Abstract	i
Figurer	iv
Tabeller	v
List of Algorithms	vi
1 Inledning	1
1.1 Syfte	1
1.2 Avgränsningar	1
2 Beskrivning av systemet	2
2.1 Fysikalisk modell	2
2.2 Translation	2
2.3 Rotation	3
2.4 Friktion	4
2.4.1 Glidande friktion	4
2.4.2 Rullande friktion	5
2.5 Kollision	6
2.5.1 Kollision med vägg	6
2.5.2 Kollision mellan bollar	8
3 Implementation	9
3.1 Numerisk metod	9
3.2 Implementation i MATLAB	9
3.2.1 Kollision	10
3.2.2 Friktion	10
3.3 Implementation i WebGL	11
4 Resultat	12
4.1 MATLAB	12

4.2 WebGL	13
5 Diskussion	15
5.1 Implementationsval	15
5.2 Vidareutveckling	15
6 Slutsats	17
Litteraturförteckning	18
A Appendix	19
A.1 Tabell över numeriska värden	19
A.2 Uppsättning av programvaran	19
B Matematiska bevis	20

Figurer

2.1	Kraften $F(t)$ uppdelad i komposanter med vinkeln θ	3
2.2	Rullande friktion för en boll	5
2.3	Hastighet av en boll över tid. Då friktionen för glidning är större än rullning minskas hastigheten längsammare efter att $\frac{5}{7}$ av starthastigheten nåtts	6
2.4	Kollision mellan boll och vägg	7
2.5	Hastighetsförlust för en boll vid kollision med väggar	7
2.6	Hastighetsriktning för två bollar före samt efter kollision	8
4.1	Uppställning i MATLAB för biljardspel	12
4.2	Slutliga uppställningen i WebGL	13
4.3	Resultat efter ett skott mot biljardbollarna, tillsammans med menyn	13
4.4	Alla bollar har satts i hålen och spelet är färdigspelat	14
B.1	Vinkeländring vid kollision med långsida	20
B.2	Vinkeländring vid kollision med kortsga	20

Tabeller

A.1 Numeriska värden använda i implementationen	19
---	----

List of Algorithms

1	Algoritmen för kollision med vägg	10
2	Algoritmen för kollision mellan två bollar	10
3	Algoritmen för bestämmandet av friktionen	10

Kapitel 1

Inledning

Fysikalisk modellering kan användas för att skapa approximationer av fysikaliska samband. Genom användning av diskreta numeriska metoder som Eulers stegmetod kan dessa modeller omformas till något som kan beräknas och renderas med ett datorprogram. Detta utgör grunden för alla grafiska visualiseringar som bygger på fysiska modeller, till exempel tygsimuleringar och vätskesimuleringar.

1.1 Syfte

Syftet med arbetet är att använda modelleringstekniker för att skapa en modell av ett tvådimensionellt partikelliknande system och omvandla detta till en numerisk metod. Denna metod används för att skapa en grafisk visualisering av modellen som är interaktiv.

Detta utförs genom att implementera en fysikalisk modell för ett biljardspel där modellen hanterar bollarnas interaktion med varandra och bordet. Denna modell omvandlas till en diskret Euler metod som används för att skapa en interaktiv visualisation av biljardspelet.

1.2 Avgränsningar

Projektet har en fast tidsram och utförs av en mindre grupp på fyra personer. Simuleringen renderas i realtid för att en användare ska ha möjligheten att interagera med den. Därav introduceras följande avgränsningar:

- Bollar överför ej rotation mellan varandra då de kolliderar.
- Endast mittpunkten av bollarna tas till hänsyn för att utföra beräkningarna.
- Stöt på bollarna antas träffa mitt på bollen, så ingen skruv uppstår.
- Bollarna kommer antingen att rulla eller glida beroende på hastighet, de kan inte rulla och glida samtidigt.

Dessa avgränsningar underlättar utvecklingen av simuleringen samt kalkyleringen under rendering vilket gör en realsidrendering möjlig.

Kapitel 2

Beskrivning av systemet

Här beskrivs de fysikaliska delarna som ingår i systemet.

2.1 Fysikalisk modell

Systemet är uppbyggt av femton bollar placerade i form av en triangel på ett vinkelrätt plan med väggar som begränsar dess rörelse. Endast bollarnas mittpunkt tas till hänsyn vid uträkningar. En boll får en rörelsemängd efter en stöt från ett biljardkö med en viss kraft. Bollen rör sig i en rak linje tills den kolliderar antingen med en av de andra femton bollarna eller en av ytans väggar. Bollarna har en friktion vilket får dess rörelsemängd att minska med tid. Systemet kan delas upp i fyra delar, translation, rotation, friktion och kollision. Vid translation används Newtons kraftekvation. Vid rotation överförs linjär hastighet till vinkelhastighet. Vid friktion hanteras två olika krafter, glidningsfriktion och rullningsfriktion. Vid kollision av bollar överförs momentkraft enligt Newtons tredje lag. För beskrivningen för de fysikaliska delarna avses de ingående delarna som finns på en boll, med undantag över kollisionen mellan två bollar där båda tas till hänsyn.

2.2 Translation

Translationen bestäms av hastighetsvektorer vilket fås av att integrera accelerationen i x - och y -axlarna, där accelerationen bestäms med Newtons kraftekvation[1]

$$a(t) = \frac{1}{m} F_{res}(t) \quad (2.1)$$

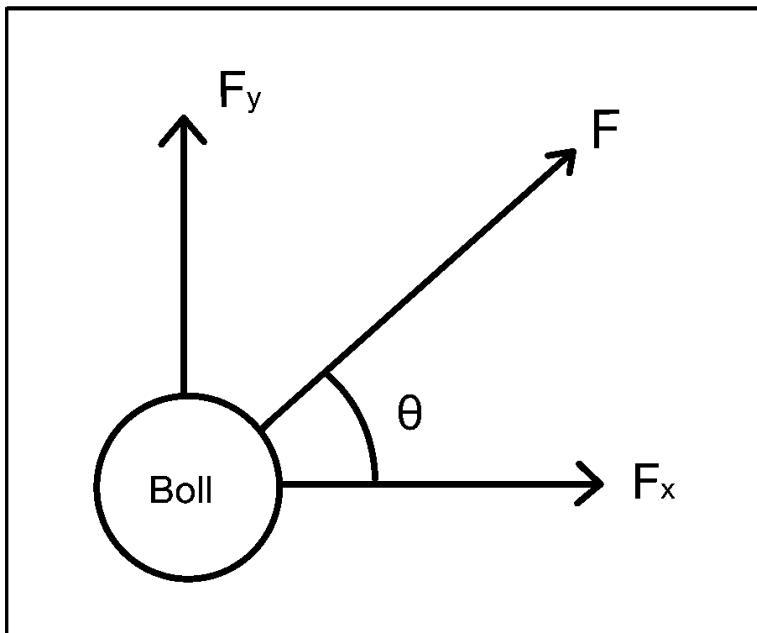
där $a(t)$ står för acceleration, $F_{res}(t)$ står för summan av alla inverkande krafter på en boll och m står för bollens massa. Bollarna får en initial stöt med en viss kraft och vinkel vilket ger $F(t)$ utifrån Ekvation 2.2 där I är impulsen och Δt är tiden då köt nuddar bollen vilket antas vara ett tidssteg i den tidsdiskreta implementationen. Kraften delas upp i x - och y -led och Ekvationerna 2.3 samt 2.4 används för att ta reda på dessa krafter. Se Figur 2.1 för illustration över komposantuppdelning och definitionen av vinkeln θ .

$$I = F(t)\Delta t \quad (2.2)$$

$$F_x(t) = \cos \theta \cdot F(t) \quad (2.3)$$

$$F_y(t) = \sin \theta \cdot F(t) \quad (2.4)$$

$F(t)$ är den applicerade kraften och θ är vinkelns för kraftens riktning. Accelerationerna tas ut med Ekvation 2.1 för att få acceleration i x -led och y -led.



Figur 2.1: Kraften $F(t)$ uppdelad i komposanter med vinkelns θ

2.3 Rotation

Efter en initial stöt rör sig bollen genom glidning vilket sedan övergår till rullning. Detta beror av den friktion som uppstår mellan bollen och bordet som skapar ett kraftmoment som ger upphov till rullningen [2]. Friktionen beskrivs noggrannare i avsnitt 2.4.

En boll som glider på ett vinkelrätt plan har en friktionskraft $F_f(t)$ enligt Ekvation 2.5 där μ är friktionskoefficienten, m är massan på bollen och g är tyngdaccelerationen.

$$F_f(t) = \mu mg \quad (2.5)$$

Då bollen glider utan ingående kraft, vilket är fallet efter den initiala stöten, är friktionskraften den enda kraften som påverkar bollen. Det ger med Newtons andra lag Ekvation 2.6 för en glidande boll, där $v'(t)$ är tidsderivatan för hastigheten.

$$F_f(t) = mv'(t) \quad (2.6)$$

Kraftmomentet $\tau(t)$ som skapas ges av Ekvation 2.7 då bollarna är sfäriska med momentarmen r som är bollens radie. Kraftmomentet vid rotation ges av Newtons andra lag i Ekvation 2.8 där I är bollens tröghetsmoment och $\omega'(t)$ är tidsderivatan för vinkelhastigheten[3].

$$\tau(t) = rF_f(t) \quad (2.7)$$

$$\tau(t) = I\omega'(t) \quad (2.8)$$

Tröghetsmomentet för en solid sfär ges av Ekvation 2.9. Genom att kombinera Ekvation 2.7, 2.8 och 2.9 erhålls Ekvation 2.10 som kan förenklas med Ekvation 2.6 och utbrytande av massan till Ekvation 2.11.

$$I = \frac{2}{5}mr^2 \quad (2.9)$$

$$F_f(t) = \frac{2}{5}mr\omega'(t) \quad (2.10)$$

$$v'(t) = -\frac{2}{5}r\omega'(t) \quad (2.11)$$

Genom att integrera 2.11 erhålls Ekvation 2.12, där v_i är initial hastighet och v_f är slutlig hastighet. Då en boll börjar glida med hastighet v_0 utan att rotera kommer rotation börja då bollens hastighet nåtts enligt 2.13, där $v_r(t)$ är nuvarande hastigheten för bollen, vilket ger Ekvation 2.14 som ger specialfallet 2.15. Detta ger slutligen att en boll endast kommer glida i början av rörelsen, tills hastigheten har nått $\frac{5}{7}$ av ursprungshastigheten, och kommer då börja rulla.

$$v_f(t) - v_i = -\frac{2}{5}r(\omega_f(t) - \omega_i) \quad (2.12)$$

$$v_r(t) = \omega(t)r \quad (2.13)$$

$$v_r(t) - v_0 = -\frac{2}{5}v_r(t) \quad (2.14)$$

$$v_r(t) = \frac{5}{7}v_0 \quad (2.15)$$

När en boll väl börjat rulla förhåller sig den linjära hastigheten till vinkelhastigheten enligt Ekvation 2.16[3].

$$v(t) = \omega(t)r \quad (2.16)$$

2.4 Friktion

Friktionen som påverkar en boll kan delas upp i två delar: En glidande friktion som uppstår när bollen börjar flytta sig direkt efter en stöt och en friktion som uppstår när bollen rullar över planet.

2.4.1 Glidande friktion

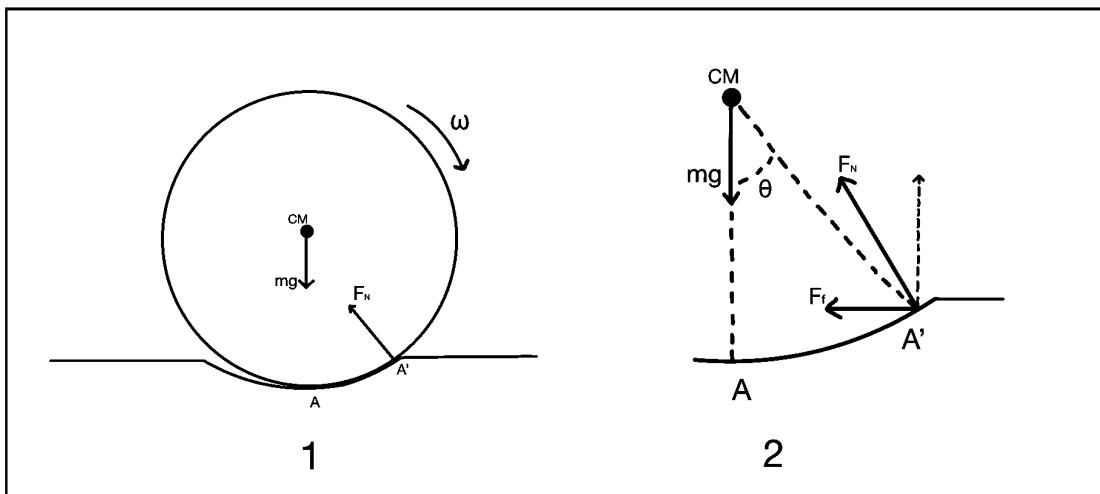
Då ett föremål glider uppstår en friktionskraft enligt

$$F_f(t) = F_N\mu \quad (2.17)$$

där $F_f(t)$ är friktionskraften, F_N normalkraften och μ friktionskoefficienten som för detta system antas vara 0,55. Då ett biljardbord står plant kan normalkraften ersättas med bollens tyngdkraft mg [1].

2.4.2 Rullande friktion

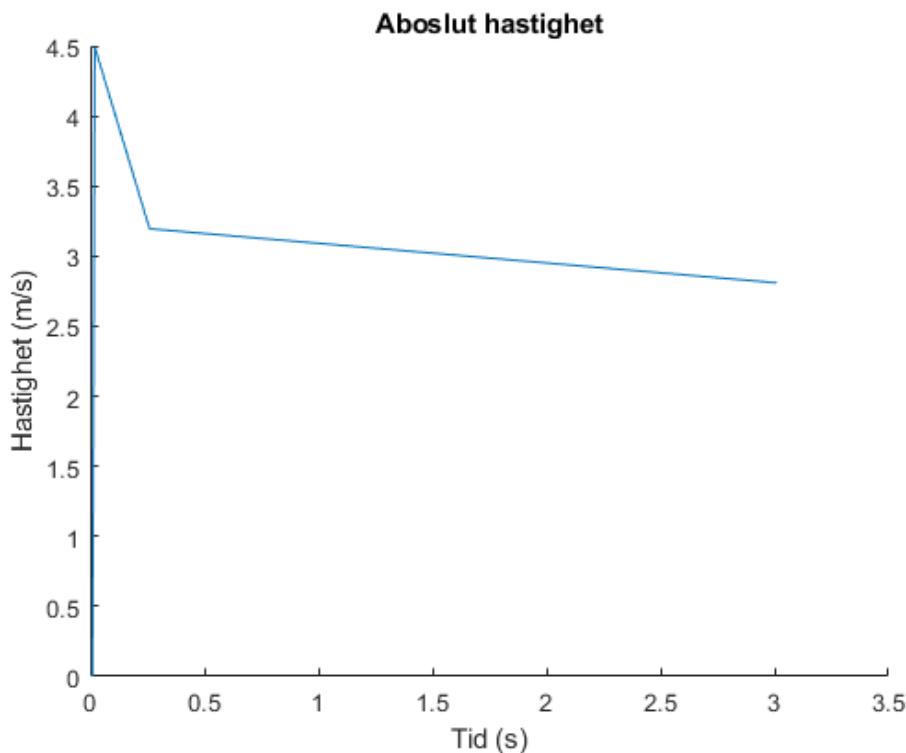
När en boll rullar skapas en friktionskraft avvikande från den i sektion 2.4. Då en biljardboll består av ett hårt material och biljardbordets matta av ett något mjukare material kommer bollen sjunka ned i detta, se Figur 2.2. Friktionskraften beror av hur mycket bollen åker ner i undermaterialet[4]. Då underlaget vid ett biljardbord är relativt hårt antas θ vara 0,02 radianer. Med hjälp av denna figur kan friktionskraften bestämmas enligt Ekvation 2.18, där $F_f(t)$ är friktionskraften, m bollens massa, g gravitationskraften och θ vinkeln mellan där bollens botten A och punkten A' möter mattan.



Figur 2.2: Rullande friktion för en boll

$$F_f(t) = \frac{5}{7}mg \quad (2.18)$$

Utifrån slutsatsen i sektion 2.3 kommer friktionskraften ändras när hastigheten uppnått ett visst värde vilket påverkar en bolls hastighetsändring som kan ses i Figur 2.3.



Figur 2.3: Hastighet av en boll över tid. Då friktionen för glidning är större än rullning minskas hastigheten längsammare efter att $\frac{5}{7}$ av starthastigheten nåtts

2.5 Kollision

För det här systemet finns det två typer av kollisioner, när en boll krockar mot en vägg och när två bollar krockar med varandra. Dessa fall måste hanteras olika vilket beskrivs här.

2.5.1 Kollision med vägg

När en boll kolliderar med en vägg antas den ha samma egenskaper som en ljusstråle som reflekterar mot ett medium med högre brytningsindex enligt reflektionslagen[1]. Den säger att den infallande vinkelns mellan bollens rörelsebana och väggens normal ska vara densamma som utfallsvinkelns. När detta översätts i systemet med bollens hastighet involverat ger det att hastighetsvektorn kan speglas i det plan som bygger upp väggen. Speglingen utförs om bollens masscentrum är mindre än en bollradie från väggen, vilket är när bollen vidrör väggen. En illustration över speglingen kan ses i Figur 2.4.

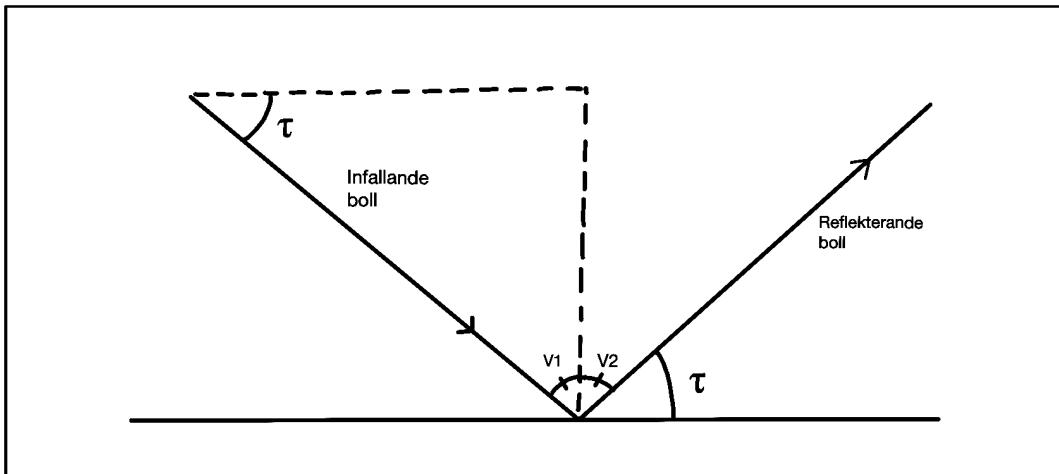
Vilken ekvation som används för att beräkna hastighetsvektorns vinkel beror på vilken vägg bollen träffar. Om bollen kolliderar med bordets långsidor används ekvation

$$\tau = -\tau \quad (2.19)$$

och om bollen träffar någon av kortsidorna används ekvation

$$\tau = \pi - \tau \quad (2.20)$$

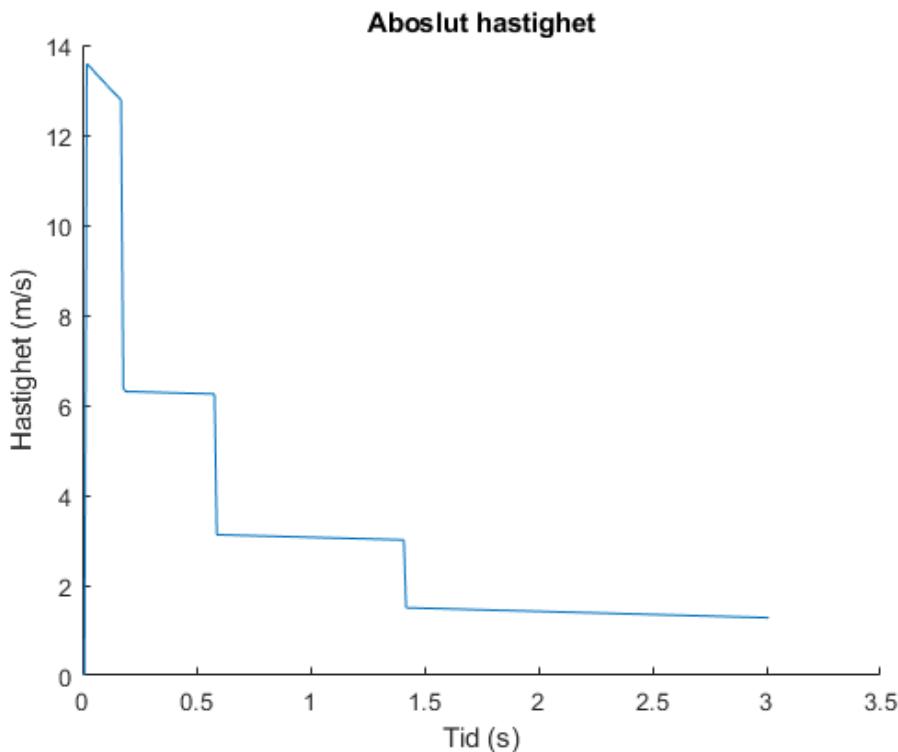
istället, där τ är vinkelns för hastighetsvektorn. Detta beror på hur bordet är placerat i koordinatsystemet. Se Bilaga B för framtagning av ekvationerna.



Figur 2.4: Kollision mellan boll och vägg

På grund av inelasticitet mellan bollen och väggen kommer hastigheten även minska vid kollisionen enligt Ekvation 2.21, där $v(t)$ är hastigheten innan kollisionen, e är elasticitetskoefficienten och $v'(t)$ är hastigheten efter kollisionen. I det här projektet antas elasticitetskoefficienten vara 0.5. Hur hastigheten ändras vid kollision med biljardbordets väggar kan ses i Figur 2.5.

$$v'(t) = ev(t) \quad (2.21)$$



Figur 2.5: Hastighetsförlust för en boll vid kollision med väggar

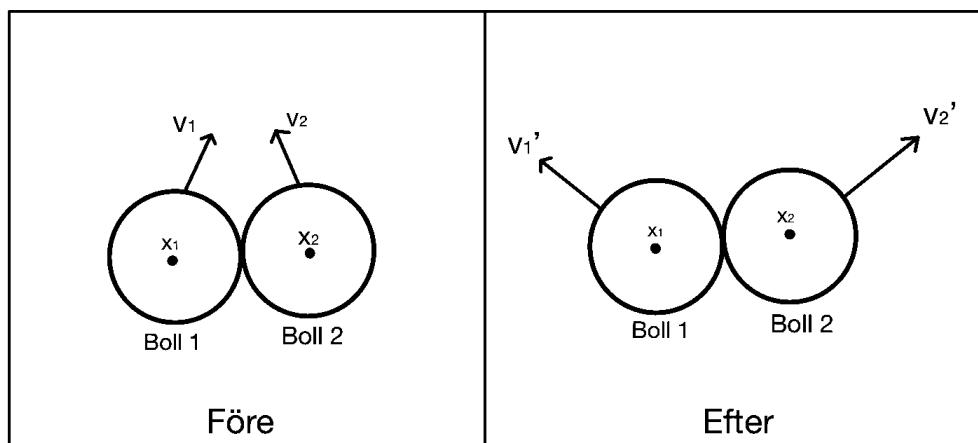
2.5.2 Kollision mellan bollar

När två bollar kolliderar med varandra måste de hanteras annorlunda från kollision med väggar. Detta på grund av deras sfäriska ytor samt att båda objekten kan vara i rörelse. Genom att hantera hastigheter och positioner som vektorer kan samtliga fall förenklas till Ekvation 2.22 och 2.23 [1]. I fallet då en av bollarna skulle vara stillastående sätts dess hastighet till nollvektorn och samma ekvationer kan användas. Operationen i täljaren avser skalärprodukt. Två bollar antas kollidera med varandra när deras masscentrum är mindre än två bollradier ifrån varandra, vilket är den första kontakten. Av samma anledning som för kollision med väggen kommer även här hastigheterna minska på grund av inelasticitet. Då bollarnas material är hårdare än väggens antas här elasticitetskoefficienten vara 0,95, som appliceras med Ekvation 2.21.

$$v'_1(t) = v_1(t) - \frac{\langle v_1(t) - v_2(t), x_1(t) - x_2(t) \rangle}{\|x_1(t) - x_2(t)\|^2} (x_1(t) - x_2(t)) \quad (2.22)$$

$$v'_2(t) = v_2(t) - \frac{\langle v_2(t) - v_1(t), x_2(t) - x_1(t) \rangle}{\|x_2(t) - x_1(t)\|^2} (x_2(t) - x_1(t)) \quad (2.23)$$

Här representerar $v_1(t)$ och $v_2(t)$ hastigheterna för två ingående bollar, $x_1(t)$ och $x_2(t)$ deras position motsvarande. $v'_1(t)$ och $v'_2(t)$ är de två bollarnas hastighet efter de kolliderat med varandra. Illustration över dessa hastigheter kan ses i Figur 2.6.



Figur 2.6: Hastighetsriktning för två bollar före samt efter kollision

Kapitel 3

Implementation

Den visuella representationen delades upp i två olika delar. Först implementerades systemet i MATLAB för att kontrollera att de fysikaliska ekvationerna var korrekta och gav ett logiskt resultat. Modellen översattes sedan till ett resultat som kunde visualiseras i webbläsaren med hjälp av WebGL. Båda implementationerna använder likvärdiga numeriska metoder som beskrivs nedan. För samtliga numeriska värden använda under projektet, se Appendix A.1.

3.1 Numerisk metod

För att simulera modellen behöver Differentialekvationen 2.1 lösas för att erhålla accelerationen. Denna ekvation lösas inte analytiskt utan med hjälp av Eulers stegmetod[5]. Eulers stegmetod kan generellt beskrivas som

$$y_{n+1} = y_n + f(t_n, y_n)h \quad (3.1)$$

där h är steglängden, n är det aktuella diskreta steget, y_n är nuvarande värde, t_n är den aktuella tiden i det diskreta intervallet och f är funktionen som beskriver derivatan av nuvarande värde. Eulers metod räknar ut nästa värde genom att addera det nuvarande värdet med derivatan av nuvarande värde i den aktuella tiden multiplicerat med steglängden. För detta system som bygger på differentialekvationen 2.1 kan den skrivas om till differensekvationen 3.2 med hjälp av 3.1.

$$v'[t_n] = \frac{1}{m} F_{res}[t_n] = \frac{1}{m} (F_{in}[t_n] - F_f[t_n]) \quad (3.2)$$

Utifrån differensekvationen stegas lösningen fram med steglängden h för att få ut bollarnas hastighet $v[t_n]$ och position $p[t_n]$ med Ekvation 3.3. Detta ger möjligheten att erhålla bollarnas position och placera de på rätt plats i simuleringen. För det här projektet är steglängden h valt till $\frac{1}{60}$.

$$\begin{aligned} v[t_n] &= v[t_n - 1] + v'[t_n] \cdot h \\ p[t_n] &= p[t_n - 1] + v[t_n] \cdot h \end{aligned} \quad (3.3)$$

3.2 Implementation i MATLAB

Simuleringen i MATLAB definierar varje boll som två vektorer, en för att bevara dess hastighet och en för position. Varje vektor har en x - och y -komponent som används i den slutliga animationen. De

har även en komponent för det diskreta tidsintervallen beskrivet av Eulers stegmetod för att skapa en animation genom att stega igenom dessa värden.

För att skapa animationen stegas samtliga värden på t_n beskrivet i 3.3 för den önskade animationens längd och kan därmed lösa differentialekvation 2.1 numeriskt. Ekvationen lösas separat för samtliga bollar för varje tidssteg.

3.2.1 Kollision

För att utföra kollision mellan en boll och biljardbordets väggar testas bollens masscentrums position relativt biljardbordets kanter. Om en boll hamnar utanför bordet appliceras nödvändiga ekvationer från sektion 2.5.1. Detta illustreras i Algoritm 1. Ett liknande test utförs om en boll är innanför hålen där bollen ska åka ner, men flyttas då utanför bordet istället för att utföra kollision.

Algorithm 1 Algoritmen för kollision med vägg

```

if ball outside table - y direction then
    velocityY  $\leftarrow -prevVelocityY$ 
    velocityX  $\leftarrow prevVelocityX$ 
    angle  $\leftarrow -prevAngle$ 
else if ball outside table - x direction then
    velocityX  $\leftarrow -prevVelocityX$ 
    velocityY  $\leftarrow prevVelocityY$ 
    angle  $\leftarrow \pi - prevAngle$ 
end if
```

Vid detektion för kollision mellan bollarna testas samtliga bollars position relativt varandra. Detta för att inte utföra kollisionsberäkningar vid varje tidssteg, endast när en kollision ska ske. Om två bollars masscentrum är närmre varandra än två bollradier har kollision uppstått, exempel kan ses i Algoritm 2.

Algorithm 2 Algoritmen för kollision mellan två bollar

```

if balls closer than  $2r$  then
    Calculate velocities according to 2.5.2
end if
```

3.2.2 Friktion

Friktion appliceras konstant vid accelerationen men friktionskraften varieras som nämnts under sektion 2.4. För att bestämma vilken friktion som används jämförs den nuvarande hastigheten för en boll med dess starthastighet, se Algoritm 3.

Algorithm 3 Algoritmen för bestämmandet av friktionen

```

if currentVelocity <  $5/7 * initialVelocity$  then
    Friction  $\leftarrow$  rolling friction
else
    Friction  $\leftarrow$  sliding friction
end if
```

När Eulers stegmetod har gått igenom alla tidssteg för bollarna presenteras en animation av händelseförfloppet med hjälp av MATLABs grafiska verktyg, se sektion 4.1.

3.3 Implementation i WebGL

För att implementera systemet i den slutliga grafiska illustrationen användes JavaScript-API:et WebGL med hjälp av biblioteket *Three.js*. Implementationen i MATLAB används som grund för att skapa WebGL-implementationen. En 3D-miljö skapades för att interagera med spelet medan simuleringen sker i ett tvådimensionellt plan.

Alla objekt utom biljardbollarna skapas med *Blender* då det är ett enklare verktyg för att skapa realistiska modeller. Här tillhör biljardbordet, rummet som spelet utspelar sig i samt biljardkön. Dessa modeller exporteras som *glTF*-filer med hjälp av *Three.js*. Då bollarna är enklare modeller kunde de skapas direkt med *Three.js*.

All kod för simuleringen av translation, rotation, friktion och kollision bygger på implementationen från MATLAB och liknar koden från sektion 3.2. Det största som skiljer sig vid implementationen i WebGL är möjliggörandet för den interaktiva delen samt en *GUI* med instruktioner för hur man spelar. Spelaren tillåts att spela biljard i realtid och styra ett kö som följer den vita bollen. Inmatning sker via tangentbordet där den vita bollens starthastighet och vinkel kan väljas. Tidsstegen för varje boll sparas inte för att förbättra prestandan, till skillnad från MATLAB där allt sparas för att utföra animationen i efterhand.

För att se kravspecifikationer av maskinvara för att köra spelet, se Appendix A.2.

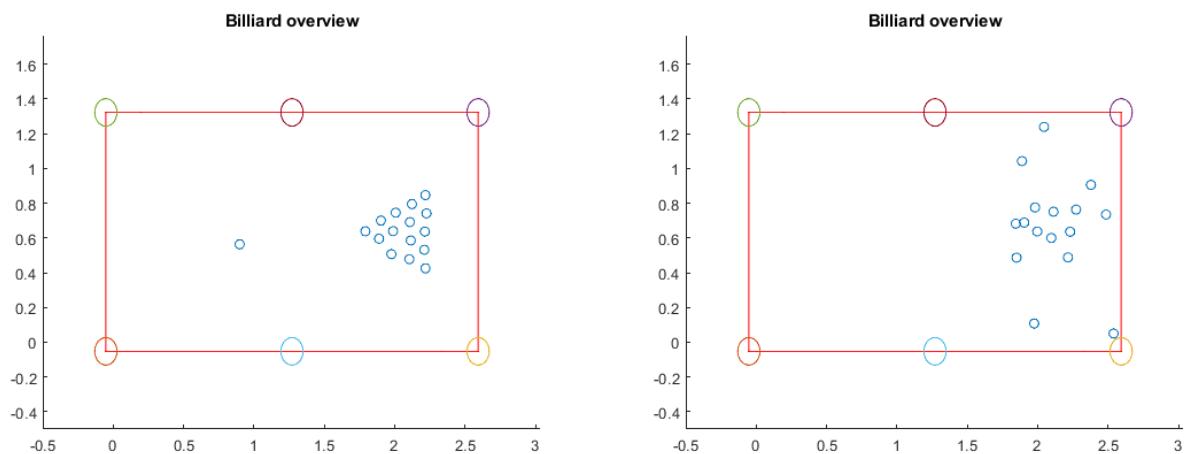
Kapitel 4

Resultat

Här presenteras de två olika resultaten från MATLAB samt WebGL. Båda simuleringarna utgår från samma fysikaliska moment men visualiseras på två olika nivåer.

4.1 MATLAB

Resultatet i MATLAB blev slutligen en simulering sett från ett tvådimensionellt perspektiv på biljardbordet. I Figur 4.1 (t.v.) ses denna modell med samtliga bollar uppställda som i början av ett spel. Den yttre linjen definierar biljardbordet och de sex större hålen är där bollarna åker ner. Om en bolls centrum hamnar över ett sådant hål flyttas bollen utanför spelet och räknas som den åkt ned i hålet. Bollarna har möjlighet att kollidera med varandra samt med väggen genom att applicera ekvationerna som nämns i sektion 2.5. Bollarna glider även i början av dess rörelse för att sedan övergå till rullning. Rullningen visualiseras dock inte på grund av den tvådimensionella simuleringen, men den fysikaliska påverkan för friktionen är implementerad.

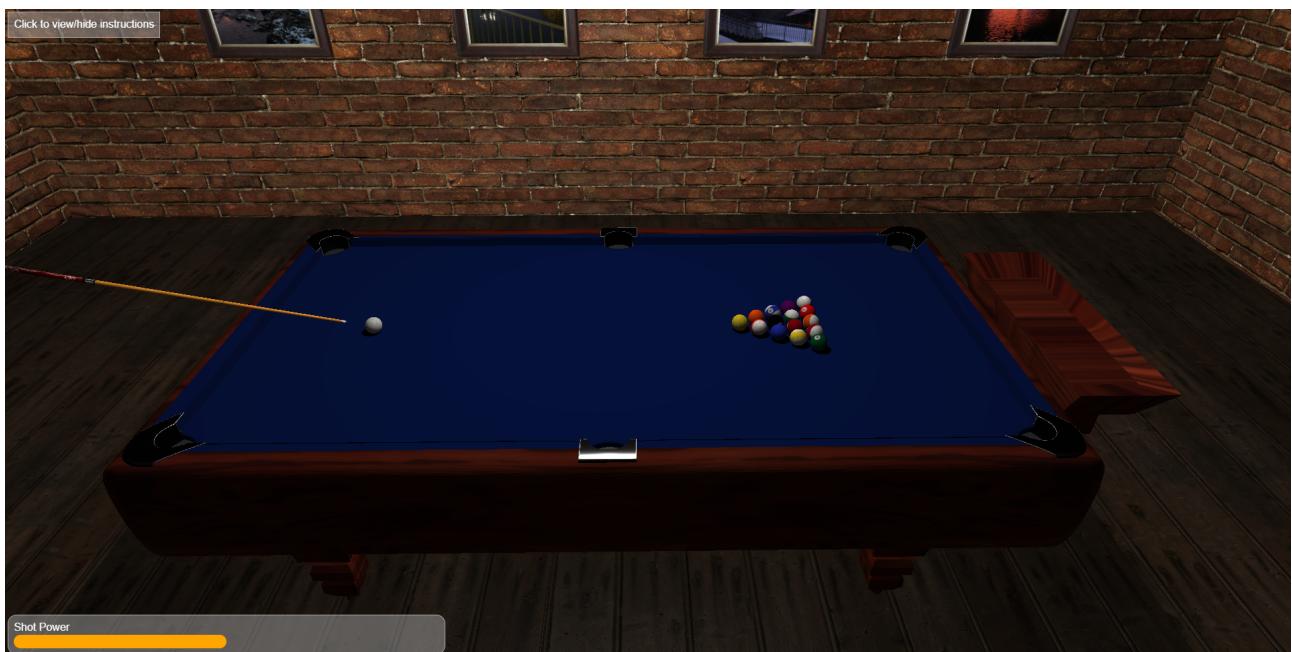


Figur 4.1: Uppställning i MATLAB för biljardspel

I Figur 4.1 (t.h.) visualiseras hur simuleringen ser ut efter ett skott har avfyrats från startpositionen och bollarna har interagerat med varandra. Denna simulering är inte interaktiv och resultatet blev endast en animering.

4.2 WebGL

Visualiseringen i WebGL resulterade i ett mer avancerat resultat. Inga fysikaliska delar skiljer sig från MATLAB-resultatet men den visuella representationen innehåller nu en tredimensionell miljö istället. I Figur 4.2 kan motsvarigheten för uppställningen ses i WebGL. Detta resultat innehåller även en interaktiv del som ger en användare möjligheten att spela denna simulering av biljard. En meny finns också implementerad här för att hjälpa användaren förstå hur den ska spela, vilket kan ses i Figur 4.3. När en boll sätts i ett hål flyttas den till en korg vid sidan av bordet och är ute ur spelet. Korgen kan ses i Figur 4.4 där alla bollar sitter.



Figur 4.2: Slutliga uppställningen i WebGL



Figur 4.3: Resultat efter ett skott mot biljardbollarna, tillsammans med meny



Figur 4.4: Alla bollar har satts i hålen och spelet är färdigspelat

Kapitel 5

Diskussion

Resultatet visar en fungerande interaktiv visualisering av ett biljardspel. Under projektet var det väldigt öppet för vad som menades med en interaktiv modell av systemet och det fanns många idéer om hur en interaktiv modell kan utföras. Den slutliga modellen implementerades i WebGL i 3D med hjälp av *Three.js*.

5.1 Implementationsval

Kollisiondetektion är den största delen av systemet. Det finns fördelar och nackdelar med varje kollisiondetektionsmetod och två stycken användes beroende på om kollisionen var mellan bollar eller mot vägg. Kollision mot väg använder en metod som gör att en boll antas som en ljusstråle där väggen agerar som en reflekterade medium. Fördelar med det är att det är lätt att implementera då bollens rörelse blir spegelvänt. Nackdelen med den är att när bollen träffar en 90 grader kant så som kanterna vid hållen så agerar de som en platt punkt istället för en vinklad punkt vilket ger en orealistisk reflektion. Enligt modellen representeras bollar endast av deras mittpunkt vilket inte stämmer med verkligheten.

Kollisionen mellan bollar använder en metod som tar till hänsyn att det är kollision mellan sfäriska ytor och inte plana ytor. Fördelen med denna metod är att ekvationen som implementeras i 2.5.2 kan använda vektorer direkt utan att behöva beräkna vinkelarna som bollarna träffar varandra med. Denna implementation tillåter dock inte skruv av bollar att överföras mellan varandra.

För att beräkna systemets differensekvationer valdes Eulers stegmetod då den var den enklaste modellen att implementera som hade kapaciteten att simulera systemet på ett övertygande sätt.

Avgränsningarna för projektet infördes för att förenkla simuleringen av systemet. Detta främst för att minska utvecklingstiden för att skapa systemet då implementationen av skruv på bollar kräver mer komplicerade funktioner och metoder. Dessa förenklingar skapades också då dessa metoder inte ansågs bidra tillräckligt mycket för att göra systemet mer visuellt realistiskt.

5.2 Vidareutveckling

Vidareutveckling som kan utföras vid ökad tidsram eller vidarerbete skulle vara att tillåta flera spelare samtidigt så att de kan tävla med varandra. Riktiga spelregler skulle även implementeras så som när vita bollen åker ner i ett hål så ska spelaren ta upp en av sina egna färgade bollar, om spelaren inte träffar någon boll så ska motståndaren få placera den vita bollen var den vill och annat. En metod för att bestämma vems tur de är att slå behöver också implementeras så när spelaren sätter en egen färgad

boll ska den fortsätta att slå.

Mer avancerade kontroller av biljardkön kan läggas till för att ge användaren mer möjligheter, till exempel så att den kan vinklas uppåt för att ge bollen skruv. En kollisionsmodell som tillåter skruv behöver då implementeras som tillåter detta samt överföring av skruv mellan bollar. Stora ändringar behöver utföras på den nuvarande modellen för att göra detta möjligt vilket nämnts i sektion 5.1.

Visuella vidarutvecklingar som en animering av biljardköns vid slag och när bollarna går ner i hålet kan även utföras. Den nuvarande modellen flyttar bollar när dem träffar hålen och biljardkön försviner direkt efter att den skjuts vilket inte är realistiskt.

Kapitel 6

Slutsats

Det resulterade systemet består av en interaktiv visualisering av biljardspellet som hanterar bollars interaktion med varandra på ett bord. Den använder Eulers stegmetod och följer avgränsningar som sattes. Det finns flera vidarutvecklingar som kan göras men den nuvarande modellen anses uppnå våra mål för projektet.

Litteraturförteckning

- [1] D. Halliday, R. Resnick & J. Walker *Principles of Physics*, Eleventh edition. John Wiley & Sons Inc, 2014
- [2] R. Rojas & M. Simon, "Freie Universität Berlin, Institut Für Informatik", *Like a rolling ball*, 2004. Hämtad 2022-01-28. [Online]. Tillgänglig: <http://robocup.mi.fu-berlin.de/buch/rolling.pdf>
- [3] S. Ulf "ITN 2020", *Formelsamling TNE043 Mekanik och Vågfysik*, 2015
- [4] J. Hierrezulo & C. Carnero, Secondary School I B Reyes Católicos" & "University of Málaga, Spain", *Sliding and rolling: the physics of a rolling ball*, 1995. Hämtad 2022-02-03. [Online]. Tillgänglig: https://billiards.colostate.edu/physics_articles/Hierrezuelo_PhysEd_95_article.pdf
- [5] J. Jönsson *MATLAB-beräkningar inom teknik och naturvetenskap*, tredje upplagan. Studentlitteratur, 2010

Bilaga A

Appendix

A.1 Tabell över numeriska värden

Tabell A.1: Numeriska värden använda i implementationen

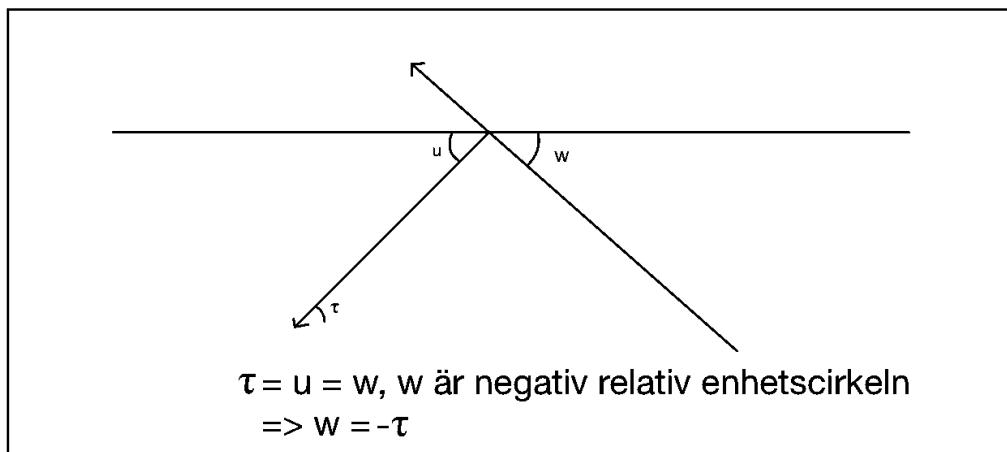
Numeriska värden	
Variabel	Värde
Bolradie r	0,0525 m
Bollmassa m	0,220 kg
Tyngdacceleration g	9,82 m/s ²
Frikitionskoefficient μ	0,55
Vinkeln θ	0,02 rad
Biljardbord bredd	1,12 m
Biljardbord längd	2,24 m
Diameter av biljardbordets kantfickor	0,15 m
Diameter av biljardbordets mittenfickor	0,092 m
Elastisitetskoefficient mellan bord och boll	0,5
Elastisitetskoefficient mellan två bollar	0,95
Bildrutor per sekund i WebGL	60
Steglängd h i WebGL	$\frac{1}{60}$
Steglängd h i MATLAB	0,01

A.2 Uppsättning av programvaran

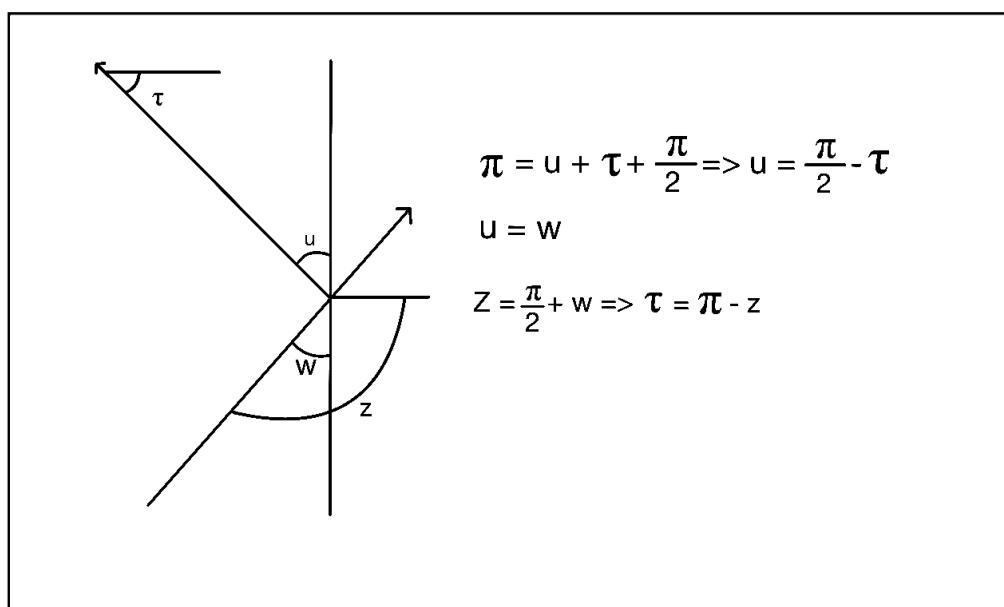
För att kunna köra simuleringen krävs att programmet ligger på en server då *Three.js* endast kan användas på en server. Filerna måste tillträdas via servern och inte öppnas manuellt. Simuleringen kan sedan köras via valfri modern webbläsare som klarar av 3D-grafik. Enheten där simuleringen sedan spelas kräver enkla möjligheter till att rendera objekten från *Three.js*. Beroende på uppkoppling till servern samt hårdvara på enheten kan det ta olika lång tid att ladda in objekten.

Bilaga B

Matematiska bevis



Figur B.1: Vinkeländring vid kollision med långsida



Figur B.2: Vinkeländring vid kollision med kortssida