Stelkroppssimulering

Tomas Forsyth Rosin, Jonas Zeitler, Emil Axelsson, Karl Johan Krantz

15 mars 2013

Sammanfattning

Sammanfattning av projektet.

Innehåll

Figurer

Tabeller

1 Inledning

1.1 Bakgrund

Fysiksimulering är en vital del av många moderna datorspel. I spelsammanhang kallas detta för ett spels *fysikmotor*. Att implementera en fysikmotor handlar om att skapa en miljö med regler till vilka en uppsättning objekt måste förhålla sig. I många spel utgörs fysikmotorns bas av en *stelkroppssimulering*, vilket innebär vissa antaganden och förenklingar av verkligheten. Mest påtagligt är att systemets objekt antas vara okomprimerbara, eller stela. Även med dessa förenklingar i beaktning är implementation av en fysikmotor ett komplext problem, som kan lösas på olika sätt beroende på aktuella förutsättningar och tillämpningsområden.

1.2 Syfte

Syftet med denna rapport är att beskriva utvecklingen av en fysikmotor, som ska kunna användas i datorspel med tvådimensionell grafik. Motorn ska ha stöd för att simulera kollisioner och friktion mellan stelkroppar modellerade som generella konvexa polygoner. Den ska också innehålla stöd för fjädrar och dämpare, som ska kunna fästas i stelkropparna. Motorn ska producera trovärdiga resultat vid såväl kollisioner som när flera stelkroppar vilar på varandra. Eftersom motorn ska kunna användas till datorspel behöver algoritmerna som används vara tillräckligt tidseffektiva för att tillåta interaktivitet. Motorn ska implementeras i C++ med hjälp av grafikbiblioteket OpenGL varpå resultatet ska utvärderas i rapporten.

2 Modell

I modellen som skulle simuleras ingick tre typer av entiteter: rörliga stelkroppar, fasta stelkroppar samt fjädrar med dämpning. Varje entitet kan ha både konstanta egenskaper och variabla tillstånd, enligt tabell A.

Tabell A: Tillstånd och egenskaper hos olika entiteter

Rörliga stelkroppar		**Fasta stelkroppar**		**Fjädrar med dämpare**	
Tillstånd	**Egenskaper**	**Tillstånd**	**Egenskaper**	**Tillstånd**	**Egenskaper**
Position	Densitet		Position		Infästnings-punkter
Vinkel	Studs-koefficient		Vinkel		Fjäder-konstant
Hastighet					Dämpnings-koefficient
Vinkel-hastighet					

Fysiksimulering är en vital del av många moderna datorspel. I spelsammanhang kallas detta för ett spels fysikmotor. Att implementera en fysikmotor handlar om att skapa en miljö med regler vilka en uppsättning objekt måste följa. I denna rapport undersöks stelkroppssimulering som utgör basen i de allra flesta fysikmotorer. Begreppet antyder att en fysikmotor för stelkroppssimulering gör vissa antaganden och förenklingar av den fysiken i en riktig värld. Mest påtagligt är att systemets objekt antas vara okomprimerbara, eller stela. Även med dessa förenklingar i beaktning är implementation av en fysikmotor ett komplext problem. Det finns många olika metoder för att nå likvärdiga resultat och att låsa projektet till en metod i ett tidigt skede kan försvåra senare utveckling. I denna rapport implementeras och utvärderas en impulsbaserad, iterativ, fysikmotor med stelkroppssimulering.

3 Kollisionsdetektion

Det första steget mot att simulera stelkroppars interaktion är att detektera alla kolliderande polygoner i scenen. Att två objekt i ett diskret system ligger dikt an varandra är sällsynt. Vanligare är att de interpenetrerar varandra, d.v.s. att ett hörn hos ett objekt ligger inuti ett annat objekt.

3.1 Detektionsmetod

I detta projekt diskuterades i huvudsak två metoder för kollisionsdetektion.

3.1.1 Separating Axis Test

För att utreda eventuell interpenetration mellan två givna objekt kan SAT [1, s. 29] *(Separating Axis Test)* användas. Testet kan avgöra om objekten inte kolliderar genom att hitta en vektor på vilken objektens punkter projiceras på disjunkta intervall (Figur **??**). Ett högre antal vektorer ökar beräkningstiden men också sannolikheten för ett korrekt resultat.

3.1.2 Gilbert-Johnson-Keerthi

För att inte göra avkall på noggrannheten har en annan metod, GJK [1, s. 30] *(Gilbert-Johnson-Keerthi)*, använts för detta projekt. Eftersom metoden kan utesluta fler möjliga kollisioner kommer senare beräkningar kräva mindre tid. GJK bygger på Minkovskidifferensen mellan de två objekten, definierad enligt (1).

$$\chi_1 \ominus \chi_2 = \{x_1 - x_2 | x_1 \in \chi_1, x_2 \in \chi_2\} \tag{1}$$

De två objekten korsar varandra om, och endast om, polygonen som Minkovskidifferensen renderar innehåller origo. GJK-algoritmen utreder detta på ett sätt som minskar antalet beräkningar från polynomiellt till linjärt beroende av antalet punkter i objekten [3].

3.2 Optimering

Att undersöka kollisioner för alla par av objekt i varje bildruta medför omfattande beräkningar och sänker programmets effektivitet betydligt. För att undvika dessa beräkningsmängder

görs först olika bedömningar av vilka objekt som riskerar att kollidera varefter de tyngre kollisionsdetektorerna appliceras endast på dessa par.

3.2.1 Bounding boxes

För att förenkla representationen av objekt användes Bounding Boxes. För varje punkt i objektet beräknas den maximala respektive minimala koordinaten i alla dimensioner. Av dessa värden konstrueras sedan nya punkter som bildar ett rätblock (rektangel i två dimensioner) med sidor parallella med koordinataxlarna. Alla punkter i objektet kommer att ligga i detta rätblock vilket gör att man snabbt kan utesluta förekomster av punkter i stora områden.

3.2.2 Quadtree

4 Kollisionslösning

5 Kollisionsrespons

5.1 Impulsbaserade system

I det system som simuleras måste någon representation av krafter mellan kroppar finnas. En vanlig förenkling är att se krafter som momentana händelser i tiden. Detta introducerar begreppen impuls och rörelsemängd. Impulser kan relateras till krafter enligt (2).

$$\bar{J} = \int_{\Delta t} \bar{F}\ dt = \int_{\Delta t} m\bar{a}\ dt = m\bar{v} \tag{2}$$

Den aktuella implementationen av (2) innehöll inget begrepp om tid utan t representerades istället av ett simuleringssteg.

5.1.1 Praktiskt förlopp

Då en kollision detekteras ska fysikmotorn lösa vilka krafter som uppstår mellan kropparna. En nödvändig förenkling är att endast hantera kollisioner som sker mellan hörn och sida för kroppar. Detta är de absolut vanligast förekommande kollisionerna och det är ur optimeringssynpunkt inte värt att ta hänsyn till andra typer av kollisioner. Den data som finns tillgänglig för kollisionsrespondern är:

- De kroppar som ingår i kollisionen
- I vilken punkt kollisionen sker
- Normalvektorn för kollisionsplanet/-sidan

Lösningen för den resulterande impulsens magnitud, j_r , illustreras bäst genom att först förstå hur impulsen appliceras på kropparnas hastigehet och vinkelhastighet.

$$\bar{v}_1' = \bar{v}_1 + \frac{j_r}{m_1}\hat{n},\ \bar{v}_2' = \bar{v}_2 + \frac{j_r}{m_2}\hat{n} \tag{3}$$

$$\omega_1' = \omega_1 + \frac{j_r}{I_1}(\bar{r}_1 \times \hat{n}),\ \omega_2' = \omega_2 + \frac{j_r}{I_2}(\bar{r}_2 \times \hat{n}) \tag{4}$$

För ekvation (3-4) är alla parametrar kända, utom $\bar{v}_i'$, ω_i' och j_r . Vektorn $\bar{r}_i$ är vektorn från i:te kroppens masscentrum till kollisionspunkten, P, vilken är gemensam för båda kropparna i kollisionen. Normalvektorn, $\hat{n}$, ges som inparameter från expanding polytope algoritmen. Punktens hastighet för respektive kropp beräknas enligt (5).

$$\bar{v}_{i_P} = \bar{v}_i + \omega \times \bar{r}_i \tag{5}$$

$$\bar{v}_r' \cdot \hat{n} = -e\bar{v}_r \cdot \hat{n} \tag{6}$$

$$\bar{v}_r = \bar{v}_{1_P} - \bar{v}_{2_P},\ \bar{v}_r' = \bar{v}_{1_P}' - \bar{v}_{2_P}' \tag{7}$$

Ekvation (6) relaterar kropparnas separationshastighet (7), före och efter kollisionen. Parametern e anger materialens studskoefficient. Genom substitution av (3-5) i (6) kan kollisionsimpulsen lösas ut enligt (8).

$$j_r = \frac{-(1+e)\bar{v}_r \cdot \hat{n}}{\frac{1}{m_1} + \frac{1}{m_2} + (\frac{1}{I_1}(\bar{r}_1 \times \hat{n}) \times \bar{r}_1 + \frac{1}{I_2}(\bar{r}_2 \times \hat{n}) \times \bar{r}_2) \cdot \hat{n}} \tag{8}$$

Genom detta skapas en grundläggande teori om hur kollisioner ska bete sig i simuleringen. En impuls beräknas i (8) och appliceras sedan i (3) och (4).

Under implementationen introducerades ett specialfall för kollision mellan rörlig och icke-rörlig kropp. Detta resulterade i en förenklad impulsmodell enligt (9).

$$j_r = \frac{-(1+e)\bar{v}_r \cdot \hat{n}}{\frac{1}{m_1} + \frac{1}{I_1}(\bar{r}_1 \times \hat{n}) \times \bar{r}_1 \cdot \hat{n}} \tag{9}$$

Ekvation (9) kan likställas med att sätta den stationära kroppens massa och tröghetsmoment till ∞.

5.2 Penalty-baserade system

Vid kollision mellan två kroppar i verkligheten är det två kontinuerliga motriktade krafter som verkar på objekten, dessa krafter är aktiva över en kort tidsperiod, något som gör att det är mer korrekt att använda än impulsbaserad kollisionslösning. Vid vilande kontakt är detta dock inte en lösning som ger trovärdiga resultat, vilket diskuterats ovan. Detta eftersom det sällan endast är ett hörn som är i kontakt mellan två kroppar i vila. Vid implementation ansågs därför att en penalty-baserad kollisionslösning var mer lämplig i dessa fall.

5.2.1 Tillvägagångssätt

Metoden medför att en utflyttning ej sker, objekt använder sig istället av mindre impulser för för att fjädra ifrån varandra. Detta kan illustreras (figur) som dämpade fjädrar som spänns mellan kropparna längs penetrationsvaktorn.

Figur

Ekvation (10) visar den impuls J som ger fjädringen på båda objekten.

$$\bar{J} = C \cdot \bar{P} \tag{10}$$

5.2.2 Övergång

En övergång mellan de två lösningsmetoderna är aktuell först när objekt vilar på varandra, vilket innebär att deras relativa hastighet i kollisionspunkten är låg. En tröskel som varierar med de aktuella egenskaperna hos systemet kan ge mer korrekta lösningar. Vid empiriska tester så sattes emellertid en fast tröskel för övergången mellan metoderna.

6 Krafter i impulsbaserade system

6.1 Kraftgeneratorer

Hittills har endast momentana krafter mellan objekt diskuterats. Det har också poängterats att detta är en förenkling av vad som sker i stötar och kontakt. En kollision mellan två objekt orsakar kompression hos materialen i respektive objekt vilket i sin tur ger upphov till krafter mellan kropparna [4].

Vissa fysikmotorer delar upp varje tidssteg i mindre delar och kan på så sätt erhålla en bättre approximation för vissa krafter. Dessa metoder kallas tidsderiverande. En tidsderiverande fysikmotor kan använda simuleringsmetoder, t.ex. Runge-Kutta, som kräver information om flera olika tillstånd hos systemet.

I simuleringen implementeras kraftgeneratorer för gravitation och fjäderkrafter.

6.2 Fjäderkrafter

En fysikmotor som implementerar massa-fjäder-system och harmonisk rörelse kräver att det finns mer sofistikerade simuleringsmetoder än Euler för icke-triviala fall. Detta motiverar att låta fjäderkrafter vara kraftgeneratorer och därmed simuleras med en Runge-Kuttalösning.

Hookes lag beskriver den kraft en fjäder påverkar andra objekt med. Kraften beror av hur utdragen eller sammanpressad fjädern är. Det vill säga, kraftgeneratorn behöver endast positionen av fjäderns infästningspunkter för att beräkna fjäderkraften.

$$f = -k \cdot \Delta l \tag{11}$$

I ekvation (11) är k fjäderkonstant och l anger hur utdragen eller sammanpressad fjädern är. Lösningen för (11) ger endast kraftens magnitud, för en flerdimensionell simulering utökas ekvationen till att även innehålla kraftens riktning (12).

Ett viktigt resultat för Hookes lag är att den resulterande kraften f verkar lika mycket i fjäderns båda ändar.

$$f = -k(|\bar{d}| - l_0)\hat{d} \tag{12}$$

$$\bar{d} = x_A - x_B \tag{13}$$

Parametrar x_A och x_B (13) är infästningspunkter med avseende på position och rotation för ett tillstånd y_t.

6.3 Friktion

Den allra vanligaste friktionsmodellen är Coloumbs modell. Den baserar sig i att den friktionskraft som påverkar en kropp är proportionell mot normalkraften mellan kroppen och underlaget. Modellen är egentligen en sammanslagning av två modeller, statisk och dynamisk friktion.

$$\bar{f}_s = \mu_s \cdot \bar{N} \tag{14}$$

$$\bar{f}_s = \mu_d \cdot \bar{N} \tag{15}$$

Där den statiska modellen (14) används för kroppar i vila och den dynamiska (15) för kroppar i rörelse. Båda modellerna kan endast ge upphov till friktionskrafter som är proportionella mot de yttre krafter som verkar på kroppen. Friktionskrafter verkar i kontaktplanets tangentriktning, $\bar{t}$.

$$f_t = \begin{cases} -(\bar{f}_{ext} \cdot \hat{t})\hat{t}, & \bar{v}_r = \bar{0}, \bar{f}_{ext} \cdot \hat{t} \leq \bar{f}_s \\ -\bar{f}_s \cdot \hat{t}, & \bar{v}_r = \bar{0}, \bar{f}_{ext} \cdot \hat{t} > \bar{f}_s \\ -\bar{f}_d \cdot \hat{t}, & \bar{v}_r \neq \bar{0} \end{cases} \tag{16}$$

Enligt (16) kan endast en extern kraft i kontaktplanet som överstiger den maximala statiska friktionskraften sätta en kropp i rörelse. I annat fall motverkas kraften av en lika stor friktionskraft vilket medför att kroppen förblir vilande.

Den friktionsmodell som används i simuleringen är ingen kraftgenerator utan tar endast hänsyn till impulser vid kontakt. En omskrivning av (14-16) ger den friktionsimpuls som motsvarar $\bar{f}_t$.

$$\bar{j}_s = \mu_s \cdot \bar{j}_r \tag{17}$$

$$\bar{j}_d = \mu_d \cdot \bar{j}_r \tag{18}$$

$$j_f = \begin{cases} -(m\bar{v}_r \cdot \hat{t})\hat{t}, & \bar{v}_r = \bar{0}, m\bar{v}_r \cdot \hat{t} \leq \bar{j}_s \\ -\bar{j}_s \cdot \hat{t}, & \bar{v}_r = \bar{0}, m\bar{v}_r \cdot \hat{t} \geq \bar{j}_s \\ -\bar{j}_d \cdot \hat{t}, & \bar{v}_r \neq \bar{0} \end{cases} \tag{19}$$

Friktionsimpulsen (19) appliceras samtidigt som kollisionsimpulsen för respektive kropp och kollision. Ekvation (17-19) är en ideal [Gravitas] friktionslösning för impulsbaserade system. Det kan emellertid visas att (17) och (18) inte ger energibevarande lösningar för höga värden på μ_s och μ_d.

Friktion simulerades utan hänsyn till friktionstal mellan specifika material. Istället har en naiv förenkling till de allra mest grundläggande egenskaperna implementerats enligt (20).

$$\bar{j}_r = \frac{|\bar{v}_r \cdot \bar{n}_\perp|}{\frac{1}{m_1} + \frac{1}{m_2} + \frac{1}{I_1}(\bar{r}_{1_\perp} \cdot \bar{n}_\perp)^2 + \frac{1}{I_2}(\bar{r}_{2_\perp} \cdot \bar{n}_\perp)^2} \tag{20}$$

Ekvation (20) kan jämföras med (impulsekvationen) som bygger på samma princip.

7 Resultat

8 Diskussion

Referenser

[1] Vella Colin, *Gravitas: An extensible physics engine framework using object-oriented and design pattern-driven software architecture principles*. 2008.

[2] Millington Ian, *Game Physics Engine Development*. 2007.

[3] Van den Bergen Gino, *A Fast and Robust GJK Implementation for Collision Detection of Convex Objects, Journal of Graphics Tool*. 4:2, 7-25, 1999.

[4] Newton, Isaac *Philosophiæ Naturalis Principia Mathematica*. 1687.