# Stelkroppssimulering

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

15 mars 2013

**Sammanfattning**

Denna rapport beskriver de fysikaliska koncept och metoder som har använts för att utveckla en fysikmotor för stelkroppar och fjädrar. I rapporten beskrivs metoder för kollisionshantering, friktionsmodeller och hantering av både kroppar i rörelse och i vila. Även Euler- och Runge-Kutta-metoder för lösning av differentialekvationer behandlas.

Resultatet som presenteras i rapporten är en prototyp av en tvådimensionell fysikmotor implementerad i C++ med grafikramverket OpenGL till stöd.

# Innehåll

# Figurer

# 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 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. 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** | |
|---|---|---|---|---|---|
| **Tillstånd** | **Egenskaper** | **Tillstånd** | **Egenskaper** | **Tillstånd** | **Egenskaper** |
| Position | Densitet | | Position | | Infästnings-punkter |
| Vinkel | Studs-koefficient | | Vinkel | | Fjäder-konstant |
| Hastighet | | | | | Fri längd |
| Vinkel-hastighet | | | | | |

Värt att notera i tabellen är att fjädrar inte har några tillståndsvariabler eftersom dessa kan hämtas från de stelkroppar de är fästa i.

# 3 Diskret simulering

Trots att den fysikaliska modell som användes i simuleringen är en förenkling av verkligheten vore det orimligt att försöka beräkna hela förloppet genom att lösa ekvationer analytiskt och göra symboliska integreringar. För simuleringar av detta slag krävs det alltså att modellen diskretiseras i tidsdomänen och att hela simuleringen delas upp i någon form av tidssteg. Många av de matematiska samband som modellen bygger på går att teckna som differentialekvationer (fjäderkrafter, gravitation etc) och det hela resulterar i att en numerisk metod för lösning av differentialekvationer måste väljas och appliceras.

## 3.1 Eulers metod

Ett enkelt sätt att hitta approximativa lösningar till differentialekvationer är att använda Eulers metod, som går ut på att uppdatera alla tillstånd enligt (1), där $\bar{y}_t$ är systemets tillstånd vid tidpunkten $t$. I varje tidsteg beräknas ett $\bar{k}$ utifrån rådande samband i det specifika systemet.

$$\bar{y}_{t+1} = \bar{y}_t + \bar{k} \tag{1}$$

För rörliga stelkroppar betecknar $\bar{y}$ position, hastighet, rotation och rotationshastighet, enligt (2), och $\bar{k}$ beräknas som den momentana derivatan av $\bar{y}$ enligt (3).

$$\bar{y} = (p_x, p_y, v_x, v_y, \theta, \omega)^T \tag{2}$$

$$\bar{k} = \left(\frac{\mathrm{d}p_x}{\mathrm{d}t}, \frac{\mathrm{d}p_y}{\mathrm{d}t}, \frac{\mathrm{d}v_x}{\mathrm{d}t}, \frac{\mathrm{d}v_y}{\mathrm{d}t}, \frac{\mathrm{d}\theta}{\mathrm{d}t}, \frac{\mathrm{d}\omega}{\mathrm{d}t}\right)^T = (v_x, v_y, a_x, a_y, \omega, \alpha)^T \tag{3}$$

## 3.2 Runge-Kutta-metoder

Ett problem med Eulers metod är att den leder till instabilitet då den används för att simulera komplexa system. För att klara av att simulera system av fjädrar i fysikmotorn användes en Runge-Kuttalösning av ordning 4. Det innebär att fyra tillståndsderivator ($k_1$-$k_4$), utgående från initialtillståndet $y_t$, beräknas och kombineras till en lösning.

$$\bar{y}_{t+1} = \bar{y}_t + \frac{1}{6}\left(\bar{k}_1 + 2\bar{k}_2 + 2\bar{k}_3 + \bar{k}_4\right) \tag{4}$$

$$\bar{k}_1 = f(\bar{y}_t) \tag{5}$$

$$\bar{k}_2 = f(\bar{y}_t + \frac{1}{2}\bar{k}_1) \tag{6}$$

$$\bar{k}_3 = f(\bar{y}_t + \frac{1}{2}\bar{k}_2) \tag{7}$$

$$\bar{k}_4 = f(\bar{y}_t + \frac{1}{2}\bar{k}_3) \tag{8}$$

I (5-8) beräknar $f$ den momentana derivatan givet ett tillstånd och i (4) vägs dessa derivator samman till ett resultat som, med en mindre avvikelse än Euler-metoden, approximerar medelderivatan mellan två tidssteg.

## 3.3 En hybrid

Medan den ovan beskrivna Runge-Kutta metoden lämpar sig för att lösa differentialekvationer är den svår att tillämpa på kollisioner mellan kroppar, då hastigheter och vinkelhastigheter ska ändras momentant. Metoden som implementerades var därför en hybrid mellan de två ovannämda tillvägagångssätten, där Runge-Kutta av ordning fyra användes för att simulera fjäderkrafter, gravitation och obehindrade förflyttningar, medan en Euler-metod applicerades i slutet av varje steg för att lösa och ge respons på de kollisioner som uppträtt under tidssteget.

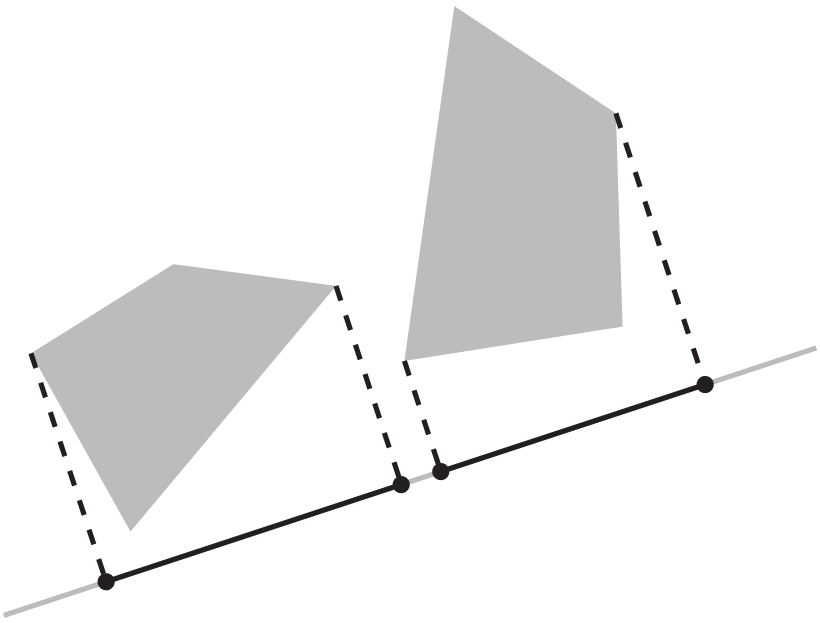
# 4 Kollisionsdetektion

## 4.1 Detektionsmetod

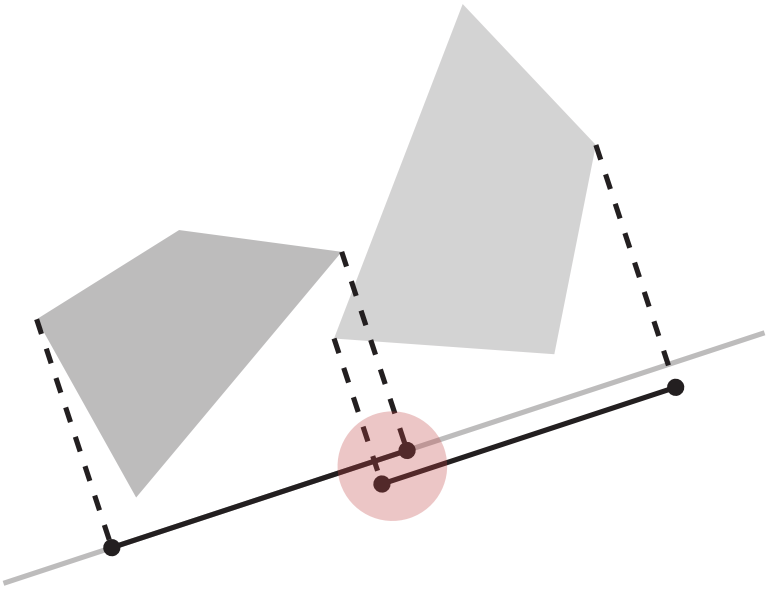
Ett första steg mot att simulera stelkroppars interaktion är att detektera alla kolliderande polygoner i scenen. Att två kroppar i ett diskret system ligger *dikt an* varandra är sällsynt. Vanligare är att de *interpenetrerar* varandra, d.v.s. att ett hörn hos en kropp ligger inuti en annan kropp.

### 4.1.1 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 kropparna *inte* kolliderar genom att hitta en vektor på vilken objektens punkter projiceras på disjunkta intervall (Figur 1). Ett högre antal vektorer ökar beräkningstiden men också sannolikheten för ett korrekt resultat. Det finns emellertid ingen garanti för att SAT, med ett ändligt antal vektorer, med säkerhet kan avgöra om två kroppar inte skär varandra (Figur 2).



Figur 1: Ett fall där SAT *kan* utesluta kollision



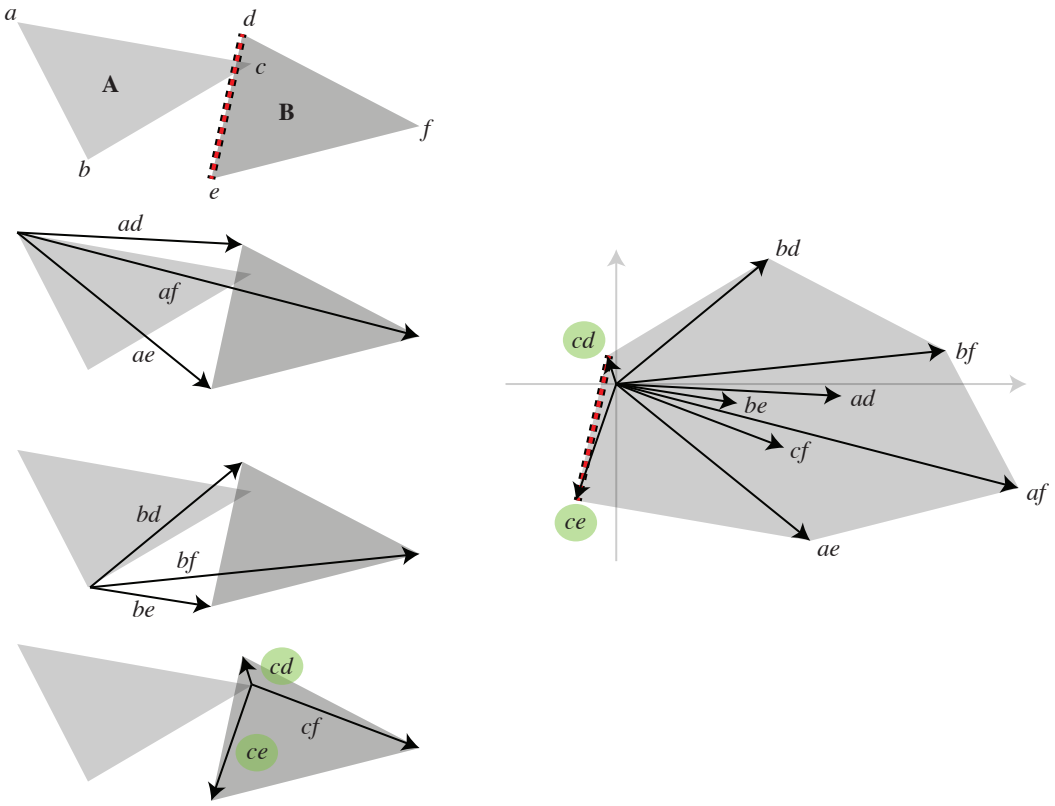
Figur 2: Ett fall där SAT *inte kan* utesluta kollision

### 4.1.2 Gilbert-Johnson-Keerthi

För att inte göra avkall på noggrannheten implementerades istället en annan metod, GJK [1, s. 30] *(Gilbert-Johnson-Keerthi)*. Eftersom metoden kan utesluta fler möjliga kollisioner kommer senare beräkningar kräva mindre tid. GJK bygger på Minkovskidifferensen mellan de två kropparna, definierad enligt (9).

$$\chi_1 \ominus \chi_2 = \{x_1 - x_2 | x_1 \in \chi_1, x_2 \in \chi_2\} \tag{9}$$

De två kropparna korsar varandra om, och endast om, polygonen som Minkovskidifferensen renderar innehåller origo (Figur 3). GJK-algoritmen utreder detta utan att behöva ta alla punkter i beaktning vilket resulterar i lägre komplexitet i beräkningarna [4].



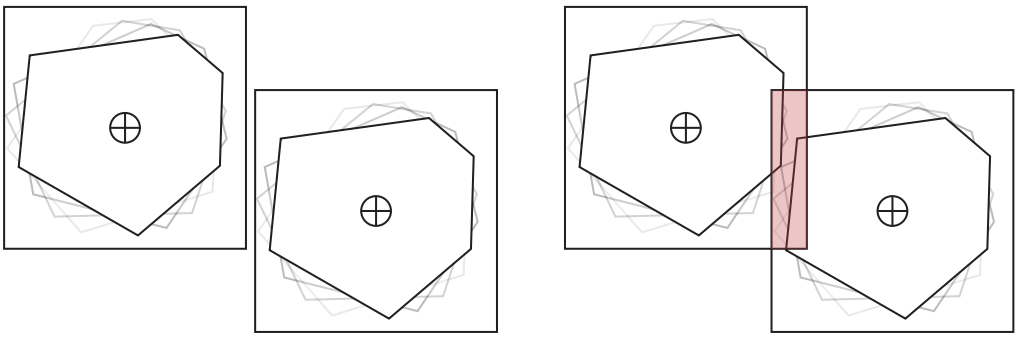

Figur 3: Minkowskidifferensen och dess uppbyggnad

## 4.2 Optimering

Att i varje tidssteg använda GJK-algoritmen för att testa kollision mellan alla kroppar i en scen skulle innebära omfattande beräkningar, vilket skulle tvinga ner fysikmotorns prestanda betydligt. För att undvika dessa beräkningsmängder görs först olika bedömningar av vilka kroppar som riskerar att kollidera varefter de mer komplexa kollisionsdetektorerna appliceras endast på dessa par.

### 4.2.1 Bounding boxes

Det är mer beräkningseffektivt att kontrollera om två kvadrater överlappar varandra än att använda GJK-algoritmen. Därför implementerades en algoritm som räknar ut de minsta kvadrater som garanterat omger alla hörnpunkter hos stelkropparnas polygoner. Detta koncept kallas *bounding boxes*. Naturligtvis går det inte att säkerställa kollisioner med denna metod utan bara avfärda dem (Figur 4).

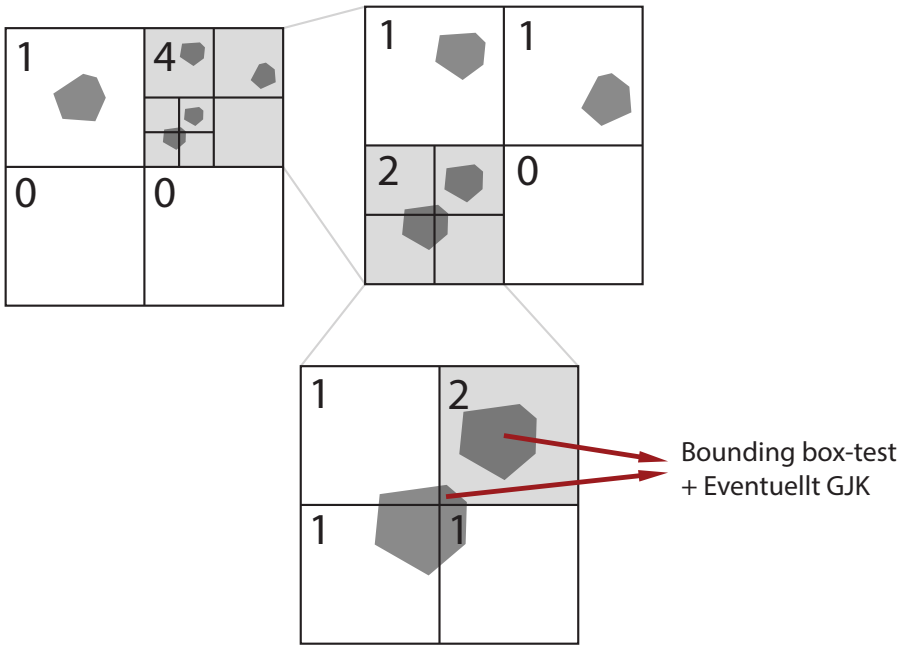


Figur 4: Bounding boxes och hur de skär varandra

### 4.2.2 Quadträd

En ytterligare optimering som kan göras är att använda ett quadträd för att dela upp simuleringsområdet och på detta sätt avgöra vilka kroppar som potentiellt kan ge upphov till kollisioner. Ett quadträd är en datastruktur som delar in ett stort område i fyra mindre likformiga områden, varpå dessa rekursivt klyvs på samma sätt tills önskad granularitet uppnås. Genom att tidigt avfärda kollisioner mellan objekt som ligger i skiljda noder i quadträdet behöver betydligt färre par av kroppar skickas till Bounding box-testet.

Ett quadträd implementerades genom att skapa kvadratiska noder som var och en gavs ett register av vilka stelkroppar som befinner sig inuti dem. Allteftersom kroppar lämnar och äntrar noder uppdateras varje nods register. Snabba uppslagningar kan därmed göras i kollisionsdetektionsprocessen. I figur 5 visas hur noder som innehåller mindre än två stelkroppar helt kan uteslutas från vidare kollisionsdetektion.




Figur 5: Quadträdets iterativa uppdelning

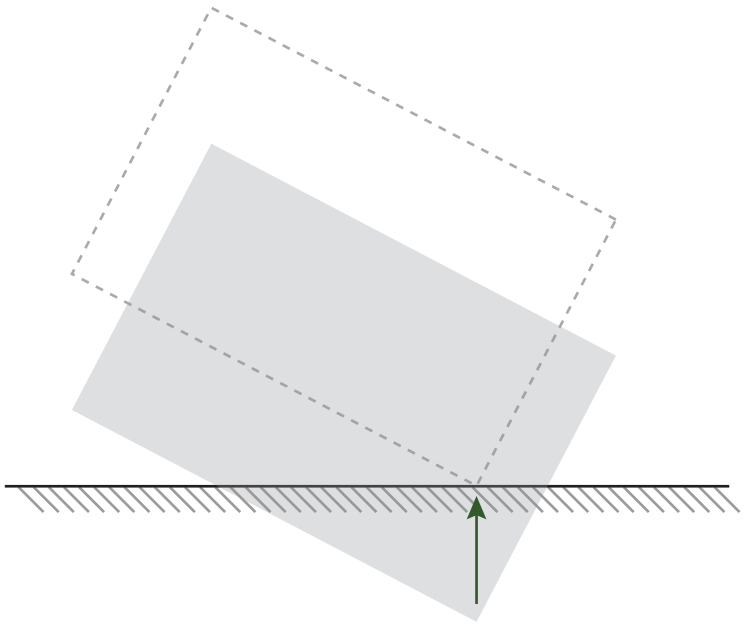
# 5 Kollisionslösning

Att två objekt interpenetrerar varandra representerar inget verkligt fysikaliskt förlopp. Det första som därför måste göras när en sådan händelse upptäckts är att flytta ut objekten från varandra. En metod för detta är att hitta det tidssteg mellan innevarande och föregående bildruta då objekten kolliderade. En analytisk lösning är emellertid komplicerad att implementera med tanke på translations- och rotationshastigheter. En enklare metod är att hitta den punkt hos polygon A som interpenetrerar längst och betrakta den sida på polygon B som ligger närmast denna punkt som en trolig kollisionssida. Hur objekten sedan separeras från varandra går att lösa på olika sätt - linjärt eller icke-linjärt.

## 5.1 Expanding Polytope Algorithm (EPA)

I denna tillämpning användes metoden EPA (Expanding Polytope Algorithm) för att hitta penetrationspunkten och en vektor mellan denna punkt och kollisionssidan, nedan kallad penetrationsvektorn. I det konvexa höljet av Minkowskidifferensen, beskriven i avsnitt 4.1.2, kommer den linje med minst avstånd till origo att vara av intresse. De punkter som spänner upp denna linje har genererats utifrån penetrationspunkten, i Figur 3 benämnd c, och de två punkter, d och e, som spänner upp kollisionssidan.

## 5.2 Linjär Kollisionslösning



Figur 6: Exempel på linjär utflyttning

En linjär kollisionslösning är det enklaste sättet att lösa interpenetration. Penetrationen löses sådant att kropparna a och b (Figur 6) translateras i penetrationsvektorns positiva respektive negativa riktning. Hur stor del av penetrationsvektorn respektive kropp skall translateras beräknas med avseende på dess omvända relativa massa enligt (10).

$$J_{part} = \frac{m_i}{m_1 + m_2} \tag{10}$$

Att translatera en rörlig kropp ut från en stationär kropp utan att påverka dess hastighet medför en förändring av systemets energi. För att motverka detta beräknas hastigheten om med avseende på utflyttningen och gravitationen.

$$|\bar{v}| = \sqrt{|v^2 - 2 \cdot |g_y| \cdot t_y|} \tag{11}$$

Ekvation (11) beskriver den nya hastigheten $v$ där $t$ är kroppens utflyttningsvektor och $g$ är den aktuella gravitationen för systemet.

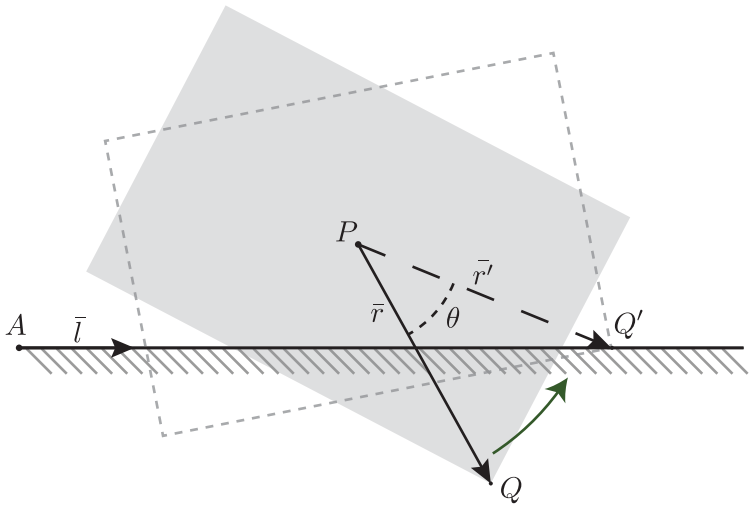
## 5.3 Icke-linjär kollisionslösning

Linjära lösningar för interpenetration skapar problem för kroppar i vila. Detta eftersom gravitationen påverkar kropparna vilket resulterar i interpenetration med underlaget. Detta resulterar i sin tur i en linjär utflytt med avseende på den punkt med längst penetrationsvektor. För kroppar i vila innebär detta att de kommer vibrera istället för att ligga stilla. Dessutom kan det visas att oönskad friktion introduceras i systemet. Ett förslag till förbättring är att dela upp utflyttning av en kropp till två komponenter. En linjär förflyttningskomponent med avseende på kroppens massa (12) och en rotationskomponent med avseende på kroppens tröghetsmoment (13).

$$lm_i = \frac{\frac{1}{m_i}}{\frac{1}{m_1} + \frac{1}{m_2} + \frac{1}{I_1} + \frac{1}{I_2}} \tag{12}$$

$$am_i = \frac{\frac{1}{m_i}}{\frac{1}{m_1} + \frac{1}{m_2} + \frac{1}{I_1} + \frac{1}{I_2}} \tag{13}$$

I simuleringen implementerades endast den linjära utflyttningskomponenten (12) och sedan en explicit lösning (Figur 7) för hur kroppen ska roteras ut.




Figur 7: Exempel på icke-linjär utflyttning

Metoden innebär att lösa ett ekvationssystem med fyra rötter och sedan testa vilken lösning som ger den optimala, dvs. minsta, rotationen ut.

$$Q' = P + \bar{r}' = A + s \cdot \bar{l} \tag{14}$$

$$|\bar{r}|^2 = |\bar{r}'|^2 = (Q - P)^2 = (Q' - P)^2 \tag{15}$$

$$\theta = \arccos \frac{\bar{r} \cdot \bar{r}'}{|\bar{r}|^2} \tag{16}$$

Från (14) och (15) löses $\bar{r}'$ut och rotationen beräknas enligt (16).
För kroppar med stor massa blir den linjära komponenten liten. Djupa penetrationer kan då resultera i att rotationen förvärrar interpenetrationen. I dessa fall kan en nedre gräns för den linjära komponenten införas för att balansera förhållandet mellan rotation och translation [2].

# 6 Kollisionsrespons

## 6.1 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 (17).

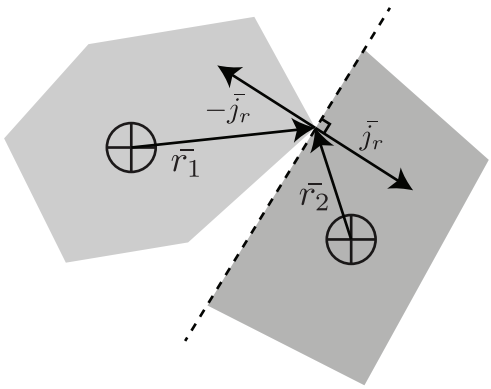
$$\bar{J} = \int_{\Delta t} \bar{F}\ dt = \int_{\Delta t} m\bar{a}\ dt = m\bar{v} \tag{17}$$

Den aktuella implementationen av (17) innehöll inget begrepp om tid utan $t$ representerades istället av ett simuleringssteg.

### 6.1.1 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 (Figur 8). Dessa 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




Figur 8: Impulser i kollisionspunkten

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

$$\bar{v}_1' = \bar{v}_1 + \frac{j_r}{m_1}\hat{n},\ \bar{v}_2' = \bar{v}_2 + \frac{j_r}{m_2}\hat{n} \tag{18}$$

$$\omega_1' = \omega_1 + \frac{j_r}{I_1}(\bar{r}_1 \times \hat{n}),\ \omega_2' = \omega_2 + \frac{j_r}{I_2}(\bar{r}_2 \times \hat{n}) \tag{19}$$

För (18-19) är alla parametrar kända, utom $\bar{v}_i'$, $\omega_i'$ 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 EPA. Punktens hastighet för respektive kropp beräknas enligt (20).

$$\bar{v}_{i_P} = \bar{v}_i + \omega \times \bar{r}_i \tag{20}$$

$$\bar{v}_r' \cdot \hat{n} = -e\bar{v}_r \cdot \hat{n} \tag{21}$$

$$\bar{v}_r = \bar{v}_{1_P} - \bar{v}_{2_P},\ \bar{v}_r' = \bar{v}_{1_P}' - \bar{v}_{2_P}' \tag{22}$$

Ekvation (21) relaterar kropparnas separationshastighet (22), före och efter kollisionen. Parametern $e$ anger materialens studskoefficient. Genom substitution av (18-20) i (21) kan kollisionsimpulsen lösas ut enligt (23).

$$j_r = \frac{-(1+e)\bar{v}_r \cdot \hat{n}}{\frac{1}{m_1} + \frac{1}{m_2} + (\frac{1}{I_1}(\bar{r}_1 \times \hat{n}) \times \bar{r}_1 + \frac{1}{I_2}(\bar{r}_2 \times \hat{n}) \times \bar{r}_2) \cdot \hat{n}} \tag{23}$$

Genom detta skapas en grundläggande teori om hur kollisioner ska bete sig i simuleringen. En impuls beräknas i (23) och appliceras sedan i (18) och (19).

Under implementationen introducerades ett specialfall för kollision mellan rörlig och stationär kropp. Detta resulterade i en förenklad impulsmodell enligt (24).

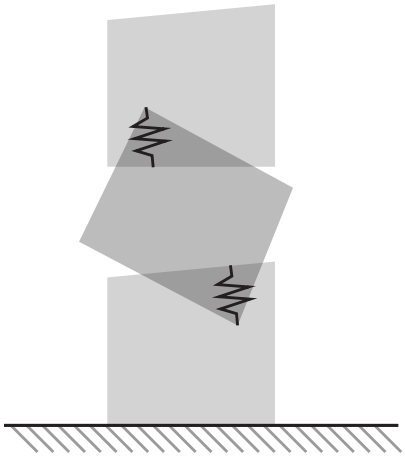
$$j_r = \frac{-(1+e)\bar{v}_r \cdot \hat{n}}{\frac{1}{m_1} + \frac{1}{I_1}(\bar{r}_1 \times \hat{n}) \times \bar{r}_1 \cdot \hat{n}} \tag{24}$$

Ekvation (24) kan likställas med att sätta den stationära kroppens massa och tröghetsmoment till $\infty$.

## 6.2 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.

Metoden medför att en utflyttning ej sker, objekt använder sig istället av mindre impulser för för att fjädra ifrån varandra. Detta kan illustreras (Figur 9) som dämpade fjädrar som spänns mellan kropparna längs penetrationsvektorn.



Figur 9: Tillämpning av dämpade fjädrar som kontaktmodell

Ekvation (25) anger den impuls $J$ som ger fjädringen på båda objekten.

$$\bar{J} = C \cdot \bar{P} \tag{25}$$

$C$ är en empiriskt framtagen konstant för att ge realistiska simuleringar och $\bar{P}$ är penetrationsvektorn mellan de två objekten.

### 6.2.1 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.

# 7 Krafter i impulsbaserade system

## 7.1 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 [5].

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.

## 7.2 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.

$$f = -k \cdot \Delta l \tag{26}$$

I (26) är $k$ fjäderkonstant och $\Delta l$ anger hur utdragen eller sammanpressad fjädern är. Lösningen för (26) ger endast kraftens magnitud, för en flerdimensionell simulering utökas ekvationen till att även innehålla kraftens riktning (27).

$$f = -k(|\bar{d}| - l_0)\hat{d} \tag{27}$$

$$\bar{d} = x_A - x_B \tag{28}$$

Parametrar $x_A$ och $x_B$ (28) är infästningspunkter med avseende på position och rotation för ett tillstånd $y_t$.

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

## 7.3 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.

$$\bar{f}_s = \mu_s \cdot \bar{N} \tag{29}$$

$$\bar{f}_s = \mu_d \cdot \bar{N} \tag{30}$$

Där den statiska modellen (29) används för kroppar i vila och den dynamiska (30) 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, $\bar{t}$.

$$f_t = \begin{cases} -(\bar{f}_{ext} \cdot \hat{t})\hat{t}, & \bar{v}_r = \bar{0}, \bar{f}_{ext} \cdot \hat{t} \leq \bar{f}_s \\ -\bar{f}_s \cdot \hat{t}, & \bar{v}_r = \bar{0}, \bar{f}_{ext} \cdot \hat{t} > \bar{f}_s \\ -\bar{f}_d \cdot \hat{t}, & \bar{v}_r \neq \bar{0} \end{cases} \tag{31}$$

Enligt (31) 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 (29-31) ger den friktionsimpuls som motsvarar $\bar{f}_t$.

$$\bar{j}_s = \mu_s \cdot \bar{j}_r \tag{32}$$

$$\bar{j}_d = \mu_d \cdot \bar{j}_r \tag{33}$$

$$j_f = \begin{cases} -(m\bar{v}_r \cdot \hat{t})\hat{t}, & \bar{v}_r = \bar{0}, m\bar{v}_r \cdot \hat{t} \leq \bar{j}_s \\ -\bar{j}_s \cdot \hat{t}, & \bar{v}_r = \bar{0}, m\bar{v}_r \cdot \hat{t} \geq \bar{j}_s \\ -\bar{j}_d \cdot \hat{t}, & \bar{v}_r \neq \bar{0} \end{cases} \tag{34}$$

Friktionsimpulsen (34) appliceras samtidigt som kollisionsimpulsen för respektive kropp och kollision. Ekvation (32-34) är en ideal [1] friktionslösning för impulsbaserade system. Det kan emellertid visas att (32) och (33) inte ger energibevarande lösningar för höga värden på $\mu_s$ och $\mu_d$.

Friktion simulerades utan hänsyn till friktionstal mellan specifika material. Istället har en naiv förenkling till de allra mest grundläggande egenskaperna implementerats enligt (35).

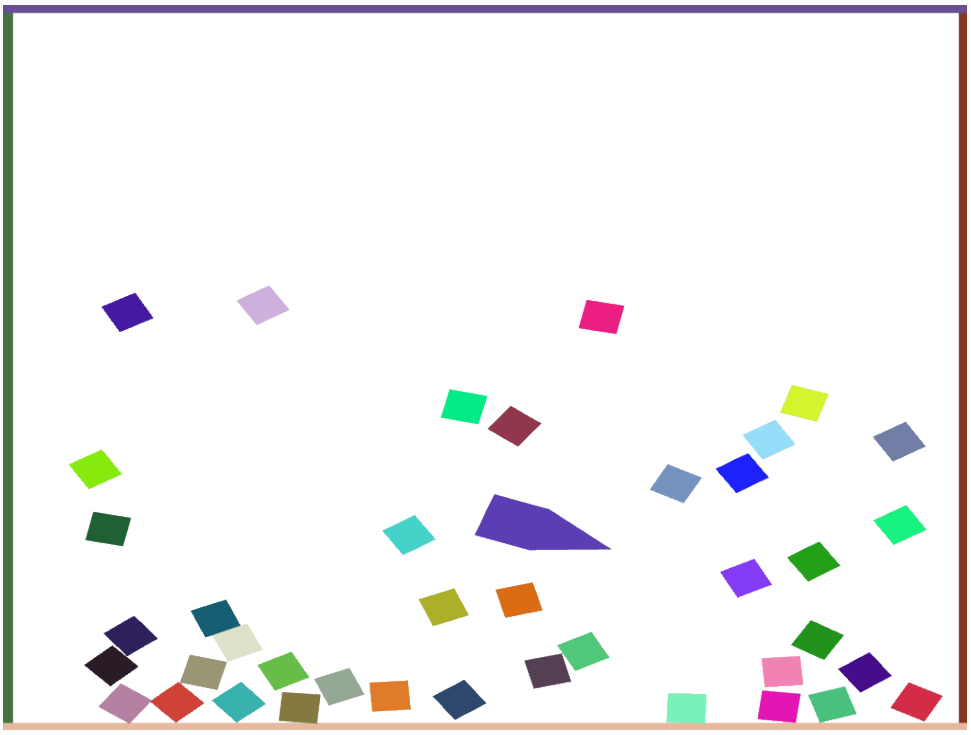
$$\bar{j}_r = \frac{|\bar{v}_r \cdot \hat{n}_\perp|}{\frac{1}{m_1} + \frac{1}{m_2} + \frac{1}{I_1}(\bar{r}_{1_\perp} \cdot \hat{n}_\perp)^2 + \frac{1}{I_2}(\bar{r}_{2_\perp} \cdot \hat{n}_\perp)^2} \tag{35}$$

Ekvation (35) kan jämföras med (23) som bygger på samma princip.

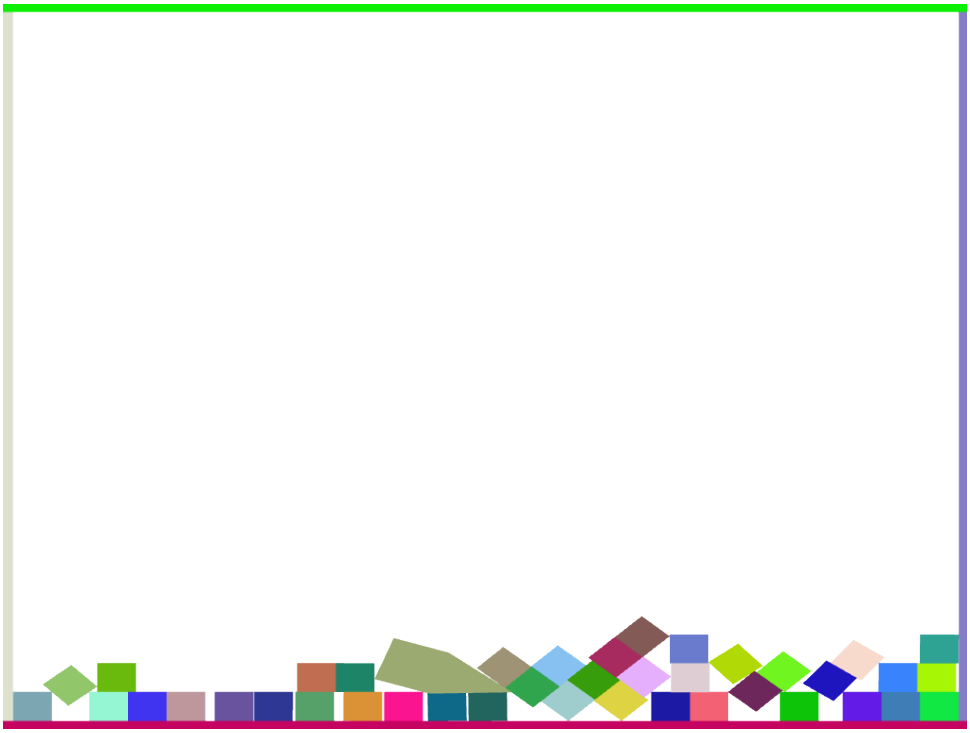
# 8 Resultat

Under projektet har en prototyp till en fysikmotor utvecklats. Motorn kan lösa sofistikerade uppsättningar av konvexa stelkroppar och producera trovärdiga resultat. Exempel på detta är:

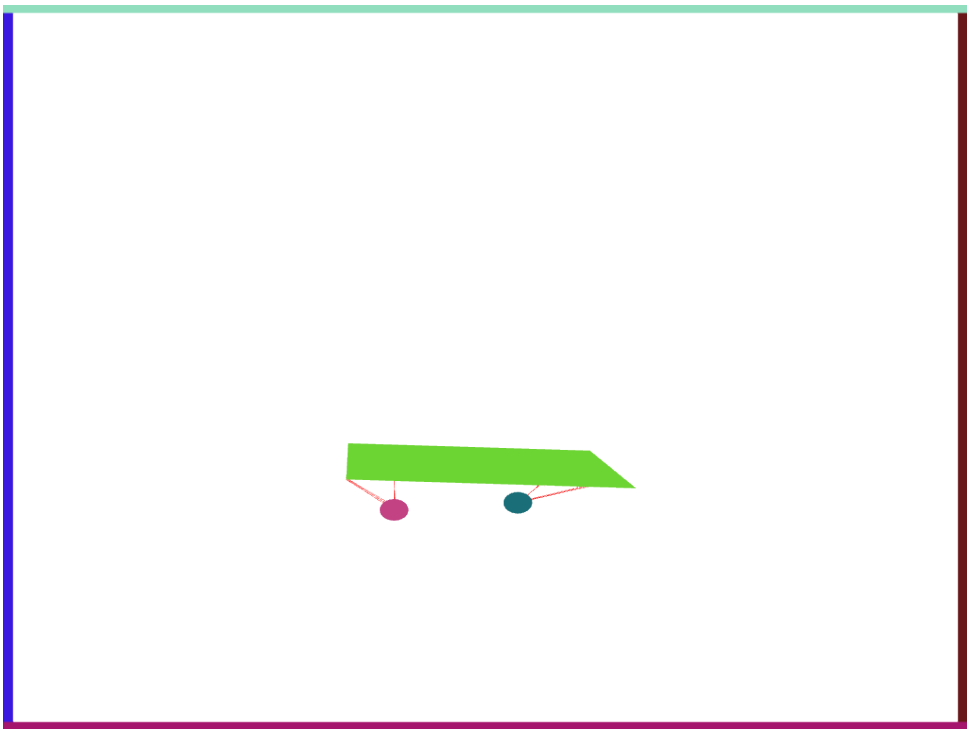
- Energibevarande kollisioner med translation och rotation (Figur 10)
- Vilande kroppar (Figur 11)
- Komplexa massa-fjäder-system (Figur 12)
- Kollisionsdetektion för godtyckliga konvexa objekt



Figur 10: Energibevarande kollisioner



Figur 11: Vilande kroppar



Figur 12: Massa-fjäder-system

# 9 Diskussion

## 9.1 Kontaktgrupper

En möjlig optimering som har diskuterats under projektets gång har varit att gruppera ihop statiska kontakter mellan kroppar. Denna optimering skulle möjliggöra att gruppera ihop kroppar som under en längre tid kolliderat och lägga dessa i vila. Inga beräkningar skulle då behöva utföras på dessa objekt då deras tillståndsvariabler ej förändras när de ligger i detta tillstånd. Den implementation som har beskrivits i rapporten har den svagheten att GJK-algoritmen kommer att behöva köras mellan alla objekt som vilar mot varandra i varje tidssteg.

## 9.2 Konkava kroppar

Att fysikmotorer begränsas till att hantera konvexa polygoner är en vanlig förenkling då det resulterar i en adekvat kompromiss mellan prestanda och generalitet. Konkava former skulle framförallt kräva en mer komplex metod för att detektera kollisioner.
Möjliga lösningar på detta problem är att antingen använda konkava kroppars konvexa hölje vid kollisionsdetektion, eller att segmentera den konkava kroppen till flera mindre konvexa kroppar.

### 9.2.1 Konvext hölje

Ett konkavt objekt kan utvidgas till att vara ett konvext objekt. Det kallas att hitta det konvexa höljet till kroppen. Syftet med detta är att förenkla kroppens kollisionsberäkningar [6, s. 29].

### 9.2.2 Segmentering

Några algoritmer där fysikmotorn automatiskt delar upp konvexa kroppar till konkava har studerats. Implementationen är dock komplex nog att utvecklingsteamet bortsåg från att implementera detta och istället fokusera på mer komplexa fysikaliska samband.

## 9.3 Simuleringsmetoder

I många fall i rapporten har påpekats att det gjorts förenklingar av förlopp som skulle vinna på att simuleras med mer komplexa metoder. Exempel på detta är friktion och penalty-metoder. Runge-Kuttametoden infördes sent i utvecklingsarbetet. Detta är en bidragande faktor till att en stor del av implementationen fortfarande använder Eulermetod. Att föra över ovan nämnda områden från simulering med en sekventiell metod till en simultan Runge-Kuttametod har hög prioritet för eventuellt fortsatt arbete.

# 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] Kumar Ashok, Etheredge Jim, Boudreaux Aaron, *Algorithmic and Architectural Gaming Design: Implementation and Development*. 2012.

[4] 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.

[5] Newton Isaac *Philosophiæ Naturalis Principia Mathematica*. 1687.

[6] Avis David, Bremnes David, Seidel Raimund, *How good are convex hull algorithms?*. 1997.