# Fluidsimulering i 2D, TNM085

Fredrik Lindner, Sebastian Piwell, Aron Tornberg, Niklas Fransson

13 mars 2014

**Sammanfattning**

Den här rapporten avhandlar metoden SPH för fluider, samt implementering i C++. SPH är en överskådlig metod för att simulera traditionellt komplexa problem och är även enkel att implementera på modern hårdvara.

# Innehåll

# 1 Bakgrund

Historiskt sett så har det alltid varit intressant att studera hur fluider rör på sig. När datorer började användas inom många industrier så kom möjligheten att sätta teorierna kring fluidsimulering i praktik. Idag används det inom stora industriprocesser och för rena visuella tillämpningar. Trots att datorer har gjort det möjligt att praktiskt simulera fluider så kvarstår det idag stora begränsningar på applicerbarheten. Därför krävs det många förenklingar för att simuleringarna skall vara ett praktiskt verktyg.

# 2 Syfte

Projektets syfte är att skapa en 2-dimensionell vätskefluidsimulering i realtid för visualisering. Vätskan skall gå att interagera med. Eventuella förenklingar av fysiken kommer att göras för att uppnå snabbhet och stabilitet.

# 3 Modell

Rörelse i en fluid kan förklaras med Navier-Stokes ekvationer. En fluid kan vara gas, vätska eller plasma. Dessa ekvationer är baserade på Newtons andra lag och bevarar moment, massa och är inkompressibel. Ekvationen ger ett flödesfält som bestämmer hastigheten för en given punkt i tid och rum i fluiden.

$$\rho(\frac{\partial \vec{u}}{\partial t} + \vec{u} \cdot \nabla \vec{u}) = -\nabla p + \nu \Delta \vec{u} \tag{1}$$

$$\nabla \cdot \vec{u} = 0 \tag{2}$$

$$\begin{aligned} \vec{u} &= \text{hastighet} \\ \rho &= \text{mass-densitet} \\ p &= \text{tryck} \\ \nu &= \text{viskositet} \end{aligned}$$

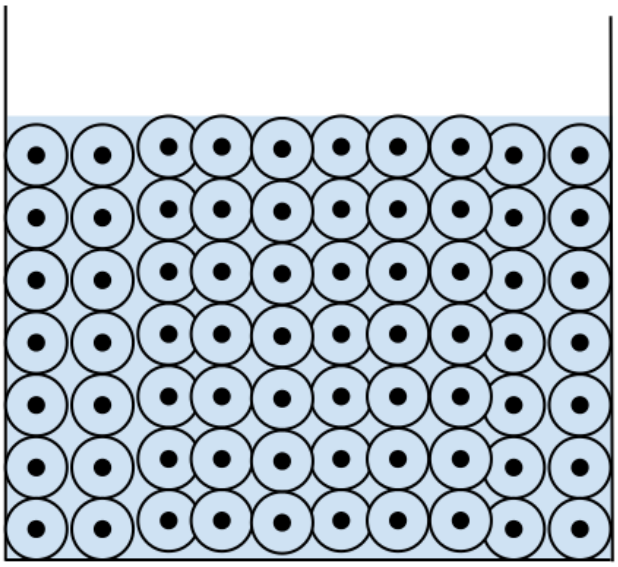
Vänsterledet är produkten av mass- och accelerationsdensiteter. Högerledet är summan av kraftdensiteterna. I högerledet kan ekvationen utvecklas med externa krafter som gravitation och ytspänning (1). Bevarandet av massan för en inkompressibel fluid beskrivs vidare (2).

I nästan alla fall är dessa ekvationer icke-linjära. Det finns ännu inte något bevis att det alltid finns lösningar till dessa ekvationer i tre dimensioner. Navier-Stokes ekvationer har varit och är fortfarande ett komplext matematiskt problem.

# 4 Simulering

Det finns två primära sätt att praktiskt simulera en fluid, Eulers- och Lagranges modell. Den Euleriska metoden går ut på att man har en fixed volym där man på bestämda positioner (celler) räknar ut fluidens egenskaper så som densitet, hastighet, tryck osv. I den Lagrangiska metoden behandlar man istället fluiden som en uppsättning partiklar som skulle kunna representera en molekyl av fluiden. Varje partikel har egenskaper som hastighet, acceleration och position men även tryck och densitet. Det Lagrangiska tillvägagångssättet att simulera fluider är vanligt och relativt enkelt att implementera.

"Smoothed Particle Hydrodynamics" (SPH)[1] är en Lagrangisk metod för att simulera fluider. Den togs fram på 70-talet i syfte att lösa astrofysiska problem. SPH delar upp fluiden i diskreta element (partiklar) där en bestämd "utjämningslängd" (även kallat interaktionsradie) jämnar ut en partikels egenskaper med hjälp av olika filterkärnor. Vilken filterkärna som används beror på vilken egenskap som ska jämnas ut. När en fluid representeras av en uppsättning partiklar är masskonserveringen given då det alltid finns en bestämd mängd partiklar med en bestämd massa. Varje egenskap hos partikeln beräknas med avseende på alla andras egenskaper inom partikelns interaktionsradie. Förenklingar i metoden SPH för vätskesimulering har gjorts för att uppnå projektets syfte [2][3].



Figur 1: En vätska representerad av partiklar med interaktionsradie.

## 4.1 Mass-Densitet

Massan för en partikel bestäms innan simuleringen och ses som en konstant. För att bestämma massdensiteten summeras massan för alla partiklar inom interaktionsradien och viktas med en filterkärna baserad på Gaussian (4). Resultatet blir massdensiteten kring partikeln.

$$\rho_i = \sum_{\lim_j} m_j \, W_{def}(r_i - r_j, h) \tag{3}$$

$$W_{def} = \frac{315}{64\pi h^9}(h^2 - r^2)^3 \tag{4}$$

## 4.2 Krafter

### 4.2.1 Tryck

Trycket är den första delen av högerledet i Navier Stokes ekvation. Trycket kan bestämmas med den ideala gaslagen (5).

$$pV = nRT \tag{5}$$

V är volymen per enhetsmassa, n är antalet partiklar i mol, R är den allmänna gaskonstanten och T är temperaturen. Högerledet är konstant för simuleringar denna rapport avser och kan därför sättas till en styvhets konstant k. För att få attraherande krafter måste det tas hänsyn till vilodensiteten. Uttrycket för trycket:

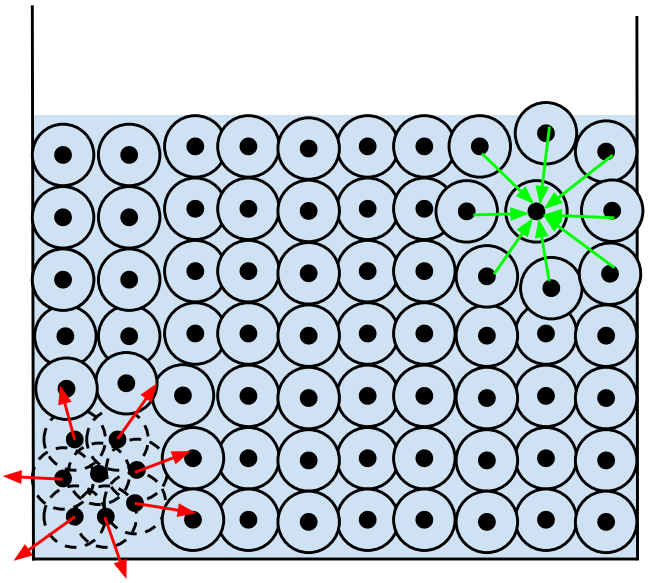
$$p = k(\rho - \rho_0) \tag{6}$$

För att få en inkompressibel fluid behöver styvhetskonstanten k vara så hög som möjligt. Det betyder att ingen partikel ska kunna gå in i den andra. Detta kan bli ett problem vilket tas upp senare i rapporten.

För att bestämma kraften från trycket summeras trycket hos partiklar inom interaktionsradien på liknande sätt som för mass-densiteten (7). Viktningskärnan som används för denna summering är till skillnad från massdensiteten nu mycket snävare och spetsig (8). För att få en jämnare kraftfördelning som mer följer Newtons tredje lag används egenskaper från närliggande partiklar.

$$f_i^{tryck} = -\sum_{j \neq i} \frac{p_i + p_j}{2} \frac{m_j}{\rho_i \rho_j} \nabla W_{tryck}(r_i - r_j, h) \tag{7}$$

$$W_{tryck} = -\frac{45}{\pi h^6}\frac{(h-r)^2}{r}\vec{r} \tag{8}$$



Figur 2: Partiklar som verkar på andra partiklar.

### 4.2.2 Viskositet

Viskositeten är den andra delen av högerledet i Navier-Stokes ekvation (1). Viskositeten står för strömningsmotståndet i fluiden. Den uppkommer ifrån friktionen mellan partiklarna och minskar den relativa hastigheten mellan dem. Viskositeten storlek bestäms av en konstant som beror på vilken typ av fluid man ska simulera. Kraften som viskositeten påverkar partikeln med fås av uttrycket:

$$f_i^{viskositet} = \mu \sum_j \frac{m_j}{\rho_i \rho_j}(u_j - u_i)\nabla^2 W_{viskositet}(r_i - r_j, h) \tag{9}$$

$$W_{viskoitet} = \frac{45}{\pi h^6}(h-r) \tag{10}$$

Där den relativa hastigheten används för att få en symmetrisk kraft som följer Newtons tredje lag. Viktningsknärnan är alltid positiv för att alltid fungera som en dämpningsterm och inte tillsätta energi i systemet och göra det instabilt.

### 4.2.3 Gravitation

Gravitationen påverkar alla partiklar lika mycket enligt uttrycket:

$$f_i^{gravitation} = \rho_i g \tag{11}$$

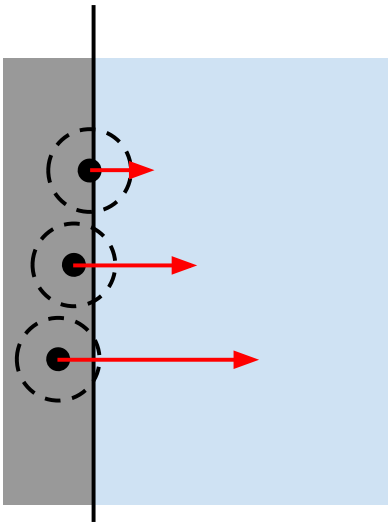
Där g är gravitationsaccelerationen som verkar i negativt y-led i simuleringen.

## 4.3 Kollisionshantering

### 4.3.1 Väggar

För att inte partiklarna ska försvinna utanför simuleringsrymden har enkla väggar vid kanterna skapats. Vid väggarna sker en kollisionshantering som aktiveras då väggen är inom partikelns radie. Kollisionsresponsen beror på styvheten på vätskan och infallsvinkeln. Den uppdaterar partikelns acceleration vilket simulerar en vägg.

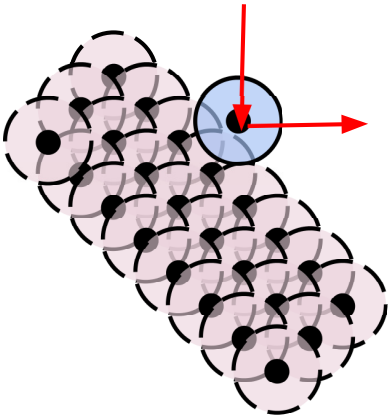
En fjäderliknande kraft i motsatt riktning från partikelns acceleration skapas baserat på hur långt partikeln har penetrerat väggen, se figur 3. Detta är för att simulera att partiklar med stor hastighet studsar hårdare mot en vägg i jämförelse med långsammare partiklar.



Figur 3: Partiklar som får en acceleration från vägg.

### 4.3.2 Kollisionspartiklar

I simuleringen kan man lägga till kanter i form av kantpartiklar. Kantpartiklarna definieras som de andra partiklarna men med tyngre massa, därmed större massdensitet och tryck. De funkar som de andra partiklar förutom att deras position inte ändras. Med deras tunga massa och fix positionering fungerar de som kanter. I en offline simulering räcker denna definition av kanter. Men för en interaktiv simulering kan inte tidssteget vara tillräckligt litet för att det ska fungera bra. För enstaka partiklar eller partiklar med höga hastigheter hinner inte kantpartiklarna påverka de inkommande partiklarna. Partiklarnas hastighet förändras då inte och åker igenom kanten.



Figur 4: Kantpartiklar

För att få bättre kanter i en realtids simulering behövs en omedelbar verkan på partikelns hastighet då de träffar kanten. Det fås genom att uppdatera partiklarnas hastighet på ett annat sätt då de är inom kantpartiklarnas interaktionsområde. Istället för Navier-Stokes ekvation för uppdatering av hastigheten används en enkel partikel mot partikel kollisionshantering. En reflektionsvinkel beräknas mellan partiklarna och hastigheten uppdateras i den riktningen med en viss dämpning.

## 4.4 Integrering

För uppdatering av position och hastighet har integreringsmetoden Leap-Frog [2] används. Den fungerar genom att ta halva steg för positionen och hastigheten. Hastigheten börjar ett halvt steg bakåt. I och med det så "hoppar" positionen och hastigheten över varandra. Hastigheten för den aktuella tiden fås approximativt av halvstegen bakom och framför. Det är en stabil och snabb integreringsmetod som lämpar sig till simuleringen.

$$u_{t+\frac{1}{2}\Delta t} = u_{t-\frac{1}{2}\Delta t} + \Delta t a_t \tag{12}$$

$$r_{t+\Delta t} = r_t + \Delta t u_{t+\frac{1}{2}\Delta t} \tag{13}$$

$$u_{-\frac{1}{2}\Delta t} = u_0 + \frac{1}{2}\Delta t a_0 \tag{14}$$

$$u_t \approx \frac{u_{t-\frac{1}{2}\Delta t} + u_{t+\frac{1}{2}\Delta t}}{2} \tag{15}$$

# 5 Implementering

Ett problem med simuleringar som denna där partiklarnas tillstånd är beroende av varandra är att tidskomplexiteten ökar kvadratiskt om man för varje partikel måste utföra beräkningar för alla andra partiklar. Två steg används för att lösa detta problem.

I steg ett delas simuleringen upp i ett rutnät med höjd och bredd lika med interaktionsradien med en lista kopplad till varje cell i rutnätet. Varje partikel i systemet koppas till den lista vars cell partikeln befinner sig i.

I steg två kontrolleras vilka andra partiklar som befinner sig inom interaktionsradien till varje partikel och dessa placeras i en lista av grannar till partikeln i fråga. För att undvika kvadratisk tidskomplexitet i denna kontroll användes faktumet att samtliga partiklar som är grannar till en specifika partikel antingen kommer befinna sig i samma cell som partikelen i fråga eller någon av de 8 omkringliggande cellerna och utför enbart kontrollen för listorna kopplade till dessa 9 celler samt så begränsas antalet grannar i listan till 64.

I uträkningen av hydrodynamiken används sedan endast de partiklar som finns i listan av grannar för den aktuella partikeln. Vilket minskar tidskomplexiteten markant.

Vid beräkningen av hydrodynamiken så beror varje partikels tillstånd enbart på tillståndet för partiklarna i föregående tidssteg. Det finns alltså inga teoretiska hinder för räkna ut hydrodynamiken för samtliga partiklar samtidigt.

OpenMP [4] har används för att räkna på flera partiklar samtidigt. *pragma*-direktiv har placerats innan de loopar som itererar genom partiklarna för att instruera kompilatorn om att dessa loopar kan köras parallellt på centralprocessorn.

# 6 Rendering

För att rita simuleringen skickas en lista med koordinater för samtliga partiklar via OpenGL till grafikkortet. Två olika metoder för utritning finns i simuleringen. Dels ett punktläge där varje partikel helt enkelt ritas ut som en punkt och dels metaballs-läget [7].

I metaball-läget används metaballs för att rita ut partiklarna. Metaballs fungerar på så sätt att man för varje partikel definierar en radialt avtagande funktion. För varje punkt i simuleringsmiljön summerar man sedan dessa funktioner och kollar ifall summan är över ett tröskelvärde. Om den är över tröskelvärdet säger man att punkten är fylld och annars tom. På detta sätt kan partiklar representeras som bollar som har egenskapen att de ser ut att smälta ihop då de kommer nära varandra. Så kallade *metaballs*.

Implementationen av metaballs i denna simuleringen fungerar på så sätt att varje partikel ritas ut som en pointsprite vars textur definieras av en fragment-shader (gradientball.frag) som texturerar pointspriten som en radialt avtagande gradient enligt funktionen:

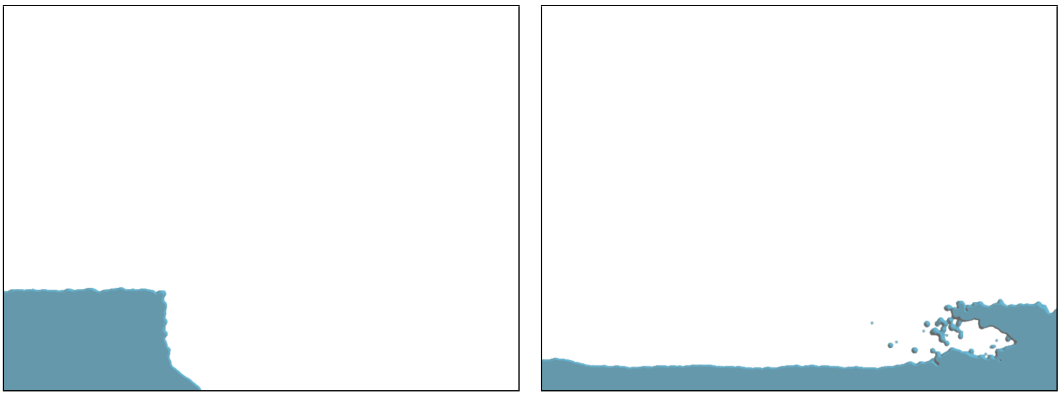
$$r^4 - r^2 + 0.25 \tag{16}$$

Som en optimeringsåtgärd räknas $r^2$ ut genom att ta skälarprodukten av den vektor som går mellan bildpunkten och partikelns koordinater med sig själv.

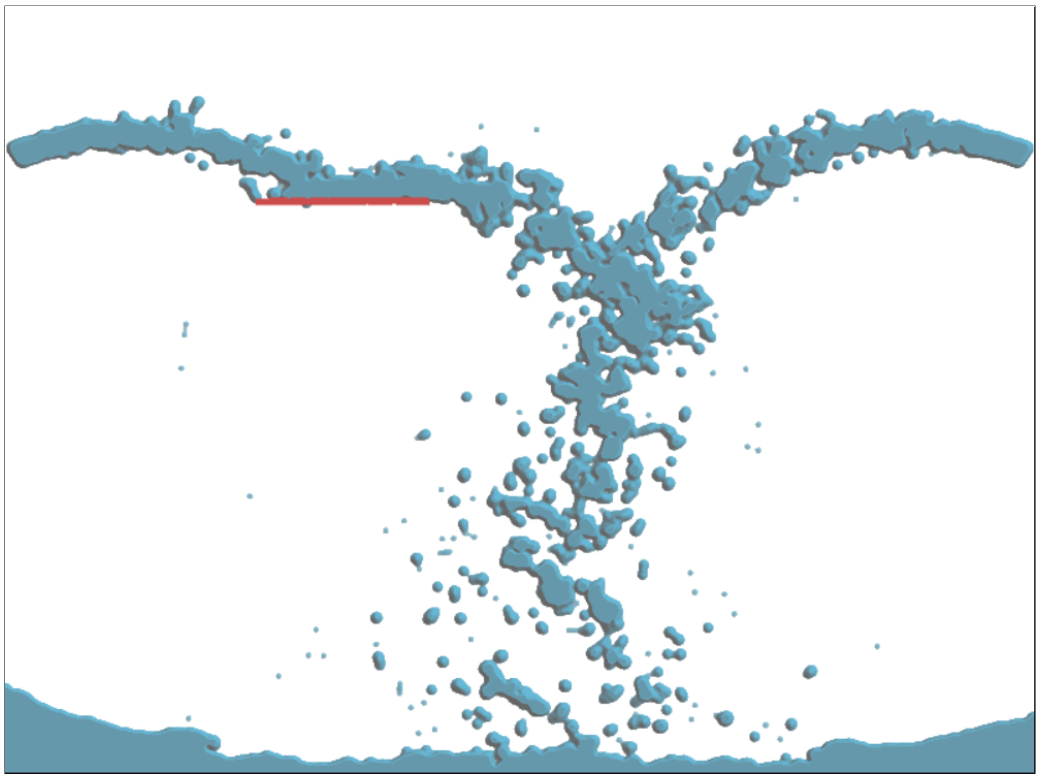
Istället för att rita partiklarna till skärmen så används efterbehandling. Bilden renderas istället till en textur som behandlas i en fragment-shader (metaballs.frag) för att sedan ritas på skärmen. I fragment-shadern så sätts alpha-värdet på texturen dels så att värden under tröskelvärdet sätts till noll men också på så att en mjuk övergång sker för att undvika en skarp pixlig kant. En normalkarta beräknas även genom att tolka den ingående texturen som en höjdkarta. Normalkartan används för att räkna ut diffust ljus till som används för att bestämma texturens RGB-värden. På så sätt åstadkoms en 3D-effekt på vätskan.

# 7 Resultat

Den interaktiva applikationen som togs fram för att demonstrera simuleringen klarar av att simulera och visa ca 16000 st partiklar samtidigt. Det gjordes ett flertal olika begynnelsevillkor för fluiden. I några fall släpptes vätskan ifrån en höjd för att studera hur komprimerad vätskan blev. Det testades också att distribuera ut vätskan över tid, ungefär som en vattenspridare.



Figur 5: Ett vanligt testfall, *dam break*, vid olika tidpunkter.



Figur 6: Exempel på partikelkälla och kantpartiklar.

# 8 Diskussion

## 8.1 Begränsningar

SPH medför begränsningar både på snabbhet och styvhet hos en fluid. För att simulera t.ex vatten som är en inkompressibel vätska så krävs det mycket hög styvhet. En hög styvhet medför att dynamiken på systemet måste vara väldigt snabbt, vilket i sin tur medför att steglängden i simuleringen måste vara väldigt kort.

Att få ett interaktivt system som använder metoden SPH för att simulera en fluid är ofta en avvägning mellan styvhet och hastighet på systemet.

## 8.2 Förbättringar

Ett flertal förbättringar när det kommer till stabilitet/styvhet och hastighet kan göras.

### 8.2.1 GPGPU

GPGPU (General-purpose computing on graphics processing units) kan tillämpas för att ytterligare parallellisera beräkningarna. Modern grafikhårdvara är speciellt gjord för att göra enkla beräkningar parallellt och därmed skulle man kunna beräkna på fler partiklar parallellt.

### 8.2.2 PCISPH

PCISPH (predictive-corrective incompressible SPH)[6] är en utökad SPH-metod. Metoden försöker att approximera hur trycket skall ändras utan att faktiskt beräkna trycket. Detta gör att ett större tidssteg kan användas utan att systemet stabilitet påverkas.

Det finns även andra metoder baserade på SPH som lyckas bibehålla en hög styvhet utan att påverka stabiliteten på systemet, tex Constraint Fluids[5] av Bodin et. al.

### 8.2.3 Sortering av grannar

När systemet delas in i ett rutnät för sortering av grannar så är inte interaktionsradien den självklara storleken på cellerna. En annan vanlig lösning är celler med en bredd och höjd som är två gånger interaktionsradien. Fördelen med detta är att det krävs mindre celler överlag samt att man endast behöver iterera genom fyra celler för att hitta en partikels grannar istället för nio. En nackdel är att den totala arean med partiklar som måste kontrolleras blir större och kommer därmed innehålla fler överflödiga partiklar. Vad som är effektivast är oklart och har ej undersökts.

# Referenser

[1] R.A. Gingold and J.J. Monaghan, *Smoothed particle hydrodynamics: theory and application to non-spherical stars*, Mon. Not. R. Astron. Soc., Vol 181 1977

[2] Micky Kelager, *Lagrangian Fluid Dynamics Using Smoothed Particle Hydrodynamics*, University of Copenhagen, 2006

[3] Marcus Vesterlund, *Simulation and Rendering of a Viscous Fluid using Smoothed Particle Hydrodynamics, 2004*

[4] http://openmp.org/

[5] K. Bodin, C. Lacoursière, M. Servin, *Constraint Fluids*, IEEE Transactions on Visualization and Computer Graphics, Jan 2011

[6] B. Solenthaler, R. Pajarola, *Predictive-corrective incompressible SPH*, University of Zurich, 2009

[7] Blinn, J. F., *A Generalization of Algebraic Surface Drawing*, ACM Transactions on Graphics 1 (3), 1982