Et billede, der indeholder tekst

Automatisk genereret beskrivelse

18. marts 2022

Frederik Cayré Hede-Andersen

3.a2 HCØL

Simulering af væskedynamik

En sop i fysik A og digital design undervisning A

Indholdsfortegnelse

[Abstract 3](#_Toc98348576)

[Indledning: 3](#_Toc98348577)

[Problemformulering 3](#_Toc98348578)

[Hoveddel 4](#_Toc98348579)

[Navier-Stokes 4](#_Toc98348580)

[Computeren og væskedynamik som algoritmer 5](#_Toc98348581)

[Feltbaseret 5](#_Toc98348582)

[Partikelbaseret 6](#_Toc98348583)

[Programmet 6](#_Toc98348584)

[Design 7](#_Toc98348585)

[Dokumentation 7](#_Toc98348586)

[Analyse af data og algoritmer 9](#_Toc98348587)

[Referencer 10](#_Toc98348588)

# Abstract

[skrives til sidst]

# Indledning:

Computere ses alle steder i verden og bruges til alle mulige ting. Fra styring af ovne til design af havelåger til beregninger af yderligere cifre af pi, computere kan en masse. Modellering af virkeligheden bliver også gjort på computere i mange industrier rundt omkring i verden. Det at kunne opleve verden og genskabe fænomener gennem en computer, uden faktisk at skulle gå ud med et kamera og måle på tingene, kan bruges til rigtig mange ting. Det kan gøre besværet med bestemte opgaver meget mindre og det kan gøre folk i stand til at opleve ting der normalt ikke sker. Naturen er vores at kommandere inde i computeren. Specifikt inden for feltet væskedynamik er computersimuleringer ekstra brugbare, da det kan være svært at måle på mange af en væskes egenskaber kontinuert. Væsker er dog svære at simulere, og det er tungt at køre på langt de fleste computere, især hvis simuleringen skal være naturlig. Der er dog mange måder at simulere væskedynamik på, ved hjælp af forskellige algoritmer. Det er det vi kigger på i denne opgave.

# Problemformulering

”Hvordan kan et program vise forskellene i simuleringer af væskedynamik, samt give indblik i relevante applikationer af simuleringer, med forskellige algoritmer.”

Denne problemformulering vil jeg svare på ved hjælp af en række underspørgsmål, samt tests af et program jeg skriver, og en bedre generel forståelse af feltet. De følgende underspørgsmål skal give os en bedre forståelse af emnet, samt få os på et rette spor når det gælder analysen.

Underspørgsmål:

* Hvad er væskedynamik og hvordan beskrives væskedynamik matematisk?
* Hvordan kan computere bruges til fluidsims?
* Hvilke algoritmer og metoder kan bruges til fluidsims?
* Hvordan kan et program designes og testes for at vise forskellene mellem forskellige fluidsims?
* Hvad er tidskompleksiteten af de implementerede algoritmer?
* Hvad kan forsøgets resultater fortælle os om algoritmerne?
* Hvordan kan algoritmerne bruges i forskellige industrier?
* Hvad fortæller forsøget om anvendelsen af fluidsims inden for computerspilsindustrien?

Note: Fluidsims = simuleringer af væskedynamik

# Hoveddel

## Navier-Stokes

Væskedynamik er det felt der håndterer bevægende stoffer på flydende og på gas form. Igennem lang tid har væsker været set på og undret over, for at kunne modellere deres bevægelse matematisk. Archimedes lov stammer fra denne undren om væsker, og der ville stille og roligt blive udviklet mere avancerede ideer om væskedynamik gennem århundrederne. Da oplysningstiden begyndte skete der virkelig noget, og i løbet af 1800-tallet fik to matematikere Claude-Louis Navier og George Gabriel Stokes udviklet en definitiv samling matematiske formler. Disse formler beskriver væsker med viskositet og er kendte for at være meget svære at løse for alle på nær de nemmeste væskesystemer. Vi kommer hovedsageligt til at arbejde med inkompressible væsker, da de er nemmere at have at gøre med. Luft er derfor ikke noget vi kommer til at se på, vand derimod er en inkompressibel væske, så den er mulig for os at modellere. Selvom alle væsker er en smule kompressible kommer det til at have en ubetydelig effekt i langt de fleste tilfælde.

For at holde formlerne nogenlunde overskuelige og for at gøre det nemmere at implementere kigger vi kun på inkompressible væsker. Et simpelt væskesystem kan derfor beskrives ved hjælp af et sæt af Navier-Stokes formlerne der ikke er så tunge. De følgende formler kan bruges til at beskrive et væskesystem med en inkompressibel væske.[[1]](#endnote-1)

Den øverste af formlerne (1) beskriver hvordan masse konserveres gennem væskesystemet. Da væsken er inkompressibel må densiteten af væsken være den samme over alle punkter. Formlen (1) beskriver hvordan der ikke kan flyde mere eller mindre væske ind i et punkt end der flyder væske ud af punktet. og ’’ er tre forskellige operatorer der arbejder på henholdsvis skalar- og vektorfelter. kan ses som en vektor af de delvist afledte af skalarfeltets dimensioner. kan beskrives således for et todimensionelt skalarfelt:[[2]](#endnote-2)

Så på et skalarfelt F bliver F til et vektorfelt, hvor hver vektor peger i retning af den største stigning af værdierne af scalarfeltet.

Kigger vi derimod på den anden operator, , agerer den på vektorfelter. Tager vi og kigger på u, vektorfeltet over hastigheder i væsken, bliver til et skalarfelt. prikkes essentielt med hastighedsvektoren, beskrevet således:

Dette beskriver den divergens der eksisterer i vektorfeltet, i dette tilfælde hastighedsvektorfeltet. Der betragtes de omkringliggende vektorer og ses på hvor meget ’hastighed’ der løber ind og ud af punktet. er negativt når vektorerne løber mere ind mod punktet end ud. Formlen (1) sætter dette til at være 0. Da densiteten er konstant over hele væsken, fordi den er inkompressibel, kan der ikke løbe mere væske ind i et punkt end der løber ud. Dette ville nemlig give at masse skulle forsvinde eller skabes i punktet, hvilket ikke ville give nogen mening.

Betragter vi den anden formel (2), får vi beskrevet hvordan hastighedsvektorerne ændres over tid. Her konserveres hastighed. Formlens forskellige led har hvert at gøre med sin egen effekt på væsken, og kan betragtes alene. Det første led har at gøre med konvektion og behandler komposanterne af hastighedsvektorerne for at få deres resulterende bevægelse. Leddet kan udvides til dette:

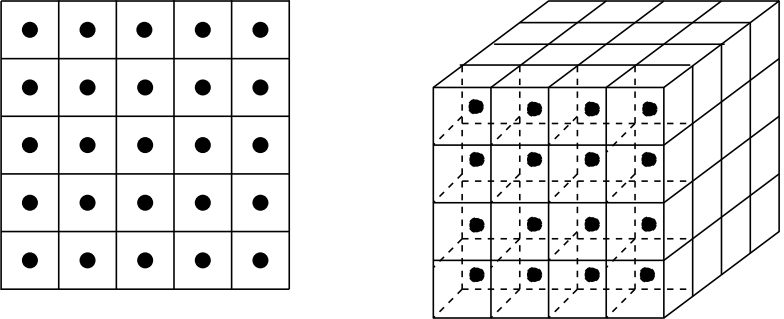
Læg mærke til at er to forskellige tegn, det første designerer hastighedsvektoren og det andet designerer x-komposanten af vektoren. Vi ender altså med en vektor der er en originale hastighedsvektor u ganget på ændringen af komposanterne over deres retninger. Det andet led i formlen (2) beskriver hvordan trykgradienten påvirker bevægelsen af væsken, hvor meget væskens hastighed er dog påvirket af densiteten. Det tredje led er diffusion, og beskriver ved hjælp af viskositeten af væsken hvordan væsken bliver påvirket af resten af væsken rundt om i forhold til viskositeten. Det sidste led er alle andre kræfter der kunne påvirke væsken. Det kunne f.eks. være tyngdekraft.1

## Computeren og væskedynamik som algoritmer

Vi har set på Navier-Stokes formlerne for inkompressible væsker, men hvordan oversætter vi det til et computerprogram. Den største forskel mellem hvordan formlen fungerer og hvordan vi kommer til at implementere den er domænet. Formlerne beskriver et kontinuert væskesystem, hvilket er besværligt at arbejde med. Da vi kommer til at simulere væsker numerisk og ikke analytisk må vi gøre væskesystemet diskret. I virkeligheden er en væske ikke kontinuer. Zoomer vi langt nok ind når vi et punkt hvor væsken er individuelle partikler, der opfører sig meget anderledes end den overordnede væske som formlerne beskriver. For at have et diskret væskesystem er der to metoder der er brugt oftest. Såkaldte feltbaserede metoder og partikelbaserede metoder. Vi kommer til at implementere to algoritmer i programmet vi skriver, en feltbaseret og en partikelbaseret metode. Vi starter med at se på den feltbaserede metode.

### Feltbaseret1

Til den feltbaserede metode består væskesystemet af et stort gitter med felter, hvor hvert celle i feltet har en samling værdier. Hvert celle har værdierne defineret ud fra centrummet af cellen, og svarer til et gennemsnit af væskens egenskaber inden for cellen. Størrelsen og mængden af celler bestemmer hvor præcis simuleringen af væsken er, dvs. hvor fin en opløsning man får. Da tid også skal være diskret bliver der taget faste ”timesteps” . Et mindre timestep giver en bedre tidsopløsning, og gør simuleringen yderligere præcis, på bekostning af at flere steps skal tages før en vis mængde tid er gået i simuleringen, altså at den bliver langsommere.



Figur 1, De diskrete felters celler har deres værdier defineret ud fra deres centrum. Ophavsret: Jos Stam, fra ”Stable Fluids” 1999

Ud fra gitteret er der flere felter, blandt andet vektorfeltet hastighedsfeltet. For at simulere væskesystemet itereres der en algoritme med en serie funktioner. Hvert celle i domænet bliver betragtet for at opdatere dens værdier. Hvert iteration over et timestep giver væsken en ny tilstand, med alle felterne opdateret, som der så kan laves en ny iteration ovenpå. Tiden det tager at beregne en iteration er hvad vi er interesserede i når det gælder de tests vi vil lave på programmet.

### Partikelbaseret[[3]](#endnote-3)

Til den partikelbaserede metode bruges konceptet om Lagrange partikler, der beskriver en bestemt metode for at beregne positionen af partikler i en væske. Der er mange forskelle mellem den feltbaserede og den partikelbaserede metode. Måden væsken gøres diskret på i denne metode er ved at have væsken være en samling partikler. Væske i virkeligheden er også en samling partikler, men metoden her simulerer ikke enkelte molekyler. Partiklerne repræsenterer dele af væsken på sammen måde som et felt repræsenterer væsken i feltet. Der er dog nogle store fordele ved den partikelbaserede metode. Der skal ikke tages specielt hensyn til massebevarelse som i den feltbaserede metode, da hvert partikel svarer til en del af væskens samlede masse, og da partiklerne hverken forsvinder eller bliver skabt (medmindre det er en ønsket effekt). Der er også mulighed for områder ikke at have væske i. Det lyder måske lidt åbenlyst, men i den feltbaserede metode vi har talt om, er densiteten af væsken den samme over hele simuleringsområdet, og der er derfor altid den samme mængde væske over det hele. I stedet for at hvert felt itereres på, itereres der på hvert partikel i væsken. Der beregnes ændringer i hastighed og position baseret på partiklerne umiddelbart omkring det partikel man kigger på. Det har den fordel at man kan have et meget stort simulerings domæne, uden at bruge så meget simuleringstid, som man ville med den feltbaserede metode. Den partikelbaserede metode skalerer nemlig simuleringstid på antallet af partikler, i modsætning til antallet af felter for den feltbaserede metode.

## Programmet

Til den feltbaserede metode kommer selve algoritmernes design til at være stærkt inspireret af artiklen ”Fluid Simulation for Dummies”, 2006, af Mike Ash, samt YouTube videoen ”Coding Challenge #132: Fluid Simulation”, 2019, af The Coding Train. Artiklen bygger på en anden artikel, ”Real-Time Fluid Dynamics for Games”, 2003, af Jos Stam, der er en efterfølger af ”Stable Fluids” som jeg har refereret til tidligere.

Til den partikelbaserede metode kommer programmet til at være baseret på pseudokoden der beskriver algoritmen, i ”Interactive 2D Particle-based Fluid Simulation for Mobile Devices”(2013), af Daniel Månsson.3

### Design

Designet af programmet skal facilitere simuleringerne og tests af simuleringerne. Programmet skrives i Processing (Java), da jeg har erfaring i det og da den feltbaserede algoritme er blevet implementeret i Processing allerede. For bedst at besvare problemformuleringen og få nogle gode testresultater, måles der på nogle få metrikker hvorefter visse forhold kan analyseres mellem dem. Tid er en stor faktor. Præcision er også en stor faktor. Der vil derfor ses på hvor lang tid det tager at beregne en enkel iteration, eller et enkelt timestep rettere sagt. Antallet af felter bliver også taget i betragtning, da det er direkte korreleret med præcisionen af simuleringen. Der vil også laves en meget begrænset visuel analyse af programmets output, for at dømme forskelle i præcision.

Et separat program skrives til den partikelbaserede metode, hvor algoritmerne implementeres baseret på den pseudokode beskrevet i den videnskabelige artikel.3 Programmet kommer til at have struktur baseret på hvad der bedst vil give mening ud fra algoritmerne beskrevet i artklen. De samme metrikker som i det feltbaserede program vil også blive målt og testet på.

### Dokumentation

#### Feltbaseret

Det fletbaserede program brugte programmet ”Coding Challenge #132: Fluid Simulation”[[4]](#endnote-4) som basis, men der er lavet store tilføjelser og ændringer. For at kunne teste programmet mest effektivt måtte programmet omskrives. En hel skulle kunne institueres for at kunne testes nemmest, og da programmet originalt var skrevet med en masse globale variabler skulle disse parameteriseres. En ny klasse blev oprettet for at holde på koden der egentlig lå i sketchens hoved klasse. De forskellige globale variabler blev derefter omskrevet til at være parametre i konstruktørerne for de tre klasser fra basis programmet. På den måde kunne hele væskesimuleringen blive lavet på ny når som helst, og med et nyt sæt parametre.

For at få et godt sæt data fra programmet blev der lavet en test metode. For at få mange datapunkter besluttede jeg mig for at lave 100 tests, hvert med en opløsning af simuleringen på test#\*10. 100 datapunkter tænkte jeg ville være rigeligt til at kunne komme frem til et ordentligt resultat. At opløsningen skulle springe med 10 pixels på hver led hver test, var da jeg tænkte at det ville give nogle mere informative resultater. Forskellen på 37x37 og 38x38 er ikke særlig stor, 30x30 og 40x40 er en markant større forskel, og ville give mere indsigt over de 100 datapunkter. For at få nogle mere repræsentative data, ville tiden det tager at simulere en enkelt frame være et gennemsnit. De første 100 frames af simuleringen for hver opløsning ville få simuleringstiden skrevet ned og derefter taget gennemsnittet af. I slutningen ville programmet spytte en .csv fil ud, der derefter kunne bearbejdes i et program som Excel.

Originalt var ideen at gennemsnittet af simuleringstiden skulle være i et givent tidsrum, som 10 sekunder. Denne fremgangsmåde ville have fordelen at hele testen ville tage en fast mængde tid, nemlig 1000 sekunder i alt. Problemet var at de senere tests ville få meget færre værdier at tage gennemsnittet af end de tidligere tests, da hver frame ville tage markant forskellige tider at beregne. De tests med meget små opløsninger som 30x30 eller 80x80 ville nemt kunne få 1000 frames på de 10 sekunder, hvor en test som 900x900 ville få måske 20 frames i alt. Jeg endte derfor med at vælge et fast antal frames.

#### Partikelbaseret

Programmet med den partikelbaserede metode var svært at skrive. Algoritmen til den partikelbaserede metode var ikke skrevet på forhånd, men bestod kun af pseudokode.3 Pseudokoden kunne følges, men var besværlig at skrive efter, da oversættelsen til datastrukturer i Processing var ret besværlig. Der bliver ofte refereret til partkel objekter der har positions og hastigheds vektorer. Da forskellige metoder i den indbyggede PVector klasse i processing, opererer og ændrer selve vektoren, skulle man passe ekstra godt på.

Figur , pseudokode fra den videnskabelige artikel3

Her ses en del af pseudokoden, der beskriver en vektor der bliver sat til en ny vektor der svarer til partikel p’s positionsvektor trukket fra partikel n’s vektor. I programmet er den indbyggede klasse PVector brugt til diverse vektorer der må være brug for. I oversættelsen af pseudokoden ville man nemt tænke på at bruge PVector.sub() til at trække den ene vektor fra den anden. Altså:

Problemet med denne linje kode er dog at dette ville ændre n.**pos**. En sideeffekt vi ikke er interesserede i, da vi kommer til at skulle bruge n.**pos** senere i programmet. Der kunne man i stedet bruge .copy() metoden i PVector til at få en kopi, så man ikke ændrer den egentlige positionsvektor.

Et billede, der indeholder tekst

Automatisk genereret beskrivelseEt andet problem der skulle løses er hvordan objekter bliver behandlet på i Processing, eller rettere sagt Java. Tager vi et bredere kig på pseudokoden fra før kan vi se et nyt problem.

Figur , pseudokode fra den videnskabelige artikel. Algoritme 3 fra underkapitlet ApplyViscosity.3

Pseudokoden henviser til at man itererer gennem en liste af partiklerne, og en liste af naboerne til partiklerne. Bruger man en klassisk for-each løkke, som , får man dog et problem længere nede. På linje 12, markeret med grønt, ændres partiklet p’s hastighedsvektor. Problemet stammer fra forskellen mellem ’pass by value’ og ’pass by reference’. giver som en kopi af partiklet, altså det giver os værdien, og er dermed ’pass by value’. Det betyder at når vi prøver at ændre p.**vel** så ændrer vi ikke det rigtige partikels vektor, men kopiens vektor hvilket vi ikke ønsker. Dette problem løses bruge en standard for-løkke. Der bruger vi for-løkkens variabel, i f.eks., sammen med partikel listens .get() metode, til at få en reference til partiklen . Ulempen ved at gøre det sådan er at der skal skrives mere, da vi skal kalde .get() af listen og eventuelt give referencen til et nyt variabel. Fordelen ved at gøre det på denne måde er at det faktisk virker, hvilket har en højere prioritet.

Der var også yderligere komplikationer, hvilket bliver diskuteret senere.

## Analyse af data og algoritmer

# Referencer

1. Stam, J. (2003). Real-Time Fluid Dynamics for Games. Hentet fra dgp.toronto.edu: <https://www.dgp.toronto.edu/public_user/stam/reality/Research/pub.html>  
    [↑](#endnote-ref-1)
2. Parth G. “*This Downward Pointing Triangle Means Grad Div and Curl in Vector Calculus (Nabla / Del) by Parth G*”, youtube.com (23-03-2021).  
   <https://youtu.be/hI4yTE8WT88> [↑](#endnote-ref-2)
3. Månsson, Daniel. (2013). “Interactive 2D Particle-based Fluid Simulation for Mobile Devices”, Bachelor’s Thesis at KTH.  
   <https://www.diva-portal.org/smash/get/diva2:676516/FULLTEXT01.pdf> [↑](#endnote-ref-3)
4. The Coding Train. ”Coding Challenge #132: Fluid Simulation”, youtube.com (12-02-2019). <https://youtu.be/alhpH6ECFvQ> [↑](#endnote-ref-4)