SIMULASI DINAMIKA FLUIDA MENGGUNAKAN METODA *SMOOTHED PARTICLE HYDRODYNAMICS* (SPH)

SKRIPSI

oleh :

RIZAL ARIFIN

0510930049-93

JURUSAN FISIKA

FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM

UNIVERSITAS BRAWIJAYA

MALANG

2009

SIMULASI DINAMIKA FLUIDA MENGGUNAKAN METODA *SMOOTHED PARTICLE HYDRODYNAMICS* (SPH)

SKRIPSI

Sebagai salah satu syarat untuk memperoleh gelar Sarjana Sains
dalam bidang fisika

oleh :
RIZAL ARIFIN
0510930049-93

JURUSAN FISIKA
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS BRAWIJAYA
MALANG

2009

LEMBAR PENGESAHAN TUGAS AKHIR

SIMULASI DINAMIKA FLUIDA MENGGUNAKAN METODA *SMOOTHED PARTICLE HYDRODYNAMICS* (SPH)

oleh :
RIZAL ARIFIN
0510930049-93

Setelah dipertahankan di depan Majelis Penguji
pada tanggal .....
dan dinyatakan memenuhi syarat untuk memperoleh gelar Sarjana Sains dalam bidang fisika

Pembimbing I	Pembimbing II
Dr. Eng. Agus Naba, MT.	**Dr. rer. nat. Abdurrouf, M. Si.**
NIP. 197208061995121001	**NIP. 197209031994121001**

Mengetahui,
Ketua Jurusan Fisika
Fakultas MIPA Universitas Brawijaya

Adi Susilo, Ph.D
NIP. 196312271991031002

LEMBAR PERNYATAAN

Saya yang bertanda tangan di bawah ini :

Nama : Rizal Arifin
NIM : 0510930049
Jurusan : Fisika
Penulis tugas Akhir berjudul : Simulasi Dinamika Fluida Menggunakan Metoda *Smoothed Particle Hydrodynamics* (SPH)

Dengan ini menyatakan bahwa :

1. **Tugas Akhir ini adalah benar-benar karya saya sendiri, dan bukan hasil plagiat dari karya orang lain. Karya-karya yang tercantum dalam Daftar Pustaka TA ini, semata-mata digunakan sebagai acuan / referensi.**
2. **Apabila kemudian hari diketahui bahwa isi TA saya merupakan hasil plagiat, maka saya bersedia menanggung akibat hukum dari keadaan tersebut.**

Demikian pernyataan ini dibuat dengan segala kesadaran.

Malang, Nopember 2009
Yang menyatakan,

(Rizal Arifin)
NIM. 0510930049

SIMULASI DINAMIKA FLUIDA MENGGUNAKAN METODA *SMOOTHED PARTICLE HYDRODYNAMICS* (SPH)

ABSTRAK

Simulasi dan analisis permasalahan dinamika fluida pada umumnya dilakukan dengan menggunakan metoda *Finite Difference Methods* (FDM), *Finite Volume Methods* (FVM), atau *Finite Element Methods* (FEM). Namun ketiga metoda tersebut kurang cocok digunakan pada kasus aliran permukaan bebas karena penggunaan *mesh* membatasi ruang kerja simulasi. Metode Penggunaan metoda *Smoothed Particle Hydrodynamics* (SPH) yang berbasis *Lagrangian* adalah pilihan yang tepat untuk permasalahan tersebut karena tidak menggunakan *mesh*. Stabilitas, akurasi dan kecepatan simulasi berbasis SPH ditentukan oleh pemilihan parameter dan fungsi kernel yang tepat. Penelitian ini bertujuan untuk menentukan parameter simulasi beserta pengaruhnya dan penggunaan fungsi kernel yang tepat untuk meningkatkan kualitas hasil simulasi. Dari penelitian ini, didapatkan jumlah partikel ideal untuk simulasi sebesar 5000 partikel setiap 0,001 m^3 dan 40 partikel berada di dalam radius kernel. Kernel yang paling sesuai adalah kernel *Quartic* dengan *Laplacian* menggunakan kernel *Viscosity*.

FLUID DYNAMICS SIMULATION BY USING SMOOTHED PARTICLE HYDRODYNAMICS (SPH) METHODS

ABSTRACT

Simulation and analysis of fluid dynamics problem was determined by scientist by using Finite Difference Methods (FDM), Finite Volume Methods (FVM), or Finite Element Methods (FEM). However, these methods are not suitable to be applied for free surface flows case because of mesh using restrict simulation domain. Smoothed Particle Hydrodynamics (SPH) methods is suitable for free surface flows because it is not using mesh. Stability, accuracy, and speed of simulation based on SPH are determined by correct selection of parameter and kernel function. This research is aimed to determine simulation parameter and its affect, and the using of correct kernel function for improving the simulation achievement quality. It has been found the number of ideal particles for simulation could be acquired 5000 particles every 0.001 m^3 and 40 particles in the kernel radius. The best kernel is Quartic kernel with using Laplacian Viscosity kernel.

KATA PENGANTAR

Alhamdulillahirobbilalamin, Sholawat dan salam semoga terlimpahkan kepada Nabi yang Mulia Muhammad *Shallallahu 'alaihi wa 'ala alihi wa sallam.*

Karya tulis sederhana ini merupakan laporan hasil penelitian penulis tentang penggunaan metoda SPH untuk simulasi dinamika fluida. Di dalam karya tulis ini akan dijelaskan teori singkat mengenai dinamika fluida dan SPH. Pembaca karya tulis ini diharapkan telah mengenal matematika dasar dan mekanika dasar agar dapat mengikuti setiap penjelasan dan persamaan di dalam karya tulis ini dengan baik. Di bagian lampiran akan disertakan *source code* utama implementasi metoda SPH dengan menggunakan bahasa C++ dan OpenGL yang dapat digunakan oleh pembaca secara bebas tanpa hak cipta (*open source*) sebagai referensi untuk pengembangan sistem yang lebih baik.

Perkenankanlah penulis untuk mengucapkan terima kasih kepada berbagai pihak yang secara langsung ataupun tidak telah membantu jalannya pengerjaan Tugas Akhir ini, khususnya kepada:

1. Orang tua dan keluarga yang telah memberikan banyak pelajaran hidup.
2. Bapak Agus Naba selaku Dosen Pembimbing I, atas motivasi, inspirasi, dan bantuannya yang banyak selama ini.
3. Bapak Abdurrouf selaku Dosen Pembimbing II, yang selalu menyempatkan waktunya untuk memberikan pelajaran pada penulis.
4. Bapak Sugeng Rianto atas inspirasi dan ilmunya tentang pemodelan sistem fisis.
5. Bapak Adi Susilo, selaku Ketua Jurusan Fisika, Universitas Brawijaya.
6. Seluruh Dosen dan karyawan Jurusan Fisika.
7. Seluruh teman Fisika, Universitas Brawijaya, khususnya teman seangkatan 2005, semoga kekeluargaan ini tak lekang dimakan zaman, teman kos Cak To, Puri, Kang Pras, Amir, Imam, Udin, Wahyu, Agung, dan semua pihak yang tidak dapat disebut satu persatu.
8. Kepada Haris atas printernya dan komputernya.
9. Teman-teman di Masjid Al-Muhajirin yang banyak membantu penulis dalam memakmurkan masjid.

10. Teman-teman guru SMP Ar-Rahmah, terima kasih atas kerjasamanya.
11. Siswa dan siswi ar-Rahmah, kalian telah menghibur saya dari kejenuhan mengerjakan skripsi ini.

Penulis menyadari akan segala kekurangan yang sangat mungkin terdapat dalam karya tulis ini. Dengan segala kekurangan dan kelebihannya semoga karya tulis ini dapat memberikan manfaat bagi orang yang ingin mempelajari pemodelan dinamika fluida dengan metoda SPH.

Malang, Nopember 2009

Penulis

DAFTAR ISI

Halaman

DAFTAR GAMBAR

Halaman

DAFTAR TABEL

Halaman

DAFTAR LAMPIRAN

Halaman

BAB I
PENDAHULUAN

1.1 Latar Belakang

Sistem fisis aliran fluida telah dikembangkan oleh Euler, Navier, dan Stokes (dari tahun 1750 sampai 1850) dan menghasilkan persamaan *Navier-Stokes*. Persamaan *Navier-Stokes* dapat digunakan untuk menyelesaikan masalah dinamika fluida secara analitis untuk kasus yang sederhana. Pada tahun 1950 para ilmuwan mulai menggunakan komputer dan mengembangkan algoritma numerik untuk menyelesaikan persamaan *Navier-Stokes* (Stam, 2003).

Simulasi dan analisis permasalahan dinamika fluida pada umumnya dilakukan dengan menggunakan metode *Finite Difference Method* (FDM), *Finite Volume Method* (FVM), atau *Finite Element Method* (FEM). Metode FDM dan FVM yang berbasis *Eulerian* ataupun FVM yang berbasis *Lagrangian* memerlukan *mesh* sebagai kerangka komputasi untuk menyelesaikan persamaannya. Sayangnya ketiga metoda tersebut memiliki keterbatasan ketika dihadapkan pada kasus dengan distorsi yang besar, material bergerak yang berinteraksi, batas (*boundary*) yang tidak tetap, dan permukaan bebas (*free-surface*). FEM tidak dapat digunakan untuk kasus distorsi yang ekstrem, sedangkan metode berbasis *Eulerian* sulit jika digunakan dalam kasus material bergerak yang berinteraksi, batas yang tidak tetap, dan permukaan bebas (Liu, 2003).

Salah satu metoda yang dipilih untuk simulasi aliran fluida adalah menggunakan sistem partikel. Partikel diasumsikan sebagai suatu entitas yang tidak mempunyai parameter ruang (dapat bergerak bebas) namun mempunyai parameter posisi, kecepatan, dan gaya interaksi antar sesamanya. Metoda *Smoothed Particle Hydrodynamics* (SPH) merupakan salah satu metoda pemodelan aliran fluida berbasis partikel yang mempunyai kelebihan berupa terdapatnya interaksi antara satu partikel dengan partikel yang lain dan interaksi partikel dengan obyek (Hamdi, 2008).

Salah satu faktor terpenting dalam simulasi fluida menggunakan metoda SPH adalah fungsi kernel. Penggunaan fungsi kernel yang sesuai menentukan akurasi dan efisiensi dalam komputasi. Berbagai bentuk fungsi kernel telah dibuat dan

dikembangkan untuk optimasi dalam simulasi metoda SPH seperti fungsi kernel *Gaussian,* fungsi kernel *Cubic-spline,* fungsi kernel *Quartic*, dan masih banyak yang lain (Liu dan Liu, 2003). Akan tetapi ada beberapa permasalahan terkait karakteristik fungsi kernel setelah diturunkan terhadap ruang yang dapat mengurangi keakuratan simulasi. Oleh karena itu, ketepatan pemilihan berbagai parameter awal simulasi seperti massa partikel, jari-jari partikel, lebar fungsi kernel, dan time-step juga mempengaruhi kualitas dari simulasi yang dilakukan.

1.2 Perumusan Masalah

Permasalahan dalam penelitian ini adalah sebagai berikut:

1. Bagaimana pengaruh penentuan parameter-parameter simulasi dinamika fluida dengan metoda SPH.
2. Bagaimana menggunakan fungsi kernel yang sesuai dengan karakteristik fisis dari parameter yang akan dihitung dalam simulasi dengan persamaan SPH.
3. Bagaimana menerapkan solusi dari permasalahan pertama dan permasalahan kedua untuk mensimulasikan dinamika fluida menggunakan metoda SPH.

1.3 Batasan Masalah

Penelitian ini terbatas pada lingkup sebagai berikut:

1. Penentuan massa partikel menggunakan persamaan kontinuitas.
2. Metoda pemodelan yang digunakan adalah SPH.
3. Parameter fisis yang dihitung dalam simulasi adalah kerapatan, tekanan, dan viskositas namun tidak ditampilkan dalam bentuk data.
4. Model yang digunakan dalam simulasi adalah dam jebol (*breaking dam*).
5. Simulasi dan visualisasi menggunakan bahasa pemrograman C++ dan OpenGL.
6. Analisis metoda hanya menggunakan hasil visualisasi.

1.4 Tujuan Penelitian

Penelitian ini mempunyai tujuan sebagai berikut:

1. Untuk menentukan pengaruh penentuan parameter-parameter simulasi dinamika fluida dengan metoda SPH
2. Untuk menentukan fungsi kernel yang sesuai dengan karakteristik fisis dari parameter yang akan dihitung dalam simulasi dengan persamaan SPH.
3. Untuk menerapkan solusi dari permasalahan pertama dan permasalahan kedua untuk mensimulasikan dinamika fluida menggunakan metoda SPH.

1.5 Manfaat Penelitian

Manfaat yang diharapkan dari penelitian ini adalah adanya peningkatan akurasi dan efisiensi simulasi dinamika fluida dengan menggunakan metoda SPH.

(halaman ini sengaja dikosongkan)

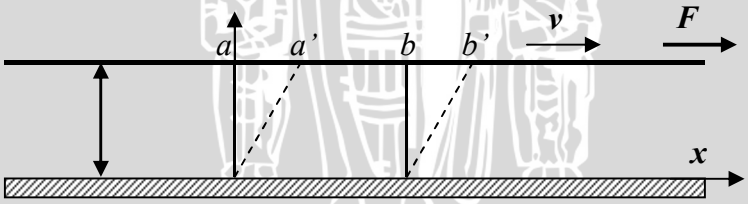
BAB II
TINJAUAN PUSTAKA

2.1 Dinamika Fluida

Dalam sub-bab ini akan dijelaskan beberapa teori mengenai dinamika fluida yang berkaitan dengan karya tulis ini, yang meliputi definisi fluida, aliran fluida, dan persamaan *Navier-Stokes* sebagai persamaan umum dinamika fluida.

2.1.1 Definisi fluida

Menurut Streeter dan Wylie (1985), fluida adalah zat yang berubah bentuk secara kontinyu bila terkena tegangan geser, betapapun kecilnya tegangan geser tersebut. Gaya geser merupakan gaya yang menyinggung permukaan, dan jika gaya geser dibagi dengan luas permukaan menghasilkan tegangan geser rata-rata pada permukaan tersebut. Suatu titik pada fluida mempunyai tegangan geser yang merupakan nilai batas perbandingan gaya geser terhadap luas, dengan definisi luas dapat berupa sebuah titik. Sebagai suatu contoh, kita tinjau suatu zat yang ditempatkan di antara dua pelat sejajar di mana jarak antara dua pelat dibuat jauh lebih kecil dari luas permukaan pelat seperti diperlihatkan pada Gambar 2.1.

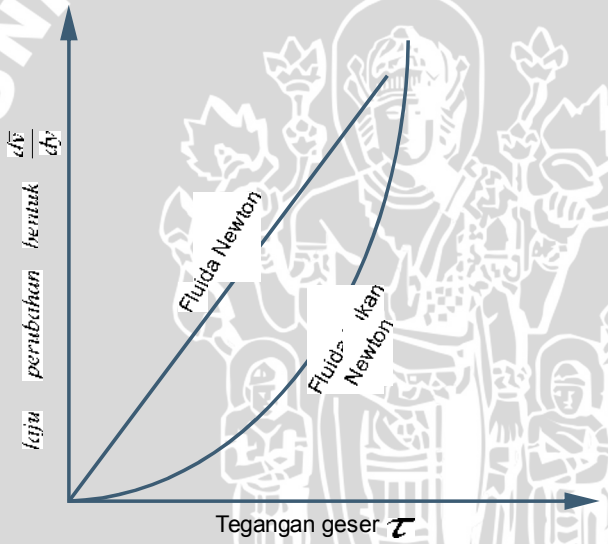


Gambar 2.1 Perubahan bentuk yang diakibatkan gaya geser konstan

Gambar 2.1 menunjukkan pelat bagian bawah terpasang tetap, sedangkan pada pelat bagian atas digerakkan dengan suatu gaya **F** yang menimbulkan tegangan geser pada zat di antara pelat tersebut sebesar –, dengan *A* merupakan luasan pergeseran yaitu sepanjang "*a* ke *a* "' kali lebar pelat.

Berdasarkan eksperimen, didapatkan suatu fakta bahwa fluida yang langsung bersentuhan dengan batas benda padat mempunyai kecepatan yang sama dengan batas tesebut.

Berdasarkan tegangan geser yang diterapkan, fluida dapat dibagi menjadi dua jenis yaitu fluida Newton dan fluida bukan Newton. Fluida Newton ditandai dengan hubungan linear antara besarnya tegangan geser dan laju perubahan bentuk yang diakibatkan. Sebaliknya dalam fluida bukan Newton hubungan antara tegangan geser dan laju perubahan bentuk fluida tidak linear. Fluida yang termasuk dalam fluida Newton adalah gas dan cairan encer, sedangkan hidrokarbon berantai panjang yang kental termasuk dalam fluida bukan Newton (Streeter dan Wylie, 1985).



Gambar 2.2 Diagram *rheologi*

Gambar 2.2 menunjukkan linearitas hubungan antara tegangan geser dengan laju perubahan bentuk pada fluida Newton dan ketidaklinearan pada fluida bukan Newton.

2.1.2 Aliran fluida

Halliday dan Resnick (1978) menyebutkan tiga karakteristik aliran fluida yaitu:

1. Aliran fluida dapat berupa aliran tunak (*steady*) atau tak tunak (*non steady*). Aliran tunak terjadi jika kecepatan fluida pada setiap titik yang diberikan adalah konstan terhadap waktu. Kondisi ini dapat terjadi pada aliran air yang tenang (mempunyai kecepatan rendah). Sedangkan pada kondisi tak tunak, kecepatan aliran merupakan sebuah fungsi waktu. Kondisi tak tunak terjadi pada air yang bergerak dipercepat seperti air terjun.
2. Aliran fluida dapat merupakan aliran melingkar (*rotational*) atau tak melingkar (*irrotational*). Aliran tak melingkar terjadi jika elemen fluida di setiap titik mempunyai kecepatan sudut netto terhadap titik tersebut.
3. Aliran fluida dapat merupakan aliran termampatkan (*compressible*) atau tak termampatkan (*incompressible*). Aliran fluida dikatakan termampatkan jika tekanannya dapat berubah sesuai dengan keadaan. Contoh dari fluida termampatkan adalah gas. Sedangkan zat cair dalam kondisi normal digolongkan sebagai aliran fluida tak termampatkan karena tekanannya akan selalu tetap pada setiap kondisi.
4. Aliran fluida dapat digolongkan sebagai aliran kental (*viscous*) atau aliran encer (*nonviscous/inviscid*). Efek viskositas terjadi karena gaya-gaya tangensial yang terjadi diantara lapisan-lapisan fluida.

2.1.3 Persamaan *Navier-Stokes* sebagai persamaan umum dinamika fluida

Persamaan *Navier-Stokes* merupakan persamaan gerak fluida yang memperhatikan gaya-gaya yang bekerja pada suatu elemen kecil fluida, seperti tegangan geser yang timbul akibat gerakan dan viskositas fluida. Secara umum persamaan *Navier-Stokes* dituliskan sebagai berikut:

$$\text{—} = \frac{1}{\text{—}}(- \quad + \quad + \quad) \qquad (2.1)$$

dimana $\boldsymbol{v}$, ρ, p, $\boldsymbol{g}$, dan secara berturut-turut adalah kecepatan, kerapatan, tekanan, percepatan gravitasi, dan koefisien viskositas fluida.

Persamaan 2.1 masih bersifat kontinyu, sehingga untuk proses komputasi perlu dilakukan diskritisasi. Metoda diskritisasi dilakukan dengan memisahkan tiap suku persamaan di ruas kanan dan masing-masing dimasukkan ke dalam algoritma SPH (Müller *et al*., 2003).

2.2 Sistem Partikel

Menurut Trina M. Roy (dalam Hamdi, 2008) partikel merupakan entitas yang tidak mempunyai parameter ruang, tetapi merupakan obyek yang mempunyai parameter posisi, kecepatan, gaya interaksi antar sesamanya, dan dapat dipengaruhi oleh gaya luar. Sistem partikel merupakan kumpulan massa titik pada ruang tiga dimensi yang dihubungkan dengan gaya tertentu dan dipengaruhi oleh gaya eksternal. Gaya eksternal yang bisa mempengaruhi sistem di antaranya adalah gaya gravitasi dan gaya gesekan yang terjadi antar partikel. Suatu partikel akan mempunyai percepatan apabila diberikan gaya, seperti yang berlaku pada hukum Newton kedua:

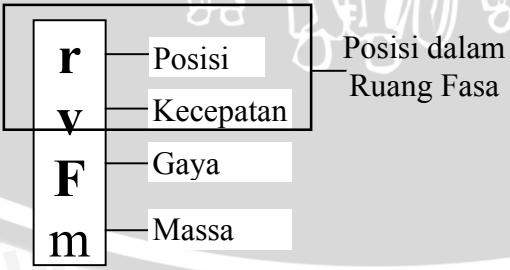
$$= \frac{}{} = \frac{}{} \qquad (2.2)$$

dimana ***F, r,*** *m,* dan ***a*** secara berturut-turut adalah gaya, posisi, massa, dan percepatan partikel. Dari persamaan di atas gaya gravitasi yang dialami partikel adalah:

$$\boldsymbol{F} = m\,\boldsymbol{g} \qquad (2.3)$$

dengan ***g*** merupakan percepatan gravitasi.

Sifat fisis suatu partikel dapat dinyatakan seperti pada Gambar 2.3.



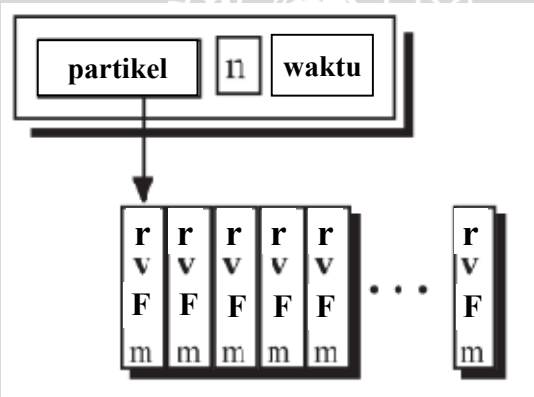
Gambar 2.3 Parameter yang dimiliki partikel

Pada gambar 2.3 partikel mempunyai besaran posisi dan kecepatan dalam ruang fasa. Sedangkan parameter lain seperti gaya dan massa tidak dapat digambarkan di dalam ruang fasa.

2.2.1 Ruang fasa

Florian Reiterer (dalam Hamdi, 2008) menyatakan bahwa ruang fasa adalah ruang untuk mempresentasikan keadaan yang mungkin bagi partikel. Setiap keadaan berkorespondensi dengan satu titik unik dalam fasa ruang. Vektor posisi $\boldsymbol{r} = [\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}]$ dan vektor kecepatan $\boldsymbol{v} = [\boldsymbol{v_1}, \boldsymbol{v_2}, \boldsymbol{v_3}]$ digabungkan ke dalam bentuk 6 vektor dalam ruang enam dimensi. Maka persamaan ruang fasa untuk gerak partikel adalah:

$$[\dot{}, \dot{}, \dot{}, \dot{}, \dot{}, \dot{}] = \quad , \quad , \quad , —, —, — \qquad (2.4)$$



Gambar 2.4 Ruang fasa pada partikel

Gambar 2.4 menyatakan ruang fasa suatu partikel, di mana setiap partikel mempunyai parameter posisi, kecepatan, gaya, dan massa.

2.2.2 Gaya

Menurut Florian Reiterer (dalam Hamdi, 2008) bahwa gaya yang bekerja pada partikel dibagi menjadi tiga macam:

1. Gaya tunggal merupakan gaya yang dikenakan pada setiap partikel yang ada di dalam sistem. Gaya ini dapat berupa gaya konstan maupun gaya yang berubah tergantung pada parameter yang dimiliki partikel seperti posisi dan massa. Gaya ini juga

berubah terhadap waktu seperti gaya yang disebabkan oleh angin.
2. Gaya antar partikel merupakan gaya yang terjadi dan saling mempengaruhi antar partikel seperti gaya pegas dan gaya gravitasi antar partikel.
3. Gaya interaksi ruang merupakan gaya yang bekerja terhadap sebagian atau seluruh partikel yang tergantung posisi seperti gaya tarik dan gaya dorong.

2.2.3 Kopling partikel

Berdasarkan pernyataan Florian Reiterer (dalam Hamdi 2008), gaya yang berkerja pada partikel dan tidak saling mempengaruhi merupakan gaya tunggal. Gaya tunggal telah diimplementasikan pada berbagai keadaan di antaranya efek asap, ledakan, hujan, aliran darah, dan lain sebagianya. Tidak adanya koneksi antar partikel membuat model partikel yang tidak tergandeng tidak cocok digunakan untuk fluida, karena gaya internal sangat penting.

Untuk mensimulasikan interaksi antar partikel telah dikembangkan beberapa model diantaranya adalah potensial Lennard-Jones yang menggambarkan gaya antar atom atau molekul netral (tidak bermuatan). Dua gaya dikenakan dalam model ini, dalam jangkauan jauh dikenakan gaya tarik menarik *Van Der Walls* dan dalam jarak dekat dikenakan gaya tolak-menolak Pauli (hasil dari *overlapping* orbital elektron). Saat di mana resultan sama dengan nol untuk jarak tertentu disebut sebagai titik keseimbangan. Potensial Lennard-Jones dapat dinyatakan dengan persamaan berikut:

$$( ) = 4 \quad - \quad - \quad - \qquad (2.5)$$

dengan : r = jarak antar partikel
σ = jarak di mana gaya antar partikel bernilai nol, dan
ε = kedalaman sumur potensial.

Persamaan Lennard-Jones di atas hanya melibatkan gaya tarik dan gaya tolak, sedangkan untuk memodelkan interaksi partikel lain seperti kerapatan, viskositas, dan lain sebagainya digunakan metoda *Smoothed Particle Hydrodynamics*.

2.3 Metoda *Smoothed Particle Hydrodynamics*

Metoda *Smoothed Particle Hydrodynamics* (SPH) dikembangkan oleh Lucy dan Gingold serta oleh Monaghan untuk mensimulasikan permasalahan-permasalahan astrofisika. Meskipun demikian, metoda SPH dapat juga digunakan untuk mensimulasikan fluida. Yang menjadi daya tarik metoda SPH adalah dihilangkannya komputasi grid ketika menghitung turunan ruang. Metoda SPH telah digunakan secara luas untuk memodelkan dinamika fluida khususnya aliran permukaan bebas. Di antara aplikasinya adalah untuk pemodelan air laut pada pantai datar oleh Monaghan dan Kos pada tahun 1999, benda padat yang jatuh pada permukaan oleh Monaghan pada tahun 2000, dan simulasi dam jebol oleh Monaghan pada tahun 1994 (Roger *et al*., 2003).

2.3.1 Pengantar SPH

SPH merupakan suatu metoda interpolasi untuk sistem partikel. Dengan SPH, kuantitas partikel hanya didefinisikan pada posisi partikel yang dapat dievaluasi pada sebarang ruang (Müller *et al*., 2003).

Menurut G. R. Liu dan M. B. Liu (2003), beberapa alasan digunakannya SPH dalam simulasi fluida adalah sebagai berikut:

1. Metoda SPH dapat digunakan untuk simulasi aliran permukaan bebas.
2. Dengan berbagai pengembangan, metoda SPH mempunyai akurasi, stabilitas, dan adaptivitas yang baik.
3. Aplikasi dari SPH sangat luas, mulai dari skala kecil, skala besar, sampai skala astronomi, dan dari sistem diskrit sampai sistem kontinyu.
4. Aplikasi SPH mencakup berbagai aspek fisis fluida seperti viskositas, gaya eksternal, gaya internal, kerapatan, dan lain sebagainya.

Hamdi (2008) menuliskan dalam tesisnya, ide-ide dasar dari metoda SPH diantaranya sebagai berikut:

1. Partikel disusun secara teratur tanpa adanya konektivitas antar partikel (*meshfree*).
2. Penggunaan metoda representasi integral untuk aproksimasi bidang pada metoda SPH dinamakan dengan aproksimasi kernel.

3. Aproksimasi kernel dilakukan dengan menggunakan partikel yang dinamakan dengan aproksimasi partikel di seluruh domain lokal.
4. Aproksimasi partikel dilakukan setiap jangka waktu tertentu, sehingga penggunaan partikel tergantung kepada distribusi partikel di saat itu.

Konsep representasi integral dari fungsi $f(\boldsymbol{r})$ yang digunakan dalam SPH dimulai dari persamaan berikut:

$$f(\boldsymbol{r}) = \int_{\Omega} f(\boldsymbol{r}')\,\delta(\boldsymbol{r}-\boldsymbol{r}')\,d\boldsymbol{r}', \qquad (2.6)$$

di mana f merupakan fungsi dari vektor posisi $\boldsymbol{r}$, dan $\delta(\boldsymbol{r}-\boldsymbol{r}')$ adalah fungsi delta Dirac yang mempunyai definisi:

$$\delta(\boldsymbol{r}-\boldsymbol{r}') = \begin{cases} 1 & \boldsymbol{r}=\boldsymbol{r}' \\ 0 & \boldsymbol{r}\neq\boldsymbol{r}' \end{cases}, \qquad (2.7)$$

Pada persaman 2.6, Ω merupakan batas volume dari daerah yang diliputi oleh $\boldsymbol{r}$. Karena menggunakan fungsi delta Dirac, persamaan SPH di atas bernilai eksak, selama $f(\boldsymbol{r})$ terdefinisi dan kontinyu dalam Ω.

2.3.2 Fungsi kernel

Jika fungsi delta Dirac digantikan dengan fungsi kernel $W(\boldsymbol{r}-\boldsymbol{r}', h)$ dengan h adalah radius kernel, persamaan SPH akan menjadi:

$$f(\boldsymbol{r}) = \int_{\Omega} f(\boldsymbol{r}')\,W(\boldsymbol{r}-\boldsymbol{r}', h)\,d\boldsymbol{r}'. \qquad (2.8)$$

Dalam SPH, operator aproksimasi kernel dituliskan dengan kurung lancip < >, sehingga persamaan SPH dapat dituliskan:

$$\langle f(\boldsymbol{r})\rangle = \int_{\Omega} f(\boldsymbol{r}')\,W(\boldsymbol{r}-\boldsymbol{r}', h)\,d\boldsymbol{r}'. \qquad (2.9)$$

Fungsi kernel yang digunakan biasanya merupakan fungsi genap, di mana fungsi kernel harus memenuhi beberapa kondisi. Kondisi pertama normalisasi dari fungsi kernel adalah:

$$\int_{\Omega} (\quad - \quad , h) \quad = 1. \qquad (2.10)$$

Kondisi kedua merupakan sifat fungsi delta yang terjadi jika fungsi kernel mendekati nol yaitu:

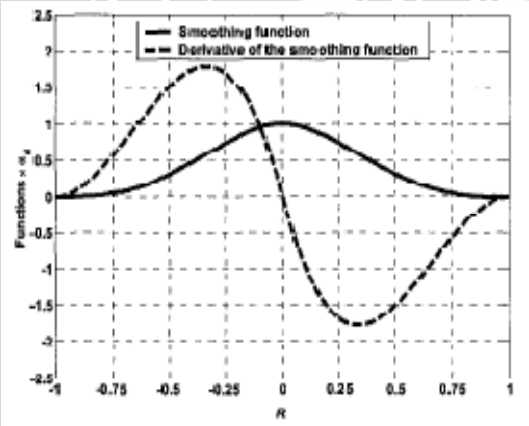
$$\lim_{\to} \quad (\quad - \quad , h) = \quad (\quad - \quad). \qquad (2.11)$$

Kondisi ketiga merupakan kondisi padat:

$$(\quad - \quad , h) = 0 \; \forall \; | \quad - \quad | > \quad h, \qquad (2.12)$$

dimana merupakan sebuah konstanta yang berhubungan dengan fungsi kernel pada titik ***r*** dan memberikan definisi area efektif (tidak nol) dari fungsi kernel. Area efektif ini disebut dengan *support domain* untuk fungsi kernel pada titik ***r*** (Liu dan Liu, 2003).

Pada tahun 1977 Lucy menggunakan fungsi kernel yang berbentuk bel, seperti ditunjukkan dalam Gambar 2.5



Gambar 2.5 Kernel *Quartic* (Liu, 2003)

Adapun persamaan matematis kernel *Quartic* adalah (Liu, 2003):

$$= \quad 1 + 3 \quad 1 \quad . \qquad (2.13)$$

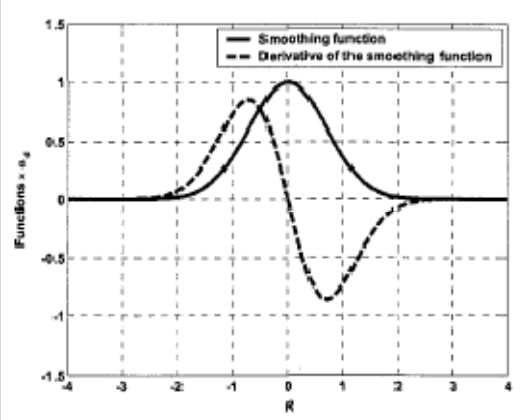
Gradien kernel ini adalah:

$$= \quad - 1 \quad , \qquad (2.14)$$

sedang *Laplacian* kernel *Quartic* adalah:

$$= \quad - 1 \quad . \qquad (2.15)$$

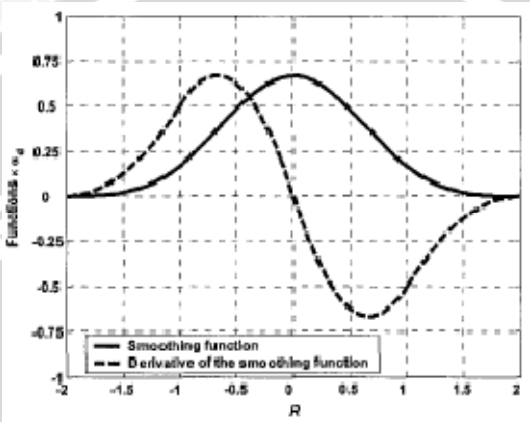
Monaghan pada tahun 1992 menyatakan (dalam Liu, 2003) bahwa untuk mendapatkan interpretasi fisis dari persamaan SPH, yang paling baik adalah mengasumsikan fungsi kernel sebagai Gaussian. Gingold dan Monaghan pada tahun 1977 memilih kernel *Gaussian* dalam Gambar 2.6 untuk mensimulasikan bintang yang tidak bulat.



Gambar 2.6 Kernel *Gaussian* (Liu, 2003)

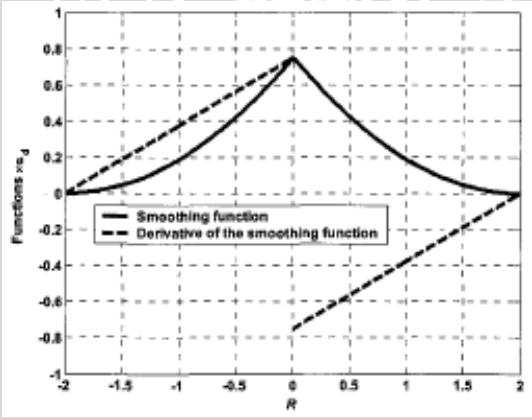
Monaghan dan Lattanzio pada tahun 1985 menemukan fungsi kernel yang didasarkan pada fungsi *Cubic-spline* yang dinamakan fungsi *B-spline* seperti pada Gambar 2.7. Fungsi *Cubic-spline* merupakan fungsi yang sering digunakan sebagai fungsi kernel

karena mempunyai bentuk yang lebih dangkal. Namun demikian turunan kedua dari *Cubic-spline* merupakan fungsi linear yang dapat mengurangi stabilitas dari kernel.



Gambar 2.7 Kernel *Cubic-spline* (Liu, 2003)

Pada tahun 1996 Johnson menggunakan fungsi *Quadratic* sebagai fungsi kernel yang didesain khusus untuk mensimulasikan permasalahan partikel dengan kecepatan yang tinggi.

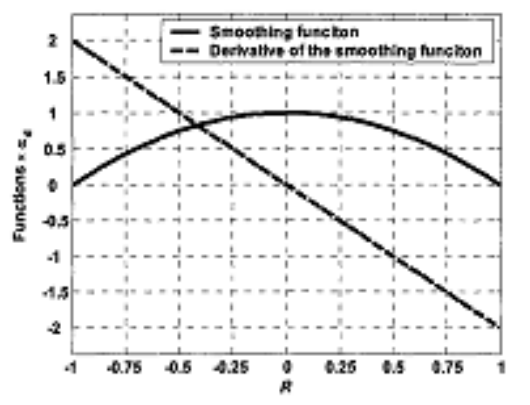


Gambar 2.8 Kernel *Quadratic* hasil karya Johnson (Liu, 2003)

Gambar 2.8 menunjukkan fungsi kernel *Quadratic* hasil karya Johnson, tidak seperti fungsi kernel yang lain, turunan dari fungsi *Quadratic* ini semakin meningkat jika jarak antar partikel semakin dekat dan semakin menurun seiring dengan bertambahnya

jarak antar partikel. Johnson menganggap ini sebagai pengembangan penting untuk mengatasi ketidakstabilan penghitungan tekanan pada SPH. Fulk menyatakan (dalam Liu, 2003) bahwa salah satu kekurangan dari fungsi kernel orde tinggi adalah kernel bernilai negatif pada beberapa daerah dalam *support domain*. Ini dapat menimbulkan permasalahan karakteristik fisis yang tidak tepat pada hidrodinamik.

Hicks dan Liebrock pada tahun 2000 menemukan kernel *Dome-shaped* yang mereka klaim cocok digunakan untuk simulasi dinamika fluida yang dibatasi oleh mesh.



Gambar 2.9 Kernel *Dome-shaped* dan gradiennya (Liu, 2003)

Gambar 2.9 memperlihatkan keadaan kernel *Dome-shaped* dan gradiennya yang nilainya membesar tanpa batas berbanding lurus dengan meningkatnya jarak antar partikel. Bentuk fungsi kernel *Dome-shaped* dalam tiga dimensi adalah (Liu, 2003):

$$= \frac{\quad}{\quad} \; 1 \frac{| \; |}{\quad} \;, \qquad (2.16)$$

sedangkan *Gradien* dari kernel *Dome-shaped* adalah:

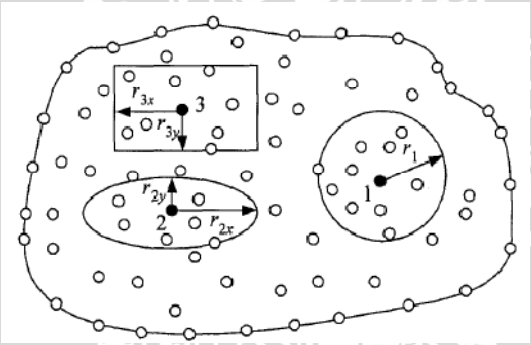
$$\nabla \quad = \frac{\quad}{\quad} \;, \qquad (2.17)$$

dan *Laplacian* dari fungsi kernel tersebut adalah:

$$\nabla \quad = \frac{\quad}{\quad}. \qquad (2.18)$$

2.3.3. *Support domain*

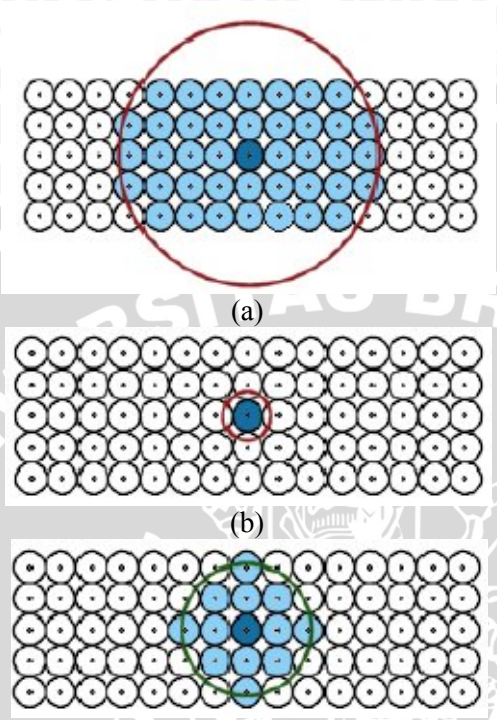
Support domain untuk titik $\boldsymbol{r} = (\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})$ didefinisikan sebagai daerah di mana informasi semua titik di dalam daerah tersebut digunakan untuk menentukan informasi pada titik $\boldsymbol{r}$. *Support domain* dalam metode SPH untuk sebuah titik bisa lokal sebagai sub region dari *problem domain*, atau global sebagai *problem domain*. Untuk mengurangi beban komputasi, biasanya dipilih lokal *support domain* yang mana hanya partikel di dalam daerah lokal dari dimensi terbatas digunakan untuk aproksimasi variabel-variabel pada titik tersebut. Dimensi dan bentuk *support domain* masing-masing titik dapat berbeda.



Gambar 2.10 Beberapa bentuk *support domain* yang sering dipergunakan (Liu, 2003)

Gambar 2.10 memperlihatkan bentuk *suppot domain* yang biasa dipergunakan adalah ellips, segi empat, dan lingkaran. *Support domain* dapat bersifat simetris atau tak simetris untuk titik di dekat *boundary*. Dalam metoda SPH, konsep *support domain* suatu partikel berhubungan erat dengan radius kernel *h* dari partikel tersebut (Liu dan Liu, 2003).

Penentuan variable *h* merupakan faktor yang sangat penting, di mana *h* jika terlalu besar atau terlalu kecil menyebabkan aproksimasi SPH menjadi tidak akurat.



Gambar 2.11. (a) radius kernel terlalu besar, (b) radius kernel terlalu kecil, (c) radius kernel yang baik.

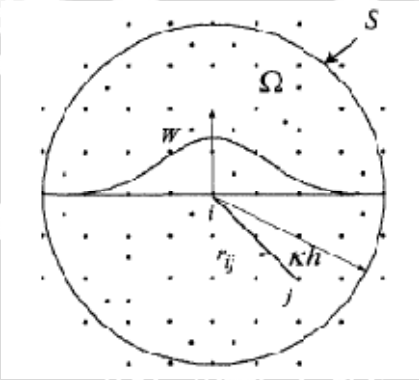
Gambar 2.11 menunjukkan besarnya radius kernel yang mempengaruhi akurasi aproksimasi partikel di mana gambar (a) menunjukkan pemilihan radius kernel yang terlalu besar, gambar (b) menunjukkan pemilihan radius kernel yang terlalu kecil, dan gambar (c) menunjukkan pemilihan radius kernel yang sesuai.

2.3.4 Aproksimasi partikel

Pada metoda SPH jumlah partikel yang digunakan untuk merepresentasikan fluida berjumlah terbatas (*finite*). Setiap partikel memiliki masa dan menempati ruang sendiri-sendiri.

Bentuk integral yang merupakan persamaan kontinyu dari SPH dapat diubah menjadi diskrit yang berbentuk penjumlahan semua partikel di dalam *support domain* seperti Gambar 2.12. Proses

penjumlahan diskrit dalam kasus SPH disebut sebagai aproksimasi partikel (*particle approximation*).



Gambar 2.12 Aproksimasi partikel dalam *support domain* yang berbentuk lingkaran yang mempunyai radius

Gambar 2.12 menunjukkan partikel yang mempunyai pengaruh kepada partikel pusat dalam suatu *support domain*.

Jika volume dalam radius ***dr'*** yang sangat kecil dari integrasi di atas pada lokasi partikel *j* digantikan dengan volume terbatas Δ dan dihubungkan dengan massa partikel maka:

$$= \Delta \quad , \qquad (2.19)$$

dengan merupakan kerapatan partikel *j* (*j* = 1, 2, 3, ….N), *N* adalah jumlah partikel di dalam *support domain*-nya partikel *j*.

Bentuk integral persamaan SPH dapat dituliskan dalam bentuk diskrit menjadi :

$$( ) = \quad (\quad - \quad , h) \frac{1}{} \quad , \qquad (2.20)$$

atau

$$( ) = \quad \frac{}{} \quad - \quad , h \ . \qquad (2.21)$$

Akhirnya persamaan aproksimasi partikel dapat dituliskan menjadi:

$$< A(\mathbf{r}) > = \int A(\mathbf{r}') W(\mathbf{r} - \mathbf{r}', h)\, d\mathbf{r}', \quad (2.22)$$

dimana W_j adalah:

$$W_j = W(\mathbf{r} - \mathbf{r}_j, h). \quad (2.23)$$

Jika fungsi $f(\mathbf{r})$ diganti dengan kerapatan ρ maka persamaannya menjadi:

$$\rho_i = \sum_j m_j W_{ij}. \quad (2.24)$$

Persamaan tersebut dikenal dengan *summation density approach* (Liu dan Liu, 2003).

2.3.5 Gaya akibat tekanan dan viskositas

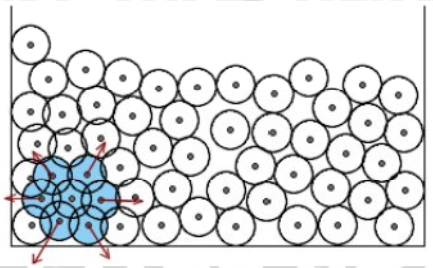
Parameter yang biasa dimiliki fluida di antaranya adalah tekanan dan viskositas. Dalam simulasi fluida berbasis partikel, tekanan yang dialami oleh sebuah partikel terjadi akibat akumulasi tekanan partikel tetangga. Besarnya pengaruh tekanan partikel tetangga bergantung pada jarak partikel tersebut terhadap partikel referensi. Partikel dengan jarak terdekat mempunyai pengaruh paling besar, sedangkan partikel yang tidak masuk dalam partikel tetangga tidak mempunyai pengaruh sama sekali (Hamdi, 2008).

Dengan mensubtitusikan bagian tekanan dari persamaan *Navier-Stokes* ke dalam persamaan SPH maka didapatkan persamaan gaya akibat tekanan:

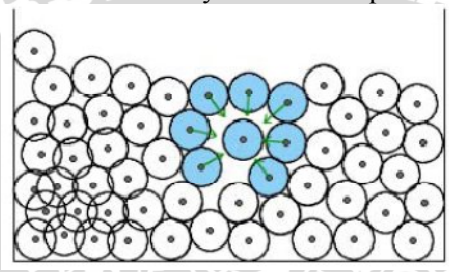
$$\mathbf{f}_i^{pressure} = -\sum_j m_j \frac{p_i + p_j}{2\rho_j} \nabla W(\mathbf{r}_i - \mathbf{r}_j, h), \quad (2.25)$$

dengan p merupakan tekanan dari partikel.

Gambar 2.13 dan 2.14 secara berturut-turut menunjukkan gaya tolak karena kerapatan partikel yang tinggi dan gaya tarik antar partikel yang mempunyai kerapatan rendah.



Gambar 2.13 Gaya tolak antar pertikel



Gambar 2.14 Gaya tarik antar pertikel

Dengan cara yang sama diperoleh persamaan gaya akibat viskositas:

$$= \quad — \quad - \quad (\quad - \quad , h), \qquad (2.26)$$

dimana $\boldsymbol{v}_i$ merupakan kecepatan partikel dan adalah koefisien viskositas fluida. Tekanan P_i dihitung dengan persamaan:

$$= \quad (\quad - \quad), \qquad (2.27)$$

dimana C merupakan konstanta kekakuan fluida dan adalah kerapatan fluida saat diam.

Percepatan fluida diberikan oleh persamaan yang merupakan gabungan dari persamaan gaya-gaya yang terjadi pada fluida:

$$= \frac{1}{} \quad + \quad + \quad , \qquad (2.28)$$

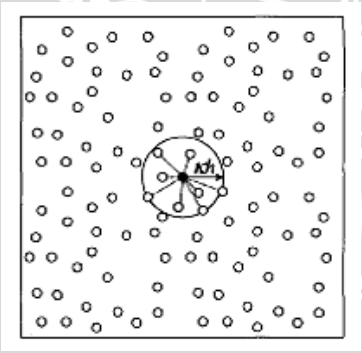
di mana gaya eksternal merupakan gaya dari luar fluida seperti gaya gravitasi dan volum fluida dianggap konstan (Müller *et. al*., 2005).

2.3.6 Pencarian partikel tetangga terdekat

Dalam metode SPH, pencarian partikel tetangga terdekat hanya difokuskan pada partikel yang berada dalam *support domain*. Tidak seperti pada *grid-base method* yang tetangga terdekatnya sudah dapat ditentukan berdasarkan grid yang ada, pencarian partikel tetangga terdekat pada SPH dapat berubah terhadap waktu. Di antara metoda yang biasa digunakan untuk pencarian partikel tetangga terdekat pada SPH adalah *all-pair search*, algoritma *linked-list*, dan algoritma *tree-search* (Liu dan Liu, 2003).

2.3.6.1 *All-pair search*

Cara yang paling mudah untuk pencarian partikel tetangga terdekat adalah dengan pendekatan *all-pair search*. Pada gambar 2.15 dapat dilihat bahwa untuk sebuah partikel *i*, pendekatan *all-pair* menghitung jarak r_{ij} dari *i* ke tiap-tiap partikel *j* (*j* = 1, 2, 3, …, *N*), dimana *N* merupakan jumlah total partikel dalam *support domain*. Jika digunakan radius kernel yang simetris, maka partikel *i* juga akan termasuk di dalam *support domain* partikel *j*.

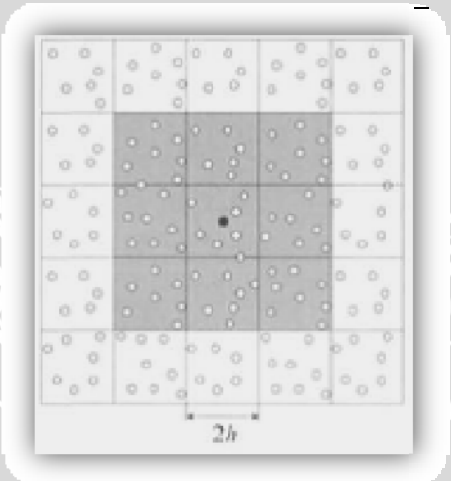


Gambar 2.15 Algoritma *all-pair search* (Liu, 2003)

Kelemahan dari metoda *all-pair search* ini adalah proses komputasi yang lama, karena harus dilakukan perhitungan setiap partikel beserta pengaruhnya dengan partikel tetangga (Liu dan Liu, 2003).

2.3.6.2 Algoritma *Linked-list*

Algoritma *linked-list* akan bekerja dengan baik untuk kasus yang mempunyai radius kernel konstan. Metoda ini menggunakan *cell* untuk menyimpan partikel menjadi beberapa kelompok. Implementasi dari algoritma *linked-list* adalah pembuatan sebuah *mesh* sementara untuk melingkupi *domain*. Jarak *mesh* dipilih berdasarkan jarak *support domain* sebesar *h* seperti dapat dilihat pada gambar 2.16. Untuk sebuah partikel *i*, partikel tetangga terdekatnya dapat terletak di *grid cell* yang sama atau *cell* yang berbatasan langsung.

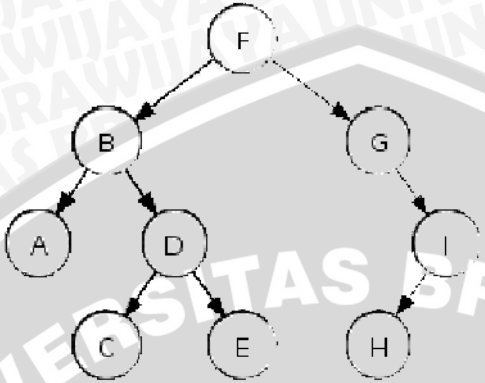


Gambar 2.16 Algoritma *linked-list* dengan jarak *cell* 2*h* (Liu, 2003)

Masalah yang ada pada metoda ini adalah jika radius kernel yang digunakan bervariasi, jarak *mesh* tidak bisa optimal untuk setiap partikel (Liu dan Liu, 2003)

2.3.6.3 Algoritma *tree-search*

Algoritma *tree-search* bekerja dengan baik untuk kasus yang mempunyai *smoothing length* bervariasi. Algoritma ini melibatkan pembuatan diagram pohon yang disusun berdasarkan posisi partikel. Setelah diagram pohon terbentuk, dapat digunakan untuk menemukan partikel tetangga terdekat secara efisien (Liu dan Liu, 2003).



Gambar 2.17 Algoritma *tree-search*

Pada gambar 2.17 menunjukkan bahwa F sebagai akar, B dan G sebagai batang, dan A, D, C, E, H, dan I sebagai cabang. Pencarian dilakukan berdasarkan tingkatan hirarki, yaitu dimulai dari akar, batang, dan kemudian cabang-cabang.

2.3.7 Integrasi waktu

Integrasi waktu dalam simulasi sistem fisis digunakan untuk menentukan keadaan sistem pada selang waktu (*time step*) berikutnya ($t + \Delta t$). Misalkan keadaan sistem pada waktu sekarang (t) diberikan posisi ($\boldsymbol{x}_t$), kecepatan ($\boldsymbol{v}_t$) dan percepatan sistem $\boldsymbol{a}(\boldsymbol{x},\boldsymbol{v},t)$, maka posisi dan kecepatan sistem untuk selang waktu selanjutnya adalah:

$$\boldsymbol{v}_{(t+\Delta t/2)} = \boldsymbol{v}_{(t-\Delta t/2)} + \boldsymbol{a}_t \Delta t \quad , \tag{2.29}$$

$$\boldsymbol{x}_{(t+\Delta t)} = \boldsymbol{x}_t + \boldsymbol{v}_{(t+\Delta t/2)} \Delta t \quad , \tag{2.30}$$

di mana Δt merupakan selang waktu yang digunakan dalam simulasi.

Persamaan (2.29) dan (2.30) merupakan salah satu persamaan integrasi waktu yang dikenal dengan *Leap-frog scheme*. Kelebihan dari *Leap-frog scheme* ini adalah persamaannya yang sederhana dan dapat digunakan untuk selang waktu yang kecil (Allard, 2009).

BAB III
METODE PENELITIAN

3.1 Waktu dan Tempat

Penelitian "Simulasi Dinamika Fluida Menggunakan Metoda *Smoothed Particle Hydrodynamics* (SPH)" ini dilaksanakan pada bulan Januari hingga Oktober 2009 di Jurusan Fisika Universitas Brawijaya Malang

3.2 Rancangan Penelitian

Penelitian dilakukan dengan membuat program berdasarkan metoda SPH yang telah digunakan secara luas (metoda yang diusulkan oleh Müller) dan digunakan untuk melakukan uji coba terhadap metoda-metoda lain yang diusulkan dalam penelitian ini.

3.3 Alat Penelitian

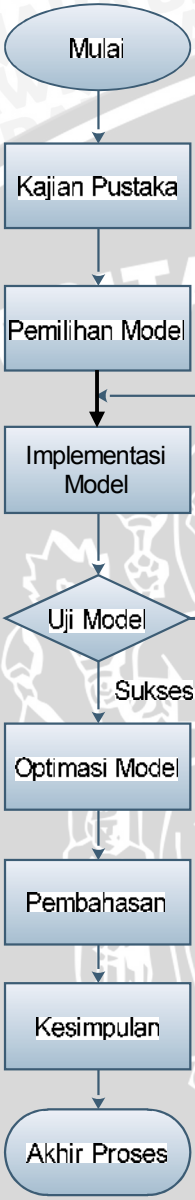
Alat-alat yang digunakan dalam penelitian ini adalah :

a. Perangkat keras (*hardware*)
   - Notebook ACER Aspire dengan CPU AMD Turion 64 bit, memori 1 GByte, dan kartu grafis NVIDIA GForce 9100.

b. Perangkat lunak (*software*)
   - Microsoft Windows XP Service Pack 2
   - Microsoft Visual C++ 2008 Express Edition
   - OpenGL
   - CoderTools Total Edit versión 5.08

3.4 Langkah Penelitian

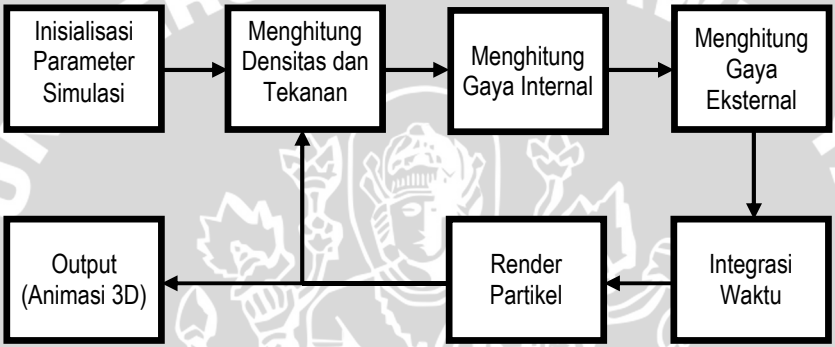
Langkah penelitian yang dilakukan dapat digambarkan dalam bentuk diagram alir pada gambar 3.1.



Gambar 3.1 Diagram alir proses penelitian

3.4.1 Implementasi model

Pada bagian ini hanya akan dijelaskan secara garis besar pembuatan model menggunakan persamaan matematis yang terdapat di dalam bab 2. Dalam proses simulasi digunakan metoda SPH. Model SPH yang digunakan merupakan model yang diajukan oleh Muller *et al*. dengan perubahan dalam penggunaan fungsi kernel. Adapun desain dari model SPH yang digunakan dalam penelitian ini ditunjukkan pada gambar 3.2 sebagai berikut:



Gambar 3.2: Diagram Alir Model Simulasi SPH

Berikut ini merupakan penjelasan secara sederhana dari gambar 3.2:

1. Inisialisasi parameter simulasi merupakan penentuan parameter dari partikel-partikel yang disimulasikan, di antaranya penentuan posisi, kecepatan, dan massa partikel.
2. Untuk menghitung densitas dan tekanan terlebih dahulu dilakukan pencarian partikel tetangga dalam *support domain*. Selanjutnya dihitung densitas dan tekanan dari masing-masing partikel beserta pengaruhnya dari pertikel tetangga.
3. Dilakukan perhitungan gaya internal yaitu gaya akibat tekanan dan gaya akibat viskositas.
4. Dilakukan perhitungan gaya eksternal yaitu gaya gravitasi.
5. Dilakukan integrasi waktu untuk menentukan posisi dan kecepatan partikel dalam time step selanjutnya.
6. Dilakukan *rendering* partikel sebagai suatu lingkaran dengan titik pusat r_i dan jari-jari R.
7. Dilakukan pembuatan animasi grafis dengan bantuan OpenGL dan berbagai *library* yang dapat ditemukan di internet.

3.4.2 Uji coba model

Uji coba model dilakukan secara kualitatif dengan cara membandingkan hasil visual yang didapatkan dari model dalam penelitian ini dengan hasil dari model lain yang sudah diakui validitasnya, dalam hal ini digunakan hasil dari penelitian Abdolmaleki *et al*. Sebagai model pembanding.

3.4.3 Variasi model

Setelah model yang dibuat dianggap memenuhi syarat, selanjutnya dilakukan variasi terhadap parameter-parameter yang dimasukkan untuk memperoleh hasil yang bervariasi. Parameter-parameter simulasi dapat ditentukan dengan dua cara yaitu secara teori dan praktis. Penentuan parameter secara teori menggunakan persamaan-persamaan dinamika fluida. Adapun parameter-parameter yang sulit ditentukan secara teori, maka dilakukan secara eksperimen. Parameter-parameter tersebut adalah massa partikel[1], radius dari radius kernel (*h*), selang waktu simulasi, dan jari-jari partikel. Partikel tersebut adalah partikel khayal yang merupakan gabungan dari partikel air.

3.4.3.1 Penentuan massa partikel

Massa tiap partikel ditentukan dengan menggunakan hubungan persamaan 3.1. sebagai persamaan dasar:

$$= —, \qquad (3.1)$$

di mana *V* merupakan volum dari fluida, *m* merupakan massa fluida, dan ρ merupakan kerapatan fluida.

Untuk mendapatkan massa partikel dilakukan dengan cara memodifikasi persamaan 3.1 menjadi persamaan 3.2 di mana massa fluida dalam suatu volume tertentu dianggap sebagai massa tiap partikel yang besarnya seragam, dikalikan dengan jumlah partikel dalam volume tersebut:

$$= ——, \qquad (3.2)$$

[1] Semua penggunaan istilah partikel dalam karya tulis ini merujuk kepada partikel khayal yang terdiri dari kumpulan partikel air.

di mana *n* merupakan jumlah partikel dalam volume tertentu. Jumlah partikel didapatkan dari pilihan pemrogram setelah melakukan eksperimen untuk menentukan jumlah yang terbaik dalam simulasi. Dengan mengacu pada persamaan 3.2, massa partikel dapat ditentukan dengan:

$$= \text{—} \tag{3.3}$$

Sebagai contoh jika dalam 1 m^3 fluida diperkirakan mempunyai 10.000 partikel dan fluida tersebut mempunyai kerapatan 1000 kg/m^3, maka massa dari tiap partikel adalah $\frac{\text{—} \times (\quad)}{\quad} = 0{,}1$.

3.4.3.2 Penentuan radius fungsi kernel

Support domain yang dipilih dalam simulasi ini adalah berbentuk lingkaran atau lebih tepatnya bola jika digunakan dalam sistem 3D, sehingga radius dari fungsi kernel dapat ditentukan dengan modifikasi persamaan volume bola.

$$= - \quad , \tag{3.4}$$

dimana V_{bola} merupakan volume dari *support domain* yang berbentuk bola dan *r* merupakan jari-jari dari bola. Jika *r* diganti dengan *h* (radius kernel) dan dengan rumus perbandingan sederhana volume *support domain* digantikan dengan parameter yang lebih umum menjadi volume fluida dikalikan dengan jumlah partikel di dalam *support domain* relatif terhadap jumlah partikel di dalam volume tersebut. Maka persamaan 3.4 menjadi persamaan 3.5 sebagai berikut

$$\text{—} = \frac{4}{3} \quad h,$$

$$h = \quad - \text{—} \quad , \tag{3.5}$$

di mana n_{sd} adalah jumlah partikel di dalam *support domain* yang didapatkan dari hasil eksperimen.

3.4.3.3 Penentuan selang waktu simulasi

Selang waktu untuk simulasi ditentukan benar-benar murni dari hasil eksperimen karena tidak ada acuan yang jelas untuk dapat dihubungkan dengan parameter dalam simulasi.

3.4.3.4 Penentuan jari-jari partikel

Penentuan jari-jari partikel berguna untuk proses rendering partikel. Ukuran jari-jari partikel yang tepat sangat penting untuk menghasilkan visualisasi yang baik Ukuran jari-jari partikel dalam simulasi ditentukan berdasarkan nilai kerapatan partikel untuk mencegah adanya tumpang tindih atau kekosongan. Tumpang tindih dan kekosongan terjadi jika partikel terlalu rapat atau partikel mempunyai kerapatan rendah sedangkan jari-jari partikel tetap. Partikel yang digunakan berbentuk bola sehingga jari-jari partikel diturunkan dari persamaan volume bola.

$$= \frac{4}{3}$$

$$— = \frac{4}{3}$$

$$= \quad \overline{\quad -\!— \quad}, \qquad (3.6)$$

di mana $r_{partikel}$ merupakan jari-jari partikel dan $V_{partikel}$ merupakan volume partikel yang berbentuk bola.

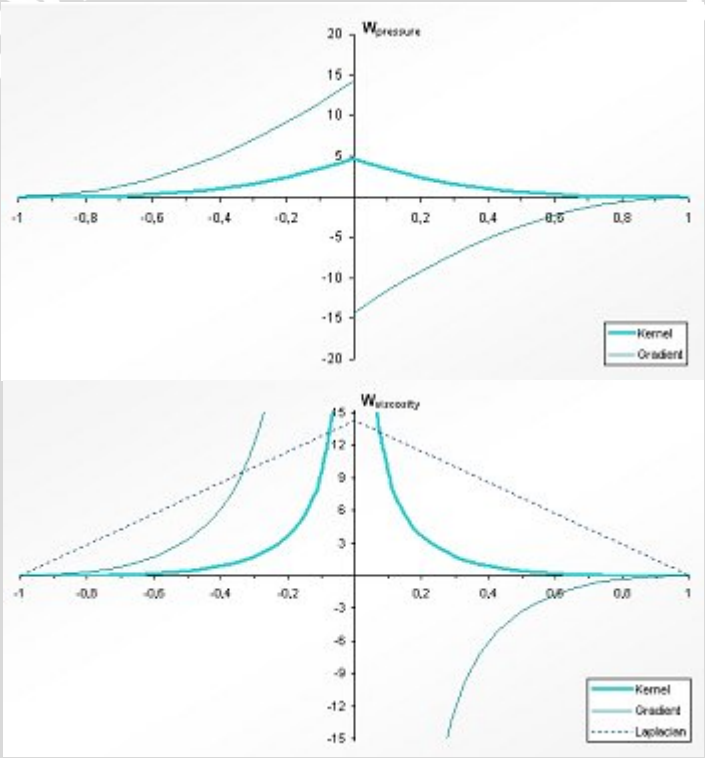
3.4.3.5 Variasi fungsi kernel

Dalam simulasi dinamika fluida menggunakan metoda SPH, fungsi kernel merupakan bagian penting simulasi untuk mendapatkan hasil yang akurat. Selain dari itu, fungsi kernel harus sederhana dan kompak agar beban komputasi komputer tidak terlalu berat sehingga simulasi dapat berjalan *real-time*. Dalam penelitian ini diujicobakan beberapa macam fungsi kernel dan dilihat pengaruhnya terhadap hasil komputasi baik berupa akurasi maupun kecepatan proses komputasi.

Fungsi kernel yang diujicobakan dalam penelitian ini adalah:

1. Kernel *Dhome-shaped*
2. Kernel *Gaussian*
3. Kernel *Quartic*

Faktor kesalahan utama penggunaan kernel terletak pada *Gradien* dan *Laplacian* kernel yang tidak dapat menggambarkan situasi fisis simulasi. Untuk mengatasi masalah ini para peneliti telah merancang beberapa jenis kernel khusus yang memenuhi logika simulasi fisis fluida. Untuk *Gradien* kernel digunakan Kernel *Spiky* dan untuk lapalacian digunakan kernel *Viscosity*.



Gambar 3.3 Kernel *Spiky* dan kernel *Viscosity* (Müller *et al*, 2003)

Gambar 3.3 memperlihatkan gambar karakteristik *Gradien* kernel *Spiky* dan kernel *Viscosity* dalam satu dimensi yang bernilai positif dalam *support domain*.

Persamaan *Gradien* kernel *Spiky* adalah

$$= - \frac{}{| |} (h - | |) \ , \qquad (3.7)$$

Laplacian kernel *Viscosity* mempunyai persamaan matematis

$$= (h - | |). \qquad (3.8)$$

BAB IV
HASIL DAN PEMBAHASAN

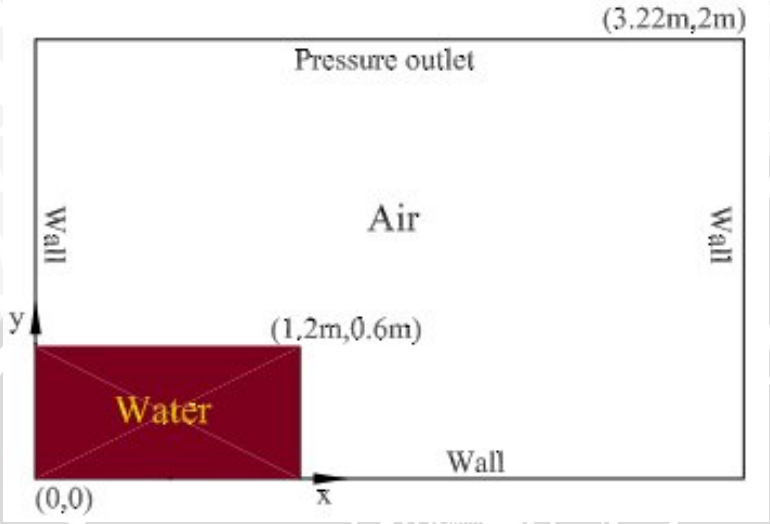
Dalam bab ini akan dijelaskan mengenai hasil tes perangkat lunak yang digunakan dalam penelitian ini dan pembahasan mengenai pemilihan parameter dan penggunaan fungsi kernel untuk meningkatkan akurasi dan kecepatan simulasi sehingga hasil dari simulasi dapat digunakan sebagai acuan untuk pengembangan perangkat lunak simulasi dinamika fluida secara interaktif. Hal yang perlu ditekankan adalah bahwa analisis hasil simulasi tidak disajikan dalam bentuk data dan grafik, namun diberikan dalam bentuk gambar *snapshoot* dari hasil simulasi. Sehingga dapat dikatakan metoda analisis yang digunakan adalah analisis kualitatif.

4.1 Uji coba perangkat lunak

Hal yang harus dilakukan dalam simulasi setelah perangkat lunak selesai dibuat adalah pengujian terhadap perangkat lunak tersebut. Dalam penelitian ini pengujian perangkat lunak dilakukan dengan membandingkan hasil yang diperoleh dari perangkat lunak yang dibuat dengan hasil dari perangkat lunak yang telah diakui validitasnya.

Dalam hal ini Abdolmaleki *et. al*. dari School of Oil and Gas Engineering The University of Western Australia, telah melakukan simulasi untuk mengetahui pola aliran air setelah menumbuk dinding vertikal dari kasus dam jebol, dengan menggunakan perangkat lunak komersial Fluent dan metoda *Navier-Stokes* (NS) solver. Metoda NS *Solver* menggunakan *Eulerian Finite Volume Method* (FVM) bersama dengan *Volume of Fluid* (VOF) *scheme*. Metode VOF dianggap cocok untuk kasus *free-surface*. (Abdolmaleki *et al*., 2004)

Gambar 4.1 menunjukkan set percobaan yang dilakukan oleh Abdolmaleki *et al*. dalam dua dimensi.



Gambar 4.1 Set percobaan dam jebol oleh Abdolmaleki *et.al*.

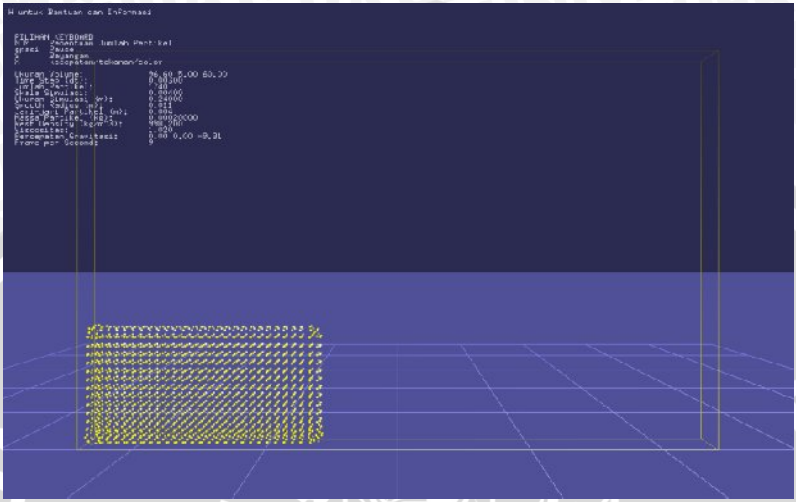
Dari gambar 4.1 dapat dilihat bahwa kotak domain simulasi berukuran tinggi 2 meter dan lebar 3,22 meter dengan dinding di bagian bawah dan kedua sisi tegaknya. Keadaan awal air adalah berbentuk persegi panjang dengan tinggi 0,6 meter dan lebar 1,2 meter (Abdolmaleki *et. al*., 2004).

Adapun set percobaan yang dilakukan dalam penelitian ini untuk kasus yang sama menggunakan metoda SPH adalah seperti yang dituliskan dalam tabel 4.1 berikut,

Tabel 4.1 Set percobaan

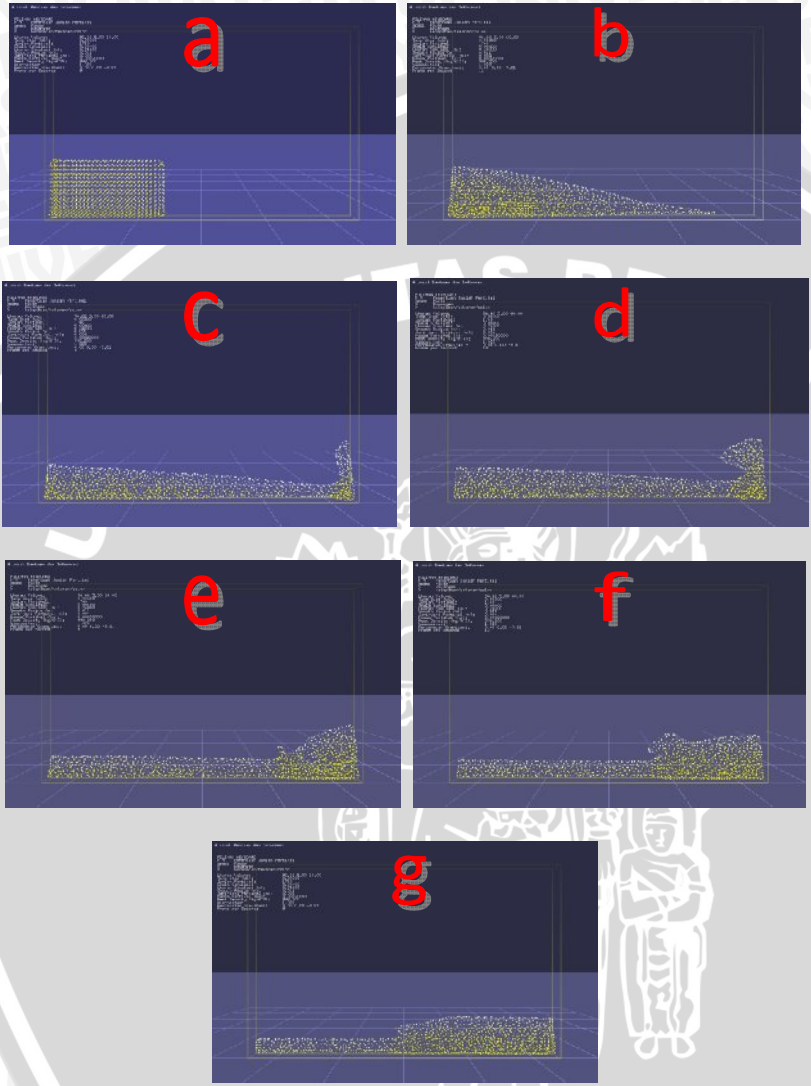
No	Parameter	Nilai
1	Dimensi Kotak	$3 \times (3{,}22 \times 0{,}2 \times 2)$ satuan
2	Jumlah partikel per 0.001 m^3	5000 partikel
3	Massa partikel	0.0002 kg
4	Radius partikel	0.0036 m
5	Jumlah partikel dalam kernel	30 partikel
6	Radius kernel	0.0113 m
7	Konstanta kekakuan	1,5

Berdasarkan parameter dari tabel 4.1, set percobaan berbentuk tiga dimensi dapat dilihat pada gambar 4.2 berikut:



Gambar 4.2 Set percobaan 3D dam jebol

Adapun hasil uji coba perangkat lunak yang dilakukan dalam penelitian ini dapat dilihat pada gambar 4.3 yang merupakan gambar tiga dimensi. Dilakukan beberapa penyesuaian terhadap gambar hasil penelitian yaitu, untuk memudahkan proses pembandingan, sudut pandang gambar dilihat dari bagian samping sehingga terlihat seperti simulasi dua dimensi. Demikian juga lebar balok domain simulasi dibuat sekecil mungkin dengan syarat masih melebihi batas radius kernel, karena pengaruh interaksi partikel dalam arah sumbu-y sangat mempengaruhi akurasi hasil simulasi. Jika dimensi dalam arah sumbu-y terlalu kecil, akan terjadi kekosongan partikel dalam radius kernel sebagaimana yang ditunjukkan dalam gambar 2.11 a. Gambar 2.43 menunjukkan *snapshot* aliran dalam waktu yang tidak berdimensi, $\tau = t(g/H)^{1/2}$ dimana t adalah waktu, g adalah percepatan gravitasi, dan H merupakan ketinggian awal air.

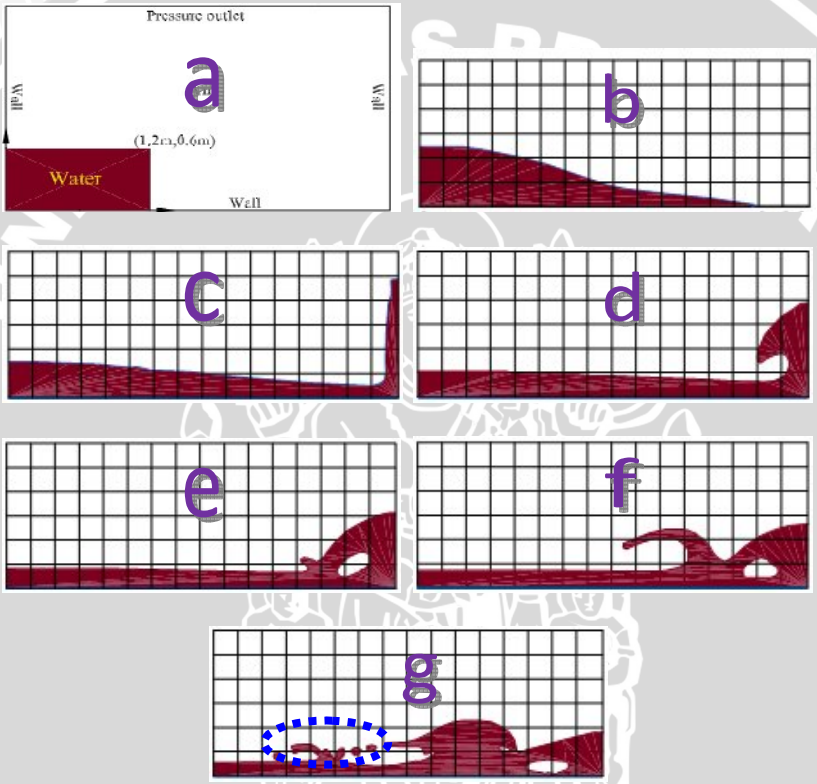


Gambar 4.3 Hasil simulasi dam jebol dengan metoda SPH, a: $\tau = 0$; b: $\tau = 2{,}02$; c: $\tau = 4{,}04$; d: $\tau = 5{,}46$; e: $\tau = 6{,}06$; f: $\tau = 7{,}08$; g: $\tau = 8{,}69$.

Hasil simulasi pada gambar 4.3 menunjukkan bahwa partikel yang mana pada kondisi awal tersusun secara teratur mulai bergerak menuju ruang kosong akibat adanya gaya gravitasi. Sampai pada

akhirnya aliran partikel menumbuk dinding, bergerak ke atas sampai ketinggian tertentu yang diakibatkan dorongan partikel dari bawah dari kemudian bergerak kembali dan menimbulkan gelombang.

Pada gambar 4.4 merupakan hasil simulasi yang dilakukan oleh Abdolmaleki *et. al.* dengan menggunakan metoda NS *Solver* sebagi pembanding hasil simulasi dengan metoda SPH.



Gambar 4.4 Hasil simulasi dam jebol dengan NS *Solver* dengan waktu yang sama seperti pada gambar 4.3 (Abdolmaleki, *et. al*., 2004)

Dari perbandingan gambar 4.3 dan gambar 4.4, pada gambar c terlihat bahwa kenaikan air pada percobaan Abdolmaleki *et. al*. lebih tinggi daripada hasil yang dilakukan penulis. Pada gambar f tidak terlihat adanya lubang dari hasil percobaan yang dilakukan penulis. Dan Pada gambar g pada hasil simulasi dengan metoda SPH tidak didapatkan riak seperti yang terdapat pada hasil simulasi

menggunakan NS *Solver* (ditandai dengan lingkaran biru putus-putus). Hal ini disebabkan semua partikel di bagian sebelah kiri tersusun secara teratur dengan tekanan yang merata sehingga tidak menghasilkan riak.

Dari hasil uji perbandingan kualitatif tersebut dapat dikatakan bahwa hasil simulasi dengan perangkat lunak yang dibuat berdasarkan metoda SPH dapat digunakan sebagai simulator namun mempunyai batasan ketelitian sampai tingkat terbentuknya dua gelombang pantul atau sampai level f dari hasil perbandingan dengan NS *Solver*. Karena yang dilakukan adalah uji kualitatif maka tidak menutup kemungkinan adanya perbedaan penafsiran di antara pembaca.

Oleh karena itu, jangkauan penelitian ini dibatasi pada keadaan yang telah dicapai yaitu sampai level f.

4.2 Penggunaan Variasi Parameter Partikel

Pada bagian ini difokuskan pada pemilihan parameter partikel dengan ketentuan sebagai berikut:

a. Partikel yang digunakan adalah partikel khayal, yang merupakan kumpulan dari beberapa partikel air.
b. Pemilihan parameter partikel yang menjadikan kecepatan simulasi berada dalam keadaan optimum.
c. Sifat-sifat fluida tidak jauh berbeda dengan hasil uji fluida.
d. Jumlah partikel uji adalah 2000 partikel.

Hasil simulasi untuk air (ρ = 998,2 Kg/m^3) dengan jumlah partikel 1000 partikel setiap 0,001 m^3 sampai dengan 5000 partikel setiap 0,001 m^3 dan 10 sampai 50 jumlah partikel setiap radius kernel dengan asumsi jumlah partikel air adalah tetap disajikan dalam lampiran 1. Penambahan jumlah partikel memberikan konsekuensi berkurangnya jumlah molekul air di dalam setiap partikel khayal.

Dari hasil uji coba didapatkan hasil terbaik pada partikel dengan jumlah 3000 sampai dengan 5000 partikel tiap 0,001 m^3. Dari hasil uji juga dapat dilihat bahwa secara umum semakin banyak jumlah partikel dalam radius kernel menyebabkan turunnya kecepatan simulasi. Hal ini disebabkan karena dengan semakin banyaknya jumlah partikel di dalam radius kernel akan menyebabkan semakin luas radius kernel sehingga daerah yang harus dihitung menjadi lebih banyak, dan beban komputasi bertambah. Untuk

kecepatan optimum dan hasil optimum sesuai dengan percobaan telah dilakukan adalah dengan jumlah partikel 4000 tiap 0,001 m^3 dengan asumsi 30 partikel berada dalam setiap radius kernel. Atau 5000 partikel tiap 0,001 m^3 dengan asumsi 40 partikel berada di dalam setiap radius kernel.

Pemilihan jumlah partikel adalah hak pengguna perangkat lunak. Jika pengguna menginginkan volume simulasi yang besar dan komputasi yang ringan, maka hendaknya memilih jumlah partikel tiap m^3 yang lebih sedikit, namun jika pengguna menginginkan volum simulasi kecil dan ketelitian yang tinggi, maka hendaknya memilih jumlah partikel tiap m^3 yang banyak, karena jumlah partikel mempengaruhi jarak antar partikel. Jika simulasi pada volum kecil menggunakan jumlah partikel per m^3 yang sedikit, maka jumlah partikel yang dapat masuk ke dalam domain simulasi akan sedikit dan mengakibatkan berkurangnya akurasi simulasi.

Ada hal lain yang harus diatur untuk menjaga kestabilan simulasi, yaitu konstanta *C* (konstanta kekakuan). Berdasarkan persamaan 2.24 nilai konstanta *C* sangat mempengaruhi reaksi antar partikel. Jika nilai *C* kecil maka reaksi antar partikel juga kecil. Sebaliknya jika nilai *C* besar, perubahan kerapatan partikel yang sedikit lebih besar daripada kerapatan awal akan menyebabkan besarnya tekanan partikel dan akan timbul gaya tolak menolak yang besar. Dari hasil pengujian didapatkan nilai *C* ideal adalah 1,5.

Selang waktu integrasi, juga mempengaruhi hasil simulasi. Pada umumnya penggunaan selang waktu yang kecil akan meningkatkan akurasi simulasi berkaitan dengan proses integrasi. Akan tetapi, jika nilai selang waktu terlalu kecil maka simulasi akan terlihat sangat lambat dan tidak realistis. Dari hasil pengujian diperoleh selang waktu ideal sebesar 0.0025.

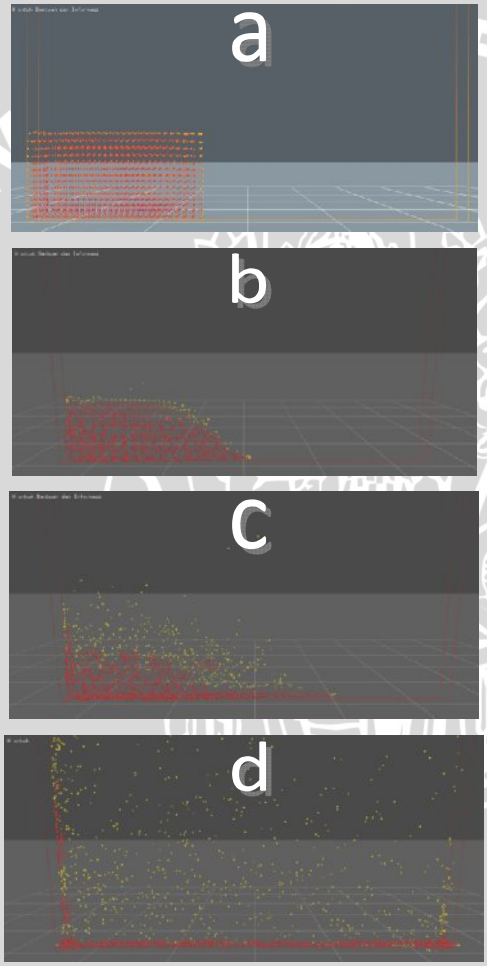
4.3 Penggunaan Variasi Fungsi Kernel

Fungsi kernel yang digunakan dalam uji coba perangkat lunak adalah fungsi kernel yang diusulkan oleh Muller *et al*. yaitu dengan menggunakan tiga macam fungsi kernel untuk menghitung kerapatan, tekanan, viskositas fluida. Dalam studi penggunaan fungsi kernel ini akan dijelaskan hasil dari uji coba masing-masing fungsi kernel yang digunakan. Adapun domain simulasi yang digunakan adalah sama dengan domain simulasi yang digunakan pada sub-bab 4.1.1 untuk menjaga konsistensi penelitian.

4.3.1 Fungsi kernel *Dome-shaped*

Smoothing kernel ini tidak memenuhi *compact support* pada turunan pertama. Pada gambar 2.9 dapat dilihat bahwa turunan fungsi kernel *Dome-shaped* selalu membesar tanpa batas.

Hasil dari simulasi menggunakan fungsi kernel *Dome-shaped* dapat dilihat pada gambar 4.5.



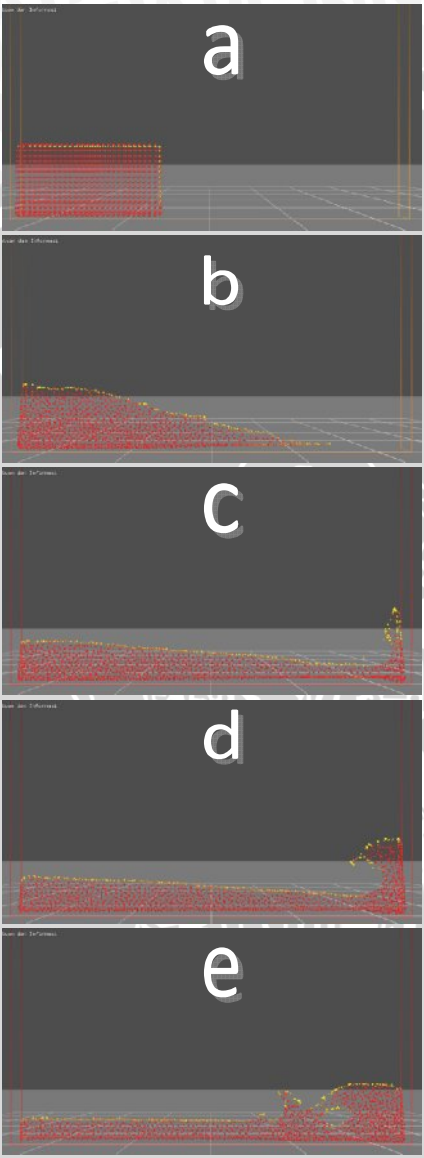
Gambar 4.5 Hasil simulasi menggunakan fungsi kernel *Dome-shaped* dengan waktu yang sama seperti gambar 4.3

Dapat dilihat bahwa pada gambar 4.5 bahwa pada hasil (a) dan (b) simulasi masih sesuai dengan yang diharapkan, namun pada keadaan (c) partikel mulai menyebar, dan pada keadaan (d) partikel menyebar tanpa batas. Sifat-sifat simulasi seperti ini sesuai dengan teori bahwa *Gradien* kernel *Dome-shaped* membesar tanpa batas sebanding dengan jarak partikel tetangga terhadap partikel pusat. Pada saat jarak partikel semakin jauh, maka gradien fungsi kernel juga semakin besar. Hal tersebut mengakibatkan semakin besarnya gaya tolak antar partikel dan partikel menjadi menyebar.

Keadaan tersebut dapat diatasi dengan mengganti *Gradien* fungsi kernel *Dome-shaped* dengan *Gradien* kernel yang lain. Pada penelitian ini, *Gradien* fungsi kernel *Dome-shaped* diganti dengan kernel *Spiky* dan untuk *Laplacian* fungsi kernel *Dome-shaped* diganti dengan kernel *Viscosity*.

Alasan dipertahankannya pengunaan fungsi kernel *Dome-shaped* adalah kesederhanaan bentuk persamaan yang jaraknya bernilai kuadrat sehingga tidak diperlukan perhitungan akar kuadrat. Dalam perhitungan komputasi dengan menggunakan komputer, pengunaan akar merupakan hal yang dapat menambah beban komputasi sehingga jika perhitungan akar dilakukan berulang-ulang untuk semua partikel, beban komputasi akan sangat tinggi. Namun demikian, pengaruh perhitungan akar tidak akan terlalu kelihatan pada penggunaan jumlah partikel yang sedikit.

Gambar 4.6 merupakan hasil visualisasi dengan menggunakan kernel *Dome-shaped* yang telah dimodifikasi.



Gambar 4.6 Hasil simulasi dam jebol dengan menggunakan fungsi kernel *Dome-shaped* termodifikasi dengan waktu yang sama seperti gambar 4.3

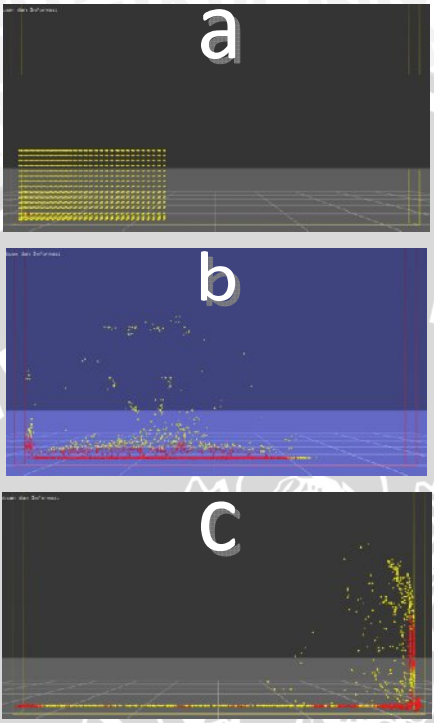
Dari gambar 4.6 diketahui bahwa dalam penelitian ini telah berhasil dibuat simulasi dam jebol dengan menggunakan fungsi kernel *Dome-shaped* termodifikasi. Dari gambar juga diketahui bahwa hasil simulasi tidak terlalu halus, dilihat pengaruh dari kernel yang digunakan, di mana fungsi kernel *Dome-shaped* tidak cepat menuju nol sehingga kestabilannya tidak maksimal.

4.3.2 Fungsi kernel *Gaussian*

Dalam anak sub-bab 2.4.2 Monaghan menyatakan bahwa *Gaussian* merupakan asumsi terbaik fungsi kernel untuk mendapatkan interpretasi fisis. Fungsi kernel ini telah digunakan oleh Monaghan dan Gingold untuk mensimulasikan bintang yang tidak bulat.

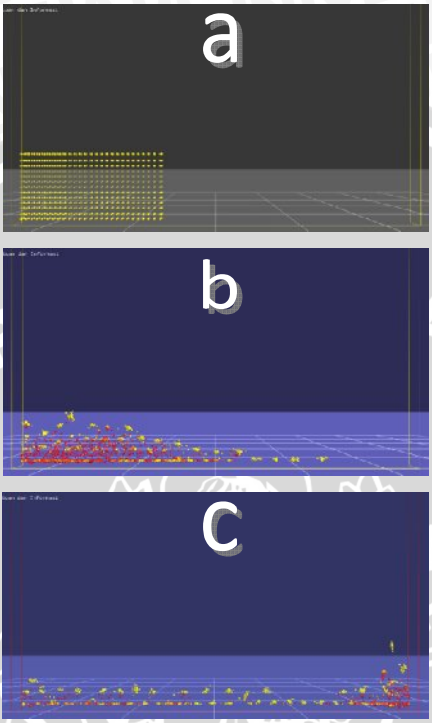
Fungsi kernel yang pada umumnya digunakan selalu mempunyai kemiripan bentuk dengan fungsi *Gaussian*. Bentuk fungsi *Gaussian* bernilai paling tinggi pada pusat dan menurun secara eksponensial seiring bertambahnya jarak. Kernel *Gaussian* dibuat dari fungsi *Gaussian* yang mana fungsi tersebut tidak pernah bernilai nol, tetapi hanya mendekati nol.

Namun demikian, setelah dilakukan uji coba penggunaan kernel *Gaussian* untuk simulasi dinamika fuida pada kasus dam jebol, didapatkan hasil yang jauh dari yang diharapkan. Gambar 4.7 memperlihatkan hasil simulasi dengan menggunakan kernel *Gaussian* pada kasus dam jebol.



Gambar 4.7 Hasil simulasi menggunakan kernel *Gaussian* dengan waktu yang sama seperti gambar 4.3

Selanjutnya dilakukan uji coba untuk mengatasi keadaan yang kurang baik ini dengan menggunakan *Gradien* dan *Laplacian* fungsi kernel yang telah berhasil digunakan pada kernel *Dome-shaped*. Gambar 4.8 menunjukkan hasil yang diperoleh setelah dilakukan modifikasi fungsi kernel. Dari sifat fisis fungsi kernel yang tidak pernah bernilai nol (hanya mendekati nol, kemudian secara tiba-tiba dipaksa bernilai nol), menyebabkan ketidakstabilan fungsi. Perubahan secara tiba-tiba ini yang diperkirakan menyebabkan suatu impuls dan mengganggu kestabilan sistem secara total.



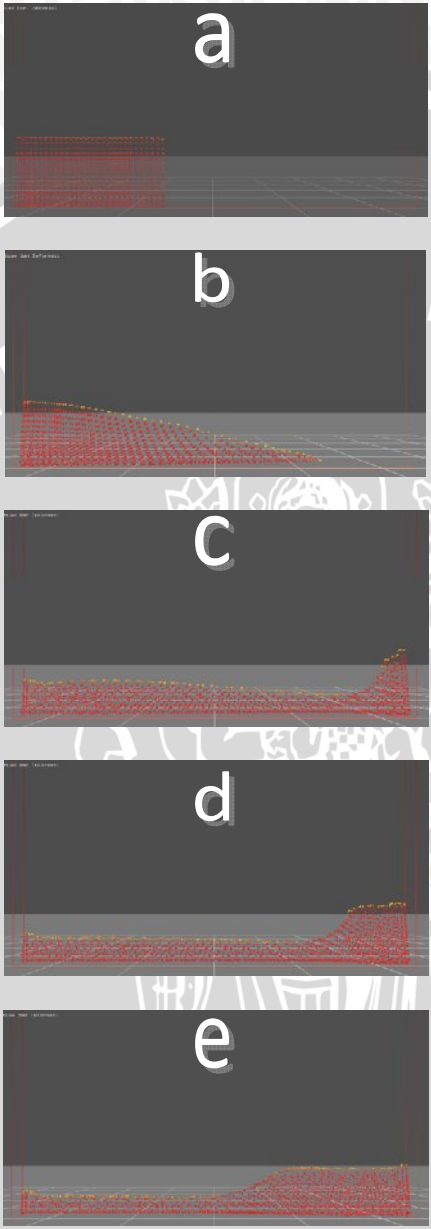
Gambar 4.8 Hasil simulasi menggunakan kernel *Gaussian* termodifikasi dengan waktu yang sama seperti gambar 4.3

Dari gambar 4.8 diketahui bahwa hasil yang didapatkan masih jauh dari situasi yang diharapkan, oleh karena itu diambil kesimpulan sementara yang menyatakan gagalnya uji coba simulasi dinamika fluida dengan menggunakan kernel *Gaussian* dan kernel tersebut tidak cocok untuk simulasi dinamika fluida.

4.3.3 Fungsi kernel *Quartic*

Kernel *Quartic* merupakan kernel klasik dengan bentuk yang sederhana. Karakteristik dari kernel *Quartic* dapat dilihat pada gambar 2.5.

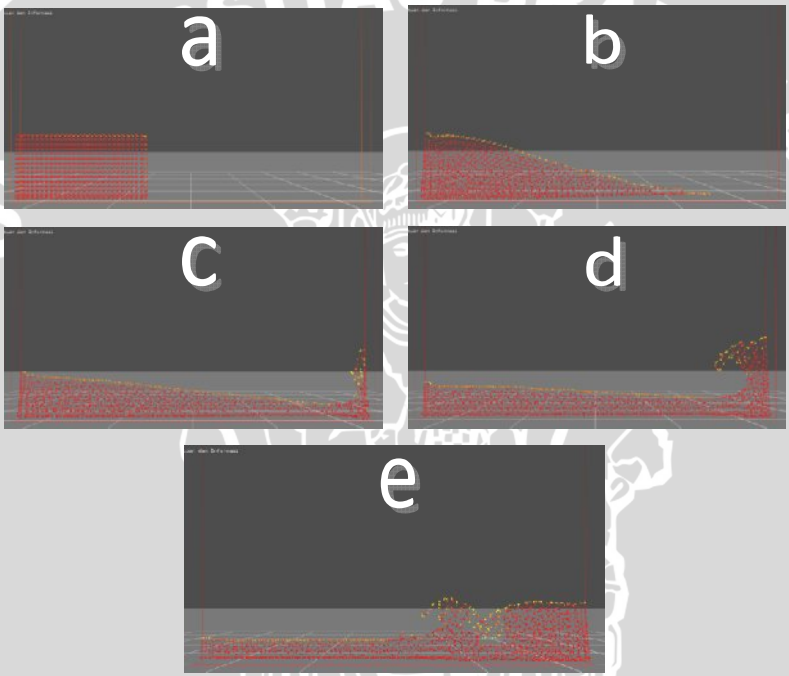
Gambar 4.9 merupakan hasil simulasi menggunakan kernel *Quartic*.



Gambar 4.9 Hasil simulasi menggunakan kernel *Quartic* dengan waktu yang sama seperti gambar 4.3

Pada gambar 4.9 a, b, dan c simulasi masih sesuai dengan hasil uji coba seperti pada gambarl 4.4. baru kemudian pada gambar 4.9 d dan e ada perbedaan karena tidak adanya aliran riak gelombang setelah pemantulan.

Untuk mengatasi permasalahan ini, penulis mencoba menggunakan *Laplacian* kernel yang lain yaitu sama dengan yang digunakan pada anak sub-bab 4.4.1. Hasil simulasinya dapat dilihat pada gambar 4.10 berikut



Gambar 4.10 Hasil simulasi dengan kernel *Quartic* termodifikasi dengan waktu yang sama seperti gambar 4.3

Dari gambar 4.10 dapat dilihat bahwa hasil simulasi dengan menggunakan kernel *Quartic* termodifikasi lebih baik daripada menggunakan kernel yang diusulkan oleh Müller *et al*. dalam hal kesesuaian dengan hasil pekerjaan Abdolmaleki *et al*. Namun demikian kernel yang digunakan Müller lebih unggul dalam hal kehalusan model. Dari gambar juga dapat diketahui bahwa tidak ada penggumpalan partikel seperti yang dikhawatirkan oleh Müller, yang

mengharuskan penggantian *Gradien* kernel normal dengan *Gradien* kernel *Spiky*. Tidak adanya penggumpalan ini terjadi karena kecepatan partikel dalam kasus dam jebol tidaklah terlalu tinggi, sehingga gaya tolak antar partikel akan bekerja menjauhkan partikel sebelum partikel berjarak nol dan kehilangan gaya tolaknya. Jika kecepatan partikel sangat tinggi, maka diperkirakan akan terjadi penggumpalan partikel karena gaya tolak tidak sempat mencegah partikel agar tidak saling tumpang tindih.

BAB V
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Dari penelitian yang dilakukan dapat diambil beberapa kesimpulan yaitu:

a. Ketepatan pemilihan jumlah partikel dapat mempengaruhi stabilitas dan kecepatan simulasi, selain itu kestabilan simulasi dipengaruhi juga oleh penentuan konstanta kekakuan.
b. Kestabilan fungsi kernel terletak pada karakteristik kernel yang bersangkutan, dan dapat digunakan beberapa kernel yang berbeda untuk mendapatkan kestabilan hasil.
c. Dari uji coba parameter dan fungsi kernel didapatkan jumlah partikel ideal untuk simulasi sebesar 5000 partikel setiap 0.001 m^3 dan 40 partikel berada di dalam radius kernel. Kernel yang paling sesuai adalah kernel *Quartic* dengan *Laplacian* menggunakan kernel *Viscosity*.

5.2 Saran

Diharapkan adanya penelitian aliran fluida dalam kasus dam jebol dengan pangamatan secara langsung dan digunakan sebagai pembanding simulasi.

(halaman ini sengaja dikosongkan)

DAFTAR PUSTAKA

Abdolmaleki K., Thiagarajan K. P., dan Morris-Thomas M. T. (2004). *Simulation of The Dam Break Problem and Impact Flows Using a Navier-Stokes Solver*. 15th Australasian Fluids Mechanics Conference. Sydney, Australia

Allard, Jeremie (2009). *Interactive Physics Tutorial: Time Integration*. INRIA/LIFL.

Halliday, David dan Robert Resnick (1978). *Fisika: Jilid 1 edisi ketiga*, terjemahan Pantur Silaban dan Erwin Sucipto 2005. Penerbit Erlangga, Jakarta.

Hamdi, Khairul (2008). *Implementasi Sistem Partikel Menggunakan Metoda Smoothed Paticle Hydrodynamics (SPH) untuk Simulasi Aliran Lava*. Tesis Master Fakultas Teknik Institut Teknologi Bandung.

Hobbs, David (2009). *Smoothed Particle Hydrodynamics (SPH): a Meshfree Particle Method for Astrophysics*, Materi kuliah pada Lund Observatory.

Liu, G. R. (2003). *Meshfree Methods: Moving beyond the Finite Element Method*. CRC Press. USA.

Liu, G. R. dan M. B. Liu (2003). *Smoothed Particle Hydrodynamics: a Meshfree Particle Method*. World Scientific Publishing Co. Pte. Ltd. Singapore.

Müller, Mattias, David Charypar, dan Markus Gross (2003). *Particle-Based Fluid Simulation for Interactive Applications*. Jurnal pada Eurographics/SIGGRAPH Symposium on Computer Animation (2003).

Müller, Mattias, *et. al*. (2005). *Particle-Based Fluid-Fluid Interaction*. Jurnal pada Eurographics/ACM SIGGRAPH Symposium on Computer Animation (2005).

Roger, Benedict D. *et. al*. (2003). *Smoothed Particle Hydrodynamics for Naval Hydrodynamics*. Paper.

Stam, Jos. (2003). *Real-Time Fluid Dynamics for Games*. Paper. Canada.

Streeter, Victor L. dan E. Benjamin Wylie (1985). *Mekanika Fluida : edisi delapan jilid 1*, terjemahan Arko Prijono 1996. Penerbit Erlangga, Jakarta.

LAMPIRAN-LAMPIRAN

(halaman ini sengaja dikosongkan)

Lampiran 1: Tabel Hasil Variasi Parameter Partikel Khayal (kumpulan dari partikel air) terhadap Kecepatan Simulasi

No	Jumlah partikel tiap 0.001 m^3	Jumlah partikel tiap radius kernel	FPS	Kesesuaian (kualitatif)
1	1000	10	6	70%
		20	5	70%
		30	3	70%
		40	6	60%
		50	3	30%
2	2000	10	6	70%
		20	4	75%
		30	3	75%
		40	3	80%
		50	3	50%
3	3000	10	6	75%
		20	4	90%
		30	4	90%
		40	3	95%
		50	3	80%
4	4000	10	7	75%
		20	5	85%
		30	4	95%
		40	4	90%
		50	3	95%
5	5000	10	7	70%
		20	5	80%
		30	4	95%
		40	4	95%
		50	3	90%

(halaman ini sengaja dikosongkan)

Lampiran 2: Kode Program

a. definisi.h

```
#ifndef COMMON_DEF
	#define COMMON_DEF

	//definisi global
	#define BAYANGAN

	#define TEX_SIZE		2048
	#define LIGHT_NEAR		0.5
	#define LIGHT_FAR		300.0
	#define DEGtoRAD		(3.141592/180.0)
	#include <windows.h>
	typedef unsigned int		uint;

	#define COLOR(r,g,b)	(	(DWORD(r*255.0f)<<24)	|
(DWORD(g*255.0f)<<16) | (DWORD(b*255.0f)<<8) )
	#define COLORA(r,g,b,a)		(	(DWORD(r*255.0f)<<24)	|
(DWORD(g*255.0f)<<16) | (DWORD(b*255.0f)<<8) | DWORD(a*255.0f) )
	#define RED(c)			(float((c>>24) & 0xFF)/255.0)
	#define GRN(c)			(float((c>>16) & 0xFF)/255.0)
	#define BLUE(c)				(float((c>>8)		&
0xFF)/255.0)
	#define ALPH(c)				(float(c		&
0xFF)/255.0)
#endif
```

b. SPHku.h

```
#ifndef SPH_SISTEM
	#define SPH_SISTEM

	#include <iostream>
	#include <vector>
	#include <stdio.h>
	#include <stdlib.h>
	#include <math.h>

	#include "set_titik.h"
	#include "vector.h"
	#include "definisi.h"

	// Parameter skalar
	#define UKURAN_SIMULASI_SPH		4
	#define SKALA_SIMULASI_SPH		5
	#define VISKOSITAS_SPH			6
	#define RESTDENSITY_SPH				7
	#define MASSA_PARTIKEL_SPH		8
	#define RADIUS_PARTIKEL_SPH		9
	#define JARAK_PARTIKEL_SPH		10
	#define SMOOTHRADIUS_SPH		11
	#define INTERNAL_STIFFNESS_SPH		12
	#define EKSTERNAL_STIFFNESS_SPH		13
	#define EKSTERNAL_DAMPEN_SPH  14
	#define LIMIT_SPH				15
	#define BATAS_BAWAH_ZMIN		16
	#define FORCE_XMAX_SIN			17
	#define FORCE_XMIN_SIN			18
	#define MAX_FRAC				19
	#define CLR_MODE				20

	// Parameter vektor
	#define VOLUME_MIN_SPH			7
	#define VOLUME_MAKS_SPH			8
	#define INITMIN_SPH			9
	#define INITMAKS_SPH			10

	// Pilihan
	#define SPH_GRID			0
	#define SPH_DEBUG			1
	#define WRAP_X				2
	#define WALL_BARRIER		3
	#define LEVY_BARRIER		4
	#define DRAIN_BARRIER		5
	#define GANGGUAN			6

	#define PARAMAX					21
	#define BFluid				2

	struct sph {
	public:
		Vector3DF		posisi;			//
Partikel dasar (harus sesuai dengan kelas partikel di Set Titik
```

```
		DWORD			clr;
		int				next;
		Vector3DF		kecepatan;

		Vector3DF		evaluasi_kecepatan;

		unsigned short age;

		float			tekanan;		//
parameter yang dihitung dalam SPH
		float			kerapatan;
		Vector3DF		force;
	};

	class SPHKu : public SetTitik {
		double
	R_kuadrat, Kernel_normal, Kernel_gradien, Kernel_laplacian;
		// Fungsi Kernel
	public:
		SPHKu ();

		// Sistem partikel
		virtual void Initialize ( int mode, int nmax );
		virtual void Reset ( int nmax );
		virtual void Run ();
		virtual void hitung_Lanjutan ();
		virtual int AddPoint ();
		virtual int AddPointReuse ();
		sph* AddFluid ()			{	return
(sph*) GetElem(0, AddPointReuse()); }
		sph* GetFluid (int n)		{	return	(sph*)
GetElem(0, n); }

		// Smoothed Partikel Hydrodynamics
		void sphSetup ();
		void sphTampil ( int n, int nmax );
		void sphDomain ();
		void sphHitungKernel ();

		//void sphHitungTekanan ();			//
O(n^2)
		void sphHitungTekananGrid ();	//	O(kn)	-
spatial grid

		//void sphHitungForce ();
	// O(n^2)
		//void sphHitungForceGrid ();		//
O(kn) - spatial grid
		void sphHitungForceGridNC ();	//	O(cn)	-
neighbor table
	};
#endif
```

c. SPHku.cpp

```
#include <conio.h>
#include <GL/glut.h>

#include "definisi.h"
#include "mtime.h"
#include "SPHku.h"

#define EPSILON                         0.00001f
        //untuk deteksi tumbukan (tresshold)

SPHKu::SPHKu ()
{
}
void SPHKu::Initialize ( int mode, int total )
{
        if ( mode != BFluid ) {
                printf   (   "ERROR:   Sistem   tidak   diinisialisasi
sebagai BFluid.\n");
        }
        SetTitik::Initialize ( mode, total );

        FreeBuffers ();
        AddBuffer ( BFluid, sizeof ( sph ), total );
        AddAttribute ( 0, "posisi", sizeof ( Vector3DF ), false );

        AddAttribute ( 0, "color", sizeof ( DWORD ), false );
        AddAttribute ( 0, "kecepatan", sizeof ( Vector3DF ), false
);
        AddAttribute ( 0, "ndx", sizeof ( unsigned short ), false
);
        AddAttribute ( 0, "age", sizeof ( unsigned short ), false
);

        AddAttribute ( 0, "tekanan", sizeof ( double ), false );
        AddAttribute ( 0, "kerapatan", sizeof ( double ), false );
        AddAttribute ( 0, "force", sizeof ( Vector3DF ), false );
        AddAttribute ( 0, "next", sizeof ( sph* ), false );
        AddAttribute ( 0, "tag", sizeof ( bool ), false );


        sphSetup ();
        Reset ( total );
}

void SPHKu::Reset ( int nmax )
{
        ResetBuffer ( 0, nmax );
        DT = 0.0025; //  0.001;

        // Mereset parameter-parameter
        Parameter [ MAX_FRAC ] = 1.0;
        Parameter [ POINT_GRAV ] = 0.0;
        Parameter [ PLANE_GRAV ] = 1.0;
```

```
        Parameter [ BATAS_BAWAH_ZMIN ] = 0.0;
        Parameter [ FORCE_XMAX_SIN ] = 0.0;
        Parameter [ FORCE_XMIN_SIN ] = 0.0;
        Pilihan [ WRAP_X ] = false;
        Pilihan [ WALL_BARRIER ] = false;
        Pilihan [ LEVY_BARRIER ] = false;
        Pilihan [ DRAIN_BARRIER ] = false;
        //Parameter [ INTERNAL_STIFFNESS_SPH ] = 1.00;
        Parameter [ VISKOSITAS_SPH ] =       1.02;
        Parameter [ INTERNAL_STIFFNESS_SPH ] = 0.01;
        Parameter [ EKSTERNAL_STIFFNESS_SPH ] = 20000;
        Parameter [ SMOOTHRADIUS_SPH ] =     0.011272517;

        vektor [ POINT_GRAV_POS ].Set ( 0, 0, 50 );
        vektor [ PLANE_GRAV_DIR ].Set ( 0, 0, -9.8 );
        vektor [ EMIT_POSISI ].Set ( 0, 0, 0 );
        vektor [ EMIT_RATE ].Set ( 0, 0, 0 );
        vektor [ EMIT_ANG ].Set ( 0, 90, 1.0 );
        vektor [ EMIT_DANG ].Set ( 0, 0, 0 );
}

int SPHKu::AddPoint ()
{
        xref ndx;
        sph* f = (sph*) AddElem ( 0, ndx );
        f->force.Set(0,0,0);
        f->kecepatan.Set(0,0,0);
        f->evaluasi_kecepatan.Set(0,0,0);
        f->next = 0x0;
        f->tekanan = 0;
        f->kerapatan = 0;
        return ndx;
}

int SPHKu::AddPointReuse ()
{
        xref ndx;
        sph* f;
        if ( NumPoints() <= mBuf[0].max-2 )
                f = (sph*) AddElem ( 0, ndx );
        else
                f = (sph*) RandomElem ( 0, ndx );

        f->force.Set(0,0,0);
        f->kecepatan.Set(0,0,0);
        f->evaluasi_kecepatan.Set(0,0,0);
        f->next = 0x0;
        f->tekanan = 0;
        f->kerapatan = 0;
        return ndx;
}

void SPHKu::Run ()
{
        bool bTiming = true;

        mint::Time start, stop;
```

```
	float ss = Parameter [ JARAK_PARTIKEL_SPH ] / Parameter[
SKALA_SIMULASI_SPH ];		// skala simulasi

	if ( vektor[EMIT_RATE].x > 0 && (++FRame) % (int)
vektor[EMIT_RATE].x == 0 ) {
		Emit ( ss );
	}


	//mulai proses penghitungan SPH
	start.SetSystemTime ( ACC_NSEC );
	hitungPosisis ();
	if ( bTiming) {
		stop.SetSystemTime ( ACC_NSEC );
		stop = stop - start;
		//printf	(	"PARTIKEL	MASUK:	%s\n",
stop.GetReadableTime().c_str() );
	}

	start.SetSystemTime ( ACC_NSEC );
	sphHitungTekananGrid ();
	if ( bTiming) {
		stop.SetSystemTime ( ACC_NSEC );
		stop = stop - start;
		//printf		(		"TEKANAN:		%s\n",
stop.GetReadableTime().c_str() );
	}

	start.SetSystemTime ( ACC_NSEC );
	sphHitungForceGridNC ();
	if ( bTiming) {
		stop.SetSystemTime ( ACC_NSEC );
		stop = stop - start;
		//printf		(		"GAYA:		%s\n",
stop.GetReadableTime().c_str() );
	}

	start.SetSystemTime ( ACC_NSEC );
	hitung_Lanjutan();
	if ( bTiming) {
		stop.SetSystemTime ( ACC_NSEC );
		stop = stop - start;
		//printf	(	"hitung_Lanjutan:	%s\n",
stop.GetReadableTime().c_str() );
	}
}



void SPHKu::sphDomain ()
{
	Vector3DF min, max;
	min = vektor[VOLUME_MIN_SPH];
	max = vektor[VOLUME_MAKS_SPH];
	min.z += 0.5;
```

```
        glColor3f ( 0.0, 0.0, 1.0 );
        glBegin ( GL_LINES );
        glVertex3f ( min.x, min.y, min.z );  glVertex3f  (  max.x,
min.y, min.z );
        glVertex3f ( min.x, max.y, min.z );  glVertex3f  (  max.x,
max.y, min.z );
        glVertex3f ( min.x, min.y, min.z );  glVertex3f  (  min.x,
max.y, min.z );
        glVertex3f ( max.x, min.y, min.z );  glVertex3f  (  max.x,
max.y, min.z );
        glEnd ();
}

void SPHKu::hitung_Lanjutan ()
{
        char *dat1, *dat1_end;
        sph* p;
        Vector3DF norm, z;
        Vector3DF dir, accel;
        Vector3DF vnext;
        Vector3DF min, max;
        double adj;
        float SL, SL2, ss, radius;
        float stiff, damp, speed, diff;
        SL = Parameter[LIMIT_SPH];
        SL2 = SL*SL;

        stiff = Parameter[EKSTERNAL_STIFFNESS_SPH];
        damp = Parameter[EKSTERNAL_DAMPEN_SPH];
        radius = Parameter[RADIUS_PARTIKEL_SPH];
        min = vektor[VOLUME_MIN_SPH];
        max = vektor[VOLUME_MAKS_SPH];
        ss = Parameter[SKALA_SIMULASI_SPH];

        dat1_end = mBuf[0].data + NumPoints()*mBuf[0].stride;
        for  (  dat1  =  mBuf[0].data;  dat1  <  dat1_end;  dat1  +=
mBuf[0].stride ) {
                p = (sph*) dat1;

                // Hitung akselerasi
                accel = p->force;
                accel *= Parameter[MASSA_PARTIKEL_SPH];

                // Batasan kecepatan
                speed   =   accel.x*accel.x   +   accel.y*accel.y   +
accel.z*accel.z;
                if ( speed > SL2 ) {
                        accel *= SL / sqrt(speed);
                }

                // Kondisi Batas

                // Dinding sumbu z
                diff  =  2  *  radius  -  (  p->posisi.z  -  min.z  -  (p-
>posisi.x          -          vektor[VOLUME_MIN_SPH].x)          *
Parameter[BATAS_BAWAH_ZMIN] )*ss;
                if (diff > EPSILON ) {
```

```
                        norm.Set ( -Parameter[BATAS_BAWAH_ZMIN], 0,
1.0 - Parameter[BATAS_BAWAH_ZMIN] );
                        adj = stiff * diff - damp * norm.Dot ( p-
>evaluasi_kecepatan );
                        accel.x += adj * norm.x; accel.y += adj *
norm.y; accel.z += adj * norm.z;
                }

                diff = 2 * radius - ( max.z - p->posisi.z )*ss;
                if (diff > EPSILON) {
                        norm.Set ( 0, 0, -1 );
                        adj = stiff * diff - damp * norm.Dot ( p-
>evaluasi_kecepatan );
                        accel.x += adj * norm.x; accel.y += adj *
norm.y; accel.z += adj * norm.z;
                }

                // Dinding sumbu x
                if ( !Pilihan[WRAP_X] ) {
                        diff = 2 * radius - ( p->posisi.x - min.x +
(sin(timing*10.0)-1+(p->posisi.y*0.025)*0.25)                    *
Parameter[FORCE_XMIN_SIN] )*ss;
                        //diff = 2 * radius - ( p->posisi.x - min.x
+ (sin(timing*10.0)-1) * Parameter[FORCE_XMIN_SIN] )*ss;
                        if (diff > EPSILON ) {
                                norm.Set ( 1.0, 0, 0 );
                                adj = (Parameter[ FORCE_XMIN_SIN ] +
1) * stiff * diff - damp * norm.Dot ( p->evaluasi_kecepatan ) ;
                                accel.x += adj * norm.x; accel.y +=
adj * norm.y; accel.z += adj * norm.z;

                        }

                        diff = 2 * radius - ( max.x - p->posisi.x +
(sin(timing*10.0)-1) * Parameter[FORCE_XMAX_SIN] )*ss;
                        if (diff > EPSILON) {
                                norm.Set ( -1, 0, 0 );
                                adj   =   (Parameter[   FORCE_XMAX_SIN
]+1) * stiff * diff - damp * norm.Dot ( p->evaluasi_kecepatan );
                                accel.x += adj * norm.x; accel.y +=
adj * norm.y; accel.z += adj * norm.z;
                        }
                }

                // Dinding sumbu y
                diff = 2 * radius - ( p->posisi.y - min.y )*ss;

                if (diff > EPSILON) {
                        norm.Set ( 0, 1, 0 );
                        adj = stiff * diff - damp * norm.Dot ( p-
>evaluasi_kecepatan );
                        accel.x += adj * norm.x; accel.y += adj *
norm.y; accel.z += adj * norm.z;
                }
                diff = 2 * radius - ( max.y - p->posisi.y )*ss;
                if (diff > EPSILON) {
                        norm.Set ( 0, -1, 0 );
```

```
                        adj = stiff * diff - damp * norm.Dot ( p-
>evaluasi_kecepatan );
                        accel.x += adj * norm.x; accel.y += adj *
norm.y; accel.z += adj * norm.z;
                }

                // Wall barrier
                if ( Pilihan[WALL_BARRIER] ) {
                        diff = 2 * radius - ( p->posisi.x - 0 )*ss;

                        if (diff < 2*radius && diff > EPSILON &&
fabs(p->posisi.y) < 3 && p->posisi.z < 10) {
                                norm.Set ( 1.0, 0, 0 );
                                adj = 2*stiff * diff - damp *
norm.Dot ( p->evaluasi_kecepatan ) ;
                                accel.x += adj * norm.x; accel.y +=
adj * norm.y; accel.z += adj * norm.z;

                        }
                }

                // Levy barrier
                if ( Pilihan[LEVY_BARRIER] ) {
                        diff = 2 * radius - ( p->posisi.x - 0 )*ss;

                        if (diff < 2*radius && diff > EPSILON &&
fabs(p->posisi.y) > 5 && p->posisi.z < 10) {
                                norm.Set ( 1.0, 0, 0 );
                                adj = 2*stiff * diff - damp *
norm.Dot ( p->evaluasi_kecepatan ) ;
                                accel.x += adj * norm.x; accel.y +=
adj * norm.y; accel.z += adj * norm.z;

                        }
                }
                // Drain barrier
                if ( Pilihan[DRAIN_BARRIER] ) {
                        diff = 2 * radius - ( p->posisi.z - min.z-15
)*ss;
                        if (diff < 2*radius && diff > EPSILON &&
(fabs(p->posisi.x)>3 || fabs(p->posisi.y)>3) ) {
                                norm.Set ( 0, 0, 1);
                                adj = stiff * diff - damp * norm.Dot
( p->evaluasi_kecepatan );
                                accel.x += adj * norm.x; accel.y +=
adj * norm.y; accel.z += adj * norm.z;
                        }
                }

                // Plane gravity
                if ( Parameter[PLANE_GRAV] > 0)
                        accel += vektor[PLANE_GRAV_DIR];

                // Point gravity
                if ( Parameter[POINT_GRAV] > 0 ) {
                        norm.x      =      (      p->posisi.x      -
vektor[POINT_GRAV_POS].x );
```

```
                    norm.y      =      (      p->posisi.y      -
vektor[POINT_GRAV_POS].y );
                    norm.z      =      (      p->posisi.z      -
vektor[POINT_GRAV_POS].z );
                    norm.Normalize ();
                    norm *= Parameter[POINT_GRAV];
                    accel -= norm;
             }

             // Integrasi Leapfrog ----------------------------
             vnext = accel;

             vnext *= DT;
             vnext += p->kecepatan;
             // v(t+1/2) = v(t-1/2) + a(t) dt
             p->evaluasi_kecepatan = p->kecepatan;
             p->evaluasi_kecepatan += vnext;
             p->evaluasi_kecepatan *= 0.5;
             // v(t+1) = [v(t-1/2) + v(t+1/2)] * 0.5
             p->kecepatan = vnext;
             vnext *= DT/ss;
             p->posisi += vnext;
             // p(t+1) = p(t) + v(t+1/2) dt

             if ( Parameter[CLR_MODE]==1.0 ) {
                    adj                                          =
fabs(vnext.x)+fabs(vnext.y)+fabs(vnext.z) / 7000.0;
                    adj = (adj > 1.0) ? 1.0 : adj;
                    p->clr = COLORA( 0, adj, 1-adj, 1 );
             }
             if ( Parameter[CLR_MODE]==2.0 ) {
                    float v = 0.5 + ( p->tekanan / 1000.0);
                    if ( v < 0.1 ) v = 0.1;
                    if ( v > 1.0 ) v = 1.0;
                    p->clr = COLORA ( v, 1-v, 0, 1. ); //p->clr
= COLORA ( v, 1-v, 0, 1 );
             }


             // Integrasi Euler -------------------------------
             //accel += -9.8;;
             //accel *= DT;
             //p->kecepatan += accel;
      // v(t+1) = v(t) + a(t) dt
             //p->evaluasi_kecepatan += accel;
             //p->evaluasi_kecepatan *= DT/ss;
             //p->posisi += p->evaluasi_kecepatan;
             //p->evaluasi_kecepatan = p->kecepatan;


             if ( Pilihan[WRAP_X] ) {
                    diff         =         p->posisi.x         -
(vektor[VOLUME_MIN_SPH].x + 2);                    //          --
Mensimulasikan objek di dalam pusat aliran
                    if ( diff <= 0 ) {
```

```
                        p->posisi.x                     =
(vektor[VOLUME_MAKS_SPH].x - 2) + diff*2;

                        p->posisi.z = 10;
                    }
                }
        }

        timing += DT;
}

//**************************SPH
Setup**************************************
//
//  Range = +/- 10.0 * 0.006 (r) =          0.12
        m (= 120 mm = 4.7 inch)
//  Volume wadah (Vc) =
        0.001728                m^3
//  Rest Density (D) =                                  1000.0
                kg / m^3
//  Massa Partikel (Pm) =                               0.00020543
        kg                                              (mass = vol *
kerapatan)
//  Jumlah Partikel (N) =                               4000.0
//  Massa air (M) =                                             0.821
                kg (= 821 grams)
//  Volum air (V) =
        0.000821    m^3 (= 3.4 cups, .21 gals)
//  Smoothing Radius (R) =                              0.02
                m (= 20 mm = ~3/4 inch)
//  Radius Partikel (Pr) =                              0.00366
        m (= 4 mm  = ~1/8 inch)
//  Volume Partikel (Pv) =                              2.054e-7
                m^3    (= .268 milliliters)
//  Rest Distance (Pd) =                                0.0059
        m
//
//  Given: D, Pm, N
//    Pv = Pm / D                       0.00020543 kg / 1000 kg/m^3 =
2.054e-7 m^3
//     Pv = 4/3*pi*Pr^3     cuberoot( 2.054e-7 m^3 * 3/(4pi) ) =
0.00366 m
//     M = Pm * N                        0.00020543  kg  *  4000.0  =
0.821 kg
//     V =  M / D                  0.821 kg / 1000 kg/m^3 = 0.000821
m^3
//     V = Pv * N                        2.054e-7   m^3   *   4000   =
0.000821 m^3
//    Pd = cuberoot(Pm/D)    cuberoot(0.00020543/1000) = 0.0059 m
//
// Ukuran grid cell ideal (gs) = 2 * smoothing radius = 0.02*2 =
0.04
// Ukuran domain ideal = k*gs/d = k*0.02*2/0.005 = k*8 = {8, 16,
24, 32, 40, 48, ..}
//    (k = Jumlah cells, gs = ukuran cell, d = Skala simulasi)

void SPHKu::sphSetup ()
```

```
{
        //parameter yang di tuning
        /*Parameter [ MASSA_PARTIKEL_SPH ] =  0.00020543;
        // kg
        Parameter [ RADIUS_PARTIKEL_SPH ] =          0.004;
                // m
        Parameter [ JARAK_PARTIKEL_SPH ] =           0.0059;
                // m
        Parameter [ SMOOTHRADIUS_SPH ] =             0.01;
                // m*/

/*
        //1000
        //10
        Parameter [ MASSA_PARTIKEL_SPH ] =           0.001;
        Parameter [ RADIUS_PARTIKEL_SPH ] =          0.006203505;
        Parameter [ JARAK_PARTIKEL_SPH ] =           0.01;
        Parameter [ SMOOTHRADIUS_SPH ] =             0.013365046;

        //20
        Parameter [ MASSA_PARTIKEL_SPH ] =           0.001;
        Parameter [ RADIUS_PARTIKEL_SPH ] =          0.006203505;
        Parameter [ JARAK_PARTIKEL_SPH ] =           0.01;
        Parameter [ SMOOTHRADIUS_SPH ] =             0.016838903;

        //30
        Parameter [ MASSA_PARTIKEL_SPH ] =           0.001;
        Parameter [ RADIUS_PARTIKEL_SPH ] =          0.006203505;
        Parameter [ JARAK_PARTIKEL_SPH ] =           0.01;
        Parameter [ SMOOTHRADIUS_SPH ] =             0.019275732;

        //40
        Parameter [ MASSA_PARTIKEL_SPH ] =           0.001;
        Parameter [ RADIUS_PARTIKEL_SPH ] =          0.006203505;
        Parameter [ JARAK_PARTIKEL_SPH ] =           0.01;
        Parameter [ SMOOTHRADIUS_SPH ] =             0.0121215688;

        //50
        Parameter [ MASSA_PARTIKEL_SPH ] =           0.001;
        Parameter [ RADIUS_PARTIKEL_SPH ] =          0.006203505;
        Parameter [ JARAK_PARTIKEL_SPH ] =           0.01;
        Parameter [ SMOOTHRADIUS_SPH ] =             0.022853907;

        //2000
        //10
        Parameter [ MASSA_PARTIKEL_SPH ] =           0.0005;
        Parameter [ RADIUS_PARTIKEL_SPH ] =          0.004923725;
        Parameter [ JARAK_PARTIKEL_SPH ] =           0.007937;
        Parameter [ SMOOTHRADIUS_SPH ] =             0.010607884;

        //20
        Parameter [ MASSA_PARTIKEL_SPH ] =           0.0005;
        Parameter [ RADIUS_PARTIKEL_SPH ] =          0.004923725;
        Parameter [ JARAK_PARTIKEL_SPH ] =           0.007937;
        Parameter [ SMOOTHRADIUS_SPH ] =             0.013365046;

        //30
```

```
Parameter [ MASSA_PARTIKEL_SPH ] =          0.0005;
Parameter [ RADIUS_PARTIKEL_SPH ] =         0.004923725;
Parameter [ JARAK_PARTIKEL_SPH ] =          0.007937;
Parameter [ SMOOTHRADIUS_SPH ] =            0.015299159;

//40
Parameter [ MASSA_PARTIKEL_SPH ] =          0.0005;
Parameter [ RADIUS_PARTIKEL_SPH ] =         0.004923725;
Parameter [ JARAK_PARTIKEL_SPH ] =          0.007937;
Parameter [ SMOOTHRADIUS_SPH ] =            0.016838903;

//50
Parameter [ MASSA_PARTIKEL_SPH ] =          0.0005;
Parameter [ RADIUS_PARTIKEL_SPH ] =         0.004923725;
Parameter [ JARAK_PARTIKEL_SPH ] =          0.007937;
Parameter [ SMOOTHRADIUS_SPH ] =            0.018139158;

//3000
//10
Parameter [ MASSA_PARTIKEL_SPH ] =          0.000333;
Parameter [ RADIUS_PARTIKEL_SPH ] =         0.004301269;
Parameter [ JARAK_PARTIKEL_SPH ] =          0.006934;
Parameter [ SMOOTHRADIUS_SPH ] =            0.009266805;

//20
Parameter [ MASSA_PARTIKEL_SPH ] =          0.000333;
Parameter [ RADIUS_PARTIKEL_SPH ] =         0.004301269;
Parameter [ JARAK_PARTIKEL_SPH ] =          0.006934;
Parameter [ SMOOTHRADIUS_SPH ] =            0.011675443;

//30
Parameter [ MASSA_PARTIKEL_SPH ] =          0.000333;
Parameter [ RADIUS_PARTIKEL_SPH ] =         0.004301269;
Parameter [ JARAK_PARTIKEL_SPH ] =          0.006934;
Parameter [ SMOOTHRADIUS_SPH ] =            0.013365046;

//40
Parameter [ MASSA_PARTIKEL_SPH ] =          0.000333;
Parameter [ RADIUS_PARTIKEL_SPH ] =         0.004301269;
Parameter [ JARAK_PARTIKEL_SPH ] =          0.006934;
Parameter [ SMOOTHRADIUS_SPH ] =            0.014710137;

//50
Parameter [ MASSA_PARTIKEL_SPH ] =          0.000333;
Parameter [ RADIUS_PARTIKEL_SPH ] =         0.004301269;
Parameter [ JARAK_PARTIKEL_SPH ] =          0.006934;
Parameter [ SMOOTHRADIUS_SPH ] =            0.015846014;

//4000
//10
Parameter [ MASSA_PARTIKEL_SPH ] =          0.00025;
Parameter [ RADIUS_PARTIKEL_SPH ] =         0.003907963;
Parameter [ JARAK_PARTIKEL_SPH ] =          0.0063;
Parameter [ SMOOTHRADIUS_SPH ] =            0.008419452;

//20
Parameter [ MASSA_PARTIKEL_SPH ] =          0.00025;
```

```
        Parameter [ RADIUS_PARTIKEL_SPH ] =          0.003907963;
        Parameter [ JARAK_PARTIKEL_SPH ] =           0.0063;
        Parameter [ SMOOTHRADIUS_SPH ] =             0.010607844;

        //30
        Parameter [ MASSA_PARTIKEL_SPH ] =           0.00025;
        Parameter [ RADIUS_PARTIKEL_SPH ] =          0.003907963;
        Parameter [ JARAK_PARTIKEL_SPH ] =           0.0063;
        Parameter [ SMOOTHRADIUS_SPH ] =             0.01214295;

        //40
        Parameter [ MASSA_PARTIKEL_SPH ] =           0.00025;
        Parameter [ RADIUS_PARTIKEL_SPH ] =          0.003907963;
        Parameter [ JARAK_PARTIKEL_SPH ] =           0.0063;
        Parameter [ SMOOTHRADIUS_SPH ] =             0.013365046;

        //50
        Parameter [ MASSA_PARTIKEL_SPH ] =           0.00025;
        Parameter [ RADIUS_PARTIKEL_SPH ] =          0.003907963;
        Parameter [ JARAK_PARTIKEL_SPH ] =           0.0063;
        Parameter [ SMOOTHRADIUS_SPH ] =             0.01439706;

        //5000
        //10
        Parameter [ MASSA_PARTIKEL_SPH ] =           0.0002;
        Parameter [ RADIUS_PARTIKEL_SPH ] =          0.003627832;
        Parameter [ JARAK_PARTIKEL_SPH ] =           0.005848;
        Parameter [ SMOOTHRADIUS_SPH ] =             0.007815926;

        //20
        Parameter [ MASSA_PARTIKEL_SPH ] =           0.0002;
        Parameter [ RADIUS_PARTIKEL_SPH ] =          0.003627832;
        Parameter [ JARAK_PARTIKEL_SPH ] =           0.005848;
        Parameter [ SMOOTHRADIUS_SPH ] =             0.00984745;
*/
        //30
        Parameter [ MASSA_PARTIKEL_SPH ] =           0.0002;
        Parameter [ RADIUS_PARTIKEL_SPH ] =          0.003627832;
        Parameter [ JARAK_PARTIKEL_SPH ] =           0.005848;
        Parameter [ SMOOTHRADIUS_SPH ] =             0.011272517;
/*
        //40
        Parameter [ MASSA_PARTIKEL_SPH ] =           0.0002;
        Parameter [ RADIUS_PARTIKEL_SPH ] =          0.003627832;
        Parameter [ JARAK_PARTIKEL_SPH ] =           0.005848;
        Parameter [ SMOOTHRADIUS_SPH ] =             0.01240701;

        //50
        Parameter [ MASSA_PARTIKEL_SPH ] =           0.0002;
        Parameter [ RADIUS_PARTIKEL_SPH ] =          0.003627832;
        Parameter [ JARAK_PARTIKEL_SPH ] =           0.005848;
        Parameter [ SMOOTHRADIUS_SPH ] =             0.013365046;


        Parameter [ MASSA_PARTIKEL_SPH ] =           0.3655/10;
        Parameter [ RADIUS_PARTIKEL_SPH ] =          0.009598/10;
        Parameter [ JARAK_PARTIKEL_SPH ] =           0.0715/10;
```

```
        Parameter [ SMOOTHRADIUS_SPH ] =            0.0443/10;
*/
        //parameter milik semua cairan (air)
        Parameter [ RESTDENSITY_SPH ] =                    998.2;
                    // kg / m^3
        Parameter [ SKALA_SIMULASI_SPH ] =          0.004;
              // satuan
        Parameter [ VISKOSITAS_SPH ] =
        0.4;//1.02;                    //pascal-second (Pa.s) = 1
kg m^-1 s^-1
        Parameter [ INTERNAL_STIFFNESS_SPH ] =      1.5;
        Parameter [ EKSTERNAL_STIFFNESS_SPH ] =     10000.0;
        Parameter [ EKSTERNAL_DAMPEN_SPH ] = 256.0;
        //kelembaban eksternal
        Parameter [ LIMIT_SPH ] =                          200.0;
                    // m / s

        Pilihan [ SPH_GRID ]= false;
        Pilihan [ SPH_DEBUG ]= false;

        sphHitungKernel ();
}

void SPHKu::sphHitungKernel ()
{
        Parameter [ JARAK_PARTIKEL_SPH ] = pow (
Parameter[MASSA_PARTIKEL_SPH] / Parameter[RESTDENSITY_SPH], 1/3.0
);
        R_kuadrat = Parameter [SMOOTHRADIUS_SPH] *
Parameter[SMOOTHRADIUS_SPH];
        //Kernel_normal = 315.0f / (64.0f * 3.141592 * pow(
Parameter[SMOOTHRADIUS_SPH], 9) ); // Wpoly6 kernel (penyebut) -
2003 Muller, p.4      //Kernel_normal = 105.0f / (16.0f * 3.141592
* pow( Parameter[SMOOTHRADIUS_SPH], 3) );    //bell shaped , lucy
1977
        //Kernel_gradien = -45.0f / (3.141592 * pow(
Parameter[SMOOTHRADIUS_SPH], 6) );                  //  Laplacian
of viscocity (penyebut): PI h^6
        //Kernel_laplacian = 45.0f / (3.141592 * pow(
Parameter[SMOOTHRADIUS_SPH], 6) );

        //Kernel_normal = 105.0f / (16.0f * 3.141592 * pow(
Parameter[SMOOTHRADIUS_SPH], 3) );   //bell shaped , lucy 1977
        //Kernel_gradien = 315.0f / (16.0f * 3.141592* pow(
Parameter[SMOOTHRADIUS_SPH], 4) );
        //Kernel_laplacian = 315.0f / (16.0f * 3.141592* pow(
Parameter[SMOOTHRADIUS_SPH], 5) );

        //kernel gauss
        Kernel_normal = 1.0f / (pow(3.141592, (3./2.)) * pow(
Parameter[SMOOTHRADIUS_SPH], 3));
        Kernel_gradien = -2.0f / (pow(3.141592, (3./2.)) * pow(
Parameter[SMOOTHRADIUS_SPH], 5));
        Kernel_laplacian = 2.0f / (pow(3.141592, (3./2.)) * pow(
Parameter[SMOOTHRADIUS_SPH], 5));
}
```

```
void SPHKu::sphTampil ( int n, int nmax )
{
	Vector3DF posisi;
	Vector3DF min, max;

	Reset ( nmax );

	switch ( n ) {
	case 0:		// Dam break
		vektor  [  VOLUME_MIN_SPH  ].Set  (  -48.3,  0,  0  );
	//Set ( -30, -14, 0 );
		vektor  [  VOLUME_MAKS_SPH  ].Set  (  48.3,  5,  60  );
	//Set ( 30, 14, 60 );
		vektor [ INITMIN_SPH ].Set ( -48, 0, 0 );
	//(0, -13, 0);
		vektor [ INITMAKS_SPH ].Set ( -12, 5, 18 );
	//(-29, 4, 30);
		vektor [ PLANE_GRAV_DIR ].Set ( 0.0, 0, -9.81 );
		break;
	case 1:		// Pancuran
		/*vektor [ VOLUME_MIN_SPH ].Set ( -30, -30, 0 );
		vektor [ VOLUME_MAKS_SPH ].Set ( 30, 30, 50 );
		vektor [ INITMIN_SPH ].Set ( -30, -30, 0 );
		vektor [ INITMAKS_SPH ].Set ( 30, 30, 40 );
		vektor [ EMIT_POSISI ].Set ( -20, -20, 22 );
		vektor [ EMIT_RATE ].Set ( 1, 4, 0 );
		vektor [ EMIT_ANG ].Set ( 0, 120, 1.5 );
		vektor [ EMIT_DANG ].Set ( 0, 0, 0 );
		vektor [ PLANE_GRAV_DIR ].Set ( 0.0, 0, -9.8 );*/
		vektor [ VOLUME_MIN_SPH ].Set ( -10, -10, 0 );
		vektor [ VOLUME_MAKS_SPH ].Set ( 10, 10, 50 );
		vektor [ INITMIN_SPH ].Set ( -10, -10, 0 );
		vektor [ INITMAKS_SPH ].Set ( 10, 10, 40 );
		vektor [ EMIT_POSISI ].Set ( -5, 0, 25 );
		vektor [ EMIT_RATE ].Set ( 1, 1, 10 );
		vektor [ EMIT_ANG ].Set ( 0, 120, 1.5 );
		vektor [ EMIT_DANG ].Set ( 0, 0, 0 );
		vektor [ PLANE_GRAV_DIR ].Set ( 0.0, 0, -9.8 );
		break;
	case 2:		// Shockwave Tube
		/*vektor [ VOLUME_MIN_SPH ].Set ( -60, -15, 0 );
		vektor [ VOLUME_MAKS_SPH ].Set ( 60, 15, 50 );
		vektor [ INITMIN_SPH ].Set ( -59, -14, 0 );
		vektor [ INITMAKS_SPH ].Set ( 59, 14, 30 );*/
		vektor [ VOLUME_MIN_SPH ].Set ( -20, -5, 0 );
		vektor [ VOLUME_MAKS_SPH ].Set ( 20, 5, 17 );
		vektor [ INITMIN_SPH ].Set ( -19, -4, 0 );
		vektor [ INITMAKS_SPH ].Set ( 19, 4, 10 );
		vektor [ PLANE_GRAV_DIR ].Set ( 0.0, 0, -9.8 );
		Pilihan [ WALL_BARRIER ] = true;
		Pilihan [ WRAP_X ] = true;
		break;
	}

	sphHitungKernel ();
```

```
        Parameter   [   UKURAN_SIMULASI_SPH   ]   =   Parameter   [
SKALA_SIMULASI_SPH     ]     *     (vektor[VOLUME_MAKS_SPH].z     -
vektor[VOLUME_MIN_SPH].z);
        Parameter    [    JARAK_PARTIKEL_SPH    ]    =    pow    (
Parameter[MASSA_PARTIKEL_SPH]  /  Parameter[RESTDENSITY_SPH], 1/3.0
);

        float  ss  =  Parameter  [  JARAK_PARTIKEL_SPH  ]*0.87  /
Parameter[ SKALA_SIMULASI_SPH ];
        printf ( "Spacing: %f\n", ss);
        AddVolume  ( vektor[INITMIN_SPH], vektor[INITMAKS_SPH], ss
);      // Membuat partikel

        float cell_size = Parameter[SMOOTHRADIUS_SPH]*2.0;
                // Grid cell size (2r) untuk kondisi ideal
        Grid_Setup             (             vektor[VOLUME_MIN_SPH],
vektor[VOLUME_MAKS_SPH], Parameter[SKALA_SIMULASI_SPH], cell_size,
1.0 );
                                // Setup grid
        hitungPosisis ();
                                // Memasukkan Partikel

        Vector3DF vmin, vmax;
        vmin =  vektor[VOLUME_MIN_SPH];
        vmin -= Vector3DF(2,2,2);
        vmax =  vektor[VOLUME_MAKS_SPH];
        vmax += Vector3DF(2,2,-2);
}

// Hitung tekanan menggunakan spatial grid
void SPHKu::sphHitungTekananGrid ()
{
        char *dat1, *dat1_end;
        sph* p;
        sph* pcurr;
        int pndx;
        int i, cnt = 0;
        float dx, dy, dz, sum, dsq, c;
        float d, d2, mR, mR2;
        float     radius     =     Parameter[SMOOTHRADIUS_SPH]     /
Parameter[SKALA_SIMULASI_SPH];
        d = Parameter[SKALA_SIMULASI_SPH];
        d2 = d*d;
        mR = Parameter[SMOOTHRADIUS_SPH];
        mR2 = mR*mR;
        float nat = 2.71828;   //bilangan natural

        dat1_end = mBuf[0].data + NumPoints()*mBuf[0].stride;
        i = 0;
        for  ( dat1  =  mBuf[0].data;  dat1  <  dat1_end;  dat1  +=
mBuf[0].stride, i++ ) {
                p = (sph*) dat1;

                sum = 0.0;
                NC[i] = 0;

                Grid_FindCells ( p->posisi, radius );
```

```
            for (int cell=0; cell < 8; cell++) {
                if ( gridCell[cell] != -1 ) {
                    pndx = grid [ gridCell[cell] ];

                    while ( pndx != -1 ) {

                        pcurr = (sph*) (mBuf[0].data
+ pndx*mBuf[0].stride);
                        if ( pcurr == p ) {pndx =
pcurr->next; continue; }
                        dx = ( p->posisi.x - pcurr-
>posisi.x)*d;          // jarak dalam cm
                        dy = ( p->posisi.y - pcurr-
>posisi.y)*d;
                        dz = ( p->posisi.z - pcurr-
>posisi.z)*d;
                        dsq = (dx*dx + dy*dy +
dz*dz);
                        if ( mR2 > dsq ) {
                            //c = R_kuadrat -
dsq;
                            //sum += c * c * c;

                            //c       =       (1+
3*sqrt(dsq)/mR)* pow((1-sqrt(dsq)/mR), 3);          //bell shaped
                            //sum += c;

                            //gaussian
                            c   =   pow(nat,    -
(dsq/mR2));
                            sum += c;

                            if    (    NC[i]    <
MAX_NEIGHBOR ) {
                                tetangga[i][
NC[i] ] = pndx;
                                NDist[i][
NC[i] ] = sqrt(dsq);
                                NC[i]++;
                            }
                        }
                        pndx = pcurr->next;
                    }
                }
                gridCell[cell] = -1;
            }
            p->kerapatan = sum * Parameter[MASSA_PARTIKEL_SPH]
* Kernel_normal ;
            p->tekanan      =      (      p->kerapatan      -
Parameter[RESTDENSITY_SPH] ) * Parameter[INTERNAL_STIFFNESS_SPH];

            p->kerapatan = 1.0f / p->kerapatan;
        }
}

// Hitung Gaya menggunakan spatial grid with saved neighbor table.
Fastest.
```

```
void SPHKu::sphHitungForceGridNC ()
{
        char *dat1, *dat1_end;
        sph *p;
        sph *pcurr;
        Vector3DF force, fcurr;
        register float pterm, vterm, dterm;
        int i;
        float c, d;
        float dx, dy, dz;
        float mR, mR2, visc;
        float nat = 2.71828;   //bilangan natural

        d = Parameter[SKALA_SIMULASI_SPH];
        mR = Parameter[SMOOTHRADIUS_SPH];
        mR2 = (mR*mR);
        visc = Parameter[VISKOSITAS_SPH];

        dat1_end = mBuf[0].data + NumPoints()*mBuf[0].stride;
        i = 0;

        for ( dat1 = mBuf[0].data; dat1 < dat1_end; dat1 +=
mBuf[0].stride, i++ ) {
                p = (sph*) dat1;

                force.Set ( 0, 0, 0 );
                for (int j=0; j < NC[i]; j++ ) {
                        pcurr    =    (sph*)    (mBuf[0].data    +
tetangga[i][j]*mBuf[0].stride);
                        dx  =  ( p->posisi.x  -  pcurr->posisi.x)*d;
                // jarak dalam cm
                        dy = ( p->posisi.y - pcurr->posisi.y)*d;
                        dz  =  ( p->posisi.z  -  pcurr->posisi.z)*d;

                //Muller
                        //c = ( mR - NDist[i][j] );
                        //pterm = -0.5f * c * Kernel_gradien * ( p-
>tekanan + pcurr->tekanan) / NDist[i][j];
                        //dterm   =   c   *   p->kerapatan   *   pcurr-
>kerapatan;
                        //vterm = Kernel_laplacian * visc;
                        //force.x += ( pterm * dx + vterm * (pcurr-
>evaluasi_kecepatan.x - p->evaluasi_kecepatan.x) ) * dterm;
                        //force.y += ( pterm * dy + vterm * (pcurr-
>evaluasi_kecepatan.y - p->evaluasi_kecepatan.y) ) * dterm;
                        //force.z += ( pterm * dz + vterm * (pcurr-
>evaluasi_kecepatan.z - p->evaluasi_kecepatan.z) ) * dterm;

                //Quartic
                        //c = pow((1 - NDist[i][j]/mR),3) - pow((1-
NDist[i][j]/mR),2) * (1+3*NDist[i][j]/mR);
                        //pterm = -0.5f * c * Kernel_gradien * ( p-
>tekanan + pcurr->tekanan);
                        //vterm  =  Kernel_laplacian  *visc*  (2*(1-
NDist[i][j]/mR)     *     (1+3*NDist[i][j]/mR)     -     6*pow((1-
NDist[i][j]/mR),2));
```

```
                    //force.x += ( pterm * dx + vterm * (pcurr-
>evaluasi_kecepatan.x - p->evaluasi_kecepatan.x) );
                    //force.y += ( pterm * dy + vterm * (pcurr-
>evaluasi_kecepatan.y - p->evaluasi_kecepatan.y) );
                    //force.z += ( pterm * dz + vterm * (pcurr-
>evaluasi_kecepatan.z - p->evaluasi_kecepatan.z) );

            //Gaussian
                    c       =       NDist[i][j]*pow(nat,       -
(NDist[i][j]*NDist[i][j]/mR2));
                    pterm = -0.5f * c * Kernel_gradien * ( p-
>tekanan + pcurr->tekanan);
                    vterm     =     Kernel_laplacian      *visc*
(2*NDist[i][j]*NDist[i][j]*pow(nat,                          -
(NDist[i][j]*NDist[i][j]/mR2))/mR2        -        pow(nat,       -
(NDist[i][j]*NDist[i][j]/mR2)));
                    force.x += ( pterm * dx + vterm * (pcurr-
>evaluasi_kecepatan.x - p->evaluasi_kecepatan.x) );
                    force.y += ( pterm * dy + vterm * (pcurr-
>evaluasi_kecepatan.y - p->evaluasi_kecepatan.y) );
                    force.z += ( pterm * dz + vterm * (pcurr-
>evaluasi_kecepatan.z - p->evaluasi_kecepatan.z) );
            }
            p->force = force;
        }
}
```

UCAPAN TERIMA KASIH

Kepada kedua orang tua dan semua keluarga yang tercinta, terima kasih atas semua yang diberikan kepada saya dengan penuh kasih sayang, semoga Allah membalas dengan kebaikan yang berlipat-lipat.

Rasa terima kasih yang sebesar-besarnya saya sampaikan kepada Bapak Agus Naba dan Bapak Abdurrouf yang telah berkenan meluangkan waktunya untuk memberikan bimbingan selama penulisan skripsi ini. Juga kepada Bapak Sugeng Rianto yang memberikan inspirasi dan dasar-dasar simulasi fisis kepada penulis.

Teristimewa kepada Bapak Khairul Hamdi, MT. yang telah memberikan dan mengijinkan penulis untuk mempelajari *source code* thesis beliau tentang SPH. Kepada Rama Hoeltzein, Dr. Liu dari NUS, Mathias Muller, terima kasih semuanya,Anda semua adalah guru-guru saya walaupun kita tidak pernah bertemu muka, semoga Allah mempertemukan kita nantinya dalam keadaan yang lebih baik.

Juga kepada seorang yang ada di hati saya, semoga Allah mempertemukan kita nantinya dalam keadaan yang baik, walaupun tidak ada kata terucap dari mulut ini karena memang belum waktunya, cukuplah hati yang berbicara.

Kepada Bapak dan Ibu dosen semua, terima kasih atas ilmunya yang berharga. Dan kepada teman-teman, terkhusus angkatan 2005 serta semua kakak dan adik tingkat atas guyonan-guyonannya sehingga suasana belajar menjadi lebih rileks.