Optimisasi *Cloth Simulation* Menggunakan *Parallel Computing* Berbasis GPU pada *Platform* Nvidia CUDA

Darian Gunamardi, Henry Novianus Palit
[1,2] Program Studi Teknik Informatika, Fakultas Teknologi Industri, Universitas Kristen Petra
Jl. Siwalankerto 121-131 Surabaya 60236, Indonesia
Email: dariangunamardi@gmail.com, hnpalit@petra.ac.id

ABSTRAK

Kemajuan di bidang sains dan teknologi membawa berbagai masalah dan tantangan baru. Salah satunya adalah kebutuhan untuk mensimulasikan properti dari suatu objek atau situasi, dimana objek atau situasi ini sulit dan bahkan tidak mungkin untuk diperoleh di dunia nyata. *Interactive cloth simulation* adalah salah satu bidang ilmu yang membutuhkan simulasi dilakukan secara *real-time*. Simulasi yang realistis membutuhkan beban komputasi yang tinggi, demikian mengakibatkan simulasi tidak dapat berjalan secara *real-time*. Metode optimisasi yang secara murni menggunakan algortima dan matematika saja cenderung mencari keseimbangan antara akurasi dan kecepatan. Penelitian ini menggunakan metode optimisasi *parallel computing* pada GPU untuk mempercepat simulasi tanpa mengurangi akurasi. Simulasi yang dilakukan berbasis *continuum mechanics* karena kemampuannya untuk menunjukkan properti fisik yang nyata dari kain. Dinamika model didiskritisasi menggunakan FEM, kemudian elastitisas kain akan dihitung dengan *corotational formulation*. Penelitian ini menggunakan CUDA *platform* yang dikembangkan oleh Nvidia untuk melakukan parallelisasi. Hasil pengujian menunjukkan adanya perubahan akurasi pada taraf yang dapat diterima. Peningkatan kecepatan terjadi ketika struktur dari suatu fungsi ideal untuk parallelisasi pada GPU. Parallelisasi secara keseluruhan dapat mengakibatkan *overhead* yang besar pada fungsi yang tidak cocok untuk diparallelisasi secara lansung. Memilih fungsi yang sesuai atau memodifikasi algoritma dari fungsi tersebut untuk memenuhi kondisi parallel yang ideal akan meningkatkan kecepatan program secara signifikan.

Kata Kunci: *Interactive Cloth Simulation, Continuum Mechanics, Finite Element Method, Corotational Formulation, Parallel Computing, Compute Unified Device Architecture, Graphics Processing Unit.*

ABSTRACT

The advancement in the field of science and technology brings forth new problems and challenges. One such problem is the need to simulate properties of objects or situations due to the real-world conditions being either costly or impossible to achieve. Interactive cloth simulation is one of the fields which requires a simulation to be run in real-time. A realistic simulation requires a heavy computational workload, and as such will hinder the purpose of running the simulation in real-time. Optimization using pure algorithm and math are likely to find balance between accuracy and speed. This research introduces an optimization using parallel computing on GPU to speed up the simulation without reducing the accuracy. The simulation is based on continuum mechanics due to its ability to display realistic physical properties of cloth. The model dynamics is discretized using FEM, then a corotational formulation is used to compute the elasticity property of cloth. This research uses CUDA platform developed by Nvidia to deliver the parallelization. The testing results shows that there are little but acceptable accuracy differences on the parallel version of the simulation. The speedup happens when the structure of a function is ideal for GPU parallelism. A full GPU parallelism causes massive overheads in functions that are not or at least directly parallel compatible. Choosing functions that are compatible or modifying ones to fit the ideal parallel condition will make significant speedup.

Keywords: *Interactive Cloth Simulation, Continuum Mechanics, Finite Element Method, Corotational Formulation, Parallel Computing, Compute Unified Device Architecture, Graphics Processing Unit.*

1. PENDAHULUAN

Interactive cloth simulation adalah salah satu bidang penelitian dari *computer graphics* yang bertujuan untuk membuat simulasi kain yang realistis secara *real-time*. Bidang ini banyak dimanfaatkan oleh berbagai industri untuk mengurangi biaya dan resiko yang harus diambil apabila membuat purwarupa fisik. Kendala yang terjadi dalam pembuatan simulasi adalah simulasi yang realistis akan mengakibatkan komputasi yang berat sehingga kecepatan yang *real-time* sulit didapatkan. Optimisasi yang berorientasi pada algoritma dan matematika murni memiliki kecenderungan untuk mencari keseimbangan antara akurasi dan kecepatan, sedangkan pada beberapa kasus dua hal tersebut tidak dapat dikorbankan.

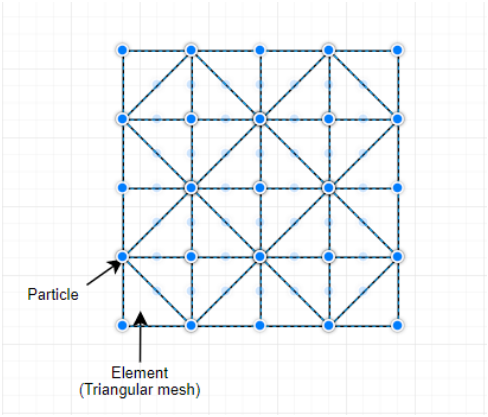
Parallel Computing adalah optimisasi berorientasi *hardware* yang dapat menanggulangi pertukaran antara kecepatan dan akurasi dengan meningkatkan kekuatan komputasi. Metode ini mengubah pemrosesan program yang biasanya berjalan secara sekuensial menjadi parallel. *Hardware* yang akan digunakan untuk *parallel computing* adalah GPU menggunakan CUDA *platform* yang dikembangkan oleh Nvidia. Alasan pemilihan GPU ketimbang CPU adalah karena GPU didesain untuk melakukan komputasi secara massif pada waktu yang bersamaan, sehingga lebih sesuai untuk menyelsaikan masalah yang membutuhkan kekuatan komputasi mentah.

Model kain yang akan digunakan dalam simulasi akan berbasis *continuum mechanics* [1]. *Continuum mechanics* dapat mewujudkan properti fisik kain sehingga sesuai untuk membuat simulasi yang realistis. *Finite Element Method* (FEM) digunakan untuk mendiskritisasi dinamika model. Untuk membuat sifat elastisitas pada kain digunakan *corotational formulation*.

2. LANDASAN TEORI

2.1 Cloth Model

Cloth model yang digunakan adalah *triangular grid* dimana tiap titik disebut partikel dan tiap *face* disebut elemen. Bentuk *grid* dapat dilihat pada Gambar 1.



Gambar 1 ***Triangular grid*.**

2.2 Hybrid Voronoi Mass

Metode ini mengasumsikan bahwa massa dari suatu partikel dinamis sebanding dengan luas segitiga yang mengelilingi partikel tersebut. Untuk suatu segitiga lancip t, dengan vertex penyusun $\mathbf{x}_i$, $\mathbf{x}_j$, dan $\mathbf{x}_k$, *voronoi area* dari suatu vertex dapat dihitung dengan:

$$A_{Voronoi}(t, \boldsymbol{x}_i) = \frac{1}{8}\left(\|\boldsymbol{x}_k - \boldsymbol{x}_i\|^2 \cot\varphi_j + \|\boldsymbol{x}_j - \boldsymbol{x}_i\|^2 \cot\varphi_k\right) \quad (1)$$

Dimana φ_i melambangkan sudut dalam segitiga pada *vertex* x_i.

Algorithm 1.

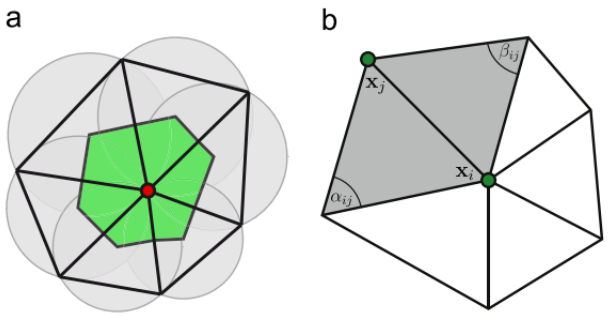
$A_{\text{hybrid}}(\mathbf{x}) = 0$
for all triangles t in the one-ring of $\mathbf{x}$ **do**
  if If t is not obtuse **then**
    $A_{\text{hybrid}}(\mathbf{x}) += A_{\text{Voronoi}}(t, \mathbf{x})$
  else
    Compute area A_t of triangle t
    if If angle at $\mathbf{x}$ in t is obtuse **then**
      $A_{\text{hybrid}}(\mathbf{x}) += \frac{1}{2} A_t$
    else
      $A_{\text{hybrid}}(\mathbf{x}) += \frac{1}{4} A_t$

Gambar 2 Algoritma *Hybrid Voronoi Mass*

Segitiga tumpul dihitung dengan cara yang berbeda. Luas yang dihitung pada sudut lancip dari segitiga tumpul akan menjadi negative jika luas total dari semua *vertex* sama dengan luas segitiga. Solusi dari masalah ini adalah metode *hybrid* yang digunakan oleh Meyer et. al [5].

2.3 Linear Stiffness

Dengan menggunakan *finite element method*, simulasi *stretch* dan *shear* dari *particle model* dapat dilakukan pada tiap elemen, sehingga hanya perlu menggunakan ruang dua dimensi dari *triangular mesh*. Demikian, deformasi pada model dapat digambarkan menggunakan *vector field* dua dimensi **u(m)** yang digunakan untuk menghitung perubahan posisi **x(m) = m + u(m)** dari sebuah *vertex* $\mathbf{m} \in R^2$.



Gambar 3 (a) menunjukkan area yang dimiliki oleh vertex merah. (b) menunjukkan titik fokus saat menghitung voronoi area dari titik xi.

Untuk melakukan simulasi berbasis *finite element*, **u(m)** hanya dievaluasi pada tiap *vertex* dari pada *triangular mesh*. Deformasi pada interior elemen diinterpolasi menggunakan tiga *shape function* $N_i(\mathbf{m})$:

$$\boldsymbol{u}(\boldsymbol{m}) = \sum_{i=1}^{3} N_i(\boldsymbol{m}) \cdot \hat{\boldsymbol{u}}_i \quad (2)$$

Vector $\hat{\mathbf{u}}_i \in R^2$ merupakan selisih perubahan polisi dari *vertex-vertex* pada elemen.

Simulasi ini menggunakan Cauchy's *linear strain tensor*:

$$\begin{pmatrix}\epsilon_x \\ \epsilon_y \\ \gamma_{xy}\end{pmatrix} = \sum_{i=1}^{3} \mathbf{B}_i \hat{\mathbf{u}}_i \quad with \quad \mathbf{B}_i = \begin{pmatrix} \frac{\partial N_i}{\partial x} & 0 \\ 0 & \frac{\partial N_i}{\partial y} \\ \frac{1}{2}\frac{\partial N_i}{\partial y} & \frac{1}{2}\frac{\partial N_i}{\partial x} \end{pmatrix} \quad (3)$$

Dimana Ni/x didapatkan dari:

$$\frac{\partial \mathbf{N}_e}{\partial \mathbf{m}} = \frac{1}{2A_e}\begin{pmatrix} y_2 - y_3 & x_3 - x_2 \\ y_3 - y_1 & x_1 - x_3 \\ y_1 - y_2 & x_2 - x_1 \end{pmatrix} \quad (4)$$

Dimana $\mathbf{N}_e = (N_1, N_2, N_3)^T$.

Berdasarkan hukum Hooke, *stress tensor* yang digunakan adalah:

$$\sigma = \mathbf{C}\varepsilon = \mathbf{C}\mathbf{B}_e\hat{\mathbf{u}} \quad (5)$$

Dimana $\mathbf{B}_e = (\mathbf{B}_1, \mathbf{B}_2, \mathbf{B}_3)$ merupakan matriks 3x6 dan $\hat{\mathbf{u}} = (\hat{\mathbf{u}}_1{}^T, \hat{\mathbf{u}}_2{}^T, \hat{\mathbf{u}}_3{}^T)^T$

C adalah *tensor* yang mendefinisikan elastisitas material. *Stress* pada kain dibagi menjadi *weft* yang horizontal dan *warp* yang vertikal. Demikian, *tensor* ini berisi E_x dan E_y untuk *weft* dan *warp*, dengan *shear modulus* E_s dan rasio Poisson v_{xy} dan v_{yx}. Rasio Poisson mewakili kontradiksi pada arah y ketika material mengalami *stretch* pada arah *x*.

$$\mathbf{C} = \begin{pmatrix} \frac{E_x}{1 - v_{xy}v_{yx}} & \frac{E_x v_{yx}}{1 - v_{xy}v_{yx}} & 0 \\ \frac{E_y v_{xy}}{1 - v_{xy}v_{yx}} & \frac{E_y}{1 - v_{xy}v_{yx}} & 0 \\ 0 & 0 & E_s \end{pmatrix} \quad (6)$$

Menggunakan komponen-komponen diatas, gaya yang bekerja pada tiga *vertex* elemen dapat dikalkulasi dengan persamaan:

$$\mathbf{f}_e = A_e \mathbf{B}_e^T \mathbf{C}\mathbf{B}_e\hat{\mathbf{u}} = \mathbf{K}_e\hat{\mathbf{u}} \quad (7)$$

Dimana A_e adalah luas awal dari elemen e, $\mathbf{K}_e \in R^{6x6}$ adalah matriks *linear stiffness* dari elemen dan *vector* $\mathbf{f}_e \in R^6$ berisi gaya yang bekerja pada ketiga *vertex*.

2.4 Corotational Formulation

Stiffness model menggunakan *linear elasticity* tidak cocok untuk digunakan pada deformasi dengan perubahan rotasi yang tinggi. Etzmuss et. al [4] dan Thomazowski et/ al [8] menggunakan *corotational formulation* untuk menyelesaikan masalah ini.

Misalkan suatu elemen dengan *vertex a, b, c,* dan normal bidang **n** = ($\mathbf{x}^b$-$\mathbf{x}^a$) x ($\mathbf{x}^c$-$\mathbf{x}^a$) dapat ditentukan *vector* bidang yang dimiliki.

$$\mathbf{P}_x = \frac{x^b - x^a}{\|x^b - x^a\|}, \mathbf{P}_y = \frac{n \times \mathbf{P}_x}{\|n \times \mathbf{P}_x\|} \tag{8}$$

Dua *vector* ini kemudian digunakan untuk menentukan matriks proyeksi terhadap bidang tersebut.

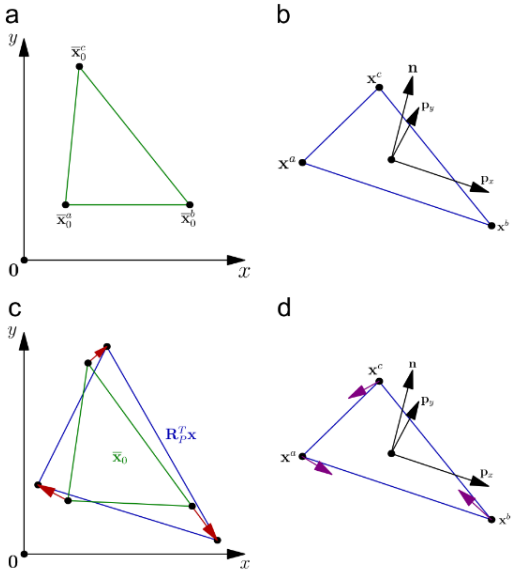
$$\mathbf{P} = \begin{pmatrix} \mathbf{P}_x^T \\ \mathbf{p}_y^T \end{pmatrix} \in R^{2x3} \tag{9}$$

Kedua elemen kemudian diproyeksikan dengan menggunakan matrix tersebut untuk mendapatkan koordinat dari *vertex* kedua elemen dalam dua dimensi $\bar{\mathbf{x}} = \mathbf{Px}$. Matrix deformasi didapatkan melalui $\mathbf{TS}^{-1}$ dimana:

$$\mathbf{S} = \left(\bar{\mathbf{x}}_0^b - \bar{\mathbf{x}}_0^a, \bar{\mathbf{x}}_0^c - \bar{\mathbf{x}}_0^a\right) \tag{10}$$

$$\mathbf{T} = \left(\bar{\mathbf{x}}^b - \bar{\mathbf{x}}^a, \bar{\mathbf{x}}^c - \bar{\mathbf{x}}^a\right) \tag{11}$$

Matrix deformasi yang dihasilkan mengandung transformasi dari $\bar{\mathbf{x}}_0$ ke $\bar{\mathbf{x}}$ tanpa translasi. Demikian, matriks rotasi $\mathbf{R} \in R^{2x2}$ yang dibutuhkan dapat diekstrak menggunakan *polar decomposition* 2 dimensi [6].



Gambar 4 (a) elemen0 (x0) diproyeksikan ke bidang dua dimensi dari elemen tersebut. (b) menunjukkan elemen yang telah terdeformasi (x). (c) elemen yang telah terdeformasi diproyeksikan ke bidang dua dimensi yang sama, kemudian diekstrak matriks deformasi rotasi.

Pada *corotational formulation*, *vertex* ditransformasikan terlebih dahulu ke bidang koordinat lokal sepert pada Gambar 4(c). Lalu gaya *linier* dikalkulasi dan hasilnya ditransformasikan kembali ke koordinat global Gambar 4(d). Transformasi ini dapat dikombinasikan dengan matriks proyeksi **P** untuk memproyeksikan *vertex* dari tiga dimensi ke dua dimensi pada posisi elemen yang berkaitan. $\mathbf{P}^T$ digunakan untuk memproyeksikan gaya pada dua dimensi kembali ke tiga dimensi. Matrix $\mathbf{R}_{P,e}$ mengkombinasikan proyeksi dan rotasi dari ketiga *vertex* pada suatu elemen.

$$\mathbf{R}_{P,e} = \begin{pmatrix} \mathbf{P}^T\mathbf{R} & 0 & 0 \\ 0 & \mathbf{P}^T\mathbf{R} & 0 \\ 0 & 0 & \mathbf{P}^T\mathbf{R} \end{pmatrix} \in R^{9\times6} \tag{12}$$

Untuk menentukan *corotated stiffness* dilakukan persamaan berikut:

$$\mathbf{K}_e^R = \mathbf{R}_{P,e}\mathbf{K}_e\mathbf{R}_{P,e}^T, \quad \tilde{\mathbf{K}}_e^R = \mathbf{R}_{P,e}\mathbf{K}_e \tag{13}$$

Menggunakan dua definisi ini, gaya yang terjadi pada ketiga *vertex* ditentukan dengan:

$$\mathbf{f}_e = \mathbf{K}_e^R\mathbf{x} + \mathbf{f}_{0,e} \quad with \quad \mathbf{f}_{0,e} = -\tilde{\mathbf{K}}_e^R\bar{\mathbf{x}}_0 \tag{14}$$

Dimana $\bar{\mathbf{x}}_0$ mengandung *vertex* dari elemen yang belum deformasi dalam dua dimensi.

2.5 Isometric Bending Model

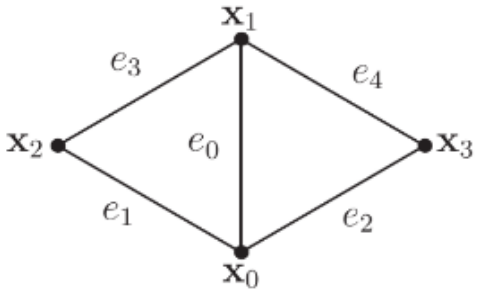
Terjadinya fenomena lipatan dan lekukan dipengaruhi oleh *bending* properti pada kain. Untuk mereproduksi properti ini pada kain digunakan metode *isometric bending model* dari Bergou et. al [2] seperti pada Gambar 5. *Model* ini memiliki keuntungan dimana *bending* menjadi fungsi kuadratik dalam posisi, sehingga Hessian **Q** hanyalah sebuah matriks yang konstan. Untuk setiap *interior edge* e_i, Hessian $\mathbf{Q}(e_i) \in R^{4x4}$ yang merupakan *local bending* dikalkulasi dari elemen t_0 dan t_1 yang berseberangan melalui e_i. $\mathbf{Q}(e_o)$ didapatkan melalui:

$$\mathbf{Q}(e_0) = \frac{3}{A_0 + A_1}\mathbf{k}_0^T\mathbf{k}_0 \tag{15}$$

Dimana A_i adalah luas dari elemen ke-i dan $\mathbf{k}_0$ adalah *row vector*:

$$\mathbf{k}_0 = (C_{03} + C_{04}, C_{01} + C_{02}, -C_{01} - C_{03}, -C_{02} - C_{04}) \tag{16}$$

Dimana $c_{jk} = \cot(e_j, e_k)$.



Gambar 5 *Diamond stencil* yang digunakan untuk menghitung *local bending* dimana e0 adalah edge yang sedang dihitung *local bending*-nya.

Kumpulan matriks *local bending* $Q(e_i)$ kemudian disusun menjadi satu matriks *global bending* $\mathbf{Q} \in R^{nxn}$ dimana n adalah jumlah partikel.

2.6 Rayleigh Damping

Damping digunakan untuk mensimulasikan kelekatan atau kekentalan kain. *Rayleigh damping* terdiri dari dua bagian, yaitu

damping pada massa dan *damping* pada *stiffness*. Matriks *damping* dari sebuah elemen dengan matriks massa $\mathbf{M}_e$ dan matriks *stiffness* $\mathbf{K}_e$ ditentukan melalui:

$$\mathbf{D}_e = \alpha \mathbf{M}_e + \beta \mathbf{K}_e \quad (17)$$

Dimana α adalah koefisien *damping* untuk massa dan β adalah koefisien *damping* untuk *stiffness.*

2.7 Time Integration

Integrasi waktu menggunakan metode *implicit euler* dengan metode *conjugate gradient* untuk menyelesaikan persamaan *linier* akhir. Berdasarkan penelitian oleh Baraff dan Witkin [1] integrasi waktu menggunakan metode ini memungkinkan simulasi yang stabil untuk *model* dengan *stiffness* yang tinggi walaupun pada iterasi waktu yang besar. Untuk melakukan integrasi waktu, perubahan kecepatan pada partikel ditentukan dengan menyelesaikan persamaan *linier* berikut:

$$(\mathbf{M} + h\mathbf{D} + \Delta t^2 \mathbf{K})\Delta \mathbf{v} = -\Delta t\,(\mathbf{K}\mathbf{x} + \mathbf{f}_0 - \mathbf{f}_{ext} + \Delta t \mathbf{K}\mathbf{v} + \mathbf{D}\mathbf{v}) \quad (18)$$

Dimana Δt adalah selisih waktu per iterasi dan *vector* $\mathbf{f}_{ext}$, **x**, dan **v** masing-masing merupakan gaya, posisi, dan kecepatan dari setiap partikel. Matriks **K** didapatkan dari menyusun matriks *corotated stiffness* $\mathbf{K}_e^R$ dan menambahkan Hessian **Q** dari *bending model*. $\mathbf{f}_0$ didapatkan dengan menyusun *vector* $\mathbf{f}_{0,e}$. Setelah menyelesaikan persamaan tersebut perubahan posisi dapat ditentukan dengan $\Delta\mathbf{x} = \Delta t(\mathbf{v} + \Delta\mathbf{v})$.

2.8 Preconditioned Conjugate Gradient

Preconditioned Conjugate Gradient (PCG) merupakan salah satu metode *iterative* untuk menyelesaikan suatu sistem persamaan linier AΔv=b, dimana input yang dibutuhkan adalah *symmetric positive semi-definite matrix* A, sebuah *symmetric positive definite preconditioning matrix* P dengan dimensi yang sama dengan A, dan *vector* b. Matriks P harus mudah dan cepat untuk diinvers. Iterasi dari metode ini berhenti saat $\| b - A\Delta v \|$ lebih kecil dari $\epsilon \| b \|$ dimana ϵ adalah faktor toleransi yang didefinisikan oleh *user*. Algoritma dari metode ini dapat dilihat pada Gambar 6 [1].

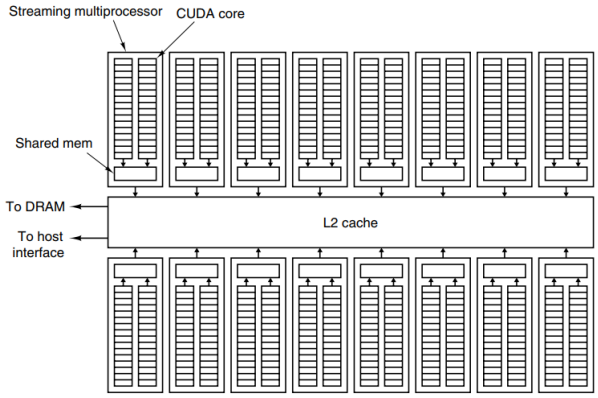
```
1   procedure pcg
2   Δv = z
3   δ0 = b^T P b
4   r = b – AΔv
5   c = P^-1 r
6   δnew = r^T c
7   while δnew > ϵ^2 δ0
8       q = Ac
9       α = δnew / (c^T q)
10      Δv = Δv + αc
11      r = r – αq
12      s = P^-1 r
13      δold = δnew
14      δnew = r^T s
15      c = s + (δnew / δold) c
```

Gambar 6 Algoritma PCG.

2.9 Nvidia CUDA Platform

CUDA (Compute Unified Device Architecture) adalah sebutan dari SIMD processor, platform, dan model pemrograman parallel computing yang dikembangkan oleh NVIDIA untuk melakukan General Purpose computing on Graphics Processing Unit (GPGPU).

GPU mengandalkan SIMD processing untuk memberikan kekuatan komputasi yang besar. SIMD atau *Single Instruction-stream Multiple Data-stream processor* tersusun dari *processor* dalam jumlah besar yang melakukan instruksi yang sama pada set data yang berbeda. Gambar 7 menunjukkan struktur dari SIMD *processor* pada GPU Nvidia dengan arsitektur Fermi. [7]



Gambar 7 Fermi GPU *Architecture.*

Thread Block adalah abstraksi pemrograman yang menjelaskan satuan yang digunakan untuk melakukan pemrosesan pada GPU, yaitu *grid*, *block*, dan *thread*. Untuk dapat menjalankan suatu proses di GPU, CPU akan memanggil sebuah instruksi yang disebut *kernel call*. Sebuah *kernel call* dapat diberikan instruksi mengenai berapa jumlah *thread* yang diinginkan untuk mengeksekusi *kernel* tersebut secara bersamaan. Jumlah *thread* yang dapat dipanggil pada tiap *block* terbatas dengan limitasi sesuai arsitektur GPU yang digunakan. Jumlah *block* pada tiap *grid* tidak terbatas, tetapi jumlah *block* yang dapat dijalankan secara bersamaan bergantung pada jumlah SM yang dimiliki oleh GPU. [3]

3. ANALISA DAN DESAIN SISTEM

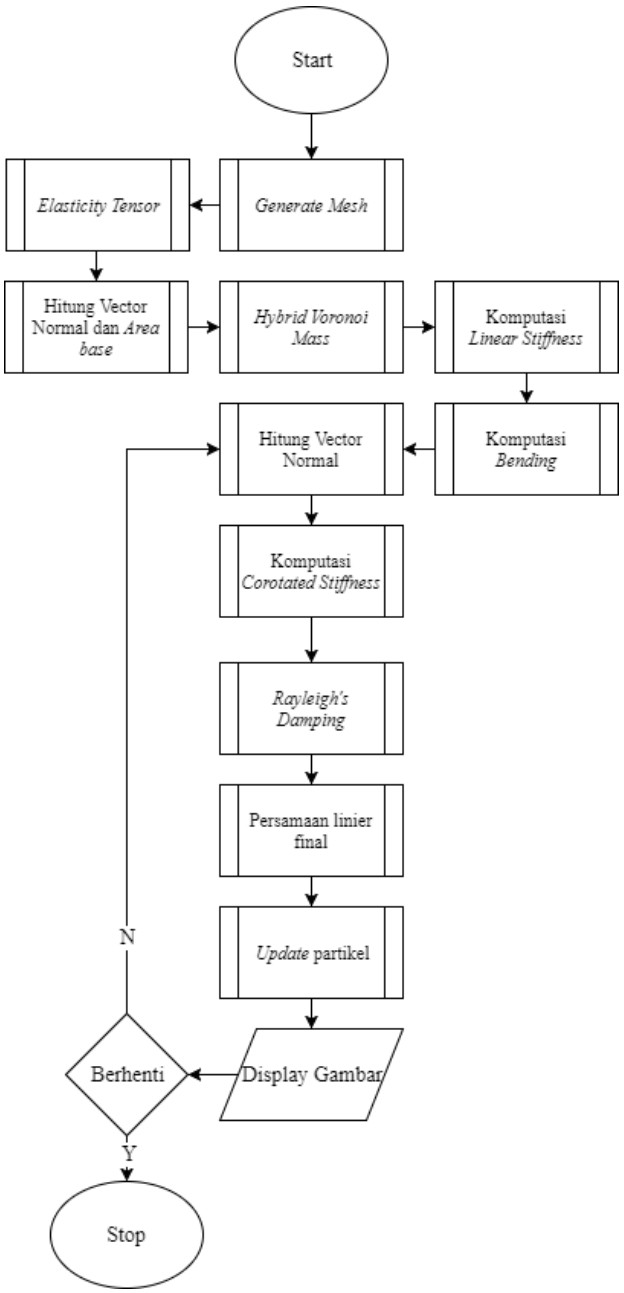
3.1 Garis Besar Kerja Sistem

Sistem terbagi menjadi dua bagian yaitu *precomputation* dan *iteration*. Proses yang terjadi pada bagian *precomputation* hanya dilakukan sekali. Sistem menyimpan dua data dari kain, yaitu data dari *base* atau model kain pada posisi 0 ($\mathbf{x}_0$), dan pada posisi terakhir setelah diberi gaya (**x**). Data dari $\mathbf{x}_0$ dan variabel yang berkaitan hanya perlu dihitung sekali sehingga dilakukan pada tahap *precomputation*. Proses yang dilakukan pada tahap *precomputation* meliputi *generate mesh* diikuti penghitungan *elasticity tensor*, *vector normal* dan *area* $\mathbf{x}_0$, *hybrid voronoi mass*, *bending*, dan *linear stiffness*. Sedangkan proses yang dilakukan pada tiap *iteration* adalah penghitungan *vector normal* **x**, *corotated stiffness*, *damping*, *final equation* dan *update* partikel. *Flowchart* garis besar sistem dapat dilihat pada Gambar 8.

4. IMPLEMENTASI

Penelitian ini menggunakan Microsoft Visual Studio 2015 sebagai *Integrated Development Environment* (IDE) dengan bahasa pemrograman C++. C++ adalah salah satu bahasa pemrograman yang didukung oleh CUDA *platform* untuk secara *native* melakukan *kernel call* sehingga *developer* dapat memparalelisasi proses yang diinginkan hingga tingkat dasar, sehingga sesuai untuk

digunakan pada penelitian ini. *Library* yang digunakan meliputi CUDA *runtime* untuk paralelisasi dan OpenGL untuk membuat tampilan simulasi. Selain itu, *plugin visual leak detector* (VLD) juga digunakan untuk mendeteksi *memory leak* pada simulasi.



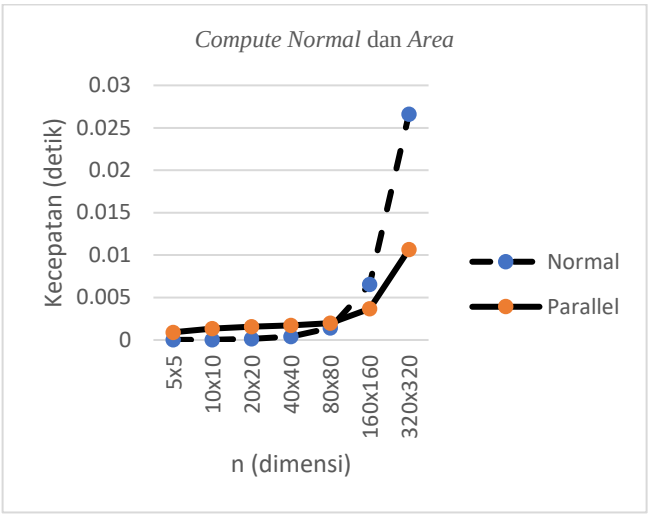
Gambar 8 *Flowchart* garis besar kerja sistem.

5. PENGUJIAN

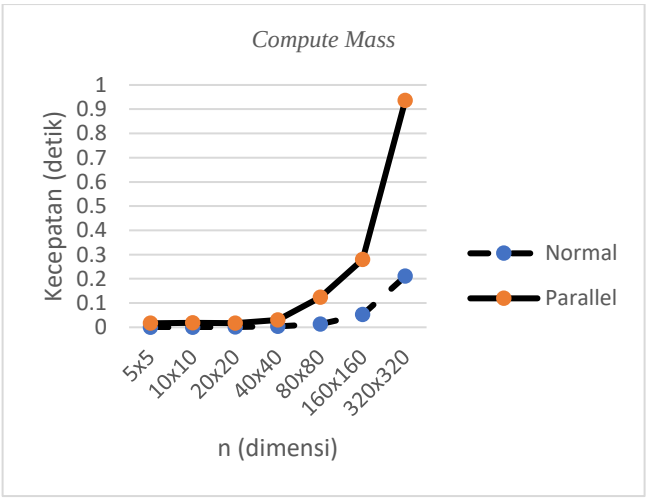
Pengujian dilakukan pada akurasi dan kecepatan dari versi *parallel* dibandingkan dengan versi *serial*. Spesifikasi mesin yang digunakan adalah Intel i3-3120, Nvidia GTX670, dengan RAM 12GB.

Perbandingan kecepatan pada Gambar 9 menunjukkan kecepatan yang cenderung landai pada versi *parallel* dibandingkan versi *serial*. Sedangkan Gambar 10 menunjukkan performa yang buruk dari versi *parallel*.

Tabel 1 menunjukkan perbedaan akurasi antara *serial* dan *parallel*.



Gambar 9 Grafik fungsi *Compute Normal* dan *Area*.



Gambar 10 Grafik fungsi *Compute Mass*.

6. KESIMPULAN

Dari pengujian yang dilakukan, dapat disimpulkan bahwa:

- Pengujian kecepatan diatas menunjukkan contoh operasi yang ideal dan tidak untuk parallelisasi GPU. Secara teori, komputasi dengan menggunakan GPU akan menghasilkan kurva kecepatan yang relatif landai, seperti Gambar 9 pada fungsi *Compute Normal* dan *Area*. Operasi tersebut adalah contoh kondisi yang ideal untuk parallelisasi dengan GPU. Sebaliknya, bentuk operasi dari fungsi *Compute Mass* kurang sesuai untuk komputasi dengan GPU sehingga menghasilkan kurva yang tajam seperti pada Gambar 10.
- Fungsi *Compute Mass* kurang ideal untuk komputasi dengan GPU karena terdapat lebih dari satu proses yang mengakses suatu memori. Untuk menanggulangi konflik *read-write* maka dilakukan operasi yang disebut *atomic operation* dari CUDA *runtime* yang menyebabkan akses terhadap suatu lokasi memori menjadi serial. Bentuk operasi seperti ini tidak

ideal untuk GPU karena menghilangkan kemampuan GPU untuk memproses secara parallel.

Berdasarkan pengujian perbandingan akurasi pada

Tabel 1, terdapat perbedaan dengan rentang 0.000001-0.000002.

Tabel 1 Perbandingan Akurasi pada Iterasi 2.

Index Partikel	Serial	Parallel
0	0.000000, 0.00000, 0.000000	0.000000, 0.00000, 0.000000
1	0.010610, -0.049168, 0.000000	0.010610, -0.049168, 0.000000
2	0.013363, -0.084712, 0.000000	0.013363, *-0.084714,* 0.000000
3	0.000536, -0.032894, 0.000000	*0.000534,* -0.032894, 0.000000
4	-0.006464, -0.044784, 0.000000	*-0.006466,* *-0.044783,* 0.000000
5	-0.004746, -0.078572, 0.000000	*-0.004748,* *-0.078573,* 0.000000
6	-0.009589, -0.049699, 0.000000	*-0.009590,* *-0.049700,* 0.000000
7	-0.013412, -0.052369, 0.000000	*-0.013413,* *-0.052368,* 0.000000
8	-0.017616, -0.082670, 0.000000	-0.017616, -0.082670, 0.000000

7. REFERENSI

[1] Baraff, D. and Witkin, A. 1998. Large Steps in Cloth Simulation. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*. ACM, New York, NY, 43-54. DOI= http://doi.acm.org/10.1145/280814.280821

[2] Bergou, M., Wardetzky, M., Harmon, D., Zorin, D., and Grinspun, E. 2006. A Quadratic Bending Model for Inextensible Surfaces. In *Proceedings of the fourth Eurographics symposium on Geometry processing*. ACM, Cagliari, 227-230.

[3] Etzmuß, O., Keckiesen, M., and Straßer, W. 2003. A fast finite element solution for cloth modelling. In *Proceedings of the 11th Pacific Conference on Computer Graphics and Applications*. IEEE, Canmore, 244-251. DOI= https//doi.acm.org/10.1109/PCCGA.2003.1238266

[4] Meyer, M., Desbrun, M., Schröder, P., & Barr, A. H. 2003. Discrete Differential-Geometry Oprators for Triangulated 2-Manifolds. In *Visualization and Mathematics III*. Springer, Berlin, 35-37.

[5] Shoemake, K. and Duff, T. 1992. Matrix Animation and Polar Decomposition. In *Proceedings of the conference on Graphics Interface* (Vancouver, Canada, May 11 – 15, 1992). Canadian Information Processing Society. Vancouver, 258-264.

[6] Tanenbaum, A. S., and Austin, T. 2013. *Structured Computer Organization.* Prentice Hall, New Jersey.

[7] Thomaszewski, B., Pabst, S., and Straßer, W. 2009. Continuum-based Strain Limiting. *Computer Graphics Forum*, 569-76. DOI= https//doi.acm.org/10.1111/j.1467-8659.2009.01397.x

[8] Urra, R., and Romero, M. t.t. *CUDA Programming Information and Resources*. URl= http://cuda.ce.rit.edu/cuda_overview/cuda_overview.htm