Penggunaan Algoritma *Divide and Conquer* Dalam *Parallel Computing* Untuk Melakukan *3D Rendering*

Giovanni Sakti Nugraha[1], Odit Ekwardo[2], Fata Mukhlish[3]

Laboratorium Ilmu dan Rekayasa Komputasi
Departemen Teknik Informatika, Institut Teknologi Bandung
Jl. Ganesha 10, Bandung

E-mail : if14096@students.if.itb.ac.id[1], if14079@students.if.itb.ac.id[2], if14084@students.if.itb.ac.id[3]

Abstrak

3D Rendering merupakan salah satu proses yang sangat penting dalam melakukan pengolahan gambar 3D. Tanpa di*render* suatu gambar yang diolah oleh perangkat lunak animasi 3D hanya akan tampil dalam bentuk kumpulan point dan *wireframe* sederhana. Proses *render* melakukan "pembungkusan" tekstur pada objek yang bersesuaian sesuai cahaya yang datang pada objek tersebut. Namun proses *render* membutuhkan daya komputasi yang sangat besar karena banyaknya titik koordinat yang harus dikomputasi, terutama jika data 3D yang diolah cukup rumit. Salah satu cara untuk memecahkan masalah tersebut adalah dengan menggunakan algoritma *Divide and Conquer* yang diterapkan kedalam metode *Parallel Computing. Divide and Conquer* merupakan salah satu strategi algoritma yang memecah suatu masalah besar menjadi beberapa bagian untuk kemudian dikerjakan satu persatu. Dalam *Parallel Computing* tiap-tiap bagian dikerjakan oleh unit pemrosesannya masing-masing, sesuai dengan kesepakatan *Divide* pada awal komputasi. *Parallel Computing* terbukti jauh lebih efektif untuk melakukan *rendering* objek 3D dibanding hanya menggunakan sebuah unit komputasi. Sebagai contoh suatu perusahaan animasi asal Jepang, membutuhkan waktu 165 tahun jika proses *render* yang dilakukan untuk membuat animasi berdurasi 100 menit hanya menggunakan sebuah unit komputasi, namun karena perusahaan tersebut menggunakan metode *Parallel Computing* proses tersebut hanya membutuhkan waktu 1 tahun saja.

***Kata kunci**: divide and conquer, parallel computing, 3D rendering*

1. Pendahuluan

Salah satu masalah kompleks yang hingga kini masih membutuhkan kemampuan komputasi yang besar adalah melakukan *proses render* terhadap objek 3D. *Proses render* objek 3D sendiri membutuhkan waktu yang cukup lama, terlebih jika objek yang diberikan cukup rumit. Salah satu cara yang digunakan untuk mempercepat penyelesaian *proses render* adalah dengan menggunakan teknik *Parallel Computing*. *Parallel Computing* merupakan teknik menggabungkan beberapa unit komputasi sekaligus untuk mengerjakan proses yang telah dibagi-bagi secara bersamaan. Ide untuk *Parallel Computing* muncul dari suatu algoritma yang cukup dikenal di dunia informatika, yaitu algoritma *Divide and Conquer*. Algoritma *Divide and Conquer* merupakan teknik pemecahan masalah dengan membagi suatu masalah menjadi beberapa bagian baru kemudian bagian-bagian tersebut dikerjakan satu persatu[5].

Sebenarnya operasi *render* objek 3D hanya merupakan kumpulan dari beberapa operasi primitif, namun operasi primitif yang dilakukan pada proses *render* sangatlah banyak. Untuk itu digunakan metode *Parallel Computing* sehingga tiap-tiap operasi primitif yang dilakukan dapat dikerjakan dengan menggunakan algoritma *Divide and Conquer* agar tiap bagian dari operasi *Divide* pada perhitungan yang dilakukan dapat dikerjakan oleh masing-masing unit komputasi. Beberapa operasi primitif yang digunakan dalam *3D Rendering* adalah operasi penghitungan jarak antara 2 titik, operasi perkalian bilangan bulat yang besar. Tentunya hampir kesemua operasi primitif tersebut dapat dipecahkan dengan menggunakan algoritma *Divide and Conquer* sehingga menjadikan proses *3D Rendering* sangat cocok untuk diselesaikan dengan algoritma *Divide and Conquer*.

2. *3D Rendering*

2.1 Pengertian *3D Rendering*

3D Rendering merupakan proses untuk membentuk sebuah gambar dari sebuah model yang dibentuk oleh perangkat lunak animasi, model tersebut berisi data geometri, titik pandang, tekstur dan cahaya yang diperlukan untuk membuat gambar yang utuh[1].

3D Rendering merupakan proses yang sangat penting dan telah digunakan untuk berbagai macam penggunaan, seperti program permainan komputer, efek spesial pada film dan program simulasi.

2.2 *3D Rendering Equation*

Dalam melakukan proses render ada rumus utama yang merupakan prinsip dasar dari proses tersebut dan pasti digunakan dalam proses. Secara mayoritas, seluruh proses *render* menggunakan rumus tersebut atau merupakan turunan dari rumus tersebut. Rumus tersebut adalah :

$$Lo(x,\vec{w}) = Le(x,\vec{w}) + \int_{\Omega} fr(x,\vec{w}',\vec{w})Li(x,\vec{w}')(\vec{w}'.\vec{n})d\vec{w}' \quad ...(1)$$

Pada suatu posisi & titik tertentu, cahaya (Lo) yang keluar merupakan jumlah dari cahaya yang memancar dan cahaya yang terefleksikan. Cahaya yang terefleksikan merupakan jumlah dari seluruh cahaya yang datang dari seluruh arah dikalikan dengan refleksi permukaan dan sudut datang. Kesimpulannya persamaan ini akan menghitung seluruh pergerakan cahaya dalam sebuah proses[1].

2.3 Hasil Proses *3D Rendering*

Terdapat banyak hasil yang dapat diperoleh dan ditampilkan dari proses *3D Rendering* pada suatu sketsa w*ireframe*, diantaranya :

1. *Shading*, variasi warna dan kecerahan yang timbul pada suatu permukaan berdasarkan pencahayaan yang dilakukan
2. *texture-mapping*, detail yang muncul pada suatu permukaan
3. *bump-mapping*, kontur yang muncul pada suatu permukaan
4. *fogging/participating medium*, bagaimana berkas cahaya berubah jika melewati udara yang tidak murni
5. *shadows*, efek dari cahaya yang terhalang
6. *soft shadows*, variasi efek dari cahaya yang terhalang tidak sempurna
7. *reflection*, refleksi yang tampak pada permukaan kaca atau gelas
8. *transparency*, transmisi cahaya yang berbeda-beda jika melewati medium tertentu
9. *translucency*, transmisi cahaya yang berbeda-beda jika memantul pada medium tertentu
10. *refraction*, cahaya yang berubah arahnya karena efek *transparency*
11. *indirect illumination*, cahaya yang datang pada suatu objek namun tidak berasal dari sumber cahaya yang sebenarnya melainkan refleksi dari permukaan objek lain
12. *caustics*, pantulan cahaya menyilaukan yang timbul pada suatu objek
13. *depth of field*, objek yang berada jauh di depan maupun di belakang objek yang menjadi fokus akan tampak buram
14. *motion blur*, objek yang bergerak dengan kecepatan tinggi atau objek yang direkam oleh kamera yang berada dalam kecepatan tinggi akan tampak buram
15. *photorealistic morphing*, teknik yang memungkinkan hasil *proses render* objek 3D menjadi tampak terlihat lebih nyata
16. *non-photorealistic rendering*, teknik yang memungkinkan hasil *proses render* objek 3D menjadi terlihat seperti hasil lukisan atau gambar

3. *Parallel Computing*

3.1 Pengertian *Parallel Computing*

Parallel Computing merupakan teknik menjalankan program untuk menjalankan suatu proses dengan menggunakan lebih dari satu unit komputasi[2].

Parallel Computing mempunyai prinsip yang bersesuaian dengan algoritma *Divide and Conquer*, yaitu membagi-bagi proses menjadi bagian-bagian yang cukup kecil dan memungkinkan untuk dikerjakan oleh sebuah unit komputasi.

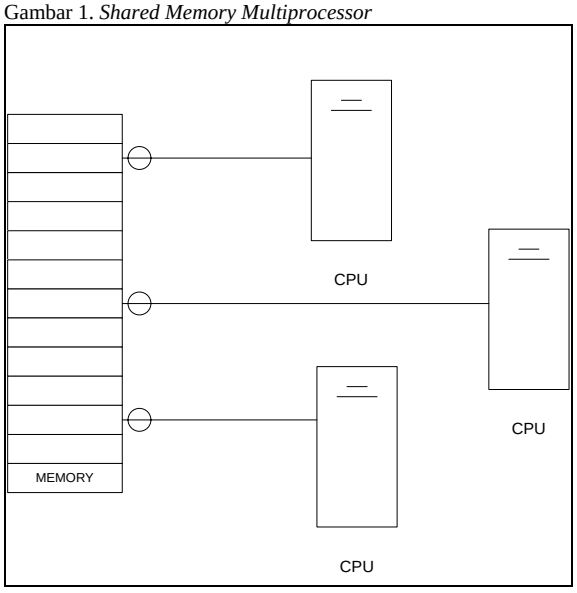
Terdapat 2 klasifikasi *parallel computer* yang penting, yaitu :

1. Sebuah komputer dengan banyak unit komputasi internal, atau lebih dikenal sebagai *Shared Memory Multiprocessor*
2. Beberapa komputer yang terhubung melalui sebuah jaringan, atau lebih dikenal sebagai *Distributed Memory Multicomputer*

Masing-masing jenis memiliki kelebihan dan kekurangannya masing

3.2 *Shared Memory Multiprocessor*

Gambar 1. *Shared Memory Multiprocessor*



Pada umumnya komputer yang digunakan untuk kebutuhan pribadi hanya menggunakan sebuah unit komputasi untuk mengerjakan proses-proses yang diberikan. Tetapi untuk meningkatkan kemampuan komputasi pada proyek-proyek berskala besar, diciptakanlah sebuah komputer dengan 2 atau lebih unit komputasi dalam sebuah CPU. Komputer jenis inilah yang disebut sebagai *Shared Memory Multiprocessor[2]*.

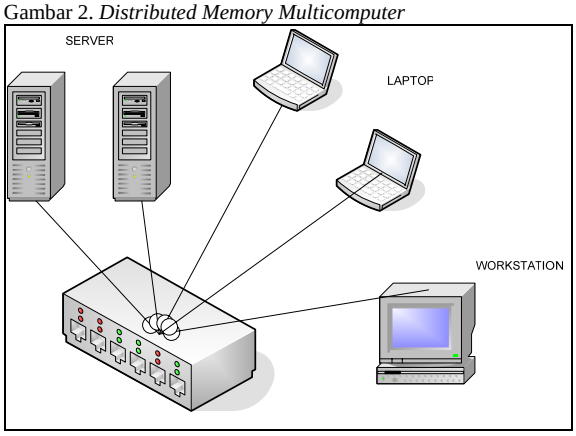
Kelebihan *Shared Memory Multiprocessor* jika dibandingkan dengan jenis *Parallel Computer* yang lain adalah lebih cepat dan efisien karena kecepatan transfer data antar unit komputasi tidak mengalami degradasi. Hal ini disebabkan karena tiap-tiap unit komputasi yang digunakan terhubung hingga tingkat bus.

Shared Memory Multiprocessor juga memiliki kekurangan yang cukup signifikan, diantaranya :

1. Sulit untuk mengkoordinasikan pembagian memori yang tersedia untuk masing-masing unit komputasi
2. Relatif lebih mahal dan lebih rumit untuk diimplementasikan
3. Tidak tahan lama, komputer jenis ini lebih sulit untuk diupgrade

3.3 *Distributed Memory Multicomputer*

Gambar 2. *Distributed Memory Multicomputer*



Distributed Memory Multicomputer atau yang seringkali disebut juga sebagai *Message-Passing Multicomputer* menggunakan jaringan sebagai penghubung antar masing-masing unit komputasi yang tersedia. Masing-masing unit komputasi pada *Distributed Memory Multicomputer* memiliki memori masing-masing dan tiap unit komputasi tersebut hanya dapat mengakses memorinya sendiri. Jika unit komputasi tersebut membutuhkan data yang disimpan pada memori dari unit komputasi lain, maka data tersebut akan dikirimkan melalui jaringan dengan menggunakan *Message-Passing[2]*.

Dalam *Distributed Memory Multicomputer* dikenal istilah *Master Processor* dan *Slave Processor*. *Master Processor* merupakan komputer yang bertugas untuk menghitung komputasi *pre-processing* dan *post-processing* seperti pembagian tugas kepada masing-masing komputer yang tersedia. Sedangkan *Slave Processor* merupakan seluruh komputer lainnya yang tidak menjadi *Master*.

Distributed Memory Multicomputer tentu jauh lebih murah dan mudah untuk diimplementasikan jika dibandingkan dengan *Shared Memory Multiprocessor* karena fleksibilitasnya yang tinggi dan harga untuk 2 buah *Processor* cenderung lebih murah jika dibandingkan dengan sebuah *Multiprocessor*.

Namun jika dibandingkan dengan metode *Shared Memory Multiprocessor* metode ini masih kurang efisien dan cepat karena unit komputasi yang tersedia hanya terhubung pada tingkat jaringan sehingga menyebabkan kecepatan komunikasi antara 2 unit komputasi mengalami degradasi yang cukup signifikan[2].

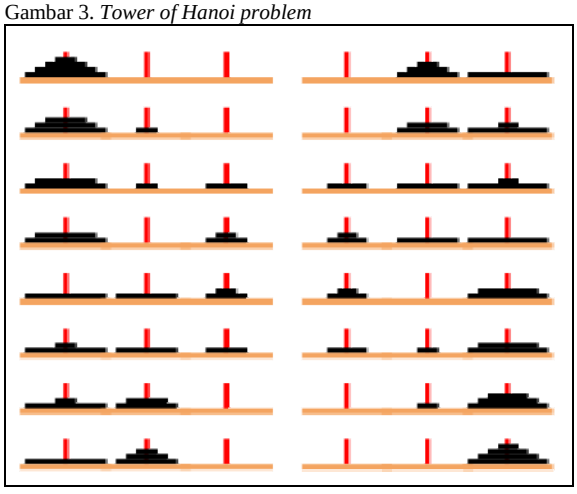
4. Algoritma *Divide and Conquer*

4.1 Pengertian Algoritma *Divide and Conquer*

Algoritma *Divide and Conquer* adalah strategi pemecahan masalah yang besar dengan cara melakukan pembagian masalah yang besar tersebut menjadi beberapa bagian yang lebih kecil secara rekursif hingga masalah tersebut dapat dipecahkan secara langsung. Solusi yang didapat dari setiap bagian kemudian digabungkan untuk membentuk sebuah solusi yang utuh.

4.2 Kemampuan Algoritma *Divide and Conquer*

Gambar 3. *Tower of Hanoi problem*



Algoritma *Divide and Conquer* memiliki kelebihan yang membuatnya banyak diterapkan dan digunakan dalam aplikasi-aplikasi dunia nyata, diantaranya :

1. Mampu menyelesaikan masalah yang sulit, algoritma ini mampu menyelesaikan masalah rumit yang hingga kini masih cukup sulit dipecahkan oleh komputer biasa, seperti *Tower of Hanoi Problem*
2. Algoritma lebih efisien untuk beberapa kasus tertentu, misalnya kasus *Fast Fourier Transform* maupun *Sorting* dapat dilakukan dengan kompleksitas algoritma *O*(n log n) dari algoritma lainnya yang hanya mampu mencapai kompleksitas *O* (n^2)
3. Algoritma ini dapat bekerja secara paralel dan dapat memaksimalkan penggunaan dari *cache memory*. Hal ini merupakan kunci dari pemrosesan secara *Parallel Computing*, dimana suatu masalah dipecah hingga sesuai dengan ukuran *cache* dari sebuah unit komputasi sehingga bagian tersebut dapat diproses hanya dalam 1 *cycle* saja[4].

5. Strategi Pemecahan Masalah

Pertama-tama kita tinjau terlebih dahulu perhitungan apa saja yang terjadi dalam proses *3D rendering*.

1. Program akan melakukan penghitungan jarak antara titik pusat dengan titik-titik *wireframe* yang bersesuaian untuk membungkus *wireframe* dengan tekstur sesuai dengan cahaya yang datang
2. Setelah koordinat titik-titik yang bersesuaian didapatkan, program mulai memberikan tekstur dan efek yang diinginkan sesuai dengan data cahaya yang tersedia dengan menggunakan persamaan *3D rendering equation*
3. Sesekali program akan menggunakan fungsi mencari pasangan titik yang jaraknya terdekat untuk membantu dalam pemberian tekstur

Dari perhitungan diatas kita dapat mengambil 2 contoh primitif yang dapat diselesaikan oleh algoritma *Divide and Conquer*, yaitu persoalan mengalikan 2 bilang bulat yang besar yang akan sering digunakan dan persoalan mencari pasangan titik yang jaraknya terdekat untuk memperhitungkan cahaya yang jatuh pada objek.

5.2 Persoalan Mengalikan 2 Bilangan Bulat yang Besar

Misalkan M dan N adalah bilangan bulat yang sangat besar, maka jika kita ingin mengalikan M dengan N, kita dapat menggunakan algoritma divide and conquer. Caranya adalah dengan membagi M dan N dengan 2 pada posisi s. M menjadi a dan b, sedangkan N menjadi c dan d. Setelah itu nilai a,b,c,d menyatakan nilai-nilai :

$a = M \text{ div } 10^s$
$b = M \text{ mod } 10^s$
$c = N \text{ div } 10^s$
$d = N \text{ mod } 10^s$

M dan N juga dapat dinyatakan dalam a,b,c,d, dan s sebagai

$M = a.10^s + b$
$N = c.\ 10^s + d$

Sehingga perkalian M dan N dapat dinyatakan sebagai

$$\begin{aligned} M \cdot N &= (a.10^s + b)\ (c.\ 10^s + d) \\ &= ac.10^{2s} + ad.10^s + bc.10^s + bd \\ &= ac.10^{2s} + (ad + bc).10^s + bd \end{aligned}$$

Dengan menggunakan algoritma ini kompleksitasnya $T(n) = O(n^2)$.

Melihat kompleksitasnya sama dengan brute force, A.A. Karatsuba berhasil menemukan metode yang lebih cepat dengan metode divide and conquer yang baru[5]. Caranya sebagai berikut :

Misalkan

$$r = (a+b)(c+d) = ac + (ad + bc) + bd \text{.....(2)}$$

maka,

$$(ad+bc) = r - ac - bd = (a+b)(c+d) - ac - bd$$

Oleh karena itu perkalian M dan N dapat dimanipulasi menjadi :

$$\begin{aligned} M \cdot N &= ac.10^{2s} + (ad + bc)\ .\ 10^s + bd \\ &= ac.10^{2s} + \{(a+b)\ (c+d) -\}\ .\ 10^s + bd \end{aligned}$$

Dengan bentuk demikian masalah perkalian M dan N dapat diselesaikan dengan kompleksitas algoritma $T(n) = O(n^{\log 3}) = O(n^{1.59})$. Dengan demikian terlihat bahwa cara Karatsuba lebih mangkus dari yang biasa[5].

5.3 Persoalan Mencari Pasangan Titik yang Jaraknya Terdekat

Misalkan kita diberikan point, Q, yang terdiri dari n buah titik (x_i,y_i) pada bidang 2D. Kita bisa mencari jarak antara 2 titik yang terdekat dengan algoritma divide and conquer. Dengan algoritma ini kita mempunyai 3 tahap yaitu :

1. Solve, ini terjadi jika n = 2. Kita bisa mendapatkan langsung jawabannya yaitu jarak kedua titik tersebut.
2. Divide, bagi daerah titik-titik tersebut menjadi 2 bagian yaitu P1 dan P2 dengan masing-masing mempunyai jumlah titik yang sama.

3. Conquer, pada bagian ini kita melakukan algoritma divide and conquer pada masing-masing bagian tersebut secara rekursif.
4. Setelah itu kita menemukan 3 kemungkinan solusi yaitu :

a. Solusi adalah jarak 2 titik pada P1
b. Solusi adalah jarak 2 titik pada P2
c. Solusi adalah jarak antara 2 titik yaitu 1 titik pada P1 dan yang lain pada P2.

Jika solusi berada pada kemungkinan ke-3 maka harus dilakukan tahap Combine.

5. Combine :

Jika ada pasangan titik P1 dan P2 yang jaraknya lebih kecil dari delta (jarak terdekat pada pasangan titik di P1 atau P2) maka kemungkinannya :

a. Absis P1 dan P2 berbeda sebesar-besarnya delta.
b. Ordinat P1 dan P2 berbeda sebesar-besarnya delta.

Maka lanjutkan pencarian ke :

a. Carilah semua titik pada P1 yang absisnya x setidaknya $x_{n/2}$ – delta.
b. Cari semua titik pada P2 yang absisnya x setidaknya $x_{n/2}$ + delta.

Ternyata dengan menggunakan algoritma divide and conquer kompleksitas yang didapatkan adalah T(n) = O(n log n).

5.4 Perbandingan Penggunaan Algoritma *Divide and Conquer* Dengan Algoritma Lain

Tabel 1. Perbandingan Kompleksitas Algoritma *Brute Force* dengan *Divide and Conquer*

No	Persoalan	Brute Force	Divide and Conquer
1	Mengalikan 2 bilangan bulat yang sangat besar	$O(n^2)$	Standar : $O(n^2)$ Karatsuba : $O(n^{1.59})$
2	Mencari pasangan titik dengan jarak terdekat.	$O(n^2)$	$O(n \log n)$

Jika dibandingkan dengan algoritma lainnya yang sering dipakai seperti algoritma *Brute Force*, maka perbedaannya tentu kurang signifikan, namun dalam proses *3D Rendering* sebuah gambar lingkaran saja memerlukan ratusan ribu point dan masing-masing point tentu akan memerlukan lebih dari sekali perhitungan. Jika suatu gambar kompleks yang di *render* maka mungkin saja akan membutuhkan milyaran point. Penggunaan algoritma yang lebih mangkus tentu akan sangat membantu dan mempercepat proses.

Dapat dikatakan bahwa algoritma *Divide and Conquer* merupakan solusi yang paling efisien untuk menyelesaikan masalah *3D Rendering* terlebih lagi karena dalam *Parallel Computing* masing-masing proses yang telah dibagi dikerjakan oleh sebuah unit komputasi sehingga tidak ada waktu yang terbuang percuma untuk menunggu proses lainnya selesai[4].

6. Kesimpulan

Melakukan *3D Rendering* tentu akan merupakan masalah yang sangat merepotkan jika tidak menggunakan algoritma memadai, dikarenakan banyaknya *point* dan *polygon* yang harus dikalkulasi setiap proses. Penggunaan algoritma *Divide and Conquer* tentu sangat membantu agar pekerjaan penghitungan yang sangat banyak dalam *proses render* dapat dikerjakan oleh beberapa unit komputasi sekaligus secara bersamaan. Hal ini juga telah dibuktikan dengan banyaknya perusahaan animasi dan film yang sering melakukan pekerjaan *proses render* menggunaan sistem *parallel computing* di dalam perusahaan mereka.

Penulis percaya, hingga ditemukan komputer yang dapat melakukan proses *3D rendering* dalam 1 *cycle* saja, algoritma *Divide and Conquer* akan tetap digunakan karena merupakan solusi untuk masalah ini yang paling efisien dan memungkinkan untuk diimplementasikan.

Daftar Pustaka

1. Akenine-Moller, Haines. 2002. *Real-time Rendering*, 2nd ed. AK Peters
2. B. Wilkinson and C. Michael Allen. 1999. *Parallel Programming : Techniques and Applications Using Networked Workstation and Parallel Computers*, 1st ed. Prentice Hall, Upper Saddle River, N.J..
3. Foley, Van Dam, Feiner, Hughes. 1990. *Computer Graphics : Principles And Practice*. Addison Wesley
4. Hardwick, Jonathan C. 1997. *Practical Parallel Divide and Conquer Algorithms*. School of Computer Science Carnegie Mellon University, Pittsburgh, P.A..
5. Munir, Ir.Rinaldi. 2005. *Strategi Algoritmik*. Lab. Ilmu Rekayasa Komputasi Program Studi Teknik Informatika ITB
6. Pharr, Humphreys. 2004. *Physically Based Rendering*. Morgan Kauffmann