ANALISIS WAKTU EKSEKUSI SISTEM TERDISTRIBUSI PADA ALCHEMI

SKRIPSI

Sebagai salah satu syarat untuk memperoleh gelar Sarjana Sains
dalam bidang Ilmu Komputer

Oleh :
ARIES SYAMSUDDIN
0210960012-96

PROGRAM STUDI ILMU KOMPUTER
JURUSAN MATEMATIKA
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS BRAWIJAYA
MALANG

2008

LEMBAR PENGESAHAN SKRIPSI

ANALISIS WAKTU EKSEKUSI
SISTEM TERDISTRIBUSI PADA ALCHEMI

Oleh :
ARIES SYAMSUDDIN
0210960012-96

Setelah dipertahankan di depan Majelis Penguji
pada tanggal 9 Januari 2008
dan dinyatakan memenuhi syarat untuk memperoleh gelar
Sarjana Sains dalam bidang Ilmu Komputer

Pembimbing I	Pembimbing II
Drs. Muh. Arif Rahman, M.Kom	Agus Wahyu Widodo, ST
NIP. 131 971 481	NIP. 132 295 994

Mengetahui,
a/n. Ketua Jurusan Matematika
Fakultas MIPA Universitas Brawijaya
Sekretaris Jurusan,

Dra. Ani Budi Astuti, Msi.
NIP. 131 993 385

ANALISIS WAKTU EKSEKUSI
SISTEM TERDISTRIBUSI PADA ALCHEMI

ABSTRAK

Komputasi terdistribusi menjadi salah satu topik yang cukup terkenal saat ini. Salah satu metode komputasi terdistribusi adalah komputasi grid. Komputasi grid merupakan suatu pendekatan inovatif yang meningkatkan infrastruktur teknologi informasi yang ada untuk mengumpulkan sumber daya, mengatur data, dan menghitung beban kerja aplikasi secara intensif. Teknologi grid dianggap sebagai suatu solusi yang sangat tangguh pada permasalahan pembagian sumber-sumber heterogen yang termasuk menghitung sumber daya, penyimpan data, dan berbagai jenis yang berbeda dari jasa yang disediakan oleh berbagai kesatuan. Di dalam skripsi ini, kami membahas tentang analisis waktu eksekusi sistem terdistribusi dengan menggunakan Alchemi. Hasil yang diperoleh bahwa waktu eksekusi proses *render* secara grid lebih cepat dibandingkan dengan secara *standalone*. Proses *render* secara grid dipengaruhi oleh perbedaan jumlah *executor*, perbedaan segmen, dan perbedaan besar ukuran gambar.

ANALISYSIS EXECUTION TIME DISTRIBUTED SYSTEM ON ALCHEMI

ABSTRACT

Distributed computing becomes one of topic which was popular currently. One of distributed computing method was grid computing. grid computing was an innovative approaching that increase information technology infrastructure which was available to gather resource, managing data, and accounts application work load intensively. Grid technology was look on as a very tough solution on about problem division heterogen resources including counting resources, data storing, and a variety type of various services that provided by every unity. In this paper, we will explain about analysis execution time distributed system by using Alchemi. Based on the results, render's execution time was faster than standalone's. Render process by grid was influenced by difference of total executor, difference of segment, and the difference of pictured measure.

LEMBAR PERNYATAAN

Saya yang bertanda tangan di bawah ini :

Nama	**:** Aries Syamsuddin
NIM	**:** 0210960012-96
Jurusan	**:** Matematika
Program Studi	**:** Ilmu Komputer
Penulis tugas akhir berjudul	**:** Analisis Waktu Eksekusi Sistem Terdistribusi pada Alchemi

Dengan ini menyatakan bahwa :

1. **Isi dari skripsi yang saya buat adalah benar-benar karya sendiri dan tidak menjiplak karya orang lain, selain nama-nama yang termaktub di isi dan tertulis di daftar pustaka dalam skripsi ini.**
2. **Apabila dikemudian hari ternyata skripsi yang saya tulis terbukti hasil jiplakan, maka saya akan bersedia menanggung segala resiko yang akan saya terima.**

Demikian pernyataan ini dibuat dengan segala kesadaran

Malang, 9 Januari 2008
Yang menyatakan,

Aries Syamsuddin
NIM. 0210960012-96

KATA PENGANTAR

Puji syukur penyusun panjatkan ke hadirat Allah SWT yang telah melimpahkan segala Rahmat, Karunia dan Hidayah-Nya sehingga penulis dapat menyelesaikan tugas akhir dengan judul "**Analisis Waktu Eksekusi Sistem Terdistribusi pada Alchemi**".

Skripsi ini diajukan sebagai syarat untuk memperoleh gelar Sarjana Komputer di Fakultas MIPA, Jurusan Ilmu Komputer, Universitas Brawijaya Malang. Atas terselesaikannya skripsi ini, penulis mengucapkan terima kasih kepada:

1. Bapak Drs. M. Arif Rahman, M.Kom, selaku pembimbing utama penulisan skripsi sekaligus selaku dosen penasehat akademik.
2. Bapak Agus Wahyu Widodo, ST., selaku pembimbing pendamping dalam penulisan skripsi ini.
3. Bapak Wayan Firdaus Mahmudy, S.Si., MT, selaku Ketua Program Studi Ilmu Komputer Jurusan Matematika FMIPA Universitas Brawijaya.
4. Istri tercinta yang selalu memberi semangat, membantu dan mendampingi penulis.
5. Kepada kedua orang tua dan mertua penulis yang tak pernah berhenti memberikan doa dan dukungannya kepada penulis.
6. Segenap bapak dan ibu dosen yang telah mendidik dan mengajarkan ilmunya kepada Penulis selama menempuh pendidikan di Program Studi Ilmu Komputer Jurusan Matematika FMIPA Universitas Brawijaya.
7. Segenap staf dan karyawan di Jurusan Matematika FMIPA Universitas Brawijaya yang telah banyak membantu penulis dalam pelaksanaan penyusunan skripsi ini.
8. Kakak dan adik penulis yang memberi dukungan kepada penulis.
9. Rekan-rekan di Program Studi Ilmu Komputer FMIPA Universitas Brawijaya yang telah banyak memberikan bantuannya demi kelancaran pelaksanaan penyusunan tugas akhir ini.
10. Semua pihak yang telah membantu terselesaikannya penyusunan skripsi ini.

Penulis menyadari bahwa dalam penyusunan skripsi ini masih banyak kekurangan, untuk itu saran dan kritik yang membangun

demi kesempurnaan penulisan selanjutnya sangat penulis harapkan. Semoga skripsi ini dapat memberikan manfaat bagi semua pihak.

Malang, 9 Januari 2008

Penulis

DAFTAR ISI

DAFTAR TABEL

DAFTAR GRAFIK

DAFTAR GAMBAR

BAB I
PENDAHULUAN

1.1 Latar Belakang

Para peneliti saat ini mulai menyadari pentingnya komputer dalam pengembangan ilmu pengetahuan dan teknologi. Komputer memungkinkan para peneliti menciptakan laboraturium virtual dalam komputer untuk melakukan eksperimen yang mahal atau bahkan tidak mungkin jika dilakukan di dalam sebuah laboraturium secara nyata. Sebagai dampak dari kebutuhan komputasi yang tinggi dan kecepatan komputasi maka kebutuhan akan komputer sangat diperlukan untuk melakukan komputasi-komputasi tersebut. Komputer sebagai sarana untuk melakukan komputasi memiliki keterbatasan dan kebanyakan komputer PC hanya dapat melakukan komputasi secara sekuensial. Hal ini menyebabkan proses komputasi menjadi sangat lambat.

Untuk memenuhi kebutuhan tersebut maka komputasi terdistribusi menjadi salah satu pilihan. Komputasi terdistribusi adalah proses mengumpulkan sumber daya dari beberapa kesatuan komputasi untuk secara kolaboratif menjalankan satu atau lebih tugas dalam suatu cara yang jelas dan terpadu. Komputasi terdistribusi memanfaatkan kehandalan dari banyak komputer atau kelompok dari komputer, untuk menyediakan suatu *supercomputer* yang maya.

Komputasi terdistribusi menjadi salah satu topik yang cukup terkenal saat ini. Salah satu metode komputasi terdistribusi adalah komputasi grid. Komputasi grid merupakan suatu pendekatan inovatif yang meningkatkan infrastruktur TI yang ada untuk mengumpulkan sumber daya, mengatur data, dan menghitung beban kerja aplikasi secara intensif. Buyya menggambarkan grid sebagai suatu jenis dari sistem paralel terdistribusi yang memungkinkan pembagian, pemilihan, dan pengumpulan secara geografis membagi-bagikan sumber daya otonomi secara dinamis pada saat bekerja yang tergantung pada ketersediaan, kemampuan, kinerja, biaya, dan persyaratan-persyaratan kualitas pelayanan para pemakai. (R. Buyya, 2002)

Teknologi grid memungkinkan para pengguna memanfaatkan sumber daya komputasi yang telah ada semaksimal mungkin. Dengan menggunakan teknologi ini, kita dapat menggabungkan

komputer-komputer yang berada di tempat yang secara geografis terpisah menjadi suatu kesatuan sistem komputer. Gabungan banyak komputer ini secara keseluruhan mampu menyediakan sumber daya komputasi yang setara atau bahkan lebih dengan komputer berkategori *supercomputer*. Lebih lanjut, sistem komputer ini dapat digunakan secara bersamaan oleh para pengguna yang juga berasal dari instansi-instansi yang lokasinya berlainan. (Bobby Nazief, 2006)

Salah satu contoh penerapan sistem berbasis grid pada dunia multimedia adalah *image-rendering*. *Image-rendering* merupakan proses dari pembangkit image dari suatu model dengan memakai program komputer. *Image-rendering* mampu menghasilkan gambar yang tampak seperti hasil fotografi yang realistik. *Image-rendering* untuk suatu citra yang sangat kompleks (gambar tiga dimensi) dibutuhkan waktu berjam-jam bahkan berhari-hari. Teknologi grid dapat digunakan untuk meningkatkan kinerja *image-rendering*. Pada suatu percobaan yang dilakukan Balder Aljaber, dibutuhkan sekitar 5 menit untuk mengubah 4 dari 100 irisan gambar yang berarti memerlukan sekitar 2 jam untuk mengubah gambar lengkap. Sedangkan diperlukan sekitar 9 hari dengan menggunakan komputer *standalone*. (Balder Aljaber, 2007)

Jumlah komputer yang digunakan untuk *image-rendering* juga berpengaruh terhadap waktu eksekusinya, yaitu semakin banyak komputer yang digunakan maka waktu eksekusinya lebih cepat. Hal ini seperti yang dilakukan oleh Gengki SW pada percobaan dalam menghitung matriks. (Gengki SW, 2004)

Alchemi merupakan salah satu teknologi grid yang saat ini terkenal digunakan untuk melakukan komputasi terdistribusi. Alchemi dapat digunakan untuk meningkatkan kinerja *image-rendering*. Alchemi memungkinkan membagi kerja *render* ke banyak komputer *client* (*executor*). Pada masing-masing *executor* akan melakukan proses *render* terhadap model yang telah ditentukan sebelumnya. Pembagian dan pengumpulan kerja dilakukan oleh *manager*, *executor* hanya menjalankan perintah yang diberikan dari *manager*. Pada penyusunan skripsi ini, penulis akan membuktikan apakah teknologi grid dapat mengingkatkan kinerja dari *image-rendering*. Aplikasi yang digunakan untuk *image-rendering* adalah POV-Ray. Dengan latar belakang tersebut maka pada penyusunan skripsi ini penulis mengambil judul : "**Analisis Waktu Eksekusi Sistem Terdistribusi pada Alchemi "**.

1.2 Rumusan Masalah

Permasalahan yang akan dijadikan objek penelitian pada skripsi ini adalah bagaimana Alchemi digunakan untuk menyelesaikan masalah proses *render* pada POV-Ray sehingga diperoleh waktu eksekusi yang lebih cepat.

1.3 Tujuan

Tujuan dari penulisan skripsi ini adalah melakukan analisis perbandingan kinerja berdasarkan waktu eksekusi antara proses *render* secara s*tandalone* dengan proses *render* secara grid

1.4 Batasan Masalah

Batasan masalah pada skripsi ini adalah :

1. Hanya membahas perancangan dan implementasi Alchemi untuk permasalahan *render* dengan menggunakan POV-Ray.
2. Parameter pengukuran kinerja yang dibahas adalah waktu proses sekuensial, waktu proses *grid computing*, overhead proses manager, speedup, efisiensi pemroses, granularitas *executor*, dan *cc ratio*
3. Keluaran dari hasil *render* dibatasi hanya berupa gambar tiga dimensi.
4. Tidak membahas *source code* file *render* dari Pov-Ray.
5. Mengabaikan faktor keamanan data pada proses pengiriman data.

1.5 Manfaat

Manfaat yang bisa diambil dari penulisan skripsi ini adalah :

1. Hasil komputasi menjadi lebih cepat dengan biaya yang murah.
2. Dengan teknologi berbasis P2P pengguna komputer dapat menyumbangkan daya komputer yang tidak terpakai untuk membangun sebuah s*upercomputer* virtual.
3. Penggunaan daya komputer untuk suatu keperluan tertentu tanpa terikat batas geografi.

BAB II
TINJAUAN PUSTAKA

2.1 Grid Computing

2.1.1. Evolusi Grid Computing

Teknologi *grid computing* merupakan teknologi yang telah dikembangkan dalam waktu yang panjang (sejak tahun 1970). Secara evolusi pengembangan teknologi sejenis mulai dari Condor, kemudian diikuti oleh PVM (*Parallel Virtual Machine*) dan MPI (*Message Passing Interface*), sampai dengan Globus Toolkit. Sejak awal, para peneliti di bidang komputasi berkinerja tinggi telah menggunakan dua pendekatan, (1) *supercomputer*, membangun sebuah komputer dengan teknologi perangkat keras berkinerja tinggi, dan (2) *multicomputer*, membangun sebuah sistem komputer dengan teknologi jaringan interkoneksi dan perangkat lunak. Pendekatan pertama umumnya menghasilkan sebuah komputer yang berkinerja tinggi, tetapi berharga amat mahal sehingga hanya dapat dimiliki oleh segelintir pihak saja. Pendekatan kedua menghasilkan suatu sistem komputer yang kinerjanya bervariasi sesuai jumlah komputer yang tergabung dan konfigurasi perangkat lunak yang digunakan. (Bobby Nazief. 2006)

Walaupun harga suatu sistem komputer berkinerja tinggi yang dibangun dengan pendekatan *multicomputer* lebih terjangkau dibandingkan dengan *supercomputer*, pemakaiannya masih terbatas. Sistem komputer berbasis jaringan tersebut umumnya diterapkan pada komputer-komputer yang terhubung dalam suatu jaringan lokal (LAN). Salah satu penyebabnya adalah masalah keamanan jaringan yang belum tertangani dengan baik. Selain itu, sistem perangkat lunak pendukung yang memungkinkan komputer-komputer tersebut bekerja sebagai satu kesatuan umumnya memiliki konfigurasi yang kompleks sehingga penggunanya harus memiliki keahlian tersendiri sebelum dapat memanfaatkan sistem komputer tersebut. (Bobby Nazief. 2006)

Sejalan dengan perkembangan teknologi Internet dan teknologi-teknologi komputer yang berkaitan lainnya seperti protokol komunikasi data, teknologi keamanan jaringan, teknologi pemgrograman terdistribusi, dan teknologi bahasa pemrograman

yang independen terhadap arsitektur komputer, maka sistem komputer berkinerja tinggi berbasis jaringan menjadi lebih mudah untuk diimplementasikan dan digunakan.

2.1.2. Grid Computing dan Solusi yang Ditawarkan

Pada beberapa tahun belakangan ini, sekelompok peneliti di bidang komputasi berkinerja tinggi secara serius memusatkan perhatian pada pengembangan sistem komputer berbasis jaringan seperti yang telah diuraikan di atas dengan menggunakan teknologi yang dikenal dengan sebutan teknologi *grid computing.*

Teknologi *grid computing* adalah suatu cara penggabungan sumber daya yang dimiliki banyak komputer yang terhubung dalam suatu jaringan sehingga terbentuk suatu kesatuan sistem komputer dengan sumber daya komputasi yang besarnya mendekati jumlah sumber daya komputasi dari komputer-komputer yang membentuknya. Lebih lanjut, sebagian atau seluruh sumber daya komputasi ini dapat dipakai oleh penggunanya sesuai kebutuhan masing-masing. Penamaan "*grid*" disini meminjam istilah yang digunakan dalam ketenagalistrikan, dimana pembangkit-pembangkit tenaga listrik dihubungkan satu sama lain untuk secara bersama-sama memasok kebutuhan tenaga listrik penggunanya. Masing-masing pengguna hanya menggunakan sebagian dari daya listrik yang dihasilkan oleh seluruh pembangkit tenaga listrik tersebut. (Bobby Nazief. 2006)

Berbeda dengan teknologi-teknologi pendahulunya seperti Condor, PVM, atau MPI, teknologi *grid computing* dilengkapi oleh komponen-komponen yang memungkinkan pemanfaatan sumber daya komputasi yang terhimpun secara lebih optimal dan aman. Untuk melihat komponen-komponen dari teknologi *grid computing* ini, akan diuraikan dengan singkat tentang Alchemi, sebuah teknologi *grid computing* yang berbasis *framework* .Net dan menggunakan teknologi *peer-to-peer*(P2P).

2.1.3. Perkembangan Grid Computing di Asia Pasifik

Teknologi *Grid Computing* saat ini dikenal di lingkungan bisnis di Asia Pasifik. Hasil survey Oracle Grid Index terakhir misalnya, memperlihatkan pertumbuhan tahunan di kawasan tersebut mencapai

83 %. *Grid Computing* adalah kegiatan memakai banyak komputer secara bersamaan untuk menyelesaikan sebuah pekerjaan. Penggunaannya banyak terdapat di industri otomotif, jasa keuangan, pemerintahan, dan pendidikan. Manfaatnya bagi mereka yaitu dapat menekan biaya operasional karena grid juga bisa membangun dengan menggunakan infrastruktur komputer yang telah ada. (M9!, 2007)

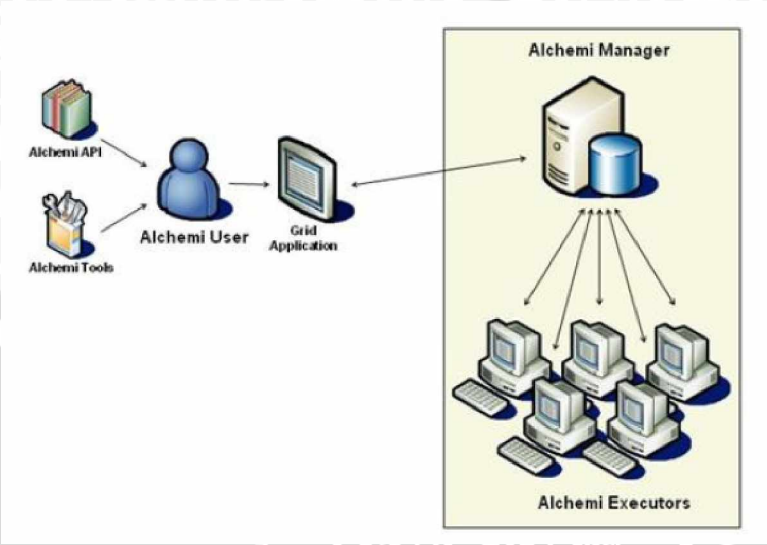
Terkait hasil survey Oracle, tingkat pertumbuhan yang berhasil dicapai di Asia Pasifik ini jelas lebih tinggi daripada Amerika Serikat uang berjumlah 45% atau Eropa yang 7% saja. Riset yang sama menunjukkan pula bila tingkat pertumbuhan dan minat konsumen di Asia Pasifik terhadap *Grid Computing* menjadi 6,1. Angka itu sama dengan indeks yang diraih Eropa Utara dan Amerika Serikat tahun 2005. Ke depan, sebagaimana dikatakan Regional Director ASEAN Oracle Coorporation Asia Pasifik Kaleem Chaudhry, grid computing ini akan semakin diakui keberadaannya. Hampir 90% organisasi di Asia Pasifik berpotensi untuk pindah ke grid computing dan 60% diantaranya manyatakan jika perpindahan tersebut pasti terjadi. (M9!, 2007)

Menurut Marketing Director PT Oracle Indonesia Goenawan Loekito, *grid computing* dapat diaplikasikan oleh berbagai jenis industri. Contohnya saja, telekomunikasi, lembaga keuangan, dan lembaga pemerintahan. Melalui kemampuannya memanfaatkan sumber daya seperti peranti keras yang sedang menganggur serta mengintegrasikan data antar departemen, pengaplikasian grid computing jelas dapat meningkatkan efisiensi di suatu perusahaan. Besarnya efisiensi tergantung pada skala perusahaan, jenis dan jumlah perangkat yang digunakan, serta banyaknya sistem dan aplikasi yang dipakai. Secara umum, kinerja yang diperoleh dalam memanfaatkan kapasitas infrastruktur bisa meningkatkan dari 40% menjadi 80%. (M9!, 2007)

2.2 Alchemi

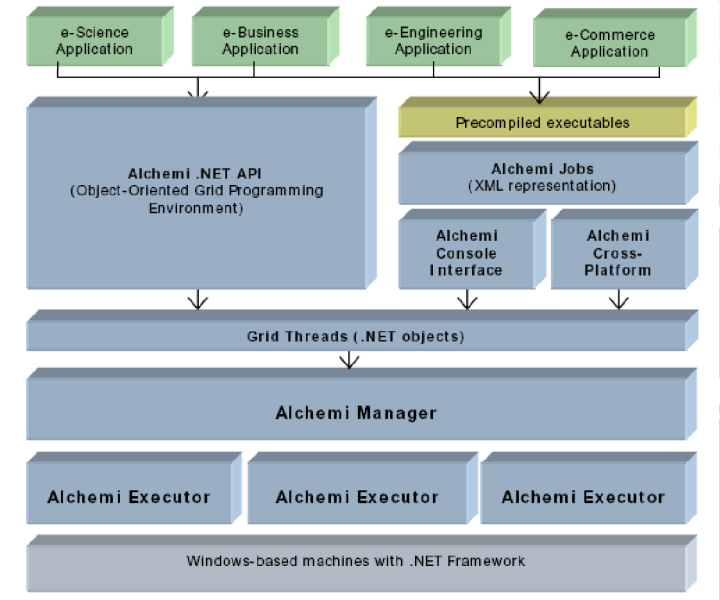
2.2.1 Alchemi Grid Computing

Ada empat jenis komponen yang dilibatkan di dalam konstruksi grid Alchemi dan aplikasi pendukungnya : *Manager, Executor, User & Cross-Platform Manager*.



Gambar 2.1 Grid Alchemi
(Sumber: Krishna Nadiminti, Akshay Luther, Rajkumar Buyya. 2005)

Pada masing-masing komputer *client* dilakukan instalasi Alchemi *executor*, kemudian pada komputer *server* dilakukan instalasi Alchemi *manager*. Sehingga nantinya masing-masing client dapat melakukan koneksi ke server yang dituju.



Gambar 2.2 Arsitektur Alchemi
(Sumber: Akshay Luther, dkk. 2005)

Alchemi dibangun dengan menggunakan *framework* .NET dan bahasa pemrograman yang digunakan adalah C#. Walaupun demikian Alchemi bersifat *opensource*.

2.2.2 Parameter Pengukuran Kerja

Dalam komputasi *grid computing* terdapat beberapa parameter pengukuran kinerja. Parameter yang diperlukan dalam melakukan pengukuran kinerja komputasi program Alchemi adalah sebagai berikut:

- Waktu proses sekuensial
  Waktu yang diukur adalah waktu eksekusi algoritma sekuensial, tidak termasuk waktu eksekusi operasi I/O. (Gengki, SWA, 2006)
- Waktu proses *grid computing*
  Waktu yang diukur adalah waktu eksekusi *grid computing* termasuk waktu *overhead*, tetapi tidak termasuk waktu operasi I/O. (Gengki, SWA, 2006)

- Overhead proses manager
  Waktu yang diperlukan untuk membuat proses thread, distribusi/pengiriman data dan pengumpulan/penerimaan data. (Gengki, SWA, 2006)
- Speedup
  Speedup diukur dengan membandingkan waktu proses sekuensial pada satu pemroses dengan waktu *grid computing* pada *p* pemroses untuk jumlah data yang sama. Bila *T(n)* didefinisikan sebagai waktu yang diperlukan oleh program sekuensial untuk memecahkan masalah dengan ukuran *n* dan *Tp(n)* sebagai waktu yang diperlukan program *grid* menggunakan *p* pemroses maka

$$sp(n) = \frac{T(n)}{Tp(n)}, \qquad \ldots\ldots\ldots 2.1$$

  speedup yang dicapai dapat melebihi jumlah pemroses yang digunakan. Hal ini bias disebabkan karena *grid computing* memiliki lebih banyak strategi pemecahan masalah sehingga kemungkinan untuk memperoleh solusi terbaik yang lebih cepat menjadi lebih besar . (Gengki, SWA, 2006)
- Efisiensi pemroses
  Efisiensi diukur dengan membagi *Speedup* yang diperoleh dengan jumlah pemroses yang digunakan. Berdasarkan definisi *speedup* diatas, maka efisiensi (Ep(n) pada system dengan *p* pemroses bias diformulasikan sebagai berikut:

$$Ep(n) = \frac{Sp(n)}{p} = \frac{T(n)}{pTp(n)} \qquad \ldots\ldots\ldots 2.2$$

  Effisiensi maksimum dicapai disaat seluruh *p* elemen pemrosesan benar-benar bekerja penuh selama periode eksekusi. (Gengki, SWA, 2006)
- Granularitas *executor*
  Granularitas adalah ukuran dari jumlah komputasi yang terlibat dalam proses. Disini granularitas *executor* bisa dihitung dengan mencatat waktu eksekusi proses *executor* pada tiap-tiap pemroses. (Gengki, SWA, 2006)

- CC Ratio
  Computation-Comunication Ratio (CC Ratio) adalah nilai perbandingan antara total waktu komputasi dengan total waktu komunikasi. (MIT Press, 1995)

2.3 Dot NET

2.3.1 Framework dot NET

Framework .NET adalah suatu komponen windows terintegrasi yang dibuat dengan tujuan untuk mensupport pengembangan berbagai macam jenis aplikasi serta agar dapat menjalankan berbagai macam aplikasi generasi mendatang termasuk pengembangan aplikasi Web Services XML. (Kurniawan, Agus, Dkk. 2004)

Framework .NET di design untuk dapat memenuhi beberapa tujuan berikut ini :

- Untuk menyediakan environment kerja yang konsisten bagi bahasa pemrograman yang berorientasi objek (*object-oriented programming - OOP*) baik kode objek itu di simpan dan di eksekusi secara lokal, atau dieksekusi secara lokal tapi didistribusikan melalui internet atau di eksekusi secara remote.
- Untuk menyediakan environment kerja di dalam mengeksekusi kode yang dapat meminimaliasi proses software *deployment* dan menghindari konflik penggunaan versi software yang di buat.
- Untuk menyediakan environment kerja yang aman dalam hal pengeksekusian kode, termasuk kode yang dibuat oleh pihak ketiga (*third party).*
- Untuk menyediakan *environment* kerja yang dapat mengurangi masalah pada persoalan performa dari kode atau dari lingkungan *interpreter* nya.
- Membuat para developer lebih mudah mengembangkan berbagai macam jenis aplikasi yang lebih bervariasi, seperti aplikasi berbasis windows dan aplikasi berbasis web.
- Membangun semua komunikasi yang ada di dalam standar industri untuk memastikan bahwa semua kode aplikasi yang

berbasis *Framework* .NET dapat berintegrasi dengan berbagai macam kode aplikasi lain.

Sebagai salah satu sarana untuk dapat memenuhi tujuan di atas, maka dibuatlah berbagai macam bahasa pemrograman yang dapat digunakan dan dapat berjalan di atas platform Framework .NET seperti bahasa C#, VB.NET, J#, Perl.NET dan lain-lain. Masing-masing bahasa tersebut mempunyai kelebihan dan kekurangannya namun yang pasti, apapun bahasa pemrograman yang digunakan, semuanya akan dapat saling berkomunikasi dan saling *compatible* satu dengan yang lainnya dengan bantuan *Framework* .NET

2.3.2 Arsitektur Framework .NET

Framework .NET terdiri dari dua buah komponen utama, yaitu *Common Language Runtime (CLR)* dan *.NET Framework Class Library* atau kadang juga sering disebut dengan *Base Class Library (BCL)*. (Kurniawan, Agus, Dkk. 2004).

Common Language Runtime (CLR) adalah pondasi utama dari *Framework* .NET. CLR merupakan komponen yang bertanggung jawab terhadap berbagai macam hal, seperti bertanggung jawab untuk melakukan managemen memory, melakukan eksekusi kode, melakukan verifikasi terhadap keamanan kode, menentukan hak akses dari kode, melakukan kompilasi kode, dan berbagai layanan sistem lainnya. Dengan adanya fungsi CLR ini, maka aplikasi berbasis .NET biasa juga disebut dengan *managed code*, sedangkan aplikasi di luar itu biasa disebut dengan *un-managed code*. (Kurniawan, Agus, Dkk. 2004)

Berikut ini beberapa hal yang disediakan CLR bagi para developer:

- Dapat lebih menyederhakan proses pengembangan aplikasi
- Memungkinkan adanya variasi dan integrasi dari berbagai bahasa pemrograman yang ada di lingkunan *Framework* .NET
- Keamanan dengan melakukan identifikasi pada kode aplikasi.
- Bersifat *Assembly* pada saat proses deployment / kompilasi
- Melakukan *versioning* sebuah komponen yang bisa didaur ulang.

- Memungkinkan penggunaan kembali kode, dengan adanya sifat inheritance.
- Melakukan pengaturan / manajemen tentang *lifetime* sebuah objek.
- Melakukan penganalisaan objek-objek secara otomatis.

CLR akan melakukan kompilasi kode-kode aplikasi kita menjadi bahasa assembly MSIL (*Microsoft Intermediate Language*). Proses kompilasi ini sendiri dilakukan oleh komponen yang bernama *Just In Time* (JIT). JIT hanya akan mengkompilasi metode-metode yang memang digunakan dalam aplikasi, dan hasil kompilasi ini sendiri di *chace* di dalam mesin dan akan dikompile kembali jika memang ada perubahan pada kode aplikasi tersebut.

.*NET Framework Class Library* atau sering juga disebut *Base Case Library* (BCL) adalah koleksi dari *reusable types* yang sangat terintegrasi secara melekat dengan CLR. *Class library* bersifat berorientasi terhadap objek yang akan menyediakan *types* dari fungsi-fungsi *managed code*. Hal ini tidak hanya berpengaruh kepada kemudahan dalam hal penggunaan, tetapi juga dapat mengurangi waktu yang diperlukan pada saat eksekusi. Dengan sifat tersebut, maka komponen pihak ketiga akan dengan mudah diaplikasikan ke dalam aplikasi yang dibuat. (Kurniawan, Agus, Dkk. 2004)

Dengan adanya BCL ini, maka kita bisa menggunakan *Framework* .NET untuk membuat berbagai macam aplikasi, seperti :

- Aplikasi *console*
- Aplikasi berbasis windowd (*Windows Form*)
- Aplikasi ASP.NET (berbasis web)
- Aplikasi Web Services XML
- Aplikasi berbasis *Windows Services*

Jika kita membuat sekumpulan *Class* untuk membuat aplikasi berbasis windows, maka *Class-class* itu bisa kita gunakan untuk jenis aplikasi lain, seperti aplikasi berbasis web (ASP.NET).

2.3.3 Keuntungan Framework .Net

Berikut merupakan beberapa keuntungan atau kelebihan dari *Framework* .NET :

a. Mudah

Kemudahan di sini lebih ke arah pada kemudahan bagi para developer untuk membuat aplikasi yang dijalankan di lingkungan *Framework* .NET. Beberapa hal yang merepotkan *developer* pada saat membuat aplikasi, telah dihilangkan atau diambil alih kemampuannya oleh *Framework* .NET, misalnya masalah *lifetime* sebuah objek yang biasanya luput dari perhatian *developer* pada saat proses pembuatan aplikasi. Masalah ini telah ditangani dan diatur secara otomatis oleh Framework .NET melalui komponen yang bernama *Garbage Collector* yang bertanggung jawab untuk mencari dan membuang objek yang sudah tidak terpakai secara otomatis.

b. Efisien

Kemudahan pada saat proses pembuatan aplikasi akan berimplikasi terhadap efisiensi suatu proses produktivitas, baik efisien dalam hal waktu pembuatan aplikasi atau juga efisien dalam hal lain, seperti biaya (*cost*).

c. Konsisten

Kemudahan-kemudahan pada saat proses pembuatan aplikasi, juga bisa berimplikasi terhadap konsistensi pada aplikasi yang kita buat. Misalnya, dengan adanya BCL, maka kita bisa menggunakan objek atau *Class* yang dibuat untuk aplikasi berbasis windows pada aplikasi berbasis web. Dengan adanya kode yang bisa dintegrasikan ke dalam berbagai macam aplikasi ini, maka konsistensi kode-kode aplikasi kita dapat terjaga.

d. Produktivitas

Semua kemudahan-kemudahan di atas, pada akhirnya akan membuat produktivitas menjadi lebih baik. Produktivitas naik, terutama produktivitas para developer, akan berdampak pada meningkatnya produktivitas suatu perusahaan.

2.3.4 C#

C# (dibaca “See-Sharp”) adalah bahasa pemrograman baru yang diciptakan oleh Microsoft (dikembangkan dibawah kepemimpinan Anders Hejlsberg yang juga telah menciptakan berbagai macam bahasa pemrograman termasuk Borland Turbo C++ dan Borland Delphi). Bahasa C# juga telah di standarisasi secara internasional oleh ECMA. (Kurniawan, Agus, Dkk. 2004).

Seperti halnya bahasa pemrograman yang lain, C# bisa digunakan untuk membangun berbagai macam jenis aplikasi, seperti

aplikasi berbasis windows (desktop) dan aplikasi berbasis web serta aplikasi berbasis *web services*. Ada beberapa alasan kenapa memilih C#, yaitu :

a. Sederhana (simple)

C# menghilangkan beberapa hal yang bersifat kompleks yang terdapat dalam beberapa macam bahasa pemrograman seperti Java dan C++, termasuk diantaranya mengilangkan *macro*, *templates*, *multiple inheritance* dan *virtual base classes*. Hal-hal tersebut yang dapat menyebabkan kebingunan pada saat menggunakannya, dan juga berpotensial menjadi masalah bagi para programmer C++. Jika anda pertama kali belajar bahasa C# sebagai bahasa pemrograman, maka hal-hal tersebut di atas tidak akan membuat waktu anda terbuang terlalu banyak untuk mempelajarinya. C# bersifat sederhana, karena bahasa ini didasarkan kepada bahasa C dan C++. Jika anda familiar dengan C dan C++ atau bahkan Java, anda akan menemukan aspek-aspek yang begitu familiar, seperti *statements*, *expression*, *operators*, dan beberapa fungsi yang diadopsi langsung dari C dan C++, tetapi dengan berbagai perbaikan yang membuat bahasanya menjadi lebih sederhana.

b. Modern

Apa yang membuat C# menjadi suatu bahasa pemrograman yang modern? Jawabannya adalah adanya beberapa fitur seperti *exception handling*, *garbage collection*, *extensible data types*, dan *code security* (keamanan kode/bahasa pemrograman). Dengan adanya fitur-fitur tersebut, menjadikan bahasa C# sebagai bahasa pemrograman yang modern.

c. Object-Oriented Language

Kunci dari bahasa pemrograman yang bersifat *Object Oriented* adalah *encapsulation*, *inheritance*, dan *polymorphism*. Secara sederhana, istilah-istilah tersebut bisa didefinisikan sebagai berikut (definisi dan penjelasan lebih jauh akan di uraikan pada bab-bab selanjutnya). *encapsulation*, dimana semua fungsi ditempatkan dalam satu paket (*single package*). *inheritance*, adalah suatu cara yang terstruktur dari suatu kode-kode pemrograman dan fungsi untuk menjadi sebuat program baru dan berbentuk suatu paket.. *Polymorphism* adalah kemampuan untuk mengadaptasi apa yang diperlukan untuk dikerjakan. Sifat-sifat tersebut di atas, telah

dimiliki oleh C# sehingga bahasa C# merupakan bahasa yang bersifat *Object Oriented*.

d. *Powerfull* dan fleksibel

C# bisa digunakan untuk membuat berbagai macam aplikasi, seperti aplikasi pengolah kata, grafik, *spreadsheets*, atau bahkan membuat kompiler untuk sebuah bahasa pemrograman.

e. Efisien

C# adalah bahasa pemrograman yang menggunakan jumlah kata-kata yang tidak terlalu banyak. C# hanya berisi kata-kata yang biasa disebut dengan *keywords*. *Keywords* ini digunakan untuk menjelaskan berbagai macam informasi. Jika anda berpikiran bahwa bahasa pemrograman yang menggunakan sangat banyak kata-kata (*keywords*) akan lebih *powerfull*, maka jawabannya adalah "pemikiran itu tidak selalu benar", karena hal itu justru bisa menambah kerumitan para developer pada saat membuat suatu aplikasi.

f. Modular

Kode C# ditulis dengan pembagian masing *Class-class* (*classes*) yang terdiri dari beberapa *routines* yang disebut sebagai *member methods*. *Class-class* dan metode-metode ini dapat digunakan kembali oleh program atau aplikasi lain. Hanya dengan memberikan informasi yang dibutuhkan oleh *Class* dan metode yang dimaksud, maka kita dapat membuat suata kode yang dapat digunakan oleh satu atau beberapa aplikasi dan program (*reusable code*).

g. C# akan menjadi populer

Dengan dukungan penuh dari Microsoft yang akan mengeluarkan produk-produk utamanya dengan dukungan *Framework* .NET, maka masa depan bahasa C# sebagai salah satu bahasa pemrograman yang ada di dalam lingkungan *Framework* .NET akan lebih baik.

2.4 Render

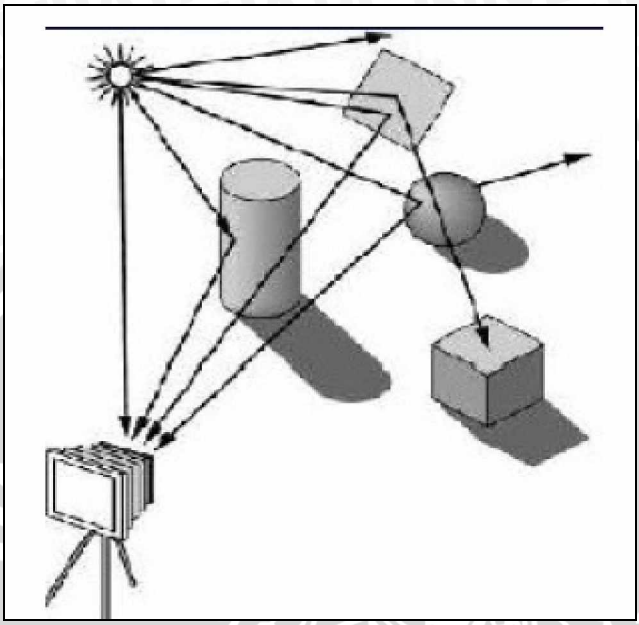
Rendering adalah proses dari pembangkit image dari suatu model dengan memakai program komputer. Model adalah suatu uraian tentang objek tiga dimensi dengan *strictly defined language* atau struktur data. Hal ini mengandung geometri, sudut pandang,

tekstur , pencahayaan , dan bayangan. Image adalah suatu gambar digital atau *raster graphics image.*

Ini merupakan salah satu sub utama topik dari grafis 3D komputer , dan dalam praktek selalu terkoneksi ke yang lain. *Render* digunakan pada arsitektur game video , simulator , bioskop atau efek khusus TV dan disain visualisasi, masing-masing mempekerjakan secara seimbang, berbeda dari fitur dan ilmu pengetahuan tentang teknik. Sebagai satu produk, bermacam-macam *renderer* tersedia. Beberapa mengintegrasikan ke dalam model lebih besar dan paket animasi, beberapa berdiri sendiri, beberapa merupakan proyek *opensource*, dan beberapa diantaranya adalah pov-ray, pixar renderman dan metalray.

2.4.1 Cluster dan Ray tracing

Ray tracing adalah suatu metode untuk me-*render* obyek 3D yang hasilnya realistik seperti foto. Metode ini dilakukan dengan cara menelusuri sinar mata atau sumber cahaya, kemudian diperiksa apakah sinar tersebut mengenai obyek atau tidak. Jika ternyata sinar yang ditelusuri tersebut mengenai suatu obyek maka selanjutnya diperhitungkan intensitas pada obyek tersebut, yaitu intensitas *ambient, diffuse* dan *specular*. Hasil dari perhitungan intensitas inilah yang terlihat oleh mata. Metode *ray tracing* dibagi menjadi dua jenis, yaitu *forward ray tracing* dan *backward ray tracing*. Pada *forward ray tracing*, sinar yang ditelusuri adalah sinar yang dipancarkan dari sumber cahaya. Satu hal yang harus diperhatikan adalah bahwa sinar yang dipancarkan oleh sumber cahaya tidak hanya berjumlah puluhan atau ratusan tetapi dapat berjumlah jutaan bahkan lebih. Semua sinar yang dipancarkan tersebut harus ditelusuri satu persatu. Bila setelah proses penelusuran dilakukan sinar yang sedang ditelusuri tersebut tidak mengenai mata, maka sinar tersebut akan diabaikan yang berarti akan banyak sekali perhitungan sia-sia yang dilakukan. Hal ini dikarenakan tidak semua sinar yang dipancarkan. dari sumber cahaya akan mengenai mata. Dengan menggunakan cara ini, maka untuk menghasilkan gambar yang diinginkan akan membutuhkan waktu yang banyak.(Sulian Mozes L. Sedubun, 2005).



Gambar 2.3 Forward Ray Tracing
(Sumber : Sulian Mozes L. Sedubun, 2005)

Sintese gambaran *Photo-realistic* secara luas digunakan di dalam banyak aplikasi seperti desain ilmu bangunan/arsitektur, rancangan industri, simulasi visual untuk reka bentuk lanskap, seni, dan seterusnya bagaimanapun, menghasilkan gambaran-gambaran foto realistis dengan mutu yang tinggi sangat memakan waktu.

Para peneliti dan ilmuwan telah mencoba menggunakan komputer paralel dan kelompok untuk mempercepat membuat gambaran-gambaran foto realistis. Ketika menggunakan komputasi kelompok, gambar itu dipecah jadi sejumlah bingkai/*frame* atau irisan-irisan baik oleh baris-baris atau kolom-kolom dan masing-masing komputer memandang suatu subset dari nomor sejumlah *frame*. *Frame-farme* itu dapat dikirim kepada masing-masing komputer di dalam potongan/bagian-bagiam yang berdekatan atau dalam satu perintah yang terselip di antara halaman. Di dalam satu kasus suatu *preview* (setiap frame ke-n{9,10,11, dst}), dari animasi itu dapat secara umum dianggap bahwa frame-frame itu sedang dikomputasi.

Kluster dasar, meskipun bermanfaat, tidak sampai cukup untuk memandang suatu gambar atau animasi yang sangat kompleks dalam

waktu yang singkat. Satu arah untuk mencapai suatu proses me-*render* yang lebih efisien adalah menggunakan kluster lebih dari satu: yaitu menerapkan teknologi komputasi Grid.

2.4.2 Pov-Ray dan Megapov

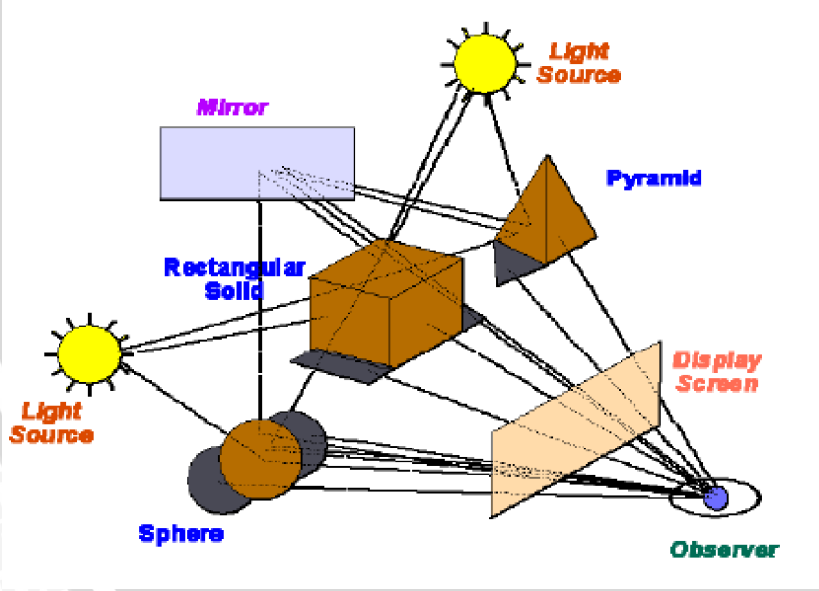
POV-RAY adalah kependekan untuk *Persistence of Vision Ray tracing*, suatu alat untuk menghasilkan grafik komputer bermutu tinggi. Program POV-Ray membuat tiga dimensi, memotret gambar realistis menggunakan suatu teknik pembuatan yang disebut *ray-tracing* (peniruan sinar). Teknik ini membaca suatu file teks berisi informasi yang menggambarkan objek, menerangi di suatu scene(tempat/bagian), dan menghasilkan satu gambar dari bagian itu dari sudut pandang itu dari suatu kamera yang juga digambarkan di dalam file teks. *Ray-tracing* adalah suatu metode menghitung intensif untuk menghasilkan gambar 3D, tetapi juga menghasilkan gambar bermutu sangat tinggi (lihat gambar 2.4)) dengan refleksi yang realistis, shading, perspektif dan efek-efek lain.

Gambar 2.4 Gambar hasil render POV-Ray
(Sumber : Balder Aljaber,dkk.2007)

Ray-tracing menghasilkan gambaran-gambaran dengan menjiplak satu berkas cahaya per piksel dari "kamera" ke dalam *scene* melalui

layar tampilan yang maya dan memotongnya dengan objek di dalam *scene* (lihat Gambar.2.5). Pada masing-masing persimpangan, kita mengikuti sinar sekunder, yang dihasilkan oleh cerminan/pemantulan, transmisi, dan bayangan-bayangan yang pada gilirannya tumpang tindih dengan objek lain, membangitkan lebih banyak sinar, dan seterusnya, untuk menghasilkan suatu sinar, yang biasanya mempunyai sekitar 10 tingkat kedalaman. Ketika pohon-sinar (*ray-tree*) diselesaikan, kita menentukan intensitas dan warna dari tiap piksel dengan menjumlahkan komponen-komponen dari dasar pohon-sinar.

Perulangan-perulangan ini 'mengikuti refleksi' memerlukan banyak persimpangan-persimpangan yang dihitung per piksel. POV-RAY dalam bentuk normal menggunakan hanya satu komputer untuk membuat gambar. Oleh karena hal ini, beberapa gambar mungkin diambil berjam-jam atau bahkan berhari-hari agar secara utuh membuat yang terbaru di PC. Animasi, sebagai suatu perluasan alami, tentu saja dapat memerlukan suatu waktu yang banyak.

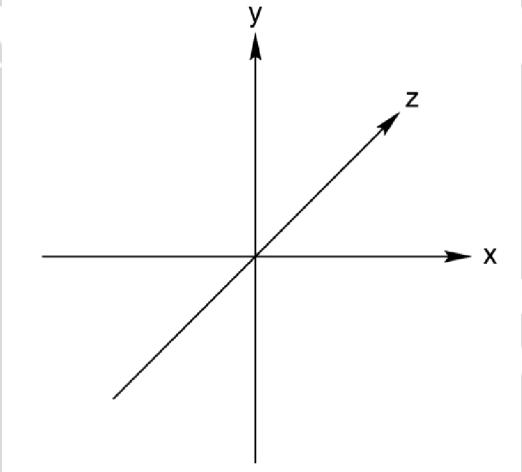


Gambar 2.5 Bagaimana POV-Ray bekerja
(Sumber : Balder Aljaber,dkk.2007)

Sedangkan Megapov sendiri adalah suatu modifikasi dan versi patch yang tidak resmi dari Pov-Ray. Sehingga beberapa fitur dan

fasilitas pada Pov-Ray mengalami perubahan dan penambahan. (MegaPOV-Team, 2005)

Pov-ray menggunakan bahasa pemrograman khusus yang disebut "*scene description language*". *Script* dapat diketik pada text editor yang disediakan POV-Ray, lalu POV-Ray akan membacanya dan menciptakan gambar dari proses render yang dilakukan. *Script* harus diketik secara *case sensitive*. Pada pov-ray digunakan koordinat 3D, sumbu y positif kearah atas, sumbu x positif ke arah kanan dan sumbu z positip ke arah dalam layar.



Gambar 2.6 Sistem koordinat POV-Ray
(Sumber : David K. Buck, Aaron A. Collins. 1999)

2.4.2.1 File Include Standar

Sebagian besar *script* yang dibuat pada POV-Ray akan menggunakan file *include* standar yang telah disediakan oleh *library* POV-Ray, walaupun sebenarnya bisa dibuat sendiri file *include* tambahan.

```
#include "colors.inc"
#include "stones.inc"
```

Kedua baris *script* diatas merupakan file *include*. Baris pertama berisi definisi untuk variabel warna sedangkan baris kedua berisi

kumpulan tekstur batu-batuan. Pada instalasi yang secara umum, file *include* akan berada di direktori \povray3\include.

2.4.2.2 Camera

Camera merupakan statemen dalam POV-Ray dimana dan bagaimana posisi sebuah tampilan dilihat atau ditempatkan. Hal ini akan menentukan posisi koordinat x,y dan z, yang biasanya disebut dengan vektor 3 dimensi.

```
camera {
    location <0, 2, -3>
    look_at  <0, 1, 2>
  }
```

`location <0,2,-3>` berarti bahwa lokasi objek akan ditempatkan pada koordinat x=0 , y=2 dan z= -3. Sedangkan `look_at <0,1,2>` berarti dilakukan rotasi terhadap posisi objek sebelumnya terhadap sumbu x=0 , sumbu y=1 dan sumbu z=2.

2.4.2.3 Object

Object merupakan deskripsi untuk sebuah objek atau benda dalam POV-Ray. *Object* bisa berupa lingkaran (bola), kotak (kubus) ataupun bentuk-bentuk yang lainnya.

```
  sphere {
    <0, 1, 2>, 2
    texture {
      pigment { color Yellow }
    }
  }
```

Script diatas menunjukkan deskripsi untuk menciptakan sebuah bentuk lingkaran (bola), dengan titik pusat (0,1,2) dan jari-jari 2 unit. Lingkaran tersebut memiliki tekstur dengan warna kuning

2.4.2.4 Texture

Texture merupakan statemen untuk mendeskripsikan tekstur pada suatu objek. *Pigment* digunakan untuk penggunaan *texture* warna dasar, beberapa warna bisa digunakan untuk *pigment*. Kata kunci yang digunakan untuk warna adalah *color*, dimana fungsi-fungsi warna yang tersedia terdapat pada file include color.inc.

Jika warna standar tidak ada, dapat dilakukan kombinasi dari warna standar yang sudah ada yaitu dari *red, green, blue*. Sebagai contoh jika ingin warna merah muda, dapat dikombinasikan dari ketiga warna tadi dengan komposisi `color red 1.0 green 0.8 blue 0.8`. Secara *script* dapat ditulis sebagai berikut :

```
color rgb <1.0, 0.8, 0.8>
```

2.4.2.5 Light

Pada awalnya layar akan berwarna hitam (gelap) sampai ditambahkan fungsi cahaya (light) pada *script* POV-Ray.

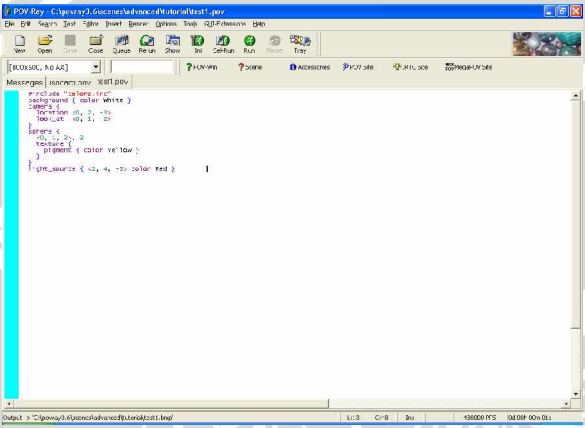
```
light_source { <2, 4, -3> color White}
```

light_source merupakan *statement* untuk memberi cahaya pada layar. `<2, 4, -3>` merupakan koordinat posisi sumber cahaya, sedangkan `color White` merupakan warna dari cahaya tersebut. Berikut sebuah contoh sederhana dari *script* POV-Ray dalam membuat sebuah gambar lingkaran (bola) dengan efek cahaya :

```
#include "colors.inc"
 background { color Cyan }
 camera {
   location <0, 2, -3>
   look_at  <0, 1,  2>
 }
 sphere {
  <0, 1, 2>, 2
  texture {
    pigment { color Yellow }
  }
```

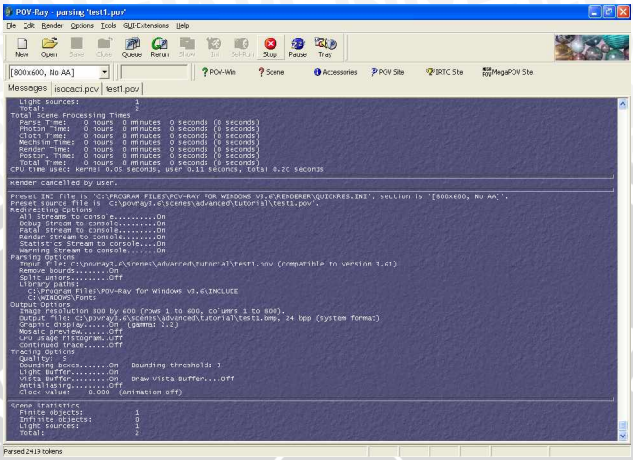
```
}
light_source { <2, 4, -3> color White}
```

Script tersebut diketik pada text editor yang dimiliki oleh pov-ray sendiri. Setelah selesai mengetik script-nya maka dengan menekan tombol run, proses render akan dilakukan.



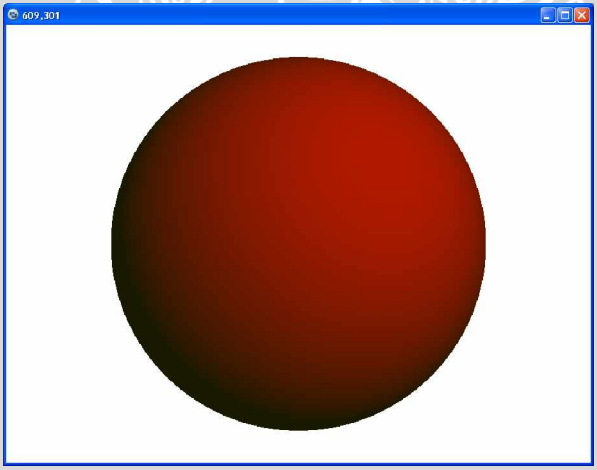
Gambar 2.7 Text editor POV-Ray

Jika proses render telah selesai, akan muncul form yang berisi gambar hasil render.



Gambar 2.8 Proses render

Berikut ini merupakan gambar hasil dari proses *render* yang telah dilakukan :



Gambar 2.9 Gambar hasil render

BAB III
METODOLOGI PENELITIAN

3.1 Subjek Penelitian

Subjek yang digunakan sebagai penelitian adalah beberapa buah PC yang terkoneksi dengan jaringan yang difungsikan untuk melakukan suatu kegiatan secara bersamaan.

3.2 Tempat dan Waktu Penelitian

Penelitian ini dilakukan di Karisma Komputer Jl. Watumujur I Malang selama bulan September 2007.

3.3 Perangkat Lunak dan Perangkat Keras Penelitian

3.3.1 Perangkat Lunak

Perangkat lunak yang digunakan pada penelitian ini adalah sebagai berikut:

- Sistem Operasi Windows XP sevices pack 2
- Alchemi 1.0.5
  Alchemi merupakan perangkat lunak yang digunakan untuk melakukan *grid computing*. Terdiri dari alchemi *manager* dan *executor*
- POV-Ray 3.6 dan Megapov 1.2.1
  POV-Ray dan megapov adalah sebuah perangkat lunak yang digunakan untuk proses *render* (*ray-tracking*).
- MySQL 5.0.21
  MySQL adalah sebuah manajemen database yang digunakan untuk menyimpanan data dari alchemi manager.
- Perangkat lunak pengolah data yang digunakan untuk mengolah data dan tabel menjadi diagram atau grafik.

3.3.2 Perangkat Keras

Perangkat keras yang digunakan pada penelitian ini adalah sebagai berikut :

- 5 buah PC Komputer sebagai *executor*
  Processor AMD Athlon XP 2000+ 1,66 GHZ
  256 MB RAM
  80 Gb HDD
  1 buah *ethernet card* Realtek RTL-8129 10/100 Mbps.
- 1 buah PC Komputer sebagai *manager*
  Processor AMD Athlon XP 2100+ 1,76 GHZ
  256 MB RAM
  80 Gb HDD
  1 buah *ethernet card* Realtek RTL-8129 10/100 Mbps.
- Kabel UTP secukupnya untuk menghubungkan perangkat yang ada sesuai topologi yang diinginkan.
- *Switch hub* 3 Com 8 port.

3.4 Setting dan Konfigurasi Sistem

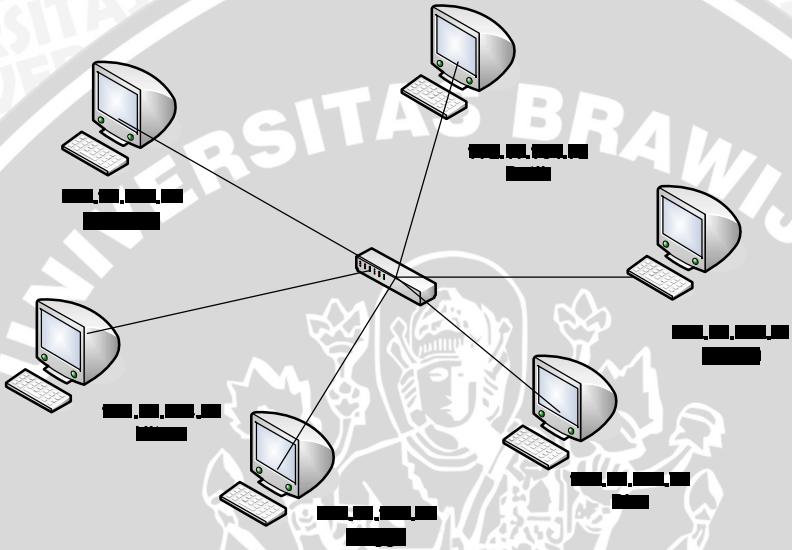
Setting dan konfigurasi sistem dari pemroses yang akan digunakan adalah 5 buah komputer yang terhubung dengan jaringan dengan topologi Star yaitu :

Tabel 3.1 Tabel spesifikasi komputer yang dipergunakan

No	Nama PC	Pemroses	Memory
1	Jingga	AMD Athlon XP 2000+	256 MB
2	Hitam	AMD Athlon XP 2000+	256 MB
3	Putih	AMD Athlon XP 2000+	256 MB
4	Biru	AMD Athlon XP 2000+	256 MB
5	Unggu	AMD Athlon XP 2000+	256 MB
6	Manager	AMD Athlon XP 2100+	256 MB

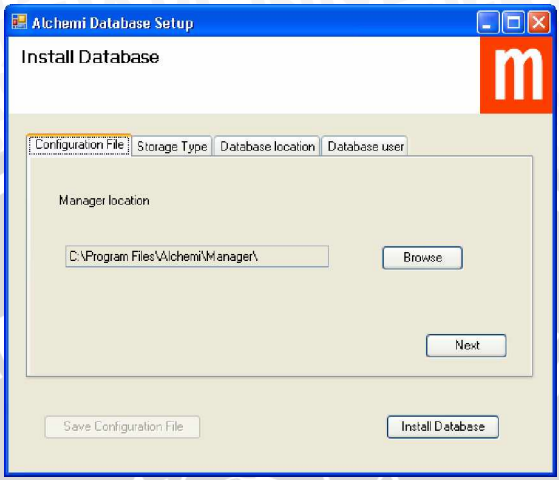
Dalam satu lingkungan jaringan komputer, eksekusi program pada saat yang berbeda dapat menghasilkan kinerja yang berbeda juga, hal ini disebebkan kondisi jaringan pada saat eksekusi program. Jika beban jaringan tinggi maka kinerja dari *grid computing* menjadi rendah dan tidak optimal, sebaliknya jika beban jaringan kecil maka

kinerja *grid computing* menjadi optimal. Mengingat hal tersebut maka diasumsikan dan dikondisikan beban jaringan kecil. konfigurasi jaringan yang digunakan adalah sebagai berikut



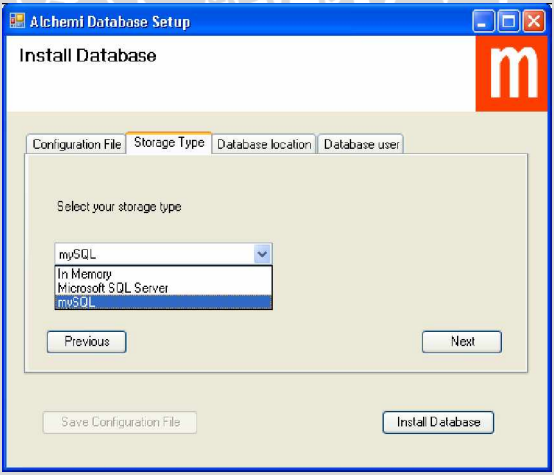
Gambar 3.1 Topologi jaringan untuk grid computing

Pada konfigurasi jaringan pada Gambar 3.1 yang bertindak sebagai server adalah komputer 'Manager' sedangkan komputer-komputer yang lain bertindak sebagai *client*. Pada komputer server dilakukan instalasi Alchemi *Manager* dan Alchemi *executor*, sedangkan pada komputer *client* hanya diinstall Alchemi *executor*. Setelah dilakukan instalasi pada komputer server maupun *client*, selanjutnya dilakukan konfigurasi koneksi database pada Alchemi *manager*. Untuk penelitian kali ini digunakan database MySQL. Secara detail untuk konfigurasi koneksi database dapat dilakukan sebagai berikut :



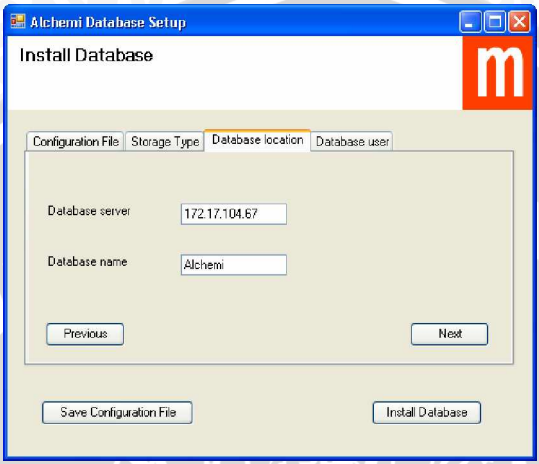
Gambar 3.2 Letak directory file Alchemi Manager

Setelah dilakukan klik dua kali pada file Alchemi.DbSetup.exe atau melalui start menu akan muncul tampilan seperti pada Gambar 3.2, hal tersebut merupakan konfigurasi letak directory yang didalamnya terdapat file Alchemi *Manager*. Klik tombol next jika letak *directory* sudah ditentukan.



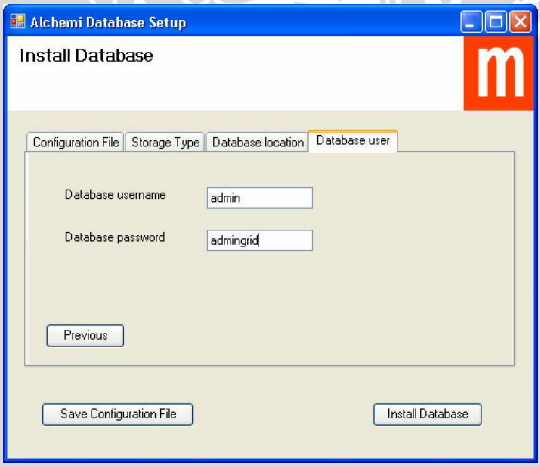
Gambar 3.3 Jenis penyimpanan data

Pada Gambar 3.3 diberikan pilihan untuk penyimpanan data. Penelitian ini penyimpanan dengan menggunakan database MySQL.



Gambar 3.4 Lokasi database

Gambar 3.4 merupakan konfigurasi untuk nama server database dan nama database yang digunakan. Untuk nama server bias menggunakan alamat IP ataupun nama host.



Gambar 3.5 Login database

Gambar 3.5 merupakan konfigurasi untuk login ke database. Login tersebut terdiri dari database *username* dan database *password*. Setelah selesai semua diisi maka tekan tombol '*install database*' dan tombol 'Save configurasi file'.

```
<?xml version="1.0" encoding="utf-8"?>
<Configuration
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<DbServer>172.17.104.67</DbServer>
<DbUsername>admin</DbUsername>
<DbPassword>admingrid<DbPassword />
<DbName>Alchemi</DbName>
<DbType>MySql</DbType>
<Id />
<ManagerHost />
<ManagerPort>0</ManagerPort>
<OwnPort>9000</OwnPort>
<ConnectVerified>false</ConnectVerified>
<InUse>false</InUse>
<Intermediate>false</Intermediate>
<Dedicated>false</Dedicated>
</Configuration>
```

Gambar 3.6 File konfigurasi koneksi database Alchemi Manager

Gambar 3.6 merupakan file hasil dari konfigurasi yang telah dilakukan. Pada file konfigurasi tersebut terdapat informasi-informasi tentang nama database, nama server, database username, database password dan informasi lainya.

3.4.1 Percobaan dengan eksekusi program standalone

Percobaan ini dilakukan untuk mengetahui bagaimana kinerja dan waktu komputasi dari proses render dengan menggunakan megapov. Pada percobaan ini akan dibandingkan dengan eksekusi program grid computing untuk mengetahui apakah dengan mengunakan grid compuitng akan diperoleh kinerja yang lebih bagus.

3.4.2 Percobaan dengan eksekusi program grid computing

Percobaan ini dilakukan untuk mengetahui bagaimana kinerja dan waktu komputasi dari proses render dengan megapov. Dengan menggunakan teknologi *grid computing*, proses render tersebut

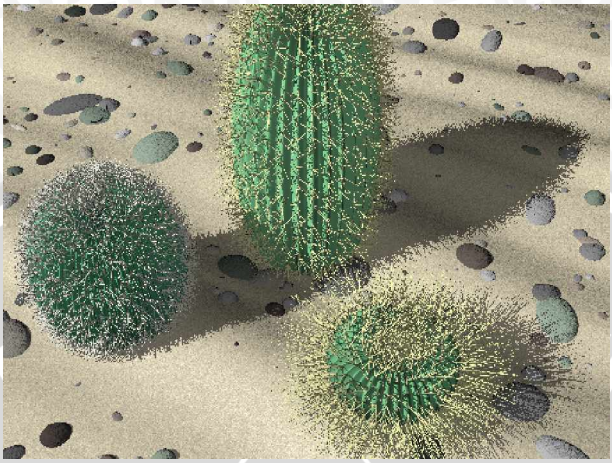
diatur dengan perangkat lunak Alchemi agar terjadi penggabungan sumber daya yang dimiliki banyak komputer yang terhubung dalam suatu jaringan sehingga terbentuk suatu kesatuan sistem komputer dengan sumber daya komputasi yang besarnya mendekati jumlah sumber daya komputasi dari komputer-komputer yang membentuknya. Pada percobaan ini akan dibandingkan dengan eksekusi program *standalone* dengan eksekusi program *grid computing*.

3.4.3 Percobaan dengan jumlah komputer yang bervariasi dari 2, 3, 4, 5 dan 6

Percobaan ini dilakukan untuk mengetahui apakah dengan menambah host maka komputasinya menjadi semakin cepat dan apakah hubungan antara waktu komunikasi, waktu proses, granularitas proses, dan speedup dengan jumlah host yang berbeda-beda.

3.5 Data percobaan

Pada percobaan ini akan digunakan beberapa data yang akan dilakukan render terhadapnya. Data tersebut nantinya akan menghasilkan sebuah gambar tiga dimensi yaitu, sebuah gambar kaktus yang mana merupakan objek yang berbentuk tiga buah kaktus yang berada di padang pasir. File yang digunakankan bernama kaktus.pov. Gambar 3.7 merupakan hasil render dari file kaktus.pov.



Gambar 3.7 Hasil render dari kaktus.pov

3.6 Desain Alchemi Renderer

3.6.1 Struktur Data Alchemi Renderer

Alchemi renderer merupakan aplikasi yang akan melakukan pengaturan proses render dengan menggunakan megapov dimana proses tersebut dilakukan secara *grid computing*. Pada alchemi renderer terdapat dua bagian yaitu alchemi renderer dan alchemi library. Didalam alchemi renderer terdapat renderer form yang merupakan bagian untuk tampilan form aplikasi dan terdapat menu untuk menjalankan proses render serta membuat koneksi ke alchemi manager, sedangkan pada alchemi library terdapat renderer thread yang merupakan bagian untuk menciptakan thread yang akan dikirim ke alchemi manager dan membuat file *temporary* untuk proses render tersebut. Adapun struktur data yang digunakan untuk renderer form terdapat pada Gambar 3.8.

```
public class RendererForm : Form
{
        private PictureBox pictureBox1;
        private Button render;
        private Label label1;
        private Label label2;
        private Label label3;
        private Label label4;

        private int imageWidth = 0;
        private int imageHeight = 0;
        private int columns = 0;
        private int rows = 0;
        private int segmentWidth = 0;
        private int segmentHeight = 0;

        private GApplication ga = null;
        private bool initted = false;

        private Bitmap composite = null;
        private String modelPath = "";
        private String[] paths = null;
        private bool drawnFirstSegment = false;

        private ComboBox widthCombo;
        private ComboBox heightCombo;
        private NumericUpDown columnsUpDown;
        private NumericUpDown rowsUpDown;
        private Label label5;
        private ComboBox modelCombo;
        private Button stop;
        private CheckBox stretchCheckBox;

        private static readonly ILog logger =
LogManager.GetLogger(MethodBase.GetCurrentMethod().Declari
ngType);

        private ProgressBar pbar;
        private Label lbProgress;

        private string basepath;
        private Container components = null;
 };
```

Gambar 3.8 Struktur data renderer form

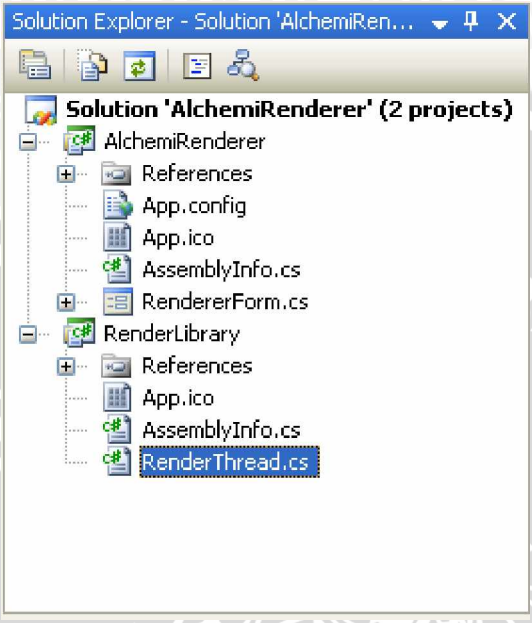
Dari Gambar 3.8 tentang struktur data alchemi renderer, diketahui variabel *imageWith* untuk mendefinisikan nilai awal lebar dari gambar hasil render, *imageHeight* untuk mendefinisikan nilai awal panjang dari gambar hasil render.

Sedangkan struktur data untuk renderer thread terdapat pada Gambar 3.9 sebagai berikut :

```
public class RenderThread : GThread
	{
		private int _startRowPixel;
		private int _startColPixel;
		private int _endRowPixel;
		private int _endColPixel;

		private string _inputFile;
		private string _megaPOV_Options;

		private int _imageWidth;
		private int _imageHeight;

		private int _segWidth;
		private int _segHeight;

		private int _col;
		private int _row;

		private Bitmap crop;

		private string _basePath;
	};
```

Gambar 3.9 Struktur data renderer thread

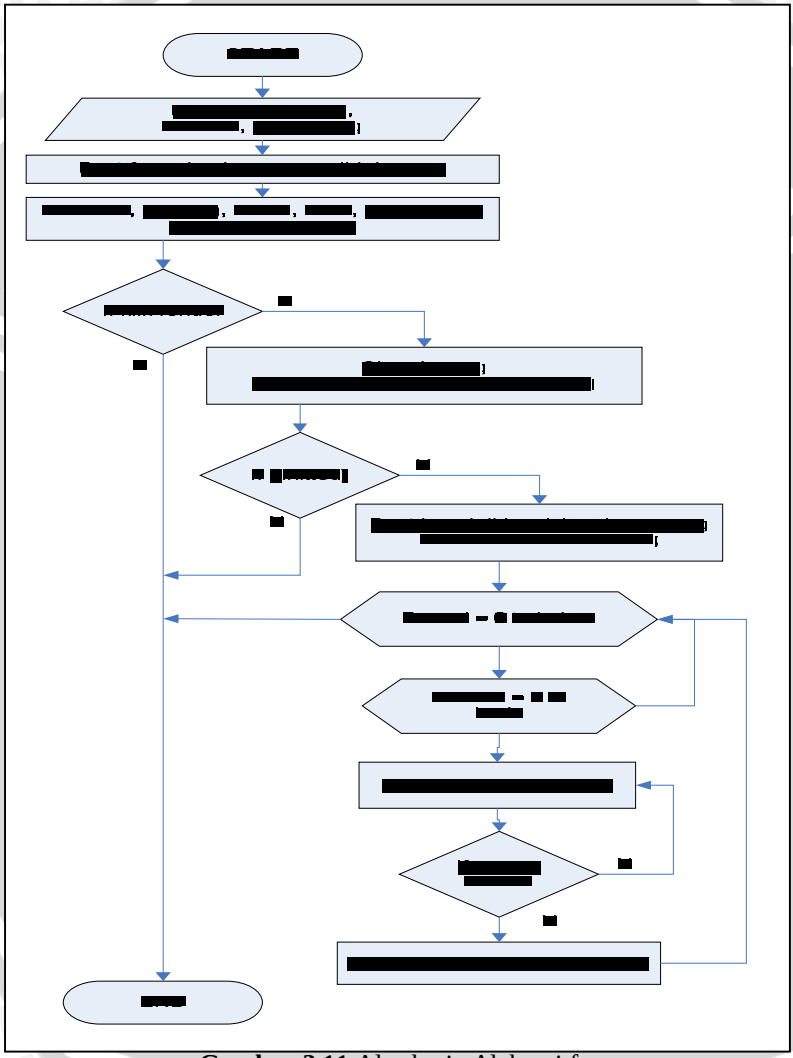
Pada Gambar 3.9, variabel *_startRowPixel* merupakan nilai awal dari pixel baris. Variabel *_startColPixel* merupakan nilai awal dari pixel kolom. Variabel *_endRowPixel* merupakan nilai akhir dari pixel baris. Variabel *_endColPixel* merupakan nilai awal dari pixel kolom. Variabel *_inputFile* merupakan nama dari input file yang akan direnderer. Variabel *_megaPov_Options* merupakan konfigurasi dari megapov. Variabel *_imageWidth* merupakan nilai lebar dari gambar hasil render. Variabel *_imageHeight* merupakan nilai panjang dari gambar hasil render. Variabel *_segWidth* merupakan nilai lebar dari segmen render. Variabel *_segHeight* merupakan nilai panjang dari segmen render. Variabel *_col* merupakan nilai kolom dan *_row* merupakan nilai baris. Variabel *crop* merupakan bitmap dari hasil segmen render. Variabel *_basePath* merupakan lokasi directory dimana hasil proses render berada.



Gambar 3.10 Alchemi form dan Alchemi thread

3.6.2 Alur kerja Alchemi Renderer

Alur kerja dari alchemi renderer terbagi menjadi dua yaitu alur kerja untuk alchemi form dan alur kerja untuk alchemi thread. Alur kerja untuk alchemi form terdapat pada Gambar 3.11.

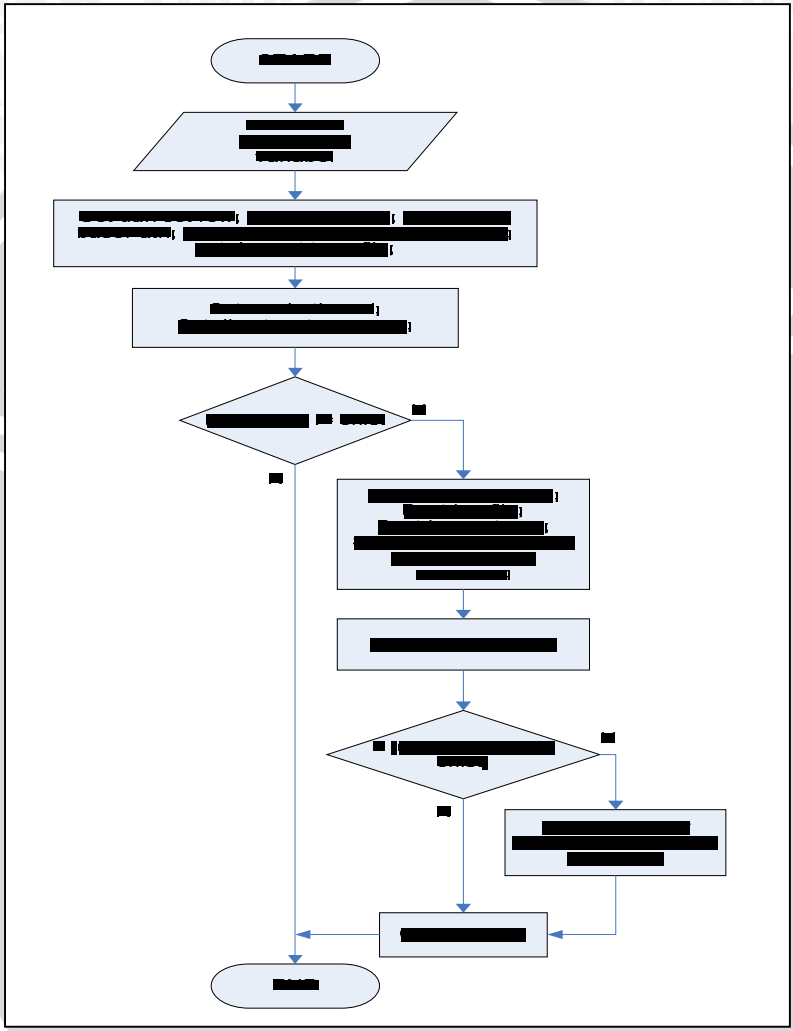


Gambar 3.11 Alur kerja Alchemi form

Berikut ini adalah penjelasan alur kerja dari renderer form :

- Inisialisasi system , variabel-variabel yang dibutuhkan dan komponen yang terdapat pada form tersebut.
- Membuat tampilan form beserta komponen-komponen yang terdapat didalamnya.
- Melakukan setting nilai lebar, panjang, besar kolom, besar baris, ukuran pixel dan nama file input yang akan dilakukan proses render.
- Melakukan koneksi ke alchemi manager, kemudian membuat thread render yang baru.
- Melakukan perulangan dari kolom sama dengan nol sampai kolom terakhir, baris sama dengan nol sampai baris terkahir. Kemudian menjalankan proses render (thread render)
- Jika proses render sudah selesai maka hasil render akan ditampilkan pada form.
- Akhir program.

Adapun untuk alur kerja renderer thread terdapat pada Gambar 3.12 sebagai berikut :



Gambar 3.12 Alur kerja Alchemi thread

Berikut ini adalah penjelasan alur kerja dari renderer thread :

- Inisialisasi system , variabel-variabel yang dibutuhkan.
- Melakukan set dan get terhadap nilai-nilai dari row, col, basepath, renderimagesegment dan tempfile.
- Melakukan setting terhadap render thread dan lokasi direktori *temporary*.
- Jika direktori belum ada maka dilakukan pembuatan direktori baru. Membuat log file, membuat *image temporary*, setting direktori lokasi megapov dan setting konfigurasi dari megapov
- Memanggil megapov dan menjalankannya untuk melakukan proses render.
- Jika nama file output ada maka proses menggambar dilakukan dan hasilnya disimpan pada file output.
- Setelah proses semua selesai, tutup file log.
- Akhir program.

3.7 Hasil dan Analisa

3.7.1 Form Untuk Pengambilan Data

Form-form yang digunakan untuk pencatatan data hasil percobaan adalah sebagai berikut :

Tabel 3.2 Contoh tabel untuk data standalone

Ukuran gambar	Waktu eksekusi (detik)
320x200	
480x320	
640x480	
800x600	
1024x800	

Tabel 3.2 merupakan contoh tabel untuk menyimpan data waktu komputasi secara standalone, dimana proses render dilakukan hanya pada satu komputer.

Tabel 3.3 Contoh tabel untuk data waktu eksekusi render secara grid pada 2, 3, 4, 5, dan 6 host.

	Waktu eksekusi (detik)				
Ukuran gambar	1x1	2x2	3x3	4x4	5x5
320x200					
480x320					
640x480					
800x600					
1024x800					

Tabel 3.3 merupakan tabel contoh untuk menyimpan data hasil dari proses render secara grid. Adapun data-data yang akan diambil dalam percobaan sebagai berikut :

a. *Standalone*
   Data yang diambil adalah data waktu komputasi. Dimana data tersebut berdasarkan beberapa resolusi yaitu : 320 x 200, 480x 320, 640 x 480, 800x 600 dan 1024 x 800.
b. *Grid computing*
   Data yang diambil hampir sama dengan standalone yaitu data waktu komputasi, overhead manager, waktu komunikasi antara executor dengan manager. Namun data tersebut diambil berdasarkan :
   - § Resolusi : 320 x 200, 480x 320, 640 x 480, 800x 600 dan 1024 x 800
   - § Jumlah komputer : 2, 3, 4, 5 dan 6
   - § Ukuran baris dan kolom : 1x1, 2x2, 3x3, 4x4 dan 5x5. Baris dan kolom merupakan segmen untuk membagi gambar yang akan dirender, seperti baris dan kolom dalam sebuah matriks.

Tabel 3.4 Contoh tabel untuk data Overhead, Granularitas dan waktu Komunikasi grid dg 2, 3, 4, 5, dan 6 PC

Ukuran gambar	Segmen														
	1 x 1			2x2			3x3			4x4			5x5		
	O	G	WK	O	G	WK	O	G	WK	O	G	WK	O	G	WK
320x200															
480x320															
640x480															
800x600															
1024x800															

Tabel 3.4 digunakan untuk menyimpan overhead manager, waktu komunikasi executor dan Granularitas Executor. Data glanularitas diperoleh melalui perhitungan dengan rumus yang sudah ditentukan.

Tabel 3.5 Contoh tabel untuk data speedup pada grid dg 2, 3, 4, 5, dan 6 PC

Ukuran gambar	Speedup				
	1x1	2x2	3x3	4x4	5x5
320x200					
480x320					
640x480					
800x600					
1024x800					

Tabel 3.5 digunakan untuk menyimpan data perhitungan speedup.

Tabel 3.6 Contoh tabel untuk data efisiensi pemroses pada grid dg 2, 3, 4, 5, dan 6 PC

Ukuran gambar	efisiensi				
	1x1	2x2	3x3	4x4	5x5
320x200					
480x320					
640x480					
800x600					
1024x800					

Tabel 3.6 digunakan untuk menyimpan data hasil perhitungan efisiensi pemroses

Tabel 3.7 Contoh tabel untuk data cc ratio pada grid dg 2, 3, 4, 5, dan 6 PC

Ukuran gambar	cc ratio				
	1x1	2x2	3x3	4x4	5x5
320x200					
480x320					
640x480					
800x600					
1024x800					

Tabel 3.7 digunakan untuk menyimpan data hasil perhitungan cc ratio.

3.7.2 Analisis Hasil Percobaan

Dari hasil pengujian tersebut akan dianalisa berdasarkan waktu komputasi dan penggunaan *resource* komputer. Berikut adalah parameter analisa data:

- Waktu Proses

Diperoleh dari hasil pengujian dan akan dilihat perbandingan rata-rata yang diproleh untuk render gambar standalone (tanpa alchemi), render gambar secara *grid computing* dengan satu komputer, dua

komputer, tiga komputer, empat komputer dan lima komputer dengan cara mencatat waktu awal proses dan waktu akhir dari proses. Dengan rumus:

$$Tp = T2 - T1 \qquad \text{…….3.1}$$

T adalah waktu proses, *T1* adalah waktu awal proses, sedangkan T2 adalah waktu akhir proses.

- Overhead proses manager

Waktu yang diperlukan proses manager untuk membuat dan mengirimkan data ke proses *executor*. Overhead proses manager diperoleh dengan cara mencatat waktu sebelum proses pembuatan thread dan mencatat waktu apabila thread telah selesai dijalankan dengan rumus :

$$Tovr = T2 - T1 \qquad \text{…….3.2}$$

Dimana *Tovr* merupakan waktu Overhead proses manager, *T2* adalah waktu setelah thread dijalankan dan *T1* adalah waktu sebelum proses pembuatan thread.

- Speedup

Diperoleh dengan membandingkan waktu proses render standalone dengan waktu *grid computing* pada *p* pemroses untuk data yang sama. Bila *T(n)* didefinisikan sebagai waktu yang diperlukan oleh proses *standalone* untuk memecahkan masalah dengan data *n* dan *Tp(n)* sebagai waktu yang diperlukan menggunakan *p* pemroses maka

$$sp(n) = \frac{T(n)}{Tp(n)}, \qquad \text{…….3.3}$$

Speedup yang dicapai dapat melebihi jumlah pemroses yang digunakan. Untuk speedup ini akan dibandingkan antara speedup dengan menggunakan satu komputer, dua komputer, tiga komputer, empat komputer dan lima komputer.

- Efisiensi Pemroses

Efisiensi diukur dengan membagi speedup yang diperoleh dengan jumlah pemroses yang digunakan. Berdasarkan definisi speedup

diatas, maka efisiensi *Ep(n)* pada sistem dengan *p* pemroses bias diformulasikan sebagai berikut:

$$Ep(n) = \frac{Sp(n)}{p} = \frac{T(n)}{pTp(n)} \qquad \text{…….3.4}$$

Effisiensi maksimum dicapai disaat seluruh *p* elemen pemrosesan benar-benar bekerja penuh selama periode eksekusi.Efisiensi pemroses akan dibandingkan antara efisiensi pemroses dengan satu komputer, dua komputer, tiga komputer, empat komputer dan lima komputer.

- Granularitas *Executor*

Diperoleh dengan menghitung waktu komputasi yang ada diproses *executor*.dengan rumus:

$$Tgran = \frac{\sum Texecutor}{nexecutor} \qquad \text{…….3.5}$$

dimana Tgran adalah waktu granularitas, $\sum Texecutor$ adalah waktu proses dari seluruh *executor* dan *nexecutor* adalah banyaknya *executor* yang menangani pekerjaan.

- CC Ratio

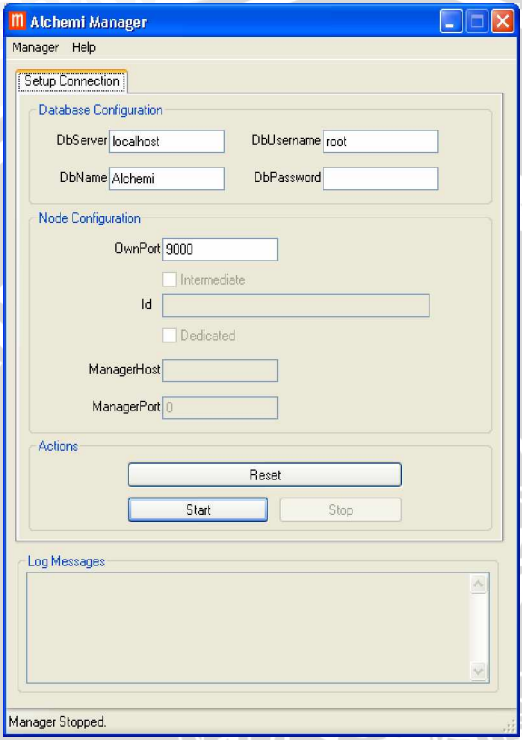
Computation-Comunication Ratio (CC Ratio) adalah nilai perbandingan antara total waktu komputasi dengan total waktu komunikasi. (MIT Press, 1995)

3.8 Menjalankan Aplikasi

Sebelum proses render dilakukan terlebih dahulu yang harus dilakukan adalah menjalankan database yang digunakan oleh alchemi manager, untuk percobaan ini digunakan MySQL 5 sebagai database, melakukan instalasi pada masing – masing komputer client aplikasi dari alchemi executor dan yang terakhir melakukan instalasi pov-ray pada masing – masing komputer client.

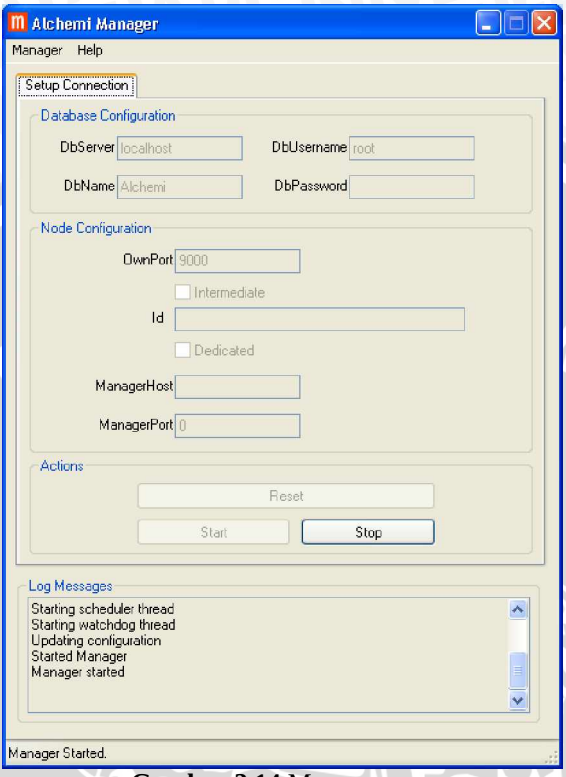
Selanjutnya jalankan aplikasi Alchemi manager, akan muncul tampilan seperti pada Gambar 3.13. Beberapa parameter yang harus

diisi adalah nama database server, nama database, nama user database, password database dan port.



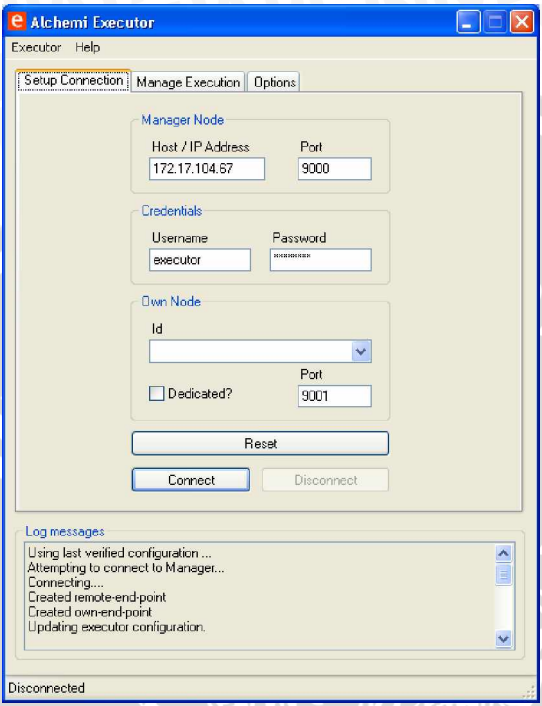
Gambar 3.13 Alchemi manager

Setelah semua parameter diisi, tekan tombol start untuk menjalankan aplikasi manager. Aplikasi manager akan melakukan koneksi ke database server. Pada Gambar 3.14 menunjukkan aplikasi menajer sedang berjalan.



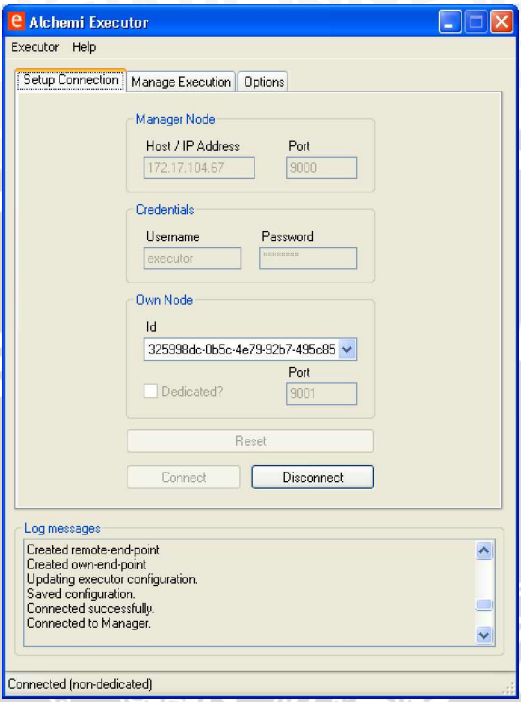
Gambar 3.14 Manager start

Setelah aplikasi manager sudah berjalan, kemudian pada masing – masing komputer client, jalankan aplikasi alchemi executor. Tampilan dari alchemi executor dapat dilihat pada Gambar 3.15.



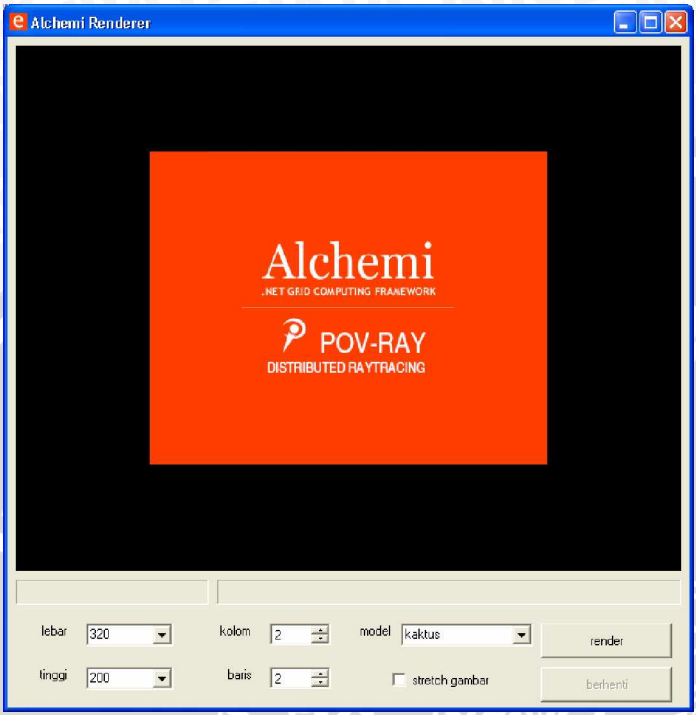
Gambar 3.15 Alchemi executor

Beberapa parameter yang harus diisi adaah nama host atau alamat IP komputer server(manager), port yang digunakan, nama pengguna, password, apakah *dedicated* atau tidak. Selanjutnya tekan tombol connect agar executor dapat terhubung ke manager. Pada Gambar 3.15 menunjukkan aplikasi executor sudah terhubung dengan manager.



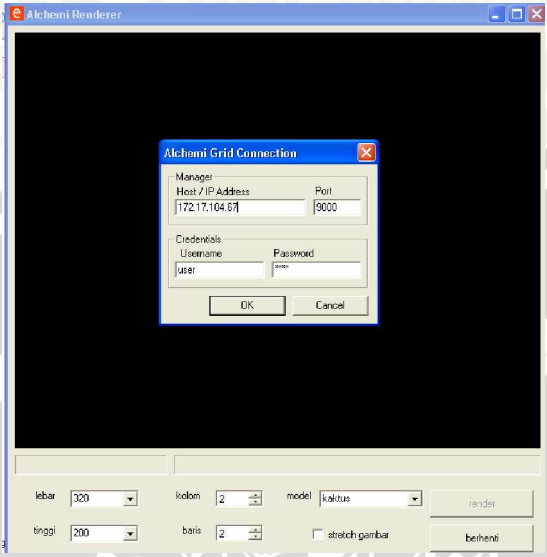
Gambar 3.16 Executor terhubung dengan manager

Jika semua aplikasi executor sudah dijalankan, selanjutnya dapat dilakukan proses render. Jalankan aplikasi alchemi pov-ray render. Tampilan alchemi pov-ray render tampak pada Gambar 3.17 beberapa parameter yang harus diisi adalah lebar gambar, tinggi gambar, kolom segmen, baris segmen, nama model yang akan dilakukan render, dan apakah tampilan haisl gambar dalam bentuk stretch atau tidak.



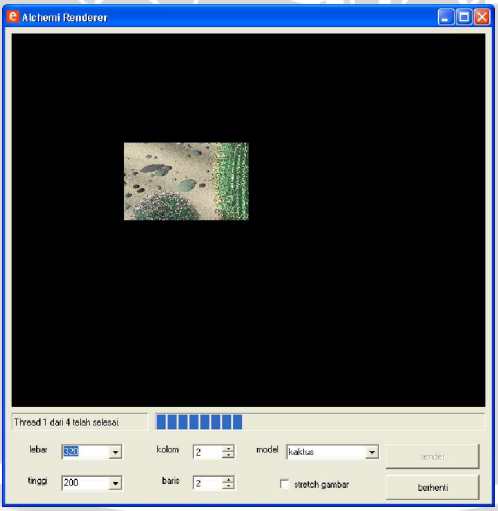
Gambar 3.17 Alchemi POV-Ray render

Setelah semua parameter diisi, selanjutnya untuk menjalankan proses render tekan tombol render maka akan muncul tampilan seperti pada Gambar 3.18 muncul form dialog yang berisikan input parameter kemana manager mana aplikasi achemi POV-Ray render akan terhubung. Adapun inputan yang harus diisi, nama host atau alamat IP, port yang digunakan, nama user dan password.



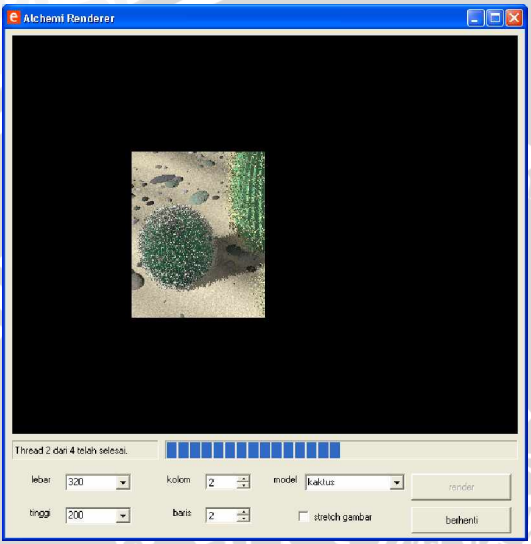
Gambar 3.18 Terhubung ke manager

Apabila aplikasi alchemi POV-Ray render sudah terhubung ke manager, proses render akan dijalankan. Tampak pada Gambar 3.19 proses render yang sedang berjalan ketika thread 1 sudah selesai, dimana terdapat 4 thread karena segmen yang dipilih sebanyak 2x2.

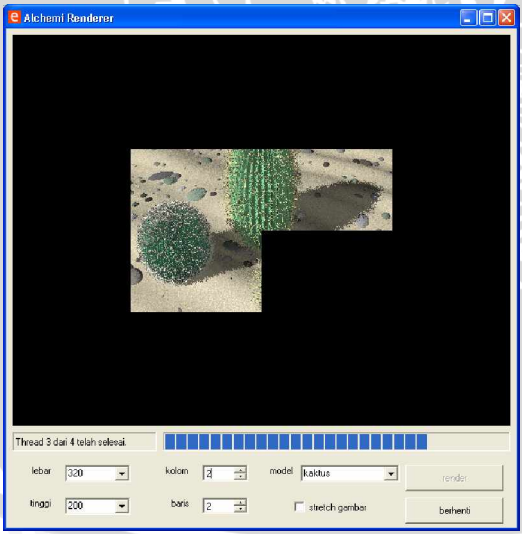


Gambar 3.19 Thread 1 dari 4 untuk render segmen 2x2

Pada Gambar 3.20 dapat dilihat bahwa proses render sedang berjalan dan thread kedua telah selesai dijalankan.

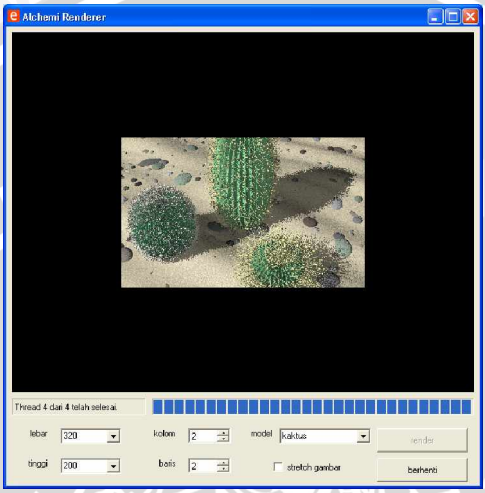


Gambar 3.20 Thread 2 dari 4 untuk render segmen 2x2



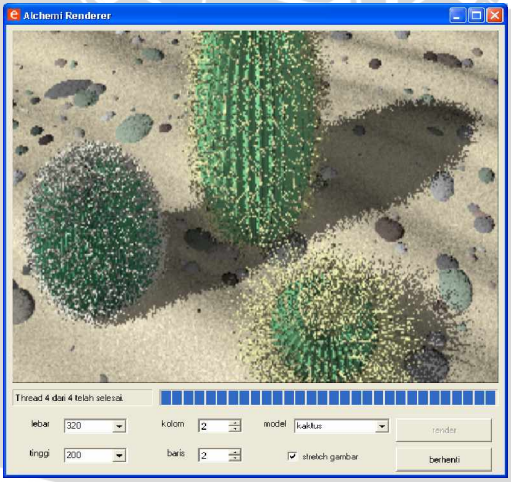
Gambar 3.21 Thread 3 dari 4 untuk render segmen 2x2

Pada Gambar 3.21 thread ketiga telah berhasil dijalankan, sedangkan pada Gambar 3.22 semua thread telah selesai dijalankan maka dapat dilihat gambar dari hasil render tersebut.



Gambar 3.22 Thread 4 dari 4 untuk render segmen 2x2

Agar tampilan gambar dapat terlihat penuh pada layar, dapat dipilih parameter stretch gambar, sebagaimana tampak pada Gambar 3.23



Gambar 3.23 Stretch gambar

BAB IV
IMPLEMENTASI DAN PEMBAHASAN

4.1 Implementasi

Implementasi POV-Ray terdistribusi pada Alchemi dibuat dengan menggunakan bahasa pemrograman C#. Dalam implementasinya terdapat beberapa *class* dan *method* yang dibuat untuk penyelesaiannya. Terdapat dua *class* yaitu *class* RenderThread dan *class* RendererForm.

4.1.1 Deskripsi Program

4.1.1.1 Class RenderThread

Class RenderThread memiliki fungsi untuk membuat *thread*, dimana thread tersebut yang akan melakukan proses *render* dengan menjalankan pov-ray secara *background*. Beberapa *method* penting yang ada pada class ini adalah *RenderThread, Start* dan beberapa *method* inisialisasi (*row,col, BasePath, RenderedImageSegment, TempFile*).

4.1.1.1.1 Method inisialisasi row,col, BasePath, RenderedImageSegment dan TempFile

Method ini berisi tentang inisialisasi fungsi-fungsi dalam mengambil nilai ataupun mengeset nilai dari *row,col, BasePath, RenderedImageSegment* dan *TempFile*.

```
public int Row
{
        get
        {
                return _row;
        }
        set
        {
                _row = value;
        }
}
```

```
public int Col
{
        get
        {
                return _col;
        }
        set
        {
                _col = value;
        }
}
public string BasePath
{
        get
        {
                return _basePath;
        }
        set
        {
                _basePath = value;
        }
}
public Bitmap RenderedImageSegment
{
        get
        {
                return crop;
        }
}
public string TempFile
{
        get
        {
                string fileToSave = Col+"_"+Row+".png";
                return fileToSave;
        }
}
```

4.1.1.1.2 Method RenderThread

Method ini berisi tentang fungsi untuk melakukan setting nilai dari variabel-variabel yang dibutuhkan oleh sebuah thread.

```
public RenderThread(string InputFile, int ImageWidth, int
ImageHeight, int SegmentWidth, int SegmentHeight, int
StartRow, int EndRow, int StartCol, int EndCol, string
MegaPOV_Options)
{
        this._inputFile = InputFile;
        this._imageWidth = ImageWidth;
        this._imageHeight = ImageHeight;
        this._segWidth = SegmentWidth;
        this._segHeight = SegmentHeight;
        this._startRowPixel = StartRow;
        this._endRowPixel = EndRow;
        this._startColPixel = StartCol;
        this._endColPixel = EndCol;
        this._megaPOV_Options = MegaPOV_Options;
}
```

4.1.1.1.3 Method Start

Method ini berisi tentang fungsi untuk melakukan proses *render* dengan menjalankan POV-Ray. File gambar hasil *render* akan disimpan didalam file sementara, lalu file tersebut akan dikirim ke manager.

```
public override void Start()
{
        string tempDir = Path.Combine(_basePath,
Path.GetFileName(WorkingDirectory));
        if (!Directory.Exists(tempDir))
                        Directory.CreateDirectory(tempDir);
        StreamWriter log =
File.CreateText(tempDir+"/tempLog_"+Col+"_"+Row+".txt");
        log.WriteLine("Direktori kerja = "+WorkingDirectory);
        log.WriteLine("Direktori Temporary = "+tempDir);
        string megaPOV_outputFilename = Path.Combine(tempDir,
Col+"_"+Row+"_tempPOV.png");
        string cmd = "cmd";
        string args = "/C " + Path.Combine(_basePath,
"bin/mpengine.exe") +
        string.Format(" +I{0} +O{1} +H{2} +W{3} +SR{4} +ER{5}
+SC{6} +EC{7} +FN16 {8}", _inputFile, megaPOV_outputFilename,
_imageHeight, _imageWidth, _startRowPixel, _endRowPixel,
_startColPixel, _endColPixel, _megaPOV_Options );
        log.WriteLine("Perintah untuk proses = :"+cmd);
        log.WriteLine("Argumen untuk proses = :"+args);
        Process megapov = new Process();
        megapov.StartInfo.FileName = cmd;
        megapov.StartInfo.Arguments = args;
```

```
        megapov.StartInfo.WorkingDirectory =
 WorkingDirectory;
        megapov.StartInfo.WindowStyle =
 ProcessWindowStyle.Hidden;
        megapov.StartInfo.UseShellExecute = true;
        megapov.StartInfo.CreateNoWindow = true;
        megapov.StartInfo.RedirectStandardError = false;
        megapov.StartInfo.RedirectStandardOutput = false;
        megapov.Start();
        megapov.WaitForExit();

        int x = (Col-1)*_segWidth;
        if (File.Exists(megaPOV_outputFilename))
```

4.1.1.2 Class RendererForm

Class RenderThread berisi method-method yang terkait dengan tampilan form , koneksi ke manager, pengaturan input untuk *render* dan menampilkan hasil dari *render*.

4.1.1.2.1 Method RendererForm

Method ini berfungsi untuk melakukan inisialisasi terhadap komponen yang dibutuhkan oleh form dan penanganan log.

```
public RendererForm()
{
        InitializeComponent();
        Logger.LogHandler += new LogEventHandler(LogHandler);
}
```

4.1.1.2.2 Method DisplayImage

Method ini berfungsi untuk menampilkan hasil gambar yang diperoleh dari proses *render*.

```
private void displayImage(Bitmap segment, int col, int row)
{
        if (!drawnFirstSegment)
        {
                clearImage();
                drawnFirstSegment = true;
        }

        Graphics g = Graphics.FromImage(composite);
        Rectangle sourceRectangle;
        Rectangle destRectangle;
        int x = 0;
        int y = 0;
        x = (col-1)*segmentWidth;
        y = (row-1)*segmentHeight;

        logger.Debug("Tampilkan segmen c, r: "+col+", "+row);
        try
        {
                sourceRectangle = new Rectangle(0, 0,
        segment.Width, segment.Height);
                destRectangle = new Rectangle(x, y,
        segment.Width, segment.Height);
                g.DrawImage(segment, destRectangle,
        sourceRectangle, GraphicsUnit.Pixel);
        }
        catch (Exception e)
        {
                logger.Debug("!!!ERROR:\n"+e.StackTrace);
        }
        pictureBox1.Image = composite;
}
```

4.1.1.2.3 Method ClearImage

Method ini berfungsi untuk membersihkan tampilan gambar sebelumnya pada form.

```
private void clearImage()
{
        if (imageWidth > 0 && imageHeight > 0)
        {
                composite = new
Bitmap(imageWidth,imageHeight);
                if (stretchCheckBox.Checked)
                {
                        pictureBox1.SizeMode =
PictureBoxSizeMode.StretchImage;
                }
                else
                {
                        pictureBox1.SizeMode =
PictureBoxSizeMode.CenterImage;
                }
                pictureBox1.Image = composite;
        }
}
```

4.1.1.2.4 Method Main

Method ini berfungsi untuk menampilkan form ketika program dijalankan.

```
static void Main()
{
        Application.Run(new RendererForm());
}
```

4.1.1.2.5 Method RenderForm_Load

Method ini berfungsi untuk mengeset variabel-variabel yang ada pada form saat form diload. Variabel tersebut terdiri antara lain resolusi, segmen, basepath, model yang akan dilakukan proses *render*.

```
private void RenderForm_Load(object sender, EventArgs e)
{
        AppDomain.CurrentDomain.UnhandledException += new
UnhandledExceptionEventHandler(DefaultErrorHandler);
        Application.ThreadException += new
ThreadExceptionEventHandler(Application_ThreadException);
        widthCombo.Items.AddRange(new object[] {"100", "160",
"200", "240", "320", "480", "512", "600", "640", "800",
"1024", "1280"});
        heightCombo.Items.AddRange(new object[] {"100",
"120", "200", "240", "320", "384", "480", "600", "640",
"768", "800", "1024"});
```

```
        widthCombo.SelectedIndex = 4;
        heightCombo.SelectedIndex = 3;

        columnsUpDown.Value = 4;
        columnsUpDown.Maximum = 1000;
        columnsUpDown.Minimum = 1;
        rowsUpDown.Value = 4;
        rowsUpDown.Maximum = 1000;
        rowsUpDown.Minimum = 1;

        basepath = "C:/povray3.6";

        paths = new String[]
{basepath+"/scenes/advanced/catur.pov
+L"+basepath+"/include",

        basepath+"/scenes/advanced/kaktus.pov
+L"+basepath+"/include",
        basepath+"/scenes/advanced/bola.pov
+L"+basepath+"/include",
        basepath+"/scenes/advanced/gelas.pov
+L"+basepath+"/include",
                        basepath+"/scenes/advanced/kotak.pov
+L"+basepath+"/include",
        };

        modelCombo.Items.AddRange(new object[]
{"catur","kaktus", "bola","gelas","kotak"});

        modelCombo.SelectedIndex = 0;
}
```

4.1.1.2.6 Method render_Click

Method ini dijalankan ketika pengguna melakukan klik pada tombol *render*. Adapun isi dari method ini sebagai berikut :

- mengambil nilai modelpath, nilai resolusi gambar(lebar dan tinggi gambar), nilai segmen(baris dan kolom)
- menulis informasi awal pada log
- menampilkan form dialog untuk koneksi ke *manager*

- melakukan koneksi ke manager
- membuat thread sebanyak segmen
- mengirim thread ke *manager* agar dikirim ke *executor*

```
private void render_Click(object sender, EventArgs e)
{
        stop.Enabled = true;
        render.Enabled = !stop.Enabled;
        drawnFirstSegment = false;
        showSplash();
        modelPath = paths[modelCombo.SelectedIndex];
        imageWidth =
Int32.Parse(widthCombo.SelectedItem.ToString());
imageHeight =
Int32.Parse(heightCombo.SelectedItem.ToString());
        columns = Decimal.ToInt32(columnsUpDown.Value);
        rows = Decimal.ToInt32(rowsUpDown.Value);
        segmentWidth = imageWidth/columns;
        segmentHeight = imageHeight/rows;
        int x = 0;
        int y = 0;
        logger.Debug("LEBAR:"+imageWidth);
        logger.Debug("TINGGI:"+imageHeight);
        logger.Debug("KOLOM:"+columns);
        logger.Debug("BARIS:"+rows);
        logger.Debug(""+modelPath);
        clearImage();

        if (!initted)
        {
                GConnectionDialog gcd = new
GConnectionDialog();
                gcd.ShowDialog();
                ga = new GApplication(true);
                ga.ApplicationName = "Alchemi POV-Ray Renderer
- ilmu komputer:UB";
                ga.Connection = gcd.Connection;
                ga.ThreadFinish += new
GThreadFinish(ga_ThreadFinish);
                ga.ThreadFailed += new
GThreadFailed(ga_ThreadFailed);
                ga.ApplicationFinish += new
GApplicationFinish(ga_ApplicationFinish);
                ga.Manifest.Add(new
ModuleDependency(typeof(RenderThread).Module));
                initted = true;
        }

        if (ga!=null && ga.Running)
        {
                ga.Stop();
        }

        pbar.Maximum = columns*rows;
        pbar.Minimum = 0;
        pbar.Value = 0;
        lbProgress.Text = "";
```

```
        for (int col=0; col<columns; col++)
        {
                for (int row=0; row<rows; row++)
                {
                        x = col*segmentWidth;
                        y = row*segmentHeight;
                        int startRowPixel = y + 1;
                        int endRowPixel = y + segmentHeight;
                        int startColPixel = x + 1;
                        int endColPixel = x + segmentWidth;

                        RenderThread rth = new
RenderThread(modelPath, imageWidth, imageHeight,
segmentWidth, segmentHeight, startRowPixel, endRowPixel,
startColPixel, endColPixel, "");
                        rth.BasePath = this.basepath;
                        rth.Col = col+1;
                        rth.Row = row+1;
                        ga.Threads.Add(rth);
                }
        }

        try
        {
                ga.Start();
        }
        catch (Exception ex)
        {
                Console.WriteLine(""+ex.StackTrace);
                MessageBox.Show("Alchemi Rendering
Failed!"+ex.ToString());
        }
}
```

4.1.1.2.7 Method StopApp

Method ini akan dijalankan ketika dilakukan klik pada tombol stop ataupun juga ketika aplikasi ditutup.

```
private void StopApp()
{
        try
        {
                if (ga != null && ga.Running)
                {
                        ga.Stop();
                        logger.Debug("Application stopped.");
                }
```

```
                        else
                        {
                                if (ga == null)
                                {
                                        logger.Debug("ga is
null");                         }
                                else
                                {
                                        logger.Debug("ga
running returned false...");
                                }
                        }
                        stop.Enabled = false;
                        render.Enabled = !stop.Enabled;
                }
                catch (Exception ex)
                {
                        MessageBox.Show("Error stopping
application: "+ex.Message);
                }
}
```

4.1.1.2.8 Method unpackThread

Method ini berfungsi untuk melakukan *unpack* terhadap data yang diperoleh dari hasil *thread* yang dijalankan oleh *executor*.

```
private void unpackThread(GThread thread)
{
        RenderThread rth = (RenderThread)thread;
        if (rth!=null)
        {
                Bitmap bit = rth.RenderedImageSegment;
                if (bit!=null)
                {
                        logger.Debug("Loading from bitmap");
                        displayImage(bit, rth.Col, rth.Row);
                }
                else
                {
                        logger.Debug ("bit is null! " +
thread.Id );
                }
        }
}
```

4.2 Pembahasan

Berdasarkan dari hasil percobaan yang telah dilakukan sebelumnya, diperoleh data-data sebagai berikut :

4.2.1 Pengamatan Waktu Komputasi Proses Render secara Standalone dan Grid

Pada percobaan ini eksekusi secara standalone dijalankan pada 1 buah PC sedangkan eksekusi secara grid menggunakan 2 buah PC, 3 buah PC, 4 buah PC, 5 buah PC dan 6 buah PC yang terhubung dengan jaringan. Untuk masing-masing percobaan yang digunakan sebagai manager adalah 1 buah PC Data yang diperoleh dari hasil percobaan adalah sebagai berikut :

Tabel 4.1 Tabel hasil percobaan secara standalone

Ukuran gambar	Waktu eksekusi (detik)
320x200	121
480x320	276
640x480	514
800x600	773
1024x800	1196

Pada Tabel 4.1 data diperoleh dari hasil percobaan secara standalone, semakin besar ukuran gambar akan menghasilkan waktu eksekusi yang semakin lama.

Tabel 4.2 Tabel hasil percobaan secara grid dengan 2 komputer

Ukuran gambar	Waktu eksekusi (detik)				
	1x1	2x2	3x3	4x4	5x5
320x200	142	87	102	110	120
480x320	337	162	171	175	232
640x480	639	268	272	276	310

800x600	798	323	346	378	425
1024x800	1224	683	703	710	712

Keterangan :
Ukuran gambar : merupakan ukuran lebar dan tinggi dari gambar hasil *render*
1x1, 2x2, dst : jumlah segmen

Tabel 4.2 merupakan data hasil percobaan secara grid dengan dua komputer. Semakin besar ukuran gambar menghasilkan nilai waktu eksekusi yang semakin lama, sedangkan untuk segmen sedikit berbeda, pada segmen 1x1 waktu eksekusi bernilai besar dan pada segmen 2x2 mengalami penurunan selanjutnya bertambah lagi pada segmen 3x3 dan seterusnya. Waktu eksekusi yang tercepat pada ukuran gambar 320x200 segmen 2x2 dengan waktu eksekusi 87 detik sedangkan waktu eksekusi yang paling lama pada ukuran gambar 1024x800 segmen 1x1 dengan waktu eksekusi 1224 detik.

Tabel 4.3 Tabel hasil percobaan secara grid dengan 3 komputer

Ukuran gambar	Waktu eksekusi (detik)				
	1x1	2x2	3x3	4x4	5x5
320x200	132	75	96	103	118
480x320	342	154	166	169	216
640x480	646	262	263	265	300
800x600	793	313	328	338	405
1024x800	1231	620	628	632	648

Keterangan :
Ukuran gambar : merupakan ukuran lebar dan tinggi dari gambar hasil *render*
1x1, 2x2, dst : jumlah segmen

Tabel 4.3 merupakan data hasil percobaan secara grid dengan tiga komputer. Pada segmen 2x2 mengalami penurunan waktu eksekusi dari segmen 1x1, namun pada segmen 3x3 mengalami kenaikan lagi.

Tabel 4.4 Tabel hasil percobaan secara grid dengan 4 komputer

Ukuran gambar	Waktu eksekusi (detik)				
	1x1	2x2	3x3	4x4	5x5
320x200	148	70	71	73	114
480x320	340	100	108	114	121
640x480	677	142	140	145	154
800x600	789	196	192	198	203
1024x800	1234	306	273	290	308

Keterangan :
Ukuran gambar : merupakan ukuran lebar dan tinggi dari gambar hasil *render*
1x1, 2x2, dst : jumlah segmen

Tabel 4.4 merupakan data hasil percobaan secara grid dengan empat komputer. Pada segmen 2x2, waktu eksekusi yang diperoleh merupakan nilai paling rendah untuk ukuran gambar 320x200 dan 480x320. Sedangkan untuk ukuran gambar 640x480, 800x600 dan 1024x800 pada segmen 3x3.

Tabel 4.5 Tabel hasil percobaan secara grid dengan 5 komputer

Ukuran gambar	Waktu eksekusi (detik)				
	1x1	2x2	3x3	4x4	5x5
320x200	164	74	83	88	108
480x320	345	158	154	142	120
640x480	662	212	200	163	152
800x600	812	282	243	221	200
1024x800	1264	389	342	328	302

Keterangan :
Ukuran gambar : merupakan ukuran lebar dan tinggi dari gambar hasil *render*
1x1, 2x2, dst : jumlah segmen

Tabel 4.5 merupakan data hasil percobaan secara grid dengan lima komputer. Untuk ukuran gambar 320x200 pada segmen 1x1 waktu eksekusi bernilai besar dan mengalami penurunan untuk segmen 2x2

namun mengalami kenaikan terus setelah segmen 3x3. Sedangkan untuk ukuran gambar yang lain dengan bertambahnya segmen, waktu eksekusi mengalami penurunan.

Tabel 4.6 Tabel hasil percobaan secara grid dengan 6 komputer

Ukuran gambar	Waktu eksekusi (detik)				
	1x1	2x2	3x3	4x4	5x5
320x200	168	94	93	96	98
480x320	348	160	142	129	118
640x480	669	236	210	187	148
800x600	823	312	263	226	198
1024x800	1298	400	364	348	300

Keterangan :
Ukuran gambar : merupakan ukuran lebar dan tinggi dari gambar hasil *render*
1x1, 2x2, dst : jumlah segmen

Tabel 4.6 merupakan data hasil percobaan secara grid dengan enam komputer. Untuk ukuran gambar 320x200 pada segmen 1x1 waktu eksekusi bernilai besar dan mengalami penurunan untuk segmen 2x2 namun mengalami kenaikan terus setelah segmen 4x4. Sedangkan untuk ukuran gambar yang lain dengan bertambahnya segmen, waktu eksekusi mengalami penurunan.

Tabel 4.7 Tabel untuk melihat Overhead, Granularitas dan waktu Komunikasi di grid untuk 2 host

Ukuran gambar	Segmen														
	1 x 1			2x2			3x3			4x4			5x5		
	O	G	WK	O	G	WK	O	G	WK	O	G	WK	O	G	WK
320x200	1,3	69,05	2,6	1,81	19,94	5,43	2,04	9,29	16,32	2,7	4,175	40,5	2,94	2,33	58,8
480x320	1,59	166,12	3,18	2,02	38,48	6,06	2,38	16,6	19,04	2,94	7,998	44,1	3,28	6,525	65,6
640x480	1,83	316,76	3,66	2,33	64,67	6,99	2,71	27,5	21,68	3,18	14,07	47,7	3,63	9,351	72,6
800x600	1,94	396,09	3,88	2,54	78,21	7,62	3,14	35,3	25,12	3,51	20,12	52,7	3,84	13,77	76,8
1024x800	2,05	608,93	4,1	2,86	167,89	8,58	3,47	74,6	27,76	3,84	40,54	57,6	4,18	24,97	83,6

Keterangan :
O : Overhead Manager di grid (detik)
G : Granularitas Executor di grid (detik)
WK : Waktu komunikasi Executor di grid (detik)

Tabel 4.7 merupakan data hasil perhitungan untuk granularitas 2 host. Pada Tabel 4.7 nilai granularitas yang terbesar pada ukuran gambar 1024x800 jumlah segmen 1x1 sebesar 608,93. Sedangkan nilai terkecil pada ukuran gambar 320x200 jumlah segmen 5x5 sebesar 2,33.

Tabel 4.8 Tabel untuk melihat Overhead, Granularitas dan waktu Komunikasi di grid untuk 3 host

Ukuran gambar	Segmen														
	1 x 1			2x2			3x3			4x4			5x5		
	O	G	WK	O	G	WK	O	G	WK	O	G	WK	O	G	WK
320x200	1,82	68,27	3,64	2,03	19,213	8,12	2,26	8,822	20,34	2,32	4,555	34,8	2,81	2,4396	56,2
480x320	2,02	165,47	4,04	2,23	37,713	8,92	2,46	16,27	22,14	2,71	8,228	40,65	2,96	6,7936	59,2
640x480	2,22	316,17	4,44	2,43	63,963	9,72	2,87	27,03	25,83	3,01	14,24	45,15	3,18	9,7688	62,6
800x600	2,52	395,22	5,04	2,84	77,2	11,4	3,17	34,92	28,53	3,42	20,21	51,3	3,62	13,999	71,4
1024x800	2,83	607,76	5,66	3,14	166,83	12,6	3,38	74,36	30,42	3,73	40,65	55,95	4,31	25,46	71,2

Keterangan :
O : Overhead Manager di grid (detik)
G : Granularitas Executor di grid (detik)
WK : Waktu komunikasi Executor di grid (detik)

Tabel 4.8 merupakan data hasil perhitungan untuk granularitas 3 host. Pada Tabel 4.8 nilai granularitas yang terbesar pada ukuran gambar 1024x800 jumlah segmen 1x1 sebesar 607,76. Sedangkan nilai terkecil pada ukuran gambar 320x200 jumlah segmen 5x5 sebesar 2,4396.

Tabel 4.9 Tabel untuk melihat Overhead, Granularitas dan waktu Komunikasi di grid untuk 4 host

Ukuran Gambar	Segmen														
	1 x 1			2x2			3x3			4x4			5x5		
	O	G	WK	O	G	WK	O	G	WK	O	G	WK	O	G	WK
320x200	2,02	69,96	6,06	2,16	14,8	8,64	2,2	6,111	13,8	2,22	2,454	31,52	2,44	2,3104	53,8
480x320	2,38	165,24	7,14	2,56	21,8	10,2	2,62	9,311	21,58	2,58	4,384	41,28	2,74	2,4584	56,8
640x480	2,69	333,12	8,07	2,87	31,913	11,5	2,89	12,34	26,01	2,92	5,96	46,72	2,94	3,6904	58,8
800x600	2,86	388,78	8,58	3,08	45,15	12,3	3,14	17,84	28,26	3,21	8,964	51,36	3,41	5,2556	68,2
1024x800	3,11	610,78	9,33	3,38	72,275	13,5	3,48	26,47	31,32	3,56	14,34	56,96	3,54	9,3064	71,8

Keterangan :
O : Overhead Manager di grid (detik)
G : Granularitas Executor di grid (detik)
WK : Waktu komunikasi Executor di grid (detik)

Tabel 4.9 merupakan data hasil perhitungan untuk granularitas 4 host. Pada Tabel 4.9 nilai granularitas yang terbesar pada ukuran gambar 1024x800 jumlah segmen 1x1 sebesar 610,78. Sedangkan nilai terkecil pada ukuran gambar 320x200 jumlah segmen 5x5 sebesar 2,3104.

Tabel 4.10 Tabel untuk melihat Overhead, Granularitas dan waktu Komunikasi di grid untuk 5 host

Ukuran Gambar	Segmen														
	1 x 1			2x2			3x3			4x4			5x5		
	O	G	WK	O	G	WK	O	G	WK	O	G	WK	O	G	WK
320x200	2,3	77,25	7,2	2,54	15,325	10,2	2,62	6,311	23,58	2,7	2,944	38,2	3,02	2,1832	50,4
480x320	2,58	167,05	8,32	2,86	35,925	11,4	2,98	13,8	26,82	2,98	5,709	47,68	3,21	2,1036	64,2
640x480	2,75	325,38	8,5	3,12	49,1	12,5	3,22	18,64	28,98	3,13	6,862	50,08	3,61	3,2476	67,2
800x600	3,06	399,85	9,24	3,37	66,038	14,5	3,38	23,24	30,42	3,68	10,22	53,88	4,22	4,4552	84,4
1024x800	3,39	624,53	11,6	3,64	92,25	16,4	3,68	33,91	33,12	3,82	16,44	61,12	4,44	8,3504	88,8

Keterangan :
O : Overhead Manager di grid (detik)
G : Granularitas Executor di grid (detik)
WK : Waktu komunikasi Executor di grid (detik)

Tabel 4.10 merupakan data hasil perhitungan untuk granularitas 5 host. Pada Tabel 4.10 nilai granularitas yang terbesar pada ukuran gambar 1024x800 jumlah segmen 1x1 sebesar 624,53. Sedangkan nilai terkecil pada ukuran gambar 480x320 jumlah segmen 5x5 sebesar 2,1036.

Tabel 4.11 Tabel untuk melihat Overhead, Granularitas dan waktu Komunikasi di grid untuk 6 host

Ukuran Gambar	Segmen														
	1 x 1			2x2			3x3			4x4			5x5		
	O	G	WK	O	G	WK	O	G	WK	O	G	WK	O	G	WK
320x200	2,36	79,1	7,44	3,58	19,28	13,32	3,85	6,61	29,65	3,98	3,021	43,68	4,03	0,5348	80,6
480x320	2,62	168,45	8,48	3,72	35,6	13,88	3,9	12	30,1	4,19	4,861	47,04	4,44	0,9904	88,8
640x480	2,84	328,9	8,36	3,8	54,5	14,2	4,09	19,3	31,81	4,59	8,061	53,44	4,68	1,9888	93,6
800x600	3,2	405,5	8,8	4,01	72,99	16,04	4,37	24,9	34,33	4,62	10,47	53,92	4,84	3,8544	96,8
1024x800	3,41	641,48	11,6	4,12	94,85	16,48	4,62	35,9	36,58	4,85	17,22	67,6	5,26	7,5816	105,2

Keterangan :
O : Overhead Manager di grid (detik)
G : Granularitas Executor di grid (detik)
WK : Waktu komunikasi Executor di grid (detik)

Tabel 4.11 merupakan data hasil perhitungan untuk granularitas 6 host. Pada Tabel 4.11 nilai granularitas yang terbesar pada ukuran gambar 1024x800 jumlah segmen 1x1 sebesar 641,48. Sedangkan nilai terkecil pada ukuran gambar 320x200 jumlah segmen 5x5 sebesar 0,5348.

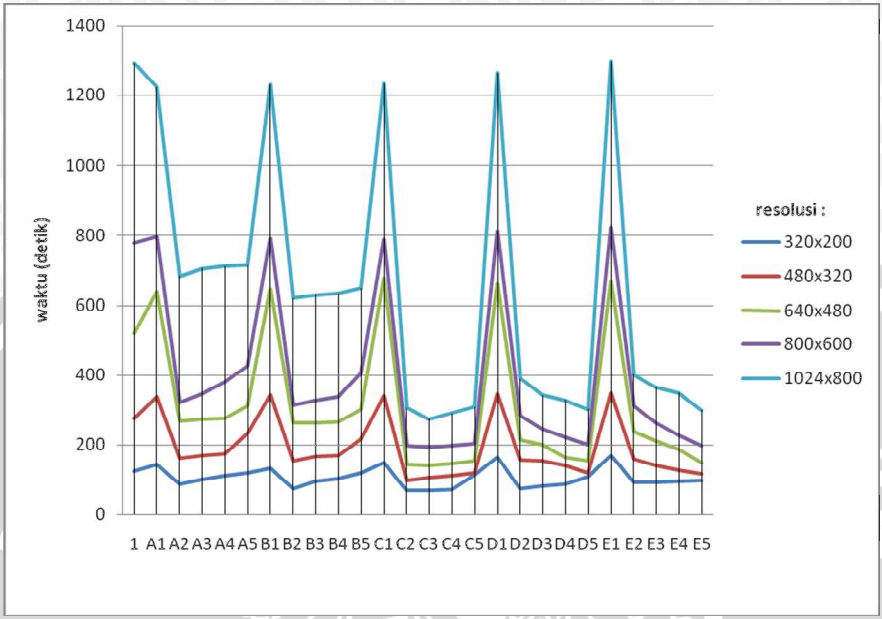
Dari Tabel 4.7 sampai Tabel 4.11 dapat dilihat bahwa dengan bertambahnya ukuran gambar, banyaknya segmen dan jumlah komputer, lama overhead proses di manager akan semakin bertambah pula dan hal tersebut juga berpengaruh terhadap waktu komunikasi antara *manager* dan *executor* sehingga menjadi lebih lama.

Untuk granularitas *executor*, terjadi penurunan mulai dari 2 host sampai pada 4 host, namun setelah 4 host besar granularitas *executor* menjadi bertambah sampai 6 host kecuali pada segmen 5x5 mengalami penurunan.

4.2.1.1 Analisis Grafik

Dari hasil pengambilan data pada sub-bab 4.2.1 maka dapat disusun sebuah grafik yang menggambarkan kinerja *render* secara standalone dengan secara grid dengan 2 PC, 3 PC, 4 PC, 5 PC dan 6 PC pada ukuran gambar dengan 320 x 200, 480x 320, 640 x 480, 800x 600 dan 1024 x 800, juga terhadap segmen 1x1, 2x2, 3x3, 4x4, dan 5x5.

Grafik 4.1 Grafik perbandingan kinerja secara standalone dan secara grid


1400
1200
1000
800
600
400
200
0
waktu (detik)
1 A1 A2 A3 A4 A5 B1 B2 B3 B4 B5 C1 C2 C3 C4 C5 D1 D2 D3 D4 D5 E1 E2 E3 E4 E5
resolusi :
320x200
480x320
640x480
800x600
1024x800

Keterangan :

A1	: 2 host dengan segmen 1x1
A2	: 2 host dengan segmen 2x2
A3	: 2 host dengan segmen 3x3
A4	: 2 host dengan segmen 4x4
A5	: 2 host dengan segmen 5x5
B1	: 3 host dengan segmen 1x1
B2	: 3 host dengan segmen 2x2
B3	: 3 host dengan segmen 3x3
B4	: 3 host dengan segmen 4x4
B5	: 3 host dengan segmen 5x5
C1	: 4 host dengan segmen 1x1
C2	: 4 host dengan segmen 2x2
C3	: 4 host dengan segmen 3x3
C4	: 4 host dengan segmen 4x4
C5	: 4 host dengan segmen 5x5
D1	: 5 host dengan segmen 1x1
D2	: 5 host dengan segmen 2x2
D3	: 5 host dengan segmen 3x3
D4	: 5 host dengan segmen 4x4
D5	: 5 host dengan segmen 5x5
E1	: 6 host dengan segmen 1x1
E2	: 6 host dengan segmen 2x2
E3	: 6 host dengan segmen 3x3
E4	: 6 host dengan segmen 4x4
E5	: 6 host dengan segmen 5x5

Dari Grafik 4.1 dapat dilihat bahwa kinerja yang paling bagus untuk ukuran gambar 320x200, dan 480x 320 pada segmen 2x2 untuk 4 host, sedangkan untuk ukuran gambar 640 x 480, 800x 600 dan 1024x800 mengalami kinerja paling bagus pada segmen 3x3 untuk 4 host.

4.2.1.2 Analisis Hasil

Pada percobaan yang telah dilakukan, penambahan jumlah komputer untuk proses *render* secara grid tidak selalu akan mengakibatkan perbaikan kinerja dari sebelumnya. Untuk ukuran gambar 320x200, dan 480x 320 dengan melakukan penambahan jumlah komputer dari 3 komputer menjadi 4 komputer mengalami kenaikan kinerja tetapi jika dilakukan penambahan jumlah komputer

lagi, kinerja akan mengalami penurunan kecuali untuk segmen 5x5. Penambahan segmen pada grid dengan 4 komputer mengakibatkan kenaikan kinerja yaitu pada segmen 1x1 dinaikan menjadi segmen 2x2, namun ketika dilakukan penambahan lagi kinerja malah mengalami penurunan dari kinerja sebelumnya. Sedangkan untuk ukuran gambar 640 x 480, 800x 600 dan 1024x800, kinerja paling bagus diperoleh pada segmen 3x3 dengan menggunakan 4 komputer. Penambahan jumlah komputer dan segmen tidak selalu akan mengakibatkan adanya hasil kinerja yang lebih bagus, hal ini disebabkan dengan adanya penambahan tersebut juga mengakibatkan bertambah pula overhead manager dan waktu komunikasi executor dengan manager.

Dari hasil percobaan yang diperoleh, kemudian dilakukan perhitungan untuk mencari regresi maka diperoleh rumusan sebagai berikut :

K = 4,03 + 0,0001 L - 0,0004 T

Dimana K adalah jumlah komputer yang terbaik untuk melakukan *render*, L adalah lebar ukuran gambar dan T adalah tinggi ukuran gambar.

4.2.2 Pengamatan Speedup Waktu Komputasi Secara Grid

Untuk menghitung *speedup* waktu komputasi secara grid digunakan rumus $sp(n) = \frac{T(n)}{Tp(n)}$ dimana n adalah jumlah komputernya. Diperoleh data sebagai berikut :

Tabel 4.12 Tabel speedup untuk 2 host

Ukuran	**Speedup**

gambar	1x1	2x2	3x3	4x4	5x5
320x200	0,880282	1,436782	1,22549	1,136364	1,041667
480x320	0,818991	1,703704	1,614035	1,577143	1,189655
640x480	0,812207	1,936567	1,908088	1,880435	1,674194
800x600	0,977444	2,414861	2,254335	2,063492	1,835294
1024x800	1,055556	1,891654	1,837838	1,819718	1,814607

Tabel 4.12 merupakan data hasil perhitungan untuk *speedup* 2 host. Pada Tabel 4.12 nilai *speedup* yang terbesar pada ukuran gambar 800x600 jumlah segmen 2x2 sebesar 2,41. Sedangkan nilai terkecil pada ukuran gambar 320x200 jumlah segmen 1x1 sebesar 0,88.

Tabel 4.13 Tabel speedup untuk 3 host

Ukuran gambar	Speedup				
	1x1	2x2	3x3	4x4	5x5
320x200	0,94697	1,666667	1,302083	1,213592	1,059322
480x320	0,807018	1,792208	1,662651	1,633136	1,277778
640x480	0,803406	1,980916	1,973384	1,958491	1,73
800x600	0,983607	2,492013	2,378049	2,307692	1,925926
1024x800	1,049553	2,083871	2,057325	2,044304	1,993827

Tabel 4.13 merupakan data hasil perhitungan untuk *speedup* 3 host. Pada Tabel 4.13 nilai *speedup* yang terbesar pada ukuran gambar 800x600 jumlah segmen 2x2 sebesar 2,49. Sedangkan nilai terkecil pada ukuran gambar 640x480 jumlah segmen 1x1 sebesar 0,803.

Tabel 4.14 Tabel speedup untuk 4 host

Ukuran gambar	Speedup				
	1x1	2x2	3x3	4x4	5x5
320x200	0,844595	1,785714	1,760563	1,712329	1,096491
480x320	0,811765	2,76	2,555556	2,421053	2,280992
640x480	0,766617	3,65493	3,707143	3,57931	3,37013
800x600	0,988593	3,979592	4,0625	3,939394	3,842365

1024x800	1,047002	4,222222	4,732601	4,455172	4,194805

Tabel 4.14 merupakan data hasil perhitungan untuk *speedup* 4 host. Pada Tabel 4.14 nilai *speedup* yang terbesar pada ukuran gambar 1024x800 jumlah segmen 3x3 sebesar 4,73. Sedangkan nilai terkecil pada ukuran gambar 640x480 jumlah segmen 1x1 sebesar 0,98.

Tabel 4.15 Tabel speedup untuk 5 host

Ukuran gambar	Speedup				
	1x1	2x2	3x3	4x4	5x5
320x200	0,762195	1,689189	1,506024	1,420455	1,157407
480x320	0,8	1,746835	1,792208	1,943662	2,3
640x480	0,783988	2,448113	2,595	3,184049	3,414474
800x600	0,960591	2,765957	3,209877	3,529412	3,9
1024x800	1,022152	3,321337	3,777778	3,939024	4,278146

Tabel 4.15 merupakan data hasil perhitungan untuk *speedup* 5 host. Pada Tabel 4.15 nilai *speedup* yang terbesar pada ukuran gambar 1024x800 jumlah segmen 5x5 sebesar 4,27. Sedangkan nilai terkecil pada ukuran gambar 320x200 jumlah segmen 1x1 sebesar 0,76.

Tabel 4.16 Tabel speedup untuk 6 host

Ukuran gambar	Speedup				
	1x1	2x2	3x3	4x4	5x5
320x200	0,744048	1,329787	1,344086	1,302083	1,27551
480x320	0,793103	1,725	1,943662	2,139535	2,338983
640x480	0,775785	2,199153	2,471429	2,775401	3,506757
800x600	0,947752	2,5	2,965779	3,451327	3,939394
1024x800	0,995378	3,23	3,549451	3,712644	4,306667

Tabel 4.16 merupakan data hasil perhitungan untuk *speedup* 6 host. Pada Tabel 4.16 nilai *speedup* yang terbesar pada ukuran gambar 1024x800 jumlah segmen 5x5 sebesar 4,30. Sedangkan nilai terkecil pada ukuran gambar 320x200 jumlah segmen 1x1 sebesar 0,74.

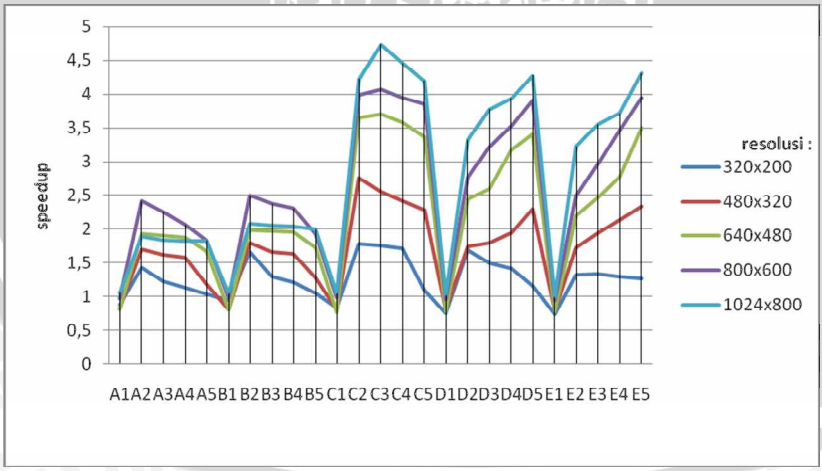
Dari Tabel 4.12 sampai Tabel 4.16 dapat dilihat perhitungan untuk 2 sampai 3 host, *speedup* mengalami kenaikan pada segmen 2x2 dari nilai pada segmen 1x1 namun menjadi turun lagi untuk penambahan segmen selanjutnya. Untuk 4 host pada ukuran gambar 320x200 dan 480x320 nilai *speedup* naik dari segmen 1x1 ke segmen 2x2 namun

menurun lagi setelah kenaikan segmen lagi, sedangkan pada ukuan gambar 640x480, 800x600, dan 1024x800 nilai *speedup* naik dari segmen 1x1 sampai 3x3, namun menurun ketika pada penambahan segmen 4x4. Sedangkan untuk 5 dan 6 host dengan bertambahnya segmen nilai *speedup* mengalami kenaikan kecuali untuk ukuran gambar 320x200 pada 5 host terjadi penurunan lagi pada segmen 3x3 dan pada 6 host terjadi penurunan lagi pada segmen 4x4. Hal ini dipengaruhi karena adanya inisialisasi aplikasi, delay sinkronisasi, waktu komunkasi dan juga *overhead* sehingga *speedup* mengalami penurunan.

4.2.2.1 Analisis Grafik

Dari hasil perhitungan data pada sub-bab 4.2.2 maka dapat disusun sebuah grafik yang menggambarkan perbandingan *speedup* dari proses *render* secara grid dengan 2 PC, 3 PC, 4 PC, 5 PC dan 6 PC pada ukuran gambar dengan 320 x 200, 480x 320, 640 x 480, 800x 600 dan 1024 x 800, juga terhadap segmen 1x1, 2x2, 3x3, 4x4, dan 5x5.

Grafik 4.2 Grafik perbandingan speedup secara grid



Keterangan :

A1	: 2 host dengan segmen 1x1
A2	: 2 host dengan segmen 2x2
A3	: 2 host dengan segmen 3x3
A4	: 2 host dengan segmen 4x4
A5	: 2 host dengan segmen 5x5
B1	: 3 host dengan segmen 1x1
B2	: 3 host dengan segmen 2x2
B3	: 3 host dengan segmen 3x3
B4	: 3 host dengan segmen 4x4
B5	: 3 host dengan segmen 5x5
C1	: 4 host dengan segmen 1x1
C2	: 4 host dengan segmen 2x2
C3	: 4 host dengan segmen 3x3
C4	: 4 host dengan segmen 4x4
C5	: 4 host dengan segmen 5x5
D1	: 5 host dengan segmen 1x1
D2	: 5 host dengan segmen 2x2
D3	: 5 host dengan segmen 3x3
D4	: 5 host dengan segmen 4x4
D5	: 5 host dengan segmen 5x5
E1	: 6 host dengan segmen 1x1
E2	: 6 host dengan segmen 2x2
E3	: 6 host dengan segmen 3x3
E4	: 6 host dengan segmen 4x4
E5	: 6 host dengan segmen 5x5

Dari Grafik 4.2 dapat dilihat bahwa *speedup* yang paling tinggi dicapai pada ukuran gambar 1024x800 dengan segmen 3x3 dengan menggunakan 4 host. Untuk ukuran gambar yang lain, ukuran gambar 320x200 dan 480x320 *speedup* yang paling tinggi terjadi pada segmen 2x2 dengan menggunakan 4 host sedangkan ukuran gambar 640x480 dan 800x600 pada segmen 3x3 dengan menggunakan 4 host. Sehingga belum tentu dengan adanya penambahan jumlah host dan segmen akan mengakibatkan kenaikan nilai *speedup*. Hal ini dipengaruhi dengan adanya inisialisasi aplikasi, *overhead* manager dan waktu komunikasi *executor* dengan manager.

4.2.3 Pengamatan Efisiensi Pemroses secara grid

Untuk mengetahui effisiensi pemroses maka yang harus dilakukan terlebih dulu adalah menghitung *Speedup* dahulu. Menghitung efisiensi yaitu dengan cara *speedup* yang diperoleh dengan jumlah pemroses yang digunakan. Berikut adalah tabel hasil perhitungan dari efisiensi pemroses :

Tabel 4.17 Tabel efisiensi untuk 2 host

Ukuran gambar	efisiensi				
	1x1	2x2	3x3	4x4	5x5
320x200	0,440141	0,718391	0,612745	0,568182	0,520833
480x320	0,409496	0,851852	0,807018	0,788571	0,594828
640x480	0,406103	0,968284	0,954044	0,940217	0,837097
800x600	0,488722	1,20743	1,127168	1,031746	0,917647
1024x800	0,527778	0,945827	0,918919	0,909859	0,907303

Tabel 4.17 merupakan data hasil perhitungan untuk efisiensi 2 host. Pada Tabel 4.17 nilai efisiensi yang terbesar pada ukuran gambar 800x600 jumlah segmen 2x2 sebesar 1,2. Sedangkan nilai terkecil pada ukuran gambar 640x480 jumlah segmen 1x1 sebesar 0,406.

Tabel 4.18 Tabel efisiensi untuk 3 host

Ukuran gambar	efisiensi				
	1x1	2x2	3x3	4x4	5x5
320x200	0,315657	0,555556	0,434028	0,404531	0,353107
480x320	0,269006	0,597403	0,554217	0,544379	0,425926
640x480	0,267802	0,660305	0,657795	0,65283	0,576667
800x600	0,327869	0,830671	0,792683	0,769231	0,641975
1024x800	0,349851	0,694624	0,685775	0,681435	0,664609

Tabel 4.18 merupakan data hasil perhitungan untuk efisiensi 3 host. Pada Tabel 4.18 nilai efisiensi yang terbesar pada ukuran gambar 800x600 jumlah segmen 2x2 sebesar 0,83. Sedangkan nilai terkecil pada ukuran gambar 640x480 jumlah segmen 1x1 sebesar 0,267.

Tabel 4.19 Tabel efisiensi untuk 4 host

Ukuran gambar	efisiensi				
	1x1	2x2	3x3	4x4	5x5
320x200	0,211149	0,446429	0,440141	0,428082	0,274123
480x320	0,202941	0,69	0,638889	0,605263	0,570248
640x480	0,191654	0,913732	0,926786	0,894828	0,842532
800x600	0,247148	0,994898	1,015625	0,984848	0,960591
1024x800	0,26175	1,055556	1,18315	1,113793	1,048701

Tabel 4.19 merupakan data hasil perhitungan untuk efisiensi 4 host. Pada Tabel 4.19 nilai efisiensi yang terbesar pada ukuran gambar 1024x800 jumlah segmen 3x3 sebesar 1,18. Sedangkan nilai terkecil pada ukuran gambar 640x480 jumlah segmen 1x1 sebesar 0,191.

Tabel 4.20 Tabel efisiensi untuk 5 host

Ukuran gambar	efisiensi				
	1x1	2x2	3x3	4x4	5x5
320x200	0,152439	0,337838	0,301205	0,284091	0,231481
480x320	0,16	0,349367	0,358442	0,388732	0,46
640x480	0,156798	0,489623	0,519	0,63681	0,682895
800x600	0,192118	0,553191	0,641975	0,705882	0,78
1024x800	0,20443	0,664267	0,755556	0,787805	0,855629

Tabel 4.20 merupakan data hasil perhitungan untuk efisiensi 5 host. Pada Tabel 4.20 nilai efisiensi yang terbesar pada ukuran gambar 1024x800 jumlah segmen 5x5 sebesar 0,85. Sedangkan nilai terkecil pada ukuran gambar 320x200 jumlah segmen 1x1 sebesar 0,152.

Tabel 4.21 Tabel efisiensi untuk 6 host

Ukuran gambar	efisiensi				
	1x1	2x2	3x3	4x4	5x5
320x200	0,124008	0,221631	0,224014	0,217014	0,212585
480x320	0,132184	0,2875	0,323944	0,356589	0,389831
640x480	0,129297	0,366525	0,411905	0,462567	0,584459
800x600	0,157959	0,416667	0,494297	0,575221	0,656566
1024x800	0,165896	0,538333	0,591575	0,618774	0,717778

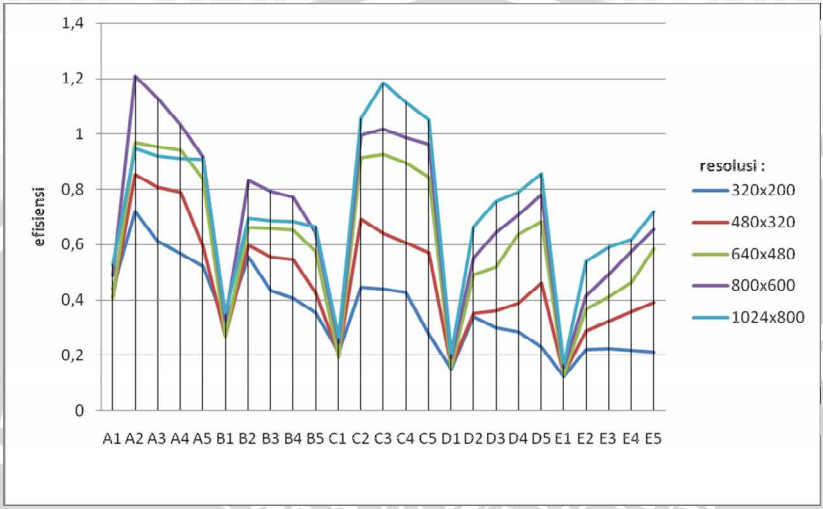
Tabel 4.21 merupakan data hasil perhitungan untuk efisiensi 6 host. Pada Tabel 4.21 nilai efisiensi yang terbesar pada ukuran gambar 1024x800 jumlah segmen 5x5 sebesar 0,717. Sedangkan nilai terkecil pada ukuran gambar 320x200 jumlah segmen 1x1 sebesar 0,124.

Dari Tabel 4.17 sampai Tabel 4.21 dapat dilihat dengan adanya penambahan besar ukuran gambar akan terjadi kenaikan nilai efisiensi, namun akan menjadi menurun jika dilakukan penambahan besar segmen, kecuali pada percobaan dengan 3 dan 4 host tepatnya ketika penambahan segmen dari 1x1 menjadi 2x2, juga pada percobaan 4 host pada ukuran gambar 800x600 ketika penambahan segmen dari 2x2 menjadi 3x3.

4.2.3.1 Analisis Grafik

Dari hasil perhitungan data pada sub-bab 4.2.3 maka dapat disusun sebuah grafik yang menggambarkan perbandingan efisiensi pemroses secara grid dengan 2 PC, 3 PC, 4 PC, 5 PC dan 6 PC pada ukuran gambar dengan 320 x 200, 480x 320, 640 x 480, 800x 600 dan 1024 x 800, juga terhadap segmen 1x1, 2x2, 3x3, 4x4, dan 5x5.

Grafik 4.3 Grafik perbandingan efisiensi pemroses secara grid



Keterangan :

A1	: 2 host dengan segmen 1x1
A2	: 2 host dengan segmen 2x2
A3	: 2 host dengan segmen 3x3
A4	: 2 host dengan segmen 4x4
A5	: 2 host dengan segmen 5x5
B1	: 3 host dengan segmen 1x1
B2	: 3 host dengan segmen 2x2
B3	: 3 host dengan segmen 3x3
B4	: 3 host dengan segmen 4x4
B5	: 3 host dengan segmen 5x5
C1	: 4 host dengan segmen 1x1
C2	: 4 host dengan segmen 2x2
C3	: 4 host dengan segmen 3x3
C4	: 4 host dengan segmen 4x4
C5	: 4 host dengan segmen 5x5
D1	: 5 host dengan segmen 1x1
D2	: 5 host dengan segmen 2x2
D3	: 5 host dengan segmen 3x3
D4	: 5 host dengan segmen 4x4
D5	: 5 host dengan segmen 5x5
E1	: 6 host dengan segmen 1x1
E2	: 6 host dengan segmen 2x2
E3	: 6 host dengan segmen 3x3
E4	: 6 host dengan segmen 4x4

E5 : 6 host dengan segmen 5x5

Dari Grafik 4.3 dapat dilihat nilai efisiensi pemroses paling tinggi untuk ukuran gambar 320 x 200, dan 480x 320 terjadi pada segmen 1x1 untuk 2 host, ukuran gambar 640x480 dan 800x600 terjadi pada segmen 3x3 untuk 2 host, sedangkan ukuran gambar 1024 x 800 terjadi pada segmen 3x3 untuk 4 host. Untuk proses grid menggunakan 2, 3, dan 4 host terjadi kenaikan pada segmen 2x2 kemudian mengalami penurunan lagi pada segmen 3x3 dan seterusnya kecuali pada ukuran gambar 1024x800 untuk 4 host, kenaikan terjadi sampai segmen 3x3 dan mengalami penurunan lagi pada segmen 4x4. Untuk proses grid dengan menggunakan 5 host dan 6 host, terjadi kenaikan ketika dilakukan penambahan segmen kecuali pada ukuran gambar 320x200 mengalami kenaikan sampai segmen 3x3 dan mengalami penurunan lagi pada segmen 4x4.

Pada grafik tersebut juga terlihat terjadi penurunan pada 3 host, kemudian dengan menambah host menjadi 4 host mengalami kenaikan namun mengalami penurunan lagi ketika dilakukan penambahan menjadi 5 host dan 6 host. Hal ini dipengaruhi dengan adanya inisialisasi aplikasi, overhead manager dan besarnya waktu komunikasi. Bertambahnya jumlah komputer juga menyebabkan bertambahnya jumlah pemroses, semakin banyak jumlah pemroses akan mengakibatkan semakin besar nilai pembagi pada rumus perhitungan efisiensi pemroses.

4.2.4 Pengamatan CC Ratio secara grid

CC Ratio dapat diperoleh dari total waktu komputasi yang dilakukan oleh executor dibagi dengan total waktu komunikasi yang terjadi saat proes *render* berjalan. Adapun hasil perhitungan yang telah dilakukan dapat dilihat pada Tabel 4.22

Tabel 4.22 Tabel cc ratio untuk 2 host

Ukuran gambar	**CC Ratio**				
	1x1	**2x2**	**3x3**	**4x4**	**5x5**
320x200	35,41	11,02	4,56	1,55	1,11

480x320	69,65	19,05	6,98	2,72	2,37
640x480	115,39	27,76	10,15	4,42	3,07
800x600	136,11	30,79	11,24	5,73	4,27
1024x800	198,02	58,70	21,51	10,56	7,11

Tabel 4.22 merupakan data hasil perhitungan untuk cc ratio 2 host. Pada Tabel 4.22 nilai cc ratio yang terbesar pada ukuran gambar 1024x800 jumlah segmen 1x1 sebesar 198,02. Sedangkan nilai terkecil pada ukuran gambar 320x200 jumlah segmen 5x5 sebesar 1,11.

Tabel 4.23 Tabel cc ratio untuk 3 host

Ukuran gambar	**CC Ratio**				
	1x1	**2x2**	**3x3**	**4x4**	**5x5**
320x200	23,18	6,39	3,25	1,45	1,00
480x320	55,44	12,81	5,75	2,60	2,27
640x480	96,00	20,56	8,16	4,50	3,06
800x600	103,89	21,04	9,35	5,18	4,20
1024x800	143,99	38,49	17,58	9,59	7,08

Tabel 4.23 merupakan data hasil perhitungan untuk cc ratio 3 host. Pada Tabel 4.23 nilai cc ratio yang terbesar pada ukuran gambar 1024x800 jumlah segmen 1x1 sebesar 143,99. Sedangkan nilai terkecil pada ukuran gambar 320x200 jumlah segmen 5x5 sebesar 1,00

Tabel 4.24 Tabel cc ratio untuk 4 host

Ukuran gambar	**CC Ratio**				
	1x1	**2x2**	**3x3**	**4x4**	**5x5**
320x200	17,32	5,48	3,44	1,16	0,09
480x320	34,71	6,81	3,46	1,60	1,03
640x480	61,92	8,90	3,84	1,92	1,49
800x600	67,97	11,73	5,11	2,63	1,83
1024x800	98,20	17,11	6,84	3,79	3,09

Tabel 4.24 merupakan data hasil perhitungan untuk cc ratio 4 host. Pada Tabel 4.24 nilai cc ratio yang terbesar pada ukuran gambar 1024x800 jumlah segmen 1x1 sebesar 98,20. Sedangkan nilai terkecil pada ukuran gambar 320x200 jumlah segmen 5x5 sebesar 0,09.

Tabel 4.25 Tabel cc ratio untuk 5 host

Ukuran gambar	CC Ratio				
	1x1	2x2	3x3	4x4	5x5
320x200	16,26	5,83	3,77	1,25	1,02
480x320	30,65	10,05	4,17	1,80	1,28
640x480	57,84	12,59	5,21	2,06	1,55
800x600	65,02	14,80	6,19	2,84	2,26
1024x800	83,55	18,45	8,29	4,05	3,24

Tabel 4.25 merupakan data hasil perhitungan untuk cc ratio 5 host. Pada Tabel 4.25 nilai cc ratio yang terbesar pada ukuran gambar 1024x800 jumlah segmen 1x1 sebesar 83,55. Sedangkan nilai terkecil pada ukuran gambar 480x320 jumlah segmen 5x5 sebesar 1,02.

Tabel 4.26 Tabel cc ratio untuk 6 host

Ukuran gambar	CC Ratio				
	1x1	2x2	3x3	4x4	5x5
320x200	16,14	4,56	1,78	1,01	0,83
480x320	30,35	8,09	3,18	1,52	0,75
640x480	57,82	12,11	4,85	2,22	1,09
800x600	64,78	14,56	5,80	2,86	1,23
1024x800	83,24	18,42	7,83	3,80	2,22

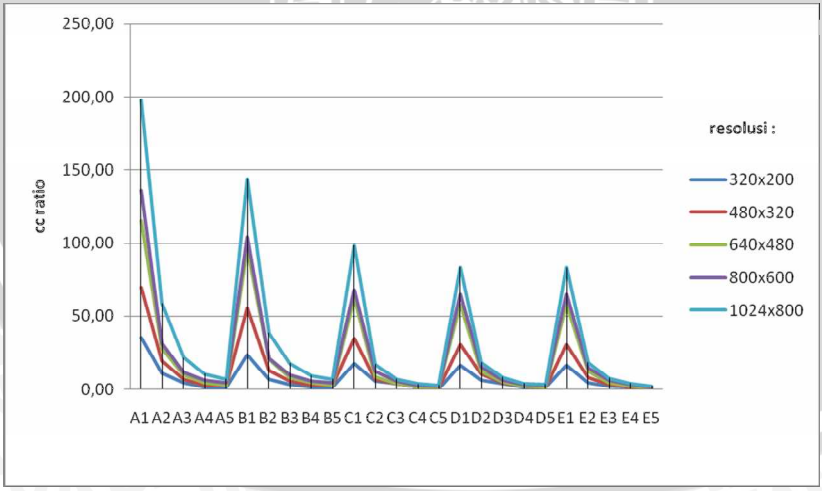
Tabel 4.26 merupakan data hasil perhitungan untuk cc ratio 6 host. Pada Tabel 4.26 nilai cc ratio yang terbesar pada ukuran gambar 1024x800 jumlah segmen 1x1 sebesar 83,24. Sedangkan nilai terkecil pada ukuran gambar 480x320 jumlah segmen 5x5 sebesar 0,75.

Dari Tabel 4.22 sampai Tabel 4.26 dapat dilihat dengan adanya penambahan besar ukuran gambar akan terjadi kenaikan nilai cc ratio, sedangkan pada ukuran gambar yang sama jika dilakukan penambahan segmen maka akan mengalami penurunan nilai cc ratio. Penurunan nilai cc ratio juga terjadi dengan adanya penambahan komputer dari 2 host menjadi 3 host, hal ini terjadi sampai 4 host sedangkan ketika menjadi 5 host terjadi kenaikan nilai cc ratio dari nilai sebelumnya tetapi pada 6 host terjadi penurunan lagi. Namun untuk segmen 1x1 dengan bertambahnya komputer mengakibatkan penurunan nilai cc ratio.

4.2.4.1 Analisis Grafik

Dari hasil perhitungan data pada sub-bab 4.2.4 maka dapat disusun sebuah grafik yang menggambarkan perbandingan cc ratio secara grid dengan 2 PC, 3 PC, 4 PC, 5 PC dan 6 PC pada ukuran gambar dengan 320 x 200, 480x 320, 640 x 480, 800x 600 dan 1024 x 800, juga terhadap segmen 1x1, 2x2, 3x3, 4x4, dan 5x5.

Grafik 4.4 Grafik perbandingan cc ratio secara grid



Keterangan :

A1	: 2 host dengan segmen 1x1
A2	: 2 host dengan segmen 2x2
A3	: 2 host dengan segmen 3x3
A4	: 2 host dengan segmen 4x4
A5	: 2 host dengan segmen 5x5
B1	: 3 host dengan segmen 1x1
B2	: 3 host dengan segmen 2x2
B3	: 3 host dengan segmen 3x3
B4	: 3 host dengan segmen 4x4
B5	: 3 host dengan segmen 5x5
C1	: 4 host dengan segmen 1x1
C2	: 4 host dengan segmen 2x2
C3	: 4 host dengan segmen 3x3
C4	: 4 host dengan segmen 4x4
C5	: 4 host dengan segmen 5x5
D1	: 5 host dengan segmen 1x1
D2	: 5 host dengan segmen 2x2
D3	: 5 host dengan segmen 3x3
D4	: 5 host dengan segmen 4x4
D5	: 5 host dengan segmen 5x5
E1	: 6 host dengan segmen 1x1
E2	: 6 host dengan segmen 2x2
E3	: 6 host dengan segmen 3x3
E4	: 6 host dengan segmen 4x4
E5	: 6 host dengan segmen 5x5

Dari Grafik 4.4 dapat dilihat nilai cc ratio mengalami kenaikan ketika terjadi penambahan ukuran gambar pada segmen yang sama, namun pada ukuran gambar yang sama ketika terjadi penambahan banyaknya segmen akan mengalami penurunan nilai ratio. Hal ini sangat dipengaruhi ketika terjadi penambahan ukuran gambar maka waktu komputasi menjadi lebih lama sedangkan waktu komunikasi dan *overhead* manager tidak banyak mengalami kenaikan, sedangkan ketika terjadi penambahan segmen akan terjadi sebaliknya yaitu, waktu komunikasi menjadi semakin lama karena semakin banyak thread yang tercipta sedangkan waktu komputasinya tidak begitu banyak mengalami kenaikan. Sedangkan penambahan jumlah komputer tidak selalu mengakibatkan penurunan nilai cc ratio, penurunan ini terjadi ketika penambahan jumlah komputer dari 2 host ke 3 host sampai 4 host, ketika penambahan dari 4 host ke 5 host nilai

cc ratio mengalami kenaikan, namun mengalami penurunan kembali ketika terjadi penambahan menjadi 6 host.

BAB V
PENUTUP

5.1 KESIMPULAN

Dari penjelasan yang telah diuraikan dalam bab sebelumnya dapat diambil beberapa kesimpulan :

1. Waktu eksekusi proses render secara grid lebih cepat dibandingkan dengan secara *standalone*. Pada proses secara grid untuk ukuran gambar 1024 x 800 waktu eksekusi terbaiknya 273 detik sedangkan secara *standalone* waktu eksekusi sebesar 1196 detik.
2. Penerapan implementasi pemrosessan secara grid dipengaruhi oleh beberapa hal, yaitu sebagi berikut :

a. Perbedaan jumlah *executor*, yaitu dimana dengan adanya penambahan jumlah *executor* maka semakin banyak *thread* yang dapat dieksekusi. Namun bukan berarti dengan semakin banyak jumlah *executo*r,kinerja program akan menjadi semakin baik. Hal ini juga sangat dipengaruhi adanya *overhead manager* dan waktu komunikasi *executor* dengan *manage*r yang semakin bertambah jika dilakukan penambahan jumlah *executor*.
b. Perbedaan segmen memiliki pengaruh terhadap kinerja program. Penambahan segmen akan berpengaruh terhadap besar ukuran gambar yang akan dihasilkan oleh masing-masing *thread*. Gambar yang relatif kecil dengan jumlah segmen yang lebih bayak akan relatif lebih menurun kinerjanya, sedangkan gambar yang relatif besar dengan jumlah segmen yang banyak maka kinerja akan relatif lebih baik. Penambahan segmen akan membuat *overhead manager* dan waktu eksekusi menjadi lebih lama.
c. Perbedaan besar ukuran gambar. Penambahan besar ukuran gambar akan mengakibatkan semakin lama waktu komputasi yang akan terjadi. Sehingga waktu eksekusi akan menjadi bertambah lama pula.

3. Penambahan jumlah komputer untuk proses render secara grid tidak selalu akan mengakibatkan perbaikan kinerja dari sebelumnya. Penurunan kinerja dipengaruhi dengan adanya kenaikan waktu komunikasi dan overhead manager yang diakibatkan dari penambahan banyaknya segmen dan jumlah host, juga dipengaruhi dengan peningkatan waktu komputasi dan proses

inisilaisasi aplikasi karena disebabkan bertambahnya besar ukuran gambar. Kesemuanya tersebut akan menghasilkan semakin lamanya waktu eksekusi yang terjadi dan kinerja komputasi akan semakin menurun.

4. Penambahan jumlah komputer dan segmen tidak selalu akan mengakibatkan adanya hasil kinerja yang lebih bagus, hal ini disebabkan dengan adanya penambahan tersebut juga mengakibatkan bertambah pula *overhead manager* dan waktu komunikasi *executor* dengan *manager*.
5. Kinerja yang paling baik untuk ukuran gambar 320x200 terjadi pada proses dengan 4 komputer untuk segmen 2x2 sebesar 70 detik, untuk ukuran gambar 480x320 terjadi pada proses dengan 4 komputer untuk segmen 2x2 sebesar 100 detik, untuk ukuran gambar 640x480 terjadi pada proses dengan 4 komputer untuk segmen 3x3 sebesar 140 detik, untuk ukuran gambar 800x600 terjadi pada proses dengan 4 komputer untuk segmen 3x3 sebesar 192 detik, sedangkan untuk ukuran gambar 1024x800 terjadi pada proses dengan empat komputer untuk segmen 3x3 sebesar 273 detik.

5.2 SARAN

1. Aplikasi ini hanya diterapkan pada proses render gambar, diharapkan dapat diterapkan untuk proses render audio, video ataupun animasi 3 dimensi.
2. Dikarenakan terbatasnya sarana dan prasarana sehingga implementasi render gambar secara grid hanya menggunakan 6 komputer dan masih menggunakan jaringan lokal. Diharapkan implementasi ini bisa dilakukan pada jaringan internet dan jumlah komputer yang lebih banyak serta untuk gambar ukuran sangat besar seperti 100 *Megapixel*.

DAFTAR PUSTAKA

1. Bader Aljaber, Thomas Jacobs, Krishna Nadiminti, Rajkumar Buyya, 2007. *Multimedia On Global Grids: A Case Study In Distributed Ray Tracing***,** Malaysian Journal of Computer Science, Malaysia
2. Bobby Nazief. 2006. *RI-GRID: Usulan Pengembangan Infrastruktur Komputasi Grid Nasional*. Prosiding e-Indonesia Initiative. Bandung.
3. Gengki, SWA, 2006 *Analisis Kinerja Dan Implementasi Parallel Prosessing Untuk Menyelesaikan Perkalian Matrik Menggunakan PVM Berbasis Linux.* Ilmu komputer Universitas Brawijaya, Malang.
4. Krishna Nadiminti, Akshay Luther, Rajkumar Buyya. 2005. *Alchemi: A .NET-based Enterprise Grid System and Framework***.** Grid Computing and Distributed Systems (GRIDS) Laboratory Dept. of Computer Science and Software Engineering The University of Melbourne, Australia.
5. Krishna Nadiminti, Rajkumar Buyya, 2006. *Enterprice Grid Computing: State of the Art.* Grid Computing and Distributed Systems (GRIDS) Laboratory Dept. of Computer Science and Software Engineering The University of Melbourne, Australia.
6. Kurniawan, Agus, Dkk. 2004. *Pengenalan Bahasa C#.* Project Otak. Jakarta.
7. Kwai Wah, Ho. 2004. *A New Grid Portal for Managing and Monitoring Application Execution on Global Grids through Multiple Devices***.** Grid Computing and Distributed Systems (GRIDS) Laboratory Dept. of Computer Science and Software Engineering The University of Melbourne, Australia.
8. M9!, 2007. *"Grid Computing" Populer di Asia Pasifik*. Officialweb Kabupaten Bandung. Bandung.
9. MegaPOV-Team, 2005. *MegaPov Documentation*. MegaPOV Community. Australia.

10. POV-Team, 2005. *Persistence of Vision Ray Tracer™ (POV-Ray™) Version 3.6.* Persistence of Vision Raytracer Pty. Ltd. Victoria, Australia.

11. Sulian Mozes L. Sedubun, 2005. *Perancangan dan Pembuatan Perangkat Lunak untuk Me-Render Mesh Object dengan Menggunakan Metode Ray Tracing.* Universitas Kristen Petra. Surabaya.

12. Team InGrid. 2007. *Rancangan Infrastruktur inGRID.* Grid Computing Research Group Fakultas Ilmu Komputer Universitas Indonesia. Jakarta.