TUGAS AKHIR – KI141502

KONSTRUKSI BOUNDING VOLUME HIERARCHY DENGAN METODE AGGLOMERATIVE CLUSTERING UNTUK MENINGKATKAN PERFORMA RAY TRACING

ARIF FATHUR MAHMUDA
NRP 51112100118

Dosen Pembimbing I
Anny Yuniarti, S.Kom.,M.Comp.Sc.

Dosen Pembimbing II
Wijayanti Nurul K., S.Kom., M.Sc

JURUSAN TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INFORMASI
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA 2017

TUGAS AKHIR – KI141502

KONSTRUKSI BOUNDING VOLUME HIERARCHY DENGAN METODE AGGLOMERATIVE CLUSTERING UNTUK MENINGKATKAN PERFORMA RAY TRACING

ARIF FATHUR MAHMUDA
NRP 51112100118

Dosen Pembimbing I
Anny Yuniarti, S.Kom.,M.Comp.Sc.

Dosen Pembimbing II
Wijayanti Nurul K., S.Kom., M.Sc

JURUSAN TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INFORMASI
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA 2017

(Halaman ini sengaja dikosongkan)

FINAL PROJECT – KI141502

BOUNDING VOLUME HIERARCHY CONSTRUCTION USING AGGLOMERATIVE CLUSTERING FOR FASTER RAY TRACING

ARIF FATHUR MAHMUDA
NRP 51112100118

Dosen Pembimbing I
Anny Yuniarti, S.Kom.,M.Comp.Sc.

Dosen Pembimbing II
Wijayanti Nurul K., S.Kom., M.Sc

DEPARTMENT OF INFORMATICS
FACULTY OF INFORMATION TECHNOLOGY
SEPULUH NOPEMBER INSTITUTE OF TECHNOLOGY
SURABAYA 2017

(Halaman ini sengaja dikosongkan)

LEMBAR PENGESAHAN

Konstruksi Bounding Volume Hierarchy Dengan Metode Agglomerative Clustering Untuk Meningkatkan Performa Ray Tracing

TUGAS AKHIR

Diajukan Untuk Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Rumpun Mata Kuliah Interaksi, Grafika, dan Seni
Program Studi S-1 Jurusan Teknik Informatika
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember

Oleh:

ARIF FATHUR MAHMUDA
NRP: 5112 100 118

Disetujui oleh Pembimbing Tugas Akhir:

Anny Yuniarti, S.Kom., M.Comp.Sc. .....
NIP: 19810622 200501 2 002 (pembimbing 1)

Wijayanti Nurul K., S.Kom., M.Sc. .....
NIP: 19860312 201212 2 004 (pembimbing 2)

SURABAYA
Januari, 2017

(Halaman ini sengaja dikosongkan)

KONSTRUKSI BOUNDING VOLUME HIERARCHY DENGAN METODE AGGLOMERATIVE CLUSTERING UNTUK MENINGKATKAN PERFORMA RAY TRACING

Nama	: Arif Fathur Mahmuda
NRP	: 5112100118
Jurusan	: Teknik Informatika, FTIf-ITS
Dosen Pembimbing I	: Anny Yuniarti, S.Kom., M.Comp.Sc.
Dosen Pembimbing II	: Wijayanti Nurul K., S.Kom., M.Sc.

Abstrak

Ray Tracing Sebagai algoritma *rendering* yang menghasilkan citra realistis memiliki beberapa kekurangan. Salah satu di antaranya adalah perhitungan persilangan *ray-object* pada tiap *pixel* yang memakan 75% waktu dari keseluruhan proses [1].

Penulis menerapkan metode yang diharapkan dapat mempersingkat proses perhitungan persilangan *ray-object* dengan membangun struktur data berupa *binary tree* (*Bounding Volume Hierarchy*) di mana masing-masing *node*-nya adalah sebuah *container* [2]. *Container* tersebut akan mencakup *container* lain atau sebuah poligon. Struktur data tersebut akan dibangun dengan metode *Approximate Agglomerative Clustering* (AAC) yang merupakan metode *bottom-up clustering* dengan *top-down preprocessing* [3].

Dengan Metode yang diterapkan, diharapkan performa Ray Tracing akan meningkat secara signifikan. Metode AAC dengan parameter yang baik dapat meningkatkan performa Ray Tracing. Metode-metode yang diterapkan sangat mudah diparalelkan sehingga performa algoritma meningkat jika dijalankan pada lingkungan paralel. Hasil uji coba penulis menunjukkan peningkatan kecepatan hingga 3 kali lipat dibandingkan tanpa menerapkan paralelisme. Pada hasil uji coba, penulis juga mendapatkan dua jenis parameter yang masing-masing memiliki karakteristik tersendiri (6= cepat, 12= HQ).

Kata kunci: Ray Tracing, Rendering, Agglomerative Clustering, Grafika, BVH

(Halaman ini sengaja dikosongkan)

BOUNDING VOLUME HIERARCHY CONSTRUCTION USING AGGLOMERATIVE CLUSTERING FOR FASTER RAY TRACING

Name	: Arif Fathur Mahmuda
NRP	: 5112100118
Department	: Informatics, FTIf-ITS
Supervisor I	: Anny Yuniarti, S.Kom., M.Comp.Sc.
Supervisor II	: Wijayanti Nurul K., S.Kom., M.Sc.

Abstract

Ray tracing as a realistic rendering method has some limitation. One of them is expensive cost for ray-object intersection that takes 75% of the whole process [1]. *Bounding Volume Hierarchy will greatly improve this step by eliminating intersection calculation with irrelevant object. We could achieve this by designing a tree that contain nodes of containers* [2]. *Each node will contains two other container or an object.*

We implement some methods that will greatly improve the construction of bounding volume hierarchy's efficiency. We implement a method based on Agglomerative Clustering with some preprocessing that will groups global nodes into smaller approximately close local groups [3], *thus the name Approximate Agglomerative Clustering (AAC) given.*

We later found optimal parameters for AAC algorithm such as container type and threshold. We also investigate some optimization to further boost AAC's efficiency such as parallelism and thread pooling. Our implementation show 3 times boost in speed in parallel environment. We also run several scenarios to get optimum parameter (container type and threshold) values for AAC.

Keywords: Ray Tracing, Rendering, Agglomerative Clustering, Graphics, BVH

(Halaman ini sengaja dikosongkan)

KATA PENGANTAR

Segala puji bagi Allah SWT yang telah melimpahkan rahmat dan anugerah-Nya sehingga penulis dapat menyelesaikan tugas akhir yang berjudul "Konstruksi Bounding Volume Hierarchy dengan Metode Agglomerative Clustering untuk Meningkatkan Performa Ray Tracing" dengan tepat waktu.

Dalam pelaksanaan dan pembuatan tugas akhir ini penulis mendapatkan banyak bantuan dari berbagai pihak. Pada kesempatan ini, ucapan terima kasih penulis berikan kepada:

1. Allah SWT, karena atas limpahan rahmat-Nya, penulis diberikan kemudahan dan kelancaran dalam mengerjakan tugas akhir ini.
2. Orang tua dan keluarga penulis yang senantiasa memberikan doa dan dukungan kepada penulis untuk menyelesaikan pengerjaan tugas akhir ini.
3. Ibu Anny Yuniarti, S.Kom., M.Comp.Sc. dan Wijayanti Nurul K., S.Kom., M.Sc. sebagai dosen pembimbing yang telah memberikan banyak arahan dan bantuan sehingga penulis dapat menyelesaikan tugas akhir ini.
4. Bapak Prof. Ir. Joko Lianto Buliali, M.Sc.,Ph.D. selaku dosen wali yang telah memberikan bimbingan dan nasihat selama masa perkuliahan.
5. Teman-teman *user* TA dan Administrator Laboratorium Interaksi, Grafika, dan Seni yang telah memberikan bantuan dalam penyelesaian tugas akhir ini.
6. Teman-teman indekos Fokusss yang telah memotivasi penulis dalam pengerjaan tugas akhir ini.
7. Semua pihak yang telah membantu terselesaikannya tugas akhir ini.

Penulis menyadari adanya banyak kekurangan dalam pengerjaan tugas akhir baik dari segi program maupun laporan. Oleh karena itu, penulis mengharapkan kritik dan saran yang membangun untuk penyempurnaan tugas akhir ini. Akhir kata,

penulis meminta maaf bila terdapat kesalahan dalam penulisan laporan tugas akhir ini. Semoga hasil dari tugas akhir ini dapat memberikan manfaat bagi pembaca pada umumnya dan penulis pada khususnya.

Surabaya, Januari 2017

Arif Fathur Mahmuda

DAFTAR ISI

DAFTAR GAMBAR

(Halaman ini sengaja dikosongkan)

DAFTAR TABEL

(Halaman ini sengaja dikosongkan)

BAB I
PENDAHULUAN

Pada bab ini akan dipaparkan garis besar tugas akhir yang meliputi latar belakang, tujuan, rumusan dan batasan permasalahan, metodologi pembuatan tugas akhir, dan sistematika penulisan.

1.1 Latar Belakang

Algoritma orisinal Ray Tracing memerlukan perhitungan untuk tiap objek pada tiap garis yang diproyeksikan [2]. Proses tersebut memakan sekitar 75% waktu yang digunakan untuk me-*render* gambar [1]. Oleh sebab itu, banyak metode yang dikembangkan untuk mengurangi kompleksitas proses tersebut. Metode-metode tersebut umumnya memerlukan sebuah struktur data yang menyimpan informasi posisi dan ukuran objek untuk membentuk *tree* sehingga tidak semua objek perlu diperiksa untuk setiap garis proyeksi.

Bounding Volume Hierarchy (BVH) dapat mempercepat proses *rendering* dengan mengabaikan objek yang tidak perlu diperhitungkan. Sayangnya konstruksi BVH bukan hal yang mudah dan konstruksi BVH secara manual tidak praktikal jika dibandingkan dengan kompleksitas *scene* yang akan di-*render*. Dalam tugas akhir ini penulis ingin menerapkan Agglomerative Clustering dengan optimalisasi untuk konstruksi BVH secara otomatis [3, 4].

Metode ini diharapkan akan menghasilkan BVH dengan kualitas baik, waktu konstruksi yang cepat, dan mempercepat proses Ray Tracing secara signifikan.

1.2 Rumusan Masalah

Berikut beberapa hal yang menjadi rumusan masalah dalam tugas akhir ini :

1. Bagaimana membuat objek-objek agar siap untuk dikelompokkan dengan Agglomerative Clustering?
2. Bagaimana mengelompokkan objek-objek dengan Agglomerative Clustering agar dapat terbentuk BVH?
3. Bagaimana melakukan *rendering* dengan BVH yang telah terbentuk?
4. Bagaimana parameter algoritma yang baik untuk algoritma AAC?
5. Bagaimana pengaruh paralelisme terhadap kecepatan algoritma?

1.3 Batasan Masalah

Berikut beberapa hal yang menjadi batasan masalah dalam pengerjaan tugas akhir ini:

1. Objek-objek didefinisikan sebagai segitiga.
2. Objek-objek memiliki posisi dan material.

1.4 Tujuan

Tujuan dari pembuatan tugas akhir ini adalah sebagai berikut.

1. Mengaplikasikan metode Ray Tracing dengan BVH.
2. Membangun struktur BVH dengan cepat dan efisien.

1.5 Metodologi

Tahap yang dilakukan untuk menyelesaikan tugas akhir ini adalah sebagai berikut:

1. **Penyusunan Proposal Tugas Akhir**

   Penulisan proposal ini merupakan tahap awal dalam pengerjaan tugas akhir. Pada proposal ini, penulis mengajukan gagasan konstruksi Bounding Volume

Hierarchy untuk dengan metode Agglomerative Clustering untuk mempercepat Ray Tracing.

2. **Studi Literatur**
   Pada tahap ini dilakukan pencarian informasi dan studi literatur sejumlah referensi tentang konstruksi BVH untuk Ray Tracing. Informasi dan studi literatur tersebut didapat dari buku, internet, dan materi-materi kuliah yang berhubungan dengan metode yang digunakan.
3. **Implementasi**
   Pada tahap ini dibangun perangkat lunak sesuai dengan rancangan yang diajukan pada proposal. Pembangunan perangkat lunak diimplementasikan sesuai dengan konsep yang telah didapatkan saat studi literatur.
4. **Pengujian dan Evaluasi**
   Pada tahapan ini dilakukan uji coba terhadap perangkat lunak yang telah dibuat. Pengujian dan evaluasi akan dilakukan dengan melihat kesesuaian dengan perencanaan. Tahap ini dimaksudkan juga untuk mengevaluasi jalannya sistem, mencari masalah yang mungkin timbul dan mengadakan perbaikan jika terdapat kesalahan.
5. **Penyusunan Buku Tugas Akhir**
   Tahap ini merupakan tahap dokumentasi dari tugas akhir. Buku tugas akhir berisi dasar teori, perancangan, implementasi, serta hasil uji coba dan evaluasi dari aplikasi yang dibangun.

1.6 Sistematika Penulisan

Buku tugas akhir ini disusun dengan sistematika penulisan sebagai berikut:

1. Bab I. Pendahuluan

   Bab pendahuluan berisi penjelasan mengenai latar belakang masalah, rumusan masalah, batasan masalah, tujuan, manfaat dan sistematika penulisan tugas akhir.

2. Bab II. Tinjauan Pustaka

   Bab tinjauan pustaka berisi penjelasan mengenai dasar teori yang mendukung pengerjaan tugas akhir.

3. Bab III. Desain dan Perancangan

   Bab analisis dan perancangan berisi penjelasan mengenai analisis kebutuhan, perancangan sistem dan perangkat yang digunakan dalam pengerjaan tugas akhir serta urutan pelaksanaan proses.

4. Bab IV. Implementasi

   Bab implementasi berisi pembangunan aplikasi Ray Tracing dengan memanfaatkan Bounding Volume Hierarchy yang sesuai dengan perancangan.

5. Bab V. Uji Coba dan Evaluasi

   Bab ini berisi hasil evaluasi aplikasi dengan menggunakan aplikasi yang dibangun. Juga disertakan analisis dari hasil evaluasi perangkat lunak.

6. Bab VI. Kesimpulan dan Saran

   Bab kesimpulan dan saran berisi kesimpulan hasil penelitian. Selain itu, bagian ini berisi saran untuk pengerjaan lebih lanjut atau permasalahan yang dialami dalam proses pengerjaan tugas akhir.

BAB II
TINJAUAN PUSTAKA

Bab tinjauan pustaka berisi penjelasan teori yang berkaitan dengan implementasi perangkat lunak. Penjelasan tersebut bertujuan untuk memberikan gambaran mengenai sistem yang akan dibangun dan berguna sebagai penunjang dalam pengembangan perangkat lunak.

2.1 Path Tracing

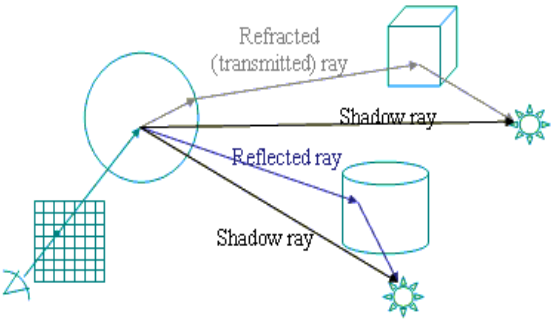
Path Tracing adalah metode me-*render* gambar dengan cara menembakkan sejumlah garis (*ray*) dan mendeteksi persilangannya dengan objek-objek pada *scene* [4]. Garis yang dimaksud adalah sebuah garis dari titik pusat (kamera) yang melewati sebuah titik pada bidang (*view plane*) yang tegak lurus terhadap arah kamera.

2.2 Recursive Ray Tracing

Ray Tracing adalah metode pengembangan dari Path Tracing yang dikenalkan oleh Turner Whitted [1]. Ray Tracing memungkinkan perhitungan *reflection*, *refraction*, dan *shadow*. Hal ini dimungkinkan dengan melakukan rekursi algoritma Path Tracing pada titik perpotongan. Pada sebuah titik perpotongan, garis dibagi menjadi tiga garis baru yang digunakan untuk menghitung;

1) *Reflection* dengan menghitung pantulan garis terhadap titik persilangan.
2) *Refraction* dengan menghitung perubahan arah garis terhadap titik perpotongan.
3) *Shadow* dengan menghitung intensitas tiap sumber cahaya terhadap titik perpotongan dengan memperhitungkan perpotongan dengan objek lain dan sumber cahaya.

Ilustrasi Ray Tracing dapat dilihat pada Gambar 2.1. Ketiga nilai tersebut dijumlahkan dengan pencahayaan global dan material objek yang berpotongan dengan garis. Metode Ray Tracing dapat menghasilkan gambar yang realistis karena memperhitungkan pencahayaan global dan objek di sekitarnya. Salah satu contoh citra yang dihasilkan metode Ray Tracing ditunjukkan pada Gambar 2.2.



Gambar 2.1 Diagram sederhana Ray Tracing [5]

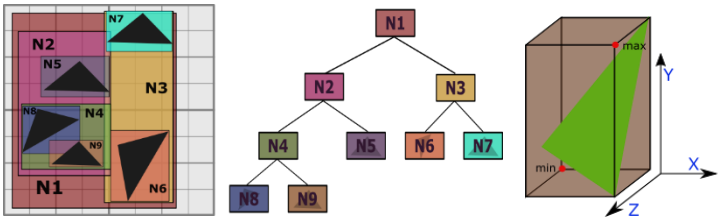
Gambar 2.2 Citra yang di-*render* dengan Ray Tracing [6]

2.3 Bounding Volume Hierarchy

Bounding Volume Hierarchy (BVH) adalah sebuah struktur data berupa *tree* yang terdiri dari *container* yang melingkupi objek atau *container* lainnya. Dengan menggunakan BVH, kita dapat mengabaikan objek yang *parent*-nya tidak berpotongan dengan garis yang ditembakkan. Dengan mengabaikan objek-objek yang tidak relevan, kita dapat mempercepat proses Ray Tracing secara signifikan. Gambar 2.3 (a,b) Menunjukkan ilustrasi BVH pada *scene* 2 dimensi dan representasi BVH pada *tree*.

2.4 Axis Aligned Bounding Box

Axis Aligned Bounding Box (AABB) adalah metode yang umum digunakan untuk mendefinisikan *bounding volume* sebuah objek. AABB didefinisikan sebagai dua titik yang merepresentasikan sebuah balok (titik *min* dan *max*). Tiap sisi pada AABB paralel dengan salah satu bidang yang dibentuk dari dua sumbu pada *world space.* Gambar 2.3 (c) mengilustrasikan AABB pada ruang tiga dimensi.



(a) (b) (c)

Gambar 2.3 (a) Ilustrasi Bounding Volume Hierarchy pada bidang 2 dimensi. (b) ilustrasi BVH dalam bentuk *tree*. (c) ilustrasi AABB sebagai *container* dalam BVH

2.5 Agglomerative Clustering

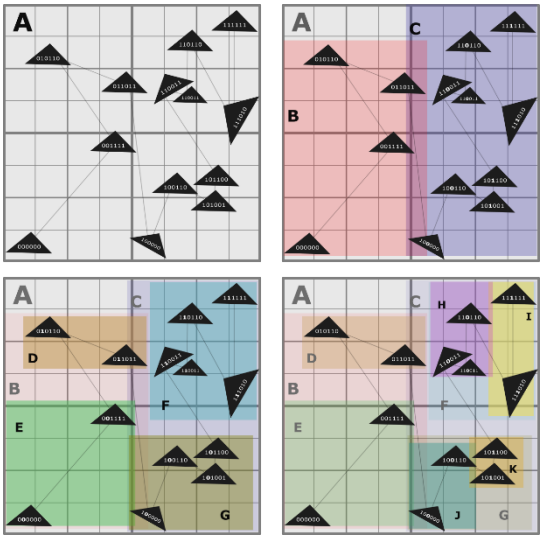
Agglomerative Clustering merupakan salah satu pendekatan konstruksi *cluster tree*. Agglomerative Clustering menggunakan pendekatan *bottom-up* di mana pada awal proses semua objek didefinisikan sebagai sebuah cluster lalu pada iterasi selanjutnya dua *cluster* dengan jarak terdekat akan membentuk *cluster* baru. Proses berakhir jika semua *cluster* telah menjadi anggota satu *cluster* atau jika jumlah *cluster* sama dengan jumlah *threshold*.

2.6 Approximate Agglomerative Clustering

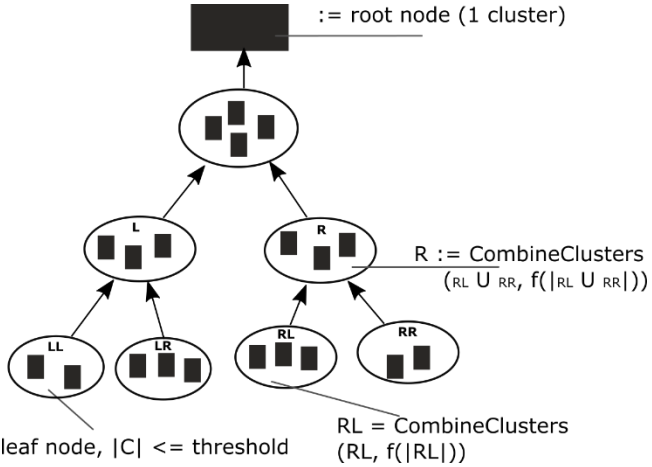
Agglomerative Clustering memiliki beberapa kelemahan yaitu biaya perhitungan yang tinggi pada tahap awal *clustering*. Hal ini dikarenakan seluruh *node* akan mencari *node* lain dengan jarak terdekat. Approximate Agglomerative Clustering (AAC) adalah algoritma yang dikembangkan untuk melakukan *clustering* dengan lebih efisien.

Fitur utama dari AAC adalah pembagian *subset* di mana sebuah *node* hanya perlu mencari tetangga terdekat dari *subset* tersebut [3]. Dengan *node* yang sudah tersortir, pembagian *subset* dapat dilakukan dengan cepat dan tidak akan mengurangi kualitas *cluster* secara signifikan.

AAC diawali dengan pengecekan apakah jumlah *node* memenuhi batas tertentu, jika dirasa jumlah *node* dalam sebuah *subset* masih terlalu besar maka *subset* tersebut akan dibagi menjadi dua *subset* lebih kecil dan seterusnya hingga jumlah *node* kurang dari batas tertentu (tahap *downward*) (Gambar 2.4). Setelah jumlah *node* dalam suatu *subset* memenuhi kriteria, akan dilakukan Agglomerative Clustering (tahap *upward*) (Gambar 2.5). Jumlah *node* yang dihasilkan bervariasi pada tiap level untuk menjaga kualitas *cluster* sampai *cluster* pada level tertinggi (*root*).



Gambar 2.4 AAC tahap I (*downward*)



Gambar 2.5 AAC tahap II (*upward*)

2.7 Morton Code

Morton Code (disebut juga dengan *z-ordering*) adalah salah satu cara merepresentasikan posisi titik pada n-dimensi menjadi 1 dimensi. Morton Code digunakan oleh Gargantini [7] sebagai metode untuk merepresentasikan *node* pada *quad tree*.

Morton Code memudahkan perhitungan fungsi jarak antar dia titik pada dimensi lebih dari satu dan memudahkan *sorting* data. Morton Code juga sering digunakan dalam pembentukan BVH. Pada implementasi kali ini Morton Code hanya digunakan sebagai pemisah dalam satu *set node* pada salah satu tahap *clustering* [3]. *Pseudocode* Morton Code dapat dilihat pada Kode Sumber 2.1.

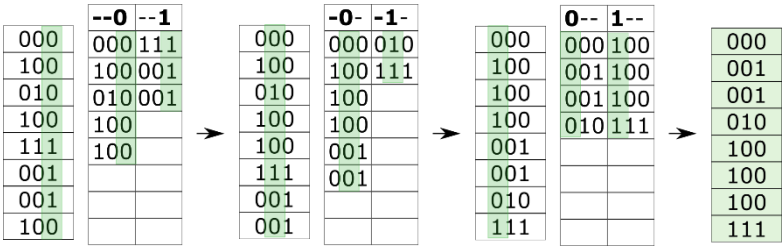
2.8 Radix Sort

Radix Sort adalah metode *sorting* dengan menyortir tiap digit dari elemen yang ingin diurutkan. Radix Sort juga merupakan algoritma *sorting* dengan stabilitas yang baik dengan kompleksitas linear. Dengan kondisi *input* tertentu, Radix Sort lebih efisien dari metode *sorting* umum seperti Quick Sort. Radix Sort juga mudah diparalelkan sehingga dapat di optimalisasi bersamaan dengan metode lain pada tugas akhir ini.

Cara kerja Radix Sort adalah dengan secara bertahap mengurutkan tiap elemen pada tiap angka. Metode ini hanya bisa diterapkan pada jenis data integer. Radix Sort juga sulit digeneralisasikan sehingga implementasi pada *standard library* sulit ditemukan. Gambar 2.6 menunjukkan ilustrasi *Radix Sort Least Significant Digit*.

```
1.  Morton Code(P)  {
2.  In  : point P = {x,y,z} (3 unsigned int 10-bit)
3.  Out : morton code       (  unsigned int 30-bit)
4.
5.   x := expandBits(x) << 2 //shift left(<<) two bits
6.   y := expandBits(y) << 1 //shift left(<<) one bit
7.   z := expandBits(z)
8.
9.   result = x + y + z
10.
11.  return result;
12. }
13.
14. Uint expandBits(int i)
15. {
16.  In  : unsigned int (10-bits)
17.        Ex  1   0   1   1   0   1   0   1   1   1
18.  Out : unsigned int (30-bits)
19.        Ex  100 000 100 100 000 100 000 100 100 100
20.
21.   i:= INSERT '00' after each bits
22.   return i
23. }
```

Kode Sumber 2.1 Pseudocode Morton Code pada dimensi tiga



Gambar 2.6 ilustrasi Radix Sort LSD dengan memanfaatkan Counting Sort

(Halaman ini sengaja dikosongkan)

BAB III
DESAIN DAN PERANCANGAN

Pada Bab 3 akan dijelaskan mengenai perancangan sistem perangkat lunak untuk mencapai tujuan dari tugas akhir. Perancangan yang akan dijelaskan pada bab ini meliputi perancangan data, perancangan proses dan perancangan antar muka. Selain itu akan dijelaskan juga desain metode secara umum pada sistem.

3.1 Perancangan Data

Perancangan data merupakan bagian yang terpenting dalam pengoperasian perangkat lunak karena diperlukan data yang tepat agar perangkat lunak dapat beroperasi dengan benar. Pada bagian ini akan dijelaskan mengenai perancangan data yang dibutuhkan untuk membangun perangkat lunak Ray Tracer dengan BVH. Data yang diperlukan dalam pengoperasian perangkat lunak adalah data masukan (*input*) dan data keluaran (*output*) yang memberikan hasil proses pengoperasian perangkat lunak untuk pengguna.

3.1.1 Data Masukan

Data masukan adalah data awal yang akan diproses pada sistem Ray Tracing. Data yang diproses berupa *file* teks yang mendefinisikan *scene* yang akan dibuat. Format data masukan mengikuti format *file* model WaveFront (.obj dan .mtl). Beberapa parameter yang dibutuhkan penulis dimasukkan dalam sebuah *file scene* (.scene). Beberapa contoh *file* dapat dilihat pada Gambar 3.1.

File scene (.scene) mendefinisikan:

1. Ukuran *scene*
2. Nama *file* keluaran
3. Kedalaman rekursi pada Ray Tracing
4. Kamera

5. Sumber Cahaya
6. *File* model (.obj)

File model (.obj) mendefinisikan:

1. *Vertex*
2. Poligon
3. Material yang digunakan
4. *File* material (.mtl)

File material (.mtl) mendefinisikan:

1. Material
2. Properti material

3.1.2 Data Keluaran

Data keluaran dari sistem ini adalah hasil Ray Tracing dalam bentuk citra digital dengan format Bitmap (.bmp). Contoh data keluaran dari program ditunjukkan pada Gambar 3.2.

3.2 Desain Metode Secara Umum

Pada tugas akhir ini akan dibangun suatu sistem untuk me-*render* gambar berdasarkan *file input* dengan metode Ray Tracing dan optimalisasi BVH. Tahap pertama yang dilakukan oleh sistem adalah tahap *preprocessing*. Pada tahap ini dilakukan proses inisialisasi *scene* yang mendefinisikan ukuran gambar, objek, pencahayaan, kamera, serta material untuk tiap objek. Tahap kedua adalah tahap membangun *BVH* sebagai optimalisasi Ray Tracing. Pada tugas akhir ini *BVH* dibangun dengan metode Approximate Agglomerative Clustering. Tahap ketiga adalah tahap Ray Tracing terhadap *scene*. Tahap yang terakhir menyimpan citra hasil Ray Tracing. Bagan dari sistem yang dibangun ditunjukkan pada Gambar 3.3.

D:\Google Drive\Kuliah\TA\Uji coba\OBJ_teap

Tools Project Preferences Help

OBJ_teapot.obj

```
1  # Blender v2.78 (sub 0) OBJ Fi
2  # www.blender.org
3  mtllib OBJ_teapot.mtl
4  o body_body_Base.001
5  v 0.631959 1.135480 0.087564
6  v 0.643907 1.134238 0.000000
7  v 0.637687 1.135480 0.000000
8  v 0.638124 1.134238 0.088418
9  v 0.650995 1.130509 0.000000
10 v 0.645148 1.130509 0.089391
11 v 0.658635 1.124296 0.000000
12 v 0.652720 1.124296 0.090441
13 v 0.666512 1.115597 0.000000
14 v 0.660526 1.115597 0.091522
15 v 0.674310 1.104412 0.000000
16 v 0.668254 1.104412 0.092593
17 v 0.681714 1.090743 0.000000
18 v 0.615250 1.135480 0.171231
19 v 0.621251 1.134237 0.172901
20 v 0.628089 1.130509 0.174804
21 usemtl Body
22 s 1
23 f 1/1/1 2/2/1 3/3/2
24 f 4/4/3 5/5/3 2/2/3
25 f 6/6/4 7/7/4 5/5/4
26 f 8/8/5 9/9/5 7/7/5
27 f 10/10/6 11/11/6 9/9/6
28 f 12/12/7 13/13/8 11/11/8
```

D:\Google Drive\Kuliah\TA\Uji coba\scene_teapot.sce

Tools Project Preferences Help

scene_teapot.scene

```
1  #size 1280 960
2  size 640 480
3  #size 320 240
4  #size 160 120
5  #size 120 80
6  #size 60 40
7
8  output untitled
9
10 camera 0 3 5 0 1 0 0 1 0 45
11
12 maxdepth 3
13
14 point 1 4 6 .5 .5 .5
15 point 2 2 5 .5 .5 .5
16 #directional -7.4 -6.5 -5.3 .2 .2 .2
```

D:\Google Drive\Kuliah\TA\Uji coba\OBJ_teapot.m

Tools Project Preferences Help

OBJ_teapot.mtl

```
1  # Blender MTL File: 'None'
2  # Material Count: 2
3
4  newmtl Body
5  Ns 96.078431
6  Ka 0.000000 0.000000 0.000000
7  Kd 0.800000 0.800000 0.800000
8  Ks 0.469027 0.469027 0.469027
9  Ke 0.000000 0.000000 0.000000
10 Ni 1.000000
11 d 1.000000
12 illum 2
13 map_Kd C:\Users\Yahya\Downloads\teap
14
```

Gambar 3.1 Contoh data masukan

Gambar 3.2 Contoh data keluaran

3.3 Perancangan Proses

Berikut ini adalah rancangan dari sistem Ray Tracing dengan BVH:

3.3.1 Tahap *Preprocessing*

Tahap *preprocessing* terbagi menjadi dua proses yaitu, proses membaca parameter pengguna dan menerjemahkan *file scene*.

3.3.1.1 Mengambil parameter algoritma

Proses yang pertama kali dilakukan adalah menetapkan parameter algoritma. Penulis juga menetapkan parameter *default* jika pengguna tidak memasukkan nilai parameter. Parameter-parameter yang dapat dimasukkan pengguna adalah:

- *File scene* (.scene)
- Jenis *container* (b = *box* / s = *sphere*)
- AAC *threshold*
- Jumlah *thread* pada *thread pool*

3.3.1.2 Operasi *parsing file*

Pada proses ini setiap baris akan dibersihkan (distandarkan) dari karakter-karakter yang tidak diperlukan seperti spasi, *tab*, atau spasi ganda. Proses ini bertujuan memudahkan proses penerjemahan baris ke instruksi yang diketahui.

Selanjutnya, dilakukan penerjemahan baris-baris yang relevan ke instruksi-instruksi yang diketahui. Gambar 3.4 menunjukkan proses iterasi tiap baris di mana tiap baris akan distandarkan untuk menghindari masalah pada saat menjalankan perintah. Setelah tiap baris distandarkan, perintah pada setiap baris pada *file* dijalankan, proses ini terlihat pada Gambar 3.5.

Adapun instruksi-instruksi yang tersedia adalah sebagai berikut :

- Pengaturan ukuran
- Nama *file* keluaran
- Batas kedalaman rekursi
- Pengaturan kamera
- Pengaturan objek pencahayaan
  - *Point light*
  - *Directional light*
- Pembentukan *vertex*
- Pembentukan objek
- Penerapan material
- Membaca *file* lain

3.3.2 Pembangunan BVH

Pada tahap ini, akan dilalukan *clustring* pada seluruh objek untuk membentuk struktur data berupa *binary tree* untuk memudahkan tahap Ray Tracing. Metode clustering yang digunakan adalah Approximate Agglomerative Clustering (AAC).

Tahapan pembangunan BVH dibagi menjadi tiga bagian penting. Bagian pertama adalah *preprocessing* data poligon dengan melakukan *sorting* dengan **Radix Sort** yang dijalankan pada awal **buildBVH** (Kode Sumber 3.1). Setelah tahapan *preprocessing*, *cluster* akan dibentuk pada metode **buildTree** (Kode Sumber 3.2) yang akan menghasilkan *cluster* sesuai parameter algoritma. Tahap terakhir adalah menggabungkan seluruh *cluster* yang tersisa menjadi satu *cluster* (*root*) dengan metode **combineClusters** (Kode Sumber 3.3).

3.3.3 Ray Tracing dan *Shading*

Pada tahap ini *scene* sudah memiliki struktur data BVH. Proses Ray Tracing dilakukan untuk menghasilkan citra digital berdasarkan metode Recursive Ray Tracing dengan model pencahayaan Blinn-Phong Shading [8] [1].

3.3.3.1 Ray Tracing

Setelah *scene* dan BVH siap, dilakukan Ray Tracing pada setiap *pixel* pada gambar. Perhitungan perpotongan akan mengikuti bentuk *tree* sehingga *node* yang tidak relevan tidak diperhitungkan. Pseudocode Ray Tracing dengan BVH ditunjukkan pada Kode Sumber 3.4.

3.3.3.2 Perhitungan warna

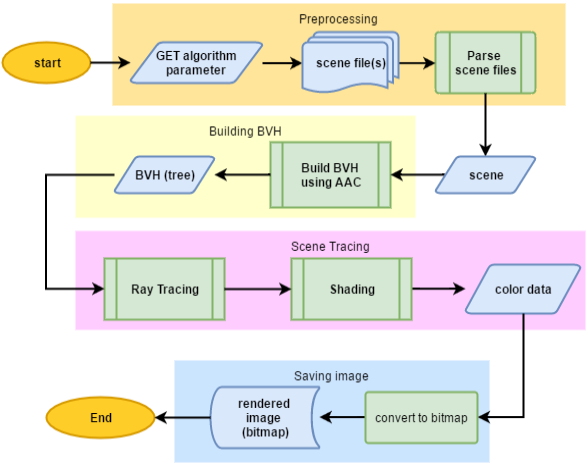
Proses Ray Tracing akan menyimpan informasi titik perpotongan *ray* dan objek. Informasi tersebut akan digunakan pada tahap pewarnaan (*shading*) objek. Pada tiap level *shading* akan dihitung warna objek dan memperhitungkan pewarnaan objek lain jika perlu.

Pada bagian inilah metode Ray Tracing melakukan proses rekursi di mana *ray* dapat membentuk *ray* baru yang dipantulkan, dibelokkan atau di arahkan sedemikian rupa dari objek yang dilewatinya.

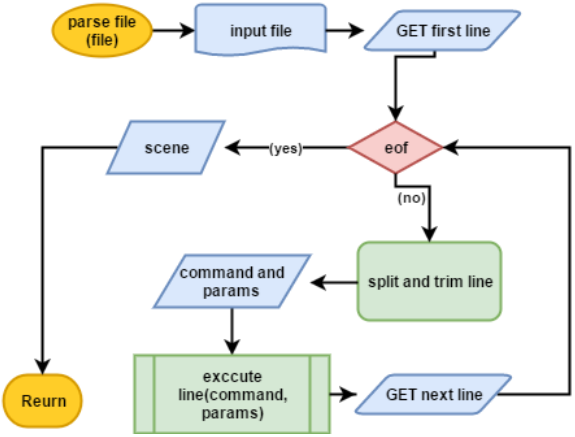
Pada Kode Sumber 3.5 ditunjukkan perhitungan warna objek itu sendiri, *reflection,* dan *refraction*. Perhitungan relevansi cahaya didapatkan dengan menembakkan *ray* ke arah sumber cahaya.

3.3.4 Menyimpan Citra

Tahap Ray Tracing akan menghasilkan informasi warna per *pixel*. Informasi tersebut digunakan untuk membangun citra digital dengan memanfaatkan *library* FreeImage.



Gambar 3.3 diagram alur rancangan sistem



Gambar 3.4 diagram alur penerjemahan *file*

Gambar 3.5 diagram alur menjalankan perintah

```
1.  buildBVH(P)
2.  In : list of polygon P
3.  Out : bvh (tree)
4.
5.    CALCULATE morton code for each P
6.    RadixSort P by morton code
7.    bvh := BuildTree(P)
8.    RETURN CombineClusters(bvh,1)
```

Kode Sumber 3.1 *pseudocode* buildBVH

```
1.  BuildTree(P)
2.  In : sublist of polygon
3.  Out : at most f(|p|) clusters containing P
4.
5.    IF (|P| < t ) THEN  //t = threshold
6.      Generate C for each P
7.      RETURN CombineClusters(C,f(t))
8.    END IF
9.    Split P to Pl and Pr
10.   C := BuildTree(Pl) U buildTree(Pr)
11.   RETURN CombineClusters(C,f(|P|))
```

Kode Sumber 3.2 *pseudocode* buildTree

```
1.  CombineClusters(C,n)
2.  In : list of cluster (C)
3.  Out : list at most n clusters
4.
5.    FOR EACH cluster in C DO
6.      CALL findBestMatch(cluster,C)
7.    END FOR
8.
9.    WHILE |C| > n DO
10.   SET best to MAX
11.   FOR EACH c in C DO
12.     IF d(c,c.closest)<best THEN
13.       best := d(c,c.closest)
14.       Cl := c ; Cr := c.closest
```

```
    END IF
  END FOR

  Cn := newCluster(Cl,Cr)
  C = C – {Cl,Cr} + Cn
  findBestMatch(Cn,C)

  FOR EACH c in C DO
    IF c.closest = {Cl || Cr} THEN
      findBestMatch(c,C)
    END IF
  END FOR

  END WHILE
  RETURN C
```

Kode Sumber 3.3 *pseudocode* combineClusters

```
Traceray(ray,bvh)
In : ray and a container
Out : -

  IF ray.intersect(bvh) THEN
    IF bvh.isLeaf THEN
      IF ray is intersecting polygon THEN
        CALCULATE collisionPoint //cPoint
        IF cPoint < ray.cPoint THEN
          Ray.object := bvh.geo
          ray.cPoint := cPoint
        END IF
      END IF
    ELSE
      CALL TraceRay with bvh.left
      CALL TraceRay with bvh.right
    END IF
  END IF
```

Kode Sumber 3.4 *pseudocode* Ray Tracing dengan BVH

```
1.  getColor(ray,bvh,limit)
2.  In : ray, container, and limit
3.  Out : color (self, reflection, refraction)
4.
5.    DECREASE limit
6.    IF limit <= 0 THEN
7.       return defaultColor
8.    END IF
9.    SET C to defaultColor
10.  //base color
11.  C += calcColor //blinn-phong shading
12.
13.   //reflection color
14.   SET reflctRay to reflectionRay
15.   CALL TraceRay with reflectRay
16.   C += coefRefl * (getColor with reflctRay)
17.
18.   //refraction color
19.   SET refrctRay to refractionRay
20.   CALL TraceRay with refrctRay
21.   C += coefRefr * (getColor with refrctRay)
22.
23.    Return C
```

Kode Sumber 3.5 Recursive Ray Tracing untuk menghitung warna

```
1.  calcColor (ray,obj,lights)
2.  In : ray, material, lighning
3.  Out : color (blinn-phong shading)
4.
5.    SET result to obj.ambient + obj.emmision
6.    CALCULATE normal                  //n
7.    GET material                      //m
8.    FOR each light in lights DO
9.      IF light is Effective THEN
10.       CALCULATE pointToLight      //p
11.       CALCULATE halfAngleToLight//h
12.       CALCULATE attenuation       //att
13.       Result += attenuation*(light.color)*
```

```
14.          ((m.diffuse*(p*n))+
15.           (m.specular*(h*n)^m.shininess))
16.     END IF
17.   END FOR
18.   Return result
```

Kode Sumber 3.6 *blinn-phong shading* untuk perhitungan warna

BAB IV
IMPLEMENTASI

Bab implementasi berisi pembahasan mengenai implementasi perangkat lunak berdasarkan perancangan yang telah dibuat. Tahap perancangan merupakan tahap yang mendasari implementasi perangkat lunak.

4.1 Lingkungan Implementasi

Lingkungan implementasi yang akan digunakan untuk melakukan implementasi meliputi perangkat keras dan perangkat lunak yang dijelaskan sebagai berikut:

1. Perangkat Keras
   a. Prosesor: Intel(R) Core (TM) i7-2670QM CPU @ 2.20GHz
   b. Memori (RAM) : 4096 MB
   c. Tipe Sistem : 64-bit
2. Perangkat Lunak
   a. *OS* : *Windows Embedded 8.1 Industry Pro* 64 bit
   b. Perangkat Pengembang : Microsoft Visual Studio Community 2015

4.2 Implementasi

Sub bab implementasi ini menjelaskan tentang implementasi proses yang sudah dijelaskan pada bab perancangan perangkat lunak.

4.2.1 Implementasi *Preprocessing*

Tahap pertama adalah membaca parameter dan *file* yang akan digunakan pada algoritma-algoritma kemudian. Pada proses ini akan dilakukan pengambilan parameter program dan algoritma (*threshold* dan jenis *container*). Proses selanjutnya adalah membaca dan menjalankan *file scene* yang berisi instruksi-instruksi.

4.2.1.1 Menerima dan memproses parameter input

Input user dimasukkan sebagai argumen pada Command Line saat memanggil program. Untuk setiap argumen terdapat nilai *default* yang penulis tentukan jika *user* tidak memasukkan argumen (Kode Sumber 4.1).

```
1.  int main(int argc, char *argv[])
2.  {
3.   //scene file name, default = "default.scene"
4.   string filename = (argc >= 2) ? argv[1] : "scene_default.scene";
5.   replace(filename.begin(), filename.end(), '\\', '/');
6.
7.   bool isAAC = true;
8.
9.   //bin type (b = box / s = sphere ), default = box;
10.  Container::TYPE binType = (argc >= 3) ? (argv[2][0] == 'b') ?
     Container::BOX : Container::SPHERE : Container::BOX;
11.
12.  //aac treshold, def=12;
13.  int thres = (argc >= 4) ? (atoi(argv[3])) : 12;
14.
15.  //thread number for aac and radix sort
16.  (argc >= 5) ? ThreadPool::setMaxThread((atoi(argv[4]) - 1)) :
     ThreadPool::setMaxThread(thread::hardware_concurrency() - 1);
17.
18.  //block number for tracing
19.  int traceTN = 8 * thread::hardware_concurrency();
20. ...
```

Kode Sumber 4.1 kode sumber parameter algoritma

4.2.1.2 *Parsing file* ke dalam *scene*

Pada tahap ini program akan membaca *file scene* yang berisi ukuran citra hasil, kamera, batas rekursi, nama *file* keluaran, pencahayaan, dan *file* model dalam format OBJ. *File* model berisi vertex, poligon, material yang digunakan, serta *file* material dalam

format MTL. *File* material berisi definisi tiap material pada *file* model.

Pada setiap baris dalam *file*, akan dilakukan *trimming* untuk memudahkan tahap selanjutnya. Implementasi metode ini terdapat pada Kode Sumber 4.2.

```
1.  void Scene::parseFile(string filename)  {
2.    string line;
3.    ifstream myfile(filename);
4.    if (myfile.is_open())   {
5.      while (getline(myfile, line))    {
6.        executeCommand(line);
7.      }
8.      myfile.close();
9.    }
10.   else cout << "Unable to open file";
11. }
12.
13. string Scene::cleanCommand(string command)  {
14.   //replace tab and multiple spaces w/ space
15.   command = regex_replace(command, regex("\t| + "), " ");
16.   //delete leading and trailing spaces
17.   command = regex_replace(command, regex("^ +| +$"), "");
18.   return command;
19. }
20.
21. vector<string> Scene::splitString(string fullcommand, char delimiter) {
22.   vector<string> elems;
23.   stringstream ss(fullcommand);
24.   string item;
25.   while (getline(ss, item, delimiter)) {
26.     if (item.length() > 0) {
27.       elems.push_back(item);
28.     }
29.   }
30.  return elems;
31. }
```

Kode Sumber 4.2 Kode sumber membaca dan membersihkan *file*

Tiap baris yang siap diterjemahkan akan dibandingkan dengan perintah-perintah yang tersedia. Implementasi tahap *parsing* tiap baris perintah ditunjukkan pada Kode Sumber 4.3 sampai Kode Sumber 4.5.

Pada setiap perintah membangun segitiga, juga dilakukan pembangunan Morton Code yang akan digunakan selanjutnya. Morton Code dibangun dengan mengubah nilai tiap komponen (x, y, dan z) yang telah dinormalkan [0.0 , 1.0] ke dalam *range* [0 , 1023] (10 bit uint) lalu menambahkan dua bit '0' pada masing-masing bit (Kode Sumber 4.6 baris 13) sehingga panjangnya menjadi 30 bit. Lalu menggeser komponen x dua bit ke kiri (x << 2)dan komponen y 1 bit ke kiri (y << 1) lalu menjumlahkan ketiga komponen tersebut(30 bit uint).

```
1.  void Scene::executeCommand(string line)  {
2.   if (line.compare("") == 0)
3.   return;
4.   if (line.find('#') != string::npos)
5.   return;
6.
7.   line = CleanCommand(line);
8.
9.   vector< string>words=splitString(line,' ');
10.  string command = words[0];
11.  vector<float> param;
12.
13.  if (command.compare("output") == 0) {
14.   outFileName = words[1] + ".bmp";
15.  }
16.
17.  else if (command.compare("geo") == 0) {
18.   parseFile(words[1]);
19.  }
20.
21.  else if (command.compare("size") == 0) {
22.   for (int i = 0; i < words.size() - 1; i++)  {
23.    param.push_back(stof(words[i + 1])); }
```

```
24.    size[0] = (int)param[0];
25.    size[1] = (int)param[1];
26.   }
27.
28.   else if (command.compare("maxdepth") == 0)
29.   {
30.    for (int i = 0; i < words.size() - 1; i++)  {
31.     param.push_back(stof(words[i + 1])); }
32.    maxDepth = (int)param[0];
33.   }
34.
35.  else if (command.compare("camera") == 0) {
36.    for (int i = 0; i < words.size() - 1; i++)    {
37.     param.push_back(stof(words[i + 1])); }
38.    Camera::getInstance()->init(&m[0]);
39.    ViewPlane::getInstance()->init(size[0], size[1]);
40.  }
41.
42.  else if (command.compare("point") == 0)   {
43.    vector<float> param;
44.    for (int i = 0; i < words.size() - 1; i++)    {
45.      param.push_back(stof(words[i + 1]));    }
46.    lights.push_back(make_shared<PointLight>(new Vec3( param[0],
       param[1], param[2], 1), new MyColor(param[3], param[4], param[5]))); }
47.
48.   else if (command.compare("directional") == 0)   {
49.    vector<float> param;
50.    for (int i = 0; i < words.size() - 1; i++)  {
51.      param.push_back(stof(words[i + 1])); }
52.    lights.push_back(make_shared<DirectionalLight>(
       new Vec3(param[0], param[1], param[2], 0),
       new MyColor(param[3], param[4], param[5]))); }
53.
54.  ...
```

Kode Sumber 4.3 Kode sumber *execute command.* Perintah-perintah *file scene*

```
1.   ...
2.    else if (command.compare("mtlib") == 0) {
3.    parseFile(words[1]);
```

```
4.    }
5.
6.    else if (command.compare("v") == 0)  {
7.     for (int i = 0; i < words.size() - 1; i++)   {
8.      param.push_back(stof(words[i + 1])); }
9.     vertices.push_back(make_shared<Vec3>(&m[0], 1));
10.   }
11.
12.   else if (command.compare("f") == 0)  {
13.    for (int i = 0; i < words.size() - 1; i++)   {
14.     string temp = splitString(words[i + 1], '/')[0];
15.     param.push_back(stof(temp));   }
16.    auto geo = createShape(&m[0]);
17.    geometries.push_back(geo);
18.   }
19.
20.   else if (command.compare("usemtl") == 0)  {
21.    for each (auto var in material)   {
22.     if (var->name == words[1])
23.      currMat = find(material.begin(), material.end(), var);
24.    }
25.   }
26.  ...
```

Kode Sumber 4.4 Kode sumber *execute command.* Perintah-perintah *file* objek

```
1.   ...
2.    else if (command.compare("newmtl") == 0) {
3.     material.push_back(make_shared<Material>());
4.     material.back()->name = words[1];  }
5.
6.    else if (command.compare("Ka") == 0)   {
7.     for (int i = 0; i < 3; i++) { param.push_back(stof(words[i + 1])); }
8.     material.back()->reflVal = MyColor(param[0], param[1], param[2]); }
9.
10.   else if (command.compare("Kd") == 0)   {
11.    for (int i = 0; i < 3; i++) {
12.     param.push_back(stof(words[i + 1])); }
13.    material.back()->diffuse = (MyColor(param[0], param[1], param[2]));
14.   }
```

```
15.
16.   else if (command.compare("Ks") == 0)   {
17.    for (int i = 0; i < 3; i++) {
18.     param.push_back(stof(words[i + 1])); }
19.    material.back()-
      >specular = (MyColor(param[0], param[1], param[2])  ); }
20.
21.   else if (command.compare("Ke") == 0)   {
22.    for (int i = 0; i < 3; i++) {
23.     param.push_back(stof(words[i + 1])); }
24.    material.back()-
      >emmission = (MyColor(param[0], param[1], param[2]));   }
25.
26.   else if (command.compare("Ns") == 0)   {
27.    material.back()->setShininess(stof(words[1]));   }
28.
29.   else if (command.compare("Ni") == 0)   {
30.    material.back()->setrefIndex(stof(words[1]));   }
31.
32.   else if (command.compare("d") == 0)  {
33.    material.back()->setRefValue(stof(words[1]));   }
34.
35.  …
```

Kode Sumber 4.5 Kode sumber *execute command.* Perintah-perintah *file* material

```
1.    uint32_t Triangle::getMortonPos()  {
2.     if (!hasMorton) {
3.      uint32_t x = expandBits(fminf(fmaxf(pos[0] * 1024.f, 0.f), 1023.f));
4.      uint32_t y = expandBits(fminf(fmaxf(pos[1] * 1024.f, 0.f), 1023.f));
5.      uint32_t z = expandBits(fminf(fmaxf(pos[2] * 1024.f, 0.f), 1023.f));
6.      mortonCode = (  x * 4 + y * 2 + z);
7.      mortonBitset = bitset<30>(mortonCode);
8.      hasMorton = true;
9.     }
10.    return mortonCode;
11.   }
12.
13.   uint32_t Triangle::expandBits(uint32_t v)  {
14.    v = (v * 0x00010001u) & 0xFF0000FFu;
```

```
15.   v = (v * 0x00000101u) & 0x0F00F00Fu;
16.   v = (v * 0x00000011u) & 0xC30C30C3u;
17.   v = (v * 0x00000005u) & 0x49249249u;
18.   return v;
19. }
```

Kode Sumber 4.6 Kode sumber untuk membangun Morton Code

4.2.2 Implementasi Pembangunan BVH

Setelah *scene* siap, Tracer akan menginisialisasi BVH *builder* (*threshold* dan jenis *container*) dengan metode AAC (Kode Sumber 4.7) lalu memanggil metode **buildBVH** untuk membangun BVH. Tahap pertama adalah pembentukan *builder* dengan parameter dari tahap pertama.

Metode **buildBVH** (Kode Sumber 4.8) menerima daftar poligon yang telah memiliki Morton Code. Hal pertama yang dilakukan adalah melakukan *sorting* poligon dengan memanfaatkan Radix Sort. Selanjutnya, dilakukan pembangunan *tree* dan *assignment tree* pada *scene*.

Metode **buildTree** (Kode Sumber 4.9) menerima daftar poligon untuk diproses secara paralel. Metode **buildTree** akan membuat *node* dengan container yang melingkupi masing-masing poligon jika jumlah poligon kurang dari *threshold*. Sedangkan jika jumlah poligon melebihi *threshold*, maka daftar poligon akan dibagi dua *sub list* berdasarkan metode **getPivot** yang memanfaatkan perubahan bit pada Morton Code sehingga menjamin *list* terbagi secara optimal. Masing-masing *sub list* tersebut akan dijadikan parameter pada metode **buildTree** yang dipanggil secara paralel dan rekursi. Selanjutnya *list container* akan dikirim ke metode **combineCluster**. Metode **buildTree** akan mengembalikan daftar *container* dengan jumlah bervariasi berdasarkan fungsi reduksi yang telah ditentukan.

Metode **combineCluster** (Kode Sumber 4.10) menerima masukan **n** *container* dan akan mengembalikan **m** *container*. Metode **combineCluster** adalah metode *clustering* berdasarkan Agglomerative Clustering di mana dua *container* terdekat akan dijadikan satu *container* baru dengan dua *container* tersebut menjadi *child node* dari *node container* yang baru. Proses ini dilakukan sehingga tepat ada **m** *container* pada *list*. Optimalisasi pada **combineCluster** dilakukan dengan menyimpan nilai jarak dua *container*.

```
1.  BVHBuilder::BVHBuilder(Container::TYPE _type, bool _isAAC, int _thresho
    ld) {
2.   type = _type;
3.   isAAC = _isAAC;
4.   threshold = _threshold;
5.  }
6.
7.  void TraceManager::buildBVH() {
8.  BVHBuilder(binType,isAAC,aacThres).buildBVH(scene.geometries);
9.  }
```

Kode Sumber 4.7 Inisialisasi BVH *builder*

```
1.  shared_ptr<Container> BVHBuilder::buildBVH(vector<shared_ptr<Triangl
    e>>& primitives) {
2.   int n = primitives.size();
3.
4.   //nothing in scene
5.   if (n == 0)
6.    return nullptr;
7.
8.   vector<shared_ptr< Container>> temp;
9.   if (!isAAC)//not optimized agglomerative clustering   {
10.   buildTree_LORD(0, temp, primitives, type);   }
11.  else {//using optimized agglomerative clustering (AAC)
12.    if (ThreadPool::tp.n_idle() > 0) {
13.     future<void> buildtree = ThreadPool::tp.push
        (buildTree_AAC, ref(temp), ref(primitives), type);
14.     buildtree.get();
```

```
15.    }
16.    else
17.     buildTree_AAC(0, temp, primitives, type);
18.    }
19.    //create complete tree
20.    combineCluster(temp, 1);
21.   }
```

Kode Sumber 4.8 Kode sumber metode buildBVH

```
1.   void BVHBuilder::buildTree_AAC(int id, vector<shared_ptr<Container>>&
     bins, vector<shared_ptr<Triangle>>& primitives, Container::TYPE _type)
     {
2.    //create clusters if polygon number < threshold
3.    if (primitives.size() <= threshold)   {
4.     ContainerFactory cf;
5.     for (int i = 0; i < primitives.size(); i++)    {
6.       bins.push_back(cf.CreateContainer(primitives[i], _type));
7.     }
8.     combineCluster(bins, f(threshold));
9.     return;
10.   }
11.
12.   //split primitives into two groups besed on pivot
13.   int pivot = getPivot(primitives);
14.   vector<shared_ptr<Triangle>>
      lPrimitives(primitives.begin(), primitives.begin() + pivot);
      // pivot included in left
15.   vector<shared_ptr<Triangle>>
      rPrimitives(primitives.begin() + pivot, primitives.end());
16.
17.   //build left and right tree separately
18.   vector<shared_ptr< Container>>& lTree = bins;
19.   vector<shared_ptr< Container>> rTree;
20.
21.   //if there's an indle thread in pool, push new thread.
22.   future<void> lBuild;
23.   switch (ThreadPool::tp.n_idle() > 0)   {
24.   case true:
```

```
25.    lBuild = ThreadPool::tp.push
                    (buildTree_AAC, ref(lTree), ref(lPrimitives), _type);
26.    buildTree_AAC(id, rTree, rPrimitives, _type);
27.    lBuild.get();
28.    break;
29.   case false:
30.    buildTree_AAC(id, lTree, lPrimitives, _type);
31.    buildTree_AAC(id, rTree, rPrimitives, _type);
32.    break;
33.   }
34.
35.   /*combine two vec and then create clusters*/
36.   move(rTree.begin(), rTree.end(), inserter(bins, bins.end()));
37.   combineCluster(bins, f(bins.size()));
38.   return;
39.  }
```

Kode Sumber 4.9 Kode sumber metode buildTree

```
1.   void BVHBuilder::combineCluster
                              (vector<shared_ptr<Container>>& bins, int limit)  {
2.    ContainerFactory cf;
3.    /*precalculate bestmatch to be used in iteration*/
4.    for (int i = 0; i < bins.size(); i++)
5.     cf.findBestMatch(bins[i], bins);
6.
7.    float bestDist;
8.    shared_ptr< Container> left;
9.    shared_ptr< Container> right;
10.
11.   while (bins.size() > limit)   {
12.    bestDist = INFINITY;
13.    left = nullptr;
14.    right = nullptr;
15.
16.    /*iterate to search best match*/
17.    for (int i = 0; i < bins.size(); i++)    {
18.     if (bins[i]->areaWithClosest < bestDist)     {
19.       bestDist = bins[i]->areaWithClosest;
20.       left = bins[i];
21.       right = bins[i]->closest;
```

```
22.    }
23.   }
24.
25.   /*   delete L and R from bins, push new bin [N]:   */
26.   auto newBin(cf.combineContainer(left, right));
27.   bins.push_back(newBin);
28.   auto indexL = find(bins.begin(), bins.end(), left);
29.   swap(*indexL, bins.back()); bins.pop_back();
30.   auto indexR = find(bins.begin(), bins.end(), right);
31.   swap(*indexR, bins.back()); bins.pop_back();
32.
33.   /* change bestmatch of bin if its bestmatch is [L] or [R] */
34.   cf.findBestMatch(newBin, bins);
35.   for (int i = 0; i < bins.size(); i++)    {
36.    if (bins[i]->closest == left || bins[i]->closest == right)
37.     cf.findBestMatch(bins[i], bins);
38.   }
39.  }
40. }
```

Kode Sumber 4.10 Kode sumber metode combineCluster

Beberapa metode pendukung yang dibutuhkan ditunjukkan pada Kode Sumber 4.11. metode **getPivot** akan membagi *set* poligon menjadi dua *subset* berdasarkan perubahan bit pada Morton Code. Fungsi **f** adalah fungsi yang menentukan reduksi *subset* dari n menjadi **f(n)** di mana **f(n)** didefinisikan sebagai $\frac{threshold^{0.5+e}}{2}\ n^{0.5-e}$, sehingga f(*threshold*) = *threshold*/2.

```
1.   /*cluster reduction function
2.   * n -> number of input clusters
3.   * return -> number of max output cluster*/
4.
5.   int BVHBuilder::f(int n) {
6.    return (n <= 2) ? 1 : powf(threshold, .5f + e) / 2.f * powf(n, .5f - e); }
7.
8.   /*list partition function,
9.   * pivot is the frst bit 'flip'
```

```
10.  * ex : [0] 00000111
11.  *       [1] 00001000
12.  *       [2] 00001000
13.  *       [3] 00100000
14.  *     pivot -> 3 (flipped on third element 000xxxxx to 001xxxxx)
15.  */
16.  int BVHBuilder::getPivot(vector<shared_ptr<Triangle>>& geo){
17.   for (int i = 0; i < 30; i++)   {
18.    for (int j = 1; j < geo.size(); j++)   {
19.     //we shift for better performance & those bit no longer needed
20.     auto last = geo[j - 1]->getMortonBits() <<= 1;
21.     auto curr = geo[j]->getMortonBits() <<= 1;
22.     if ((curr[0] != last[0]))
23.      return j;
24.    }
25.   }
26.   return geo.size() / 2;
27.  }
```

Kode Sumber 4.11 Kode sumber metode lain pada tahap pembentukan BVH

4.2.3 Implementasi Recursive Ray Tracing dan *Shading*

Proses Recursive Ray Tracing dibagi menjadi dua bagian utama. Bagian pertama (Ray Tracing) akan menentukan objek terdekat yang bersilangan dengan *ray*. Bagian kedua akan menghitung warna dan menembakkan tiga *ray* baru jika dibutuhkan.

4.2.3.1 Ray Tracing

Proses Ray Tracing adalah proses membandingkan *ray* dengan tiap segitiga yang relevan dan mencari satu objek terdekat yang bersilangan dengan *ray*. Karena penulis menerapkan BVH sebagai metode optimalisasi dan Ray Tracing yang akan diimplementasikan memiliki sifat *tail recursion*. Untuk

menghindari penggunaan *memory overhead* berlebihan, penulis menerapkan metode iterasi dengan *object queue* sebagai alternatif *tail recursion* (Kode Sumber 4.12).

```
1.  void RayManager::traceRay(Ray& ray, shared_ptr<Container> bvh)  {
2.    vector<shared_ptr<Container>> bins;
3.    bins.push_back(bvh);
4.    while (bins.size() > 0)   {
5.      shared_ptr<Container> currBin = bins.back();
6.      bins.pop_back();
7.      if (currBin->isIntersecting(ray)) {
8.        if (currBin->geo != nullptr) {
9.          if (currBin->geo->isIntersecting(ray))
10.             ray.intersectWith = currBin->geo;
11.       }
12.       else {
13.         bins.push_back(currBin->rChild);
14.         bins.push_back(currBin->lChild);
15.       }
16.     }
17.   }
18. }
```

Kode Sumber 4.12 Kode sumber Ray Tracing dengan BVH menggunakan iterasi

4.2.3.2 Perhitungan warna

Proses perhitungan warna dimulai dengan menghitung warna pada titik persilangan dengan memperhitungkan material objek. Perhitungan warna pada sebuah titik memperhitungkan relevansi dan atenuasi titik terhadap sumber cahaya (Kode Sumber 4.14). Untuk mendapatkan sumber cahaya yang relevan, dilakukan Ray Tracing dari titik tersebut ke arah sumber cahaya. Jika ada objek yang menghalangi maka dapat disimpulkan sumber cahaya tersebut tidak relevan.

Tahap selanjutnya pada perhitungan cahaya adalah menghitung *reflection* dan *refraction* jika diperlukan. Perhitungan *ray reflection* dan *refraction* dapat dilihat pada Kode Sumber 4.15 dan Kode Sumber 4.16. untuk menghitung warna pada sebuah titik digunakan Blinn-Phong Shading.

```
1.  MyColor RayManager::getColor(Ray &ray, Scene &scene, int bounce) {
2.   if (bounce <= 0 || ray.intersectWith == nullptr)
3.   {
4.    return MyColor();
5.    //return scene.defColor;
6.   }
7.   else
8.   {
9.    vector<shared_ptr< Light>> effectiveLights = populateLights(ray, scene.lig
    hts, scene.bin);
10.   MyColor color =
11.    ray.intersectWith->mat.refrVal *((MyColor(1, 1, 1) - ray.intersectWith->
                        mat.reflVal)* calcColor(ray, effectiveLights, scene.getAtt()) +
12.    ray.intersectWith->mat.reflVal * getRefl(ray, scene, bounce - 1)) +
13.    (1 - ray.intersectWith->mat.refrVal) * getRefr(ray, scene, bounce - 1);
14.   return color;
15.  }
16. }
```

Kode Sumber 4.13 kode sumber perhitungan warna

```
1.  MyColor RayManager::calcColor(const Ray & ray, vector<shared_ptr<Light>
    > effectiveLights, Attenuation & attenuation)
2.  {
3.   MyColor result =
       ray.intersectWith->mat.ambient +
       ray.intersectWith->mat.emmission;
4.   Vec3 normal = ray.intersectWith->getNormal(ray.getHitMin());
5.   float cosI = ray.intersectWith->getNormal(ray.getHit()) * ray.direction;
6.   if (cosI > 0)
7.   {
8.    normal *= -1.f;
9.   }
10.
```

```
11. for (int i = 0; i < effectiveLights.size(); i++)
12. {
13.  shared_ptr< Light> light = effectiveLights[i];
14.  Vec3 pointToLight = light->getPointToLight(ray.getHitMin());
15.  Vec3 halfAngleToLight = ((ray.direction * -1.0f) + pointToLight);
16.  halfAngleToLight = halfAngleToLight.normalize();
17.
18.  Material material = ray.intersectWith->mat;
19.
20.  float attenuationValue = light->getAttValue(ray.getHitMin(), attenuation);
21.  pointToLight = pointToLight.normalize();
22.  result +=
23.    attenuationValue * (*(light->color)) *
24.    ((material.diffuse * (pointToLight * normal)) +
25.     (material.specular * powf(halfAngleToLight * normal,
                                  material.shininess)));
26. }
27. return result;
28. }
```

Kode Sumber 4.14 Kode sumber Blinn-phong Shading

```
1.  MyColor RayManager::getRefl(const Ray & ray, Scene & scene, int bounce)
    {
2.  float cosI = ray.intersectWith->getNormal(ray.getHit()) * ray.direction;
3.  Vec3 normal = ray.intersectWith->getNormal(ray.getHit());
4.  if (cosI < 0)   {  normal *= -1.f;   }
5.  Vec3 newDir = ray.direction + (normal * 2.0f * -(normal * ray.direction));
6.  Ray reflectRay(ray.getHitMin(), newDir);
7.  reflectRay.type = Ray::REFLECTION;
8.  traceRay(reflectRay, scene.bin);
9.  return getColor(reflectRay, scene, bounce);
10. }
```

Kode Sumber 4.15 Kode sumber pembentukan *reflection ray*

```
1.  MyColor RayManager::getRefr(const Ray & ray, Scene& scene, int bounce) {
2.   if (ray.intersectWith->mat.refrVal != 1.0)
3.   {
4.    float cosI = ray.intersectWith->getNormal(ray.getHit()) * ray.direction;
5.    Vec3 normal;
6.    float n1, n2, n;
7.    normal = ray.intersectWith->getNormal(ray.getHit());
8.    if (cosI > 0)
9.    {
10.    n1 = ray.intersectWith->mat.refractIndex;
11.    n2 = 1;
12.    normal *= -1.0f;
13.   }
14.   else
15.   {
16.    n1 = 1;
17.    n2 = ray.intersectWith->mat.refractIndex;
18.    cosI = -cosI;
19.   }
20.   n = n1 / n2;
21.   float sinT2 = n * n * (1.0f - cosI * cosI);
22.   float cosT2 = (1.0f - sinT2);
23.   float cosT = (float)sqrtf(1.0f - sinT2);
24.
25.   float rn = (n1 * cosI - n2 * cosT) / (n1 * cosI + n2 * cosT);
26.   float rt = (n2 * cosI - n1 * cosT) / (n2 * cosI + n2 * cosT);
27.   rn *= rn;
28.   rt *= rt;
29.   float refl = (rn + rt) * .5f;
30.   float trans = 1.0f - refl;
31.
32.   if (cosT2 < 0)
33.   {
34.    //if (bounce - 1 == 0)
35.    //    return scene.defColor;
36.    return ray.intersectWith->mat.reflVal *getRefl(ray, scene, bounce - 1);
37.   }
38.
39.   Vec3 newDir = ray.direction * n + normal * (n * cosI - cosT);
40.   Ray refractRay(ray.getHitPlus(), newDir);
41.   refractRay.type = Ray::REFRACTION;
```

```
42.  traceRay(refractRay, scene.bin);
43.  return (getColor(refractRay, scene, bounce));
44. }
45. else return MyColor();
46. }
```

Kode Sumber 4.16 Kode sumber pembentukan *refraction ray*

Dua algoritma di atas (Kode Sumber 4.15 dan Kode Sumber 4.16) membentuk *ray* baru (*reflection* dan *refraction*). Kedua metode di ataslah yang membedakan Recursive Ray Tracing dengan metode Path Tracing. Pada metode Recursive Ray Tracing, setiap warna pada titik persilangan juga memperhitungkan pantulan dan transparansi material sehingga tampak lebih realistis.

4.2.4 Implementasi Penyimpanan Citra

Assignment warna pada data Bitmap dan penyimpanan citra di-*handle* oleh *library* FreeImage. Tiap *pixel* didefinisikan dengan 3 *channel* warna (rgb-24 bit). Masing-masing *channel* memiliki nilai antara 0 sampai 255 (8 bit / *channel*). Proses *assignment* warna dan penyimpanan *file* dapat dilihat pada Kode Sumber 4.17.

```
1.  ...
2.  FreeImage_SetPixelColor(image, currCol, currRow, &color);
3.  ...
4.  FreeImage_FlipVertical(image);
5.  FreeImage_Save(FIF_BMP, image, (outFileName).c_str(), 0);
6.  ...
```

Kode Sumber 4.17 Kode sumber mewarnai *pixel* dan menyimpan citra

4.3 Optimalisasi

Pada tahap implementasi, penulis melakukan beberapa optimalisasi pada algoritma. Pada Kode Sumber 4.10 terlihat bahwa nilai perhitungan jarak tidak dihitung tiap iterasi. Nilai jarak disimpan pada tiap objek dan di akhir iterasi nilai jarak yang tidak relevan akan dihitung ulang. Optimalisasi ini sangat membantu efisiensi algoritma Agglomerative Clustering.

Optimalisasi lain yang penulis terapkan adalah paralelisme. Beberapa metode yang penulis terapkan paralelisme di dalamnya adalah; Radix Sort, pembentukan BVH, dan Ray Tracing pada *scene*. Untuk menghilangkan risiko *deadlock* dan *memory overhead*, penulis menerapkan Thread Pooling dalam penerapan paralelisme. Optimalisasi ini sangat terasa manfaatnya jika program dijalankan pada mesin dengan banyak *core*.

Pada algoritma rekursi seperti pembangunan *tree* dan Radix Sort, *programmer* perlu memperhatikan kondisi-kondisi yang dapat mengakibatkan *deadlock* dan *memory overhead* yang berlebihan. Misalkan pada kasus membangun *tree*, jika jumlah poligon lebih dari *threshold* maka *set* poligon akan dibagi menjadi dua *subset* yang dapat dijalankan secara paralel. Jika dua *subset* tersebut dijalankan pada dua *thread* baru, akan ada satu *thread* yang *idle* karena menunggu dua *thread* tersebut. Kondisi ini dapat mengakibatkan *memory overhead* berlebih. Salah satu cara mengatasinya adalah dengan memanfaatkan *thread* yang sudah ada untuk menjalankan pembentukan *sub tree* dari salah satu *subset* sehingga hanya mesin hanya perlu membuat satu *thread* baru (atau menggunakan *thread* yang tersedia di *thread pool*).

Contoh penggunaan Thread Pool dapat dilihat pada Kode Sumber 4.18. Pada kode sumber tersebut terlihat bahwa tidak semua sub-process akan dilaksanakan pada *thread* yang baru. Jika tidak ada *thread* yang *idle* maka proses dilaksanakan secara runtut (*sequential*). Jumlah *thread* yang tersedia diatur pada awal

eksekusi program berdasarkan *input* pengguna dengan syarat jumlah *thread* tidak lebih dari jumlah *logical processor* mesin dan tidak kurang dari satu.

```
7.  ...
8.  if (ThreadPool::tp.n_idle() > 0)
9.  {
10.   auto leftSort = ThreadPool::tp.push
                          (countSort, ref(arr), start, mid - 1, step - 1);
11.   countSort(id, arr, mid, end, step - 1);
12.   leftSort.get();
13. }
14. else
15. {
16.   countSort(id, arr, start, mid - 1, step - 1);
17.   countSort(id, arr, mid, end, step - 1);
18. }
19. ...
```

Kode Sumber 4.18 Kode sumber penggunaan Thread Pool pada algoritma Radix Sort

BAB V
UJI COBA DAN EVALUASI

Pada bab ini dijelaskan mengenai rangkaian uji coba perangkat lunak. Pada bab ini juga akan dibahas mengenai perhitungan parameter dalam algoritma AAC. Tujuan dari uji coba ini adalah untuk mengetahui apakah metode yang diterapkan dapat meningkatkan performa Ray Tracing. Selain itu juga pada bab ini akan dipaparkan pembahasan yang meliputi lingkungan uji coba, data uji coba, skenario uji coba, hasil uji coba, dan evaluasi.

5.1 Lingkungan Uji Coba

Lingkungan uji coba yang akan digunakan untuk melakukan uji coba meliputi perangkat keras dan perangkat lunak yang dijelaskan sebagai berikut:

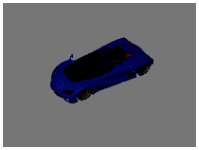
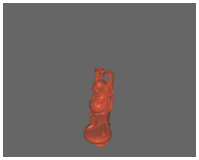
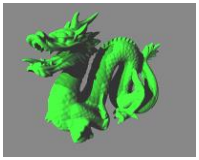
1) Perangkat Keras
   a. Prosesor: Intel(R) Core (TM) i7-2670QM CPU @ 2.20GHz
   b. Memori (RAM) : 4096 MB
2) Perangkat Lunak
   a. Sistem operasi : *Windows Embedded 8.1 Industry Pro* 64 bit

5.2 Data Uji Coba

Data masukkan uji coba adalah beberapa *file* obj dari *scene* yang sering digunakan pada tes *rendering*. *Scene* yang penulis gunakan digunakan pada uji coba kali ini ditunjukkan pada Tabel 5.1. Satu *scene* terdiri dari;

- 1 *file scene* yang berisi keterangan ukuran citra, posisi kamera dan sumber.
- *file* objek (.obj Wavefront) yang berisi *list vertex* dan *polygon*
- *file* material (.mtl Wavefront) yang berisi *list* material

Tabel 5.1 *Scene* Uji Coba

No	Nama *scene*	Jumlah *polygon*	Citra
1	teapot	15704	
2	sponza	108301	
3	car	225302	
4	conference	331179	
5	Buddha	543724	
6	dragon	871306	

5.3 Skenario Uji Coba

Berikut skenario uji coba yang dilakukan secara bertahap dengan tujuan dan metode seperti yang akan dijelaskan.

5.3.1 Skenario Uji Coba 1

- Tujuan uji coba

  Uji coba pertama akan menguji kebenaran sistem secara umum. Parameter yang akan diujikan adalah citra hasil eksekusi program.

- Metode uji coba

  Penulis akan mempersiapkan beberapa *scene* dengan tingkat kompleksitas berbeda. Pada uji coba kali ini, penulis menggunakan tiga *scene* sebagai *file* masukan yaitu *scene* “teapot”, “conference”, dan “Buddha” . Citra yang dihasilkan harus sesuai atau mirip dengan citra yang dihasilkan dengan kakas bantu lain. Hasil akan diinspeksi secara visual dengan fokus bentuk benda dan *shading* secara umum.

5.3.2 Skenario Uji Coba II

- Tujuan uji coba

  Uji coba kedua adalah membandingkan performa masing-masing jenis *container* untuk mendapatkan jenis *container* yang optimal. Variabel yang diujikan adalah bentuk *container* berupa *sphere* dan *box*.

- Metode uji coba

  Pada skenario kedua, penulis akan memilih *scene* dengan tingkat kompleksitas beragam dan menjalankan masing-masing *scene* sebanyak dua kali dengan parameter yang berbeda (*container* berupa *sphere* dan *box*).

Efisiensi *container* ditinjau dari rasio rata-rata pemanggilan fungsi persilangan (*ray-bin* dan *ray-triangle*) per *pixel* dan nilai rasio waktu masing-masing proses (membangun BVH dan *tracing scene*) dari dua jenis *container*. Penulis juga mengamati kualitas BVH berdasarkan rata-rata pemanggilan fungsi persilangan *ray-bin* dan *ray-triangle*.

5.3.3 Skenario Uji Coba III

- Tujuan uji coba

Uji coba kedua adalah menentukan parameter algoritma AAC yang optimal. Pada uji coba kali ini akan digunakan *scene* "teapot", "conference", dan "dragon". Variabel yang diujikan adalah *threshold* AAC dengan *range* [4,50].

- Metode uji coba

Parameter efisiensi *threshold* adalah waktu pembentukan BVH dan waktu *tracing scene*. Pada uji coba ini penulis berharap akan menemukan nilai *threshold* yang sesuai dengan kebutuhan (cepat dan kualitas baik).

5.3.4 Skenario Uji Coba IV

- Tujuan uji coba

Algoritma AAC dan Ray Tracing sangat mudah diparalelkan. Pada pengujian kali ini penulis akan menganalisis pengaruh jumlah *core* pada kecepatan algoritma yang diterapkan. Nilai *core* yang akan diujikan adalah 1, 2 , 4, 6, dan 8.

- Metode uji coba

Penulis mengubah jumlah *core* yang digunakan dengan mengubah jumlah *thread* pada Thread Pool yang

diterapkan pada program. Tiap *scene* akan dieksekusi dengan parameter jumlah *core* yang ingin diutilitaskan. Hasil yang akan dianalisis adalah waktu eksekusi dan dibandingkan dengan jika hanya menggunakan 1 *core.*

5.4 Hasil dan Analisa Uji Coba

Berikut adalah hasil uji coba berdasarkan skenario yang telah dijelaskan sebelumnya.

5.4.1 Hasil dan Analisa Uji Coba 1

- Hasil

  Tabel 5.2 menunjukkan hasil uji coba skenario pertama. Hasil yang diharapkan ditunjukkan pada kolom "expected". Hasil yang diharapkan mengacu pada hasil *render* pada kakas bantu Blender. Pencahayaan dan kamera didefinisikan secara manual sehingga ada ketidakcocokan sudut pandang dan pencahayaan pada citra hasil.

- Analisa hasil

  Hasil uji coba I menunjukkan hasil yang sesuai dengan harapan penulis dengan bentuk objek yang sama dengan harapan penulis. Kendala pada tahap ini adalah meletakkan kamera dan sumber cahaya pada posisi yang sama pada kakas bantu yang digunakan dan menyamakan standar material pada dua program yang berbeda.

5.4.2 Hasil dan Analisa Uji Coba 1I

- Hasil

  Hasil Uji Coba kedua dapat dilihat pada Tabel 5.3. *sphere* dan *box* adalah jenis *container* di mana nilai yang dianalisis adalah waktu *building* BVH dan waktu *tracing* masing-masing jenis *container*. Tabel 5.4 menunjukkan

perbandingan waktu jika menggunakan *container sphere* dengan *box*.

- Analisa hasil

Pada Tabel 5.3 terlihat bahwa *container* jenis *box* menunjukkan lebih baik jika dibandingkan dengan jenis *sphere*. Perbandingan waktu pembangunan BVH konsisten pada semua *scene*.

Sphere sebagai bangun tiga dimensi membutuhkan volume lebih besar untuk mencakup bangun segitiga jika dibandingkan dengan *box*. Hal ini menyebabkan jumlah persilangan *ray-container* yang lebih tinggi pada BVH dengan *sphere container* sehingga mengurangi efisiensi BVH secara umum.

5.4.3 Hasil dan Analisa Uji Coba III

- Hasil

Hasil Uji Coba ketiga dapat dilihat pada Tabel 5.5. nilai yang dianalisis adalah waktu pembentukan BVH, waktu *tracing scene*, dan waktu keseluruhan proses untuk masing-masing *scene* dengan nilai *threshold* yang bervariasi. Gambar 5.1 menunjukkan sifat dari waktu (*build* dan *trace*) terhadap perubahan nilai *threshold*.

- Analisa hasil

Pada Tabel 5.5 terlihat bahwa nilai *threshold* berbanding lurus kualitas BVH (kolom *trace*) dan berbanding terbalik dengan kecepatan pembentukan BVH (kolom *build*). Nilai *threshold* yang terlalu tinggi mengurangi efisiensi metode, terlihat nilai 12 memberikan nilai terbaik dan nilai 6 memberikan waktu pembentukan

cepat (Gambar 5.1). Berdasarkan hasil di atas penulis menetapkan dua jenis AAC(fast-t=6. HQ-t=12).

Tabel 5.6 menunjukkan rerata pemanggilan fungsi perpotongan *ray-container* dan *ray-triangle*. Pada kasus Ray Tracing tanpa BVH nilai tersebut akan sama dengan jumlah poligon dalam *scene*. Dengan menerapkan BVH, pemanggilan fungsi *intersection* berkurang secara signifikan.

5.4.4 Hasil dan Analisa Uji Coba 1V

- Hasil

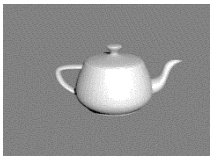
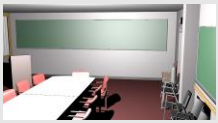
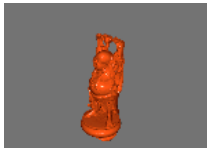
Hasil uji coba keempat ditunjukkan pada Tabel 5.7. Nilai yang akan dianalisis adalah perbandingan waktu membangun BVH pada *n-core* dengan 1-*core*.

- Analisa hasil

Hasil uji coba pada Tabel 5.7 menunjukkan peningkatan kecepatan yang signifikan pada perubahan jumlah *core*. Dikarenakan keterbatasan perangkat uji coba, penulis tidak dapat menentukan berapa besar peningkatan kecepatan pada jumlah *core* yang lebih besar. Percepatan berdasarkan jumlah *core* dapat dilihat pada Gambar 5.2. Berdasarkan hasil uji coba, utilitas delapan *core*[1] menghasilkan peningkatan kecepatan lebih dari tiga kali lipat jika dibandingkan dengan satu *core*. Peningkatan kecepatan ini konsisten pada seluruh *scene* dengan jumlah poligon yang bervariasi.

[1] Penulis menggunakan 8 *logical processors* pada 4 *core.* Hasil pada jumlah core sebenarnya akan lebih baik.

Tabel 5.2 Hasil Uji Coba I *Input-Output*

No	Scene	polygon count	Expected	Result
1	Teapot	15704		
2	Conference	331179		
3	Buddha	815588		

Tabel 5.3 Hasil Uji Coba II Jenis *Container* BVH

no	scene	polygon count	box			sphere		
			build *	trace **	avg. intersects test ***	build *	trace **	avg. intersects test ***
1	teapot	15704	92	626	19	126	3985	49
2	sponza	108301	815	3373	119	1083	32462	430
3	car	225302	1839	1526	42	2348	11418	114
4	conference	331179	2904	2915	114	3634	51966	706
5	Buddha	543724	4425	497	15	5774	2339	34
6	dragon	871306	7807	5042	101	9824	30999	228

** BVH build time (ms) ** scene tracing time (ms)*

**** Average ray-bin and ray-obj intersect function called per ray*

Tabel 5.4 Uji Coba II Rasio Kecepatan (*BOX* : *SPHERE)*

no	scene	Ratio (box : sphere)		
		build (ms)*	**trace (ms)****	**avg. intersects test *****
1	teapot	0.73	0.16	0.38
2	sponza	0.75	0.10	0.28
3	car	0.78	0.13	0.37
4	conference	0.80	0.06	0.16
5	Buddha	0.77	0.21	0.45
6	dragon	0.79	0.16	0.44
Average		0.77	0.14	0.35

** BVH build time (ms) ** scene tracing time (ms)*

**** Average ray-bin and ray-obj intersect function called per ray*

Tabel 5.5 Hasil Uji Coba III *Threshold* AAC

Thr *	Teapot (15704)			Conference (331179)			Dragon (871306)		
	build (ms) **	trace (ms) ***	**total (ms)**	build (ms) **	trace (ms) ***	**total (ms)**	build (ms) **	trace (ms) ***	**total (ms)**
4	48	802	**850**	2105	4736	**6841**	5531	8275	**13806**
6	58	684	**742**	2222	3527	**5749**	6272	7270	**13542**
8	69	670	**739**	2452	3188	**5640**	6590	6258	**12848**
10	81	655	**736**	2678	2930	**5608**	7298	5855	**13153**
12	93	654	**747**	2847	2915	**5762**	7847	4996	**12843**
14	106	623	**729**	3051	2800	**5851**	8427	4926	**13353**

Thr *	build (ms) **	trace (ms) ***	**total (ms)**	build (ms) **	trace (ms) ***	**total (ms)**	build (ms) **	trace (ms) ***	**total (ms)**
16	114	637	**751**	3295	2696	**5991**	8876	4790	**13666**
18	131	643	**774**	3608	2680	**6288**	9488	4618	**14106**
20	139	639	**778**	3781	2600	**6381**	10166	4797	**14963**
25	157	604	**761**	4127	2538	**6665**	11519	4533	**16052**
30	193	601	**794**	4745	2502	**7247**	12963	4221	**17184**
35	212	569	**781**	5324	2478	**7802**	14284	4029	**18313**
40	239	625	**864**	6093	2486	**8579**	16105	4359	**20464**
45	273	596	**869**	6393	2446	**8839**	17681	4705	**22386**
50	304	611	**915**	7038	2431	**9469**	25139	12661	**37800**

threshold ** BVH build time (ms) * scene tracing time (ms)*

Tabel 5.6 Hasil Uji Coba III *Threshold* AAC. Perhitungan perpotongan *ray-object*

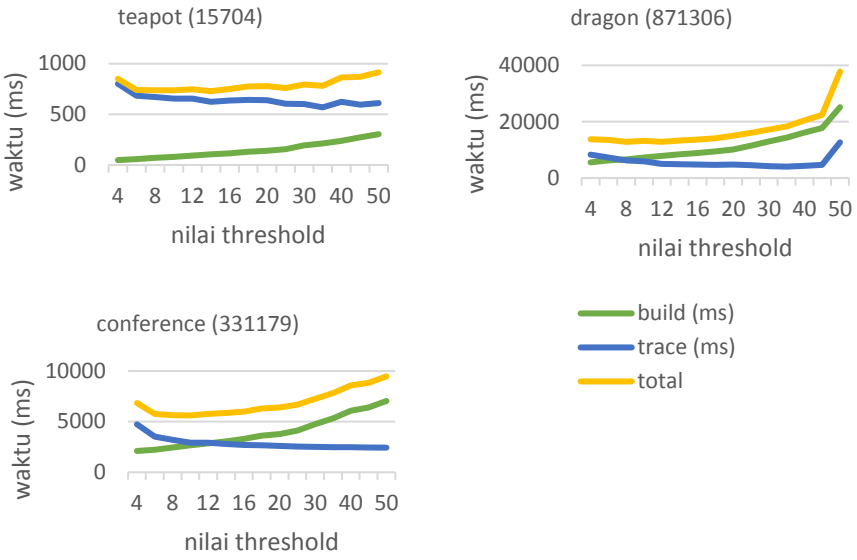
Thr*	**Teapot (15704)**	**Conference (331179)**	**Dragon (871306)**
4	23	227	176
6	21	153	136
8	20	129	117
10	19	115	108
12	19	114	101
14	18	110	95
16	18	105	93
18	18	102	90
20	18	99	88
25	18	93	85

Thr*	Teapot (15704)	Conference (331179)	Dragon (871306)
30	17	88	82
35	17	88	80
40	17	87	78
45	17	86	77
50	17	85	76

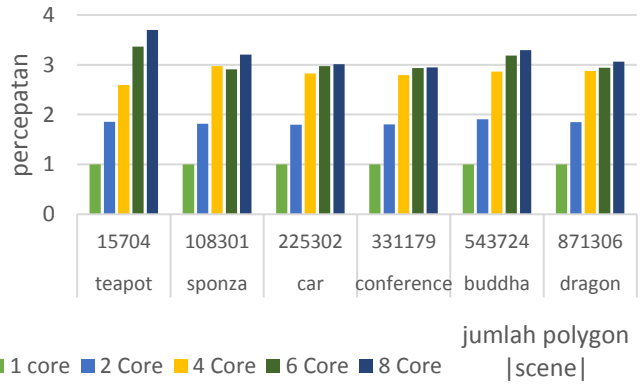
**threshold*

Tabel 5.7 Hasil Uji Coba Jumlah *Core* Pada Waktu Pembentukan BVH

no	Scene (polygon)	Build time (ms)				
		1 core	**2 Core**	**4 Core**	**6 Core**	**8 Core**
1	teapot (15704)	451 (1.00 x)	243 (**1.86** x)	174 (**2.59** x)	134 (**3.37** x)	122 (**3.70** x)
2	sponza (108301)	3373 (1.00 x)	1859 (**1.81** x)	1135 (**2.97** x)	1159 (**2.91** x)	1053 (**3.20** x)
3	car (225302)	7215 (1.00 x)	4021 (**1.79** x)	2552 (**2.83** x)	2425 (**2.98** x)	2394 (**3.01** x)
4	conference (331179)	11273 (1.00 x)	6253 (**1.80** x)	4038 (**2.79** x)	3843 (**2.93** x)	3827 (**2.95** x)
5	Buddha (543724)	19384 (1.00 x)	10179 (**1.90** x)	6777 (**2.86** x)	6092 (**3.18** x)	5882 (**3.30** x)
6	dragon (871306)	31445 (1.00 x)	17005 (**1.85** x)	10927 (**2.88** x)	10692 (**2.94** x)	10268 (**3.06** x)
	average	1.00	**1.84**	**2.82**	**3.05**	**3.20**



Gambar 5.1 Pengaruh *threshold* pada kecepatan konstruksi dan *rendering* pada tiga *scene* yang berbeda



Gambar 5.2 Percepatan konstruksi *container* berdasarkan jumlah *core*

BAB VI
KESIMPULAN DAN SARAN

Pada bab ini akan diberikan kesimpulan yang diambil selama pengerjaan tugas akhir serta saran-saran tentang pengembangan yang dapat dilakukan terhadap tugas akhir ini di masa yang akan datang.

6.1 Kesimpulan

Ray Tracing sebagai metode *rendering* realistis sangat terbantu oleh struktur data BVH. Pembentukan BVH dengan AAC adalah metode yang *feasible* dan seluruh proses tersebut sangat efektif jika diterapkan secara paralel. Dari hasil uji coba juga dapat disimpulkan bahwa jenis *container box* lebih baik dari pada *sphere* dan *threshold* AAC optimal adalah 6 (*fast*) dan 12 (*high quality*).

6.2 Saran

Dengan makin berkembangnya teknologi GPU penulis sangat menyarankan mengembangkan metode ini agar dapat diterapkan dengan memanfaatkan kekuatan paralelisme dari GPU.

(Halaman ini sengaja dikosongkan)

DAFTAR PUSTAKA

[1] T. Whitted, "An improved illumination model for shaded display," *ACM SIGGRAPH Computer Graphics,* vol. 13, no. 2, p. 14, 1979.

[2] J. Goldsmith dan J. Salmon, "Automatic Creation of Object Hierarchies for Ray Tracing," *Computer Graphics and Applications,* vol. 7, no. 5, pp. 14-20, 1987.

[3] G. Yan, H. Yong, K. Fatahalian and G. Blelloch, "Efficient BVH construction via approximate agglomerative clustering," *Proceedings of the 5th High-Performance Graphics Conference,* pp. 81-88, 2013.

[4] A. Appel, "Some techniques for shading machine renderings of solids," 1968.

[5] D. Vrajitoru, "Ray Tracing," [Online]. Available: http://www.cs.iusb.edu/~danav/teach/c481/c481_15_raytrace.html. [Diakses 3 10 2016].

[6] G. Tran, "Glasses, pitcher, ashtray and dice (POV-Ray)," 2006. [Online]. Available: http://www.oyonale.com/modeles.php?lang=en&page=40.

[7] I. Gargantini, "An effective way to represent quadtrees," *Communication of the ACM,* vol. 25, pp. 905-910, 1982.

[8] B. T. Phong, "Illumination for Computer Generated Pictures," *Communications of the ACM,* vol. 18, no. 6, pp. 311-317, 1975.

(Halaman ini sengaja dikosongkan)

BIODATA PENULIS

Arif Fathur Mahmuda, penulis dari buku tugas akhir ini lahir di kabupaten Kutai Timur tanggal 27 Juni 1995. Penulis telah menempuh pendidikan di SDN 002 Sangatta Selatan (2001-2007), SMPN 1 Sangatta Utara (2007-2009), SMAN 1 Sangatta Utara (2009-2012) dan Teknik Informatika ITS Surabaya (2012-2017). Selama masa perkuliahan, penulis pernah menjadi asisten pada mata kuliah Grafika Komputer dan Topik Khusus Interaksi, Grafika, dan Seni. Penulis juga aktif sebagai anggota organisasi Pencinta Alam Mahasiswa Informatika (PAMOR). Penulis memilih bidang minat Interaksi, Grafika, dan Seni (IGS) dan tertarik pada topik *Game Development* serta Grafika Komputer. Penulis dapat dihubungi melalui email AFMahmuda@gmail.com.