Accelerated Ray Tracing dengan *Bounding Volume Hierarchy* dan *Compute Unified Device Architecture*

Rina Savista Halim[1], Liliana, M.Eng[2], Prof. Rolly Intan, Dr. Eng.[3]
Program Studi Teknik Informatika,
Fakultas Teknologi Industri, UK Petra
Jln. Siwalankerto 121–131 Surabaya 60236
Telp. (031)-2983455, Fax. (031)-8417658
rinsavs@gmail.com[1], lilian@petra.ac.id[2], rintan@petra.ac.id[3]

ABSTRAK

Ray tracing adalah teknik dalam grafika komputer yang menampilkan gambar atau citra dengan memancarkan sinar dari mata ke setiap *pixel* pada *screen*, dan dilakukan pengecekan apakah sinar tersebut mengenai objek. *Ray tracing* dapat menghasilkan gambar yang fotorealistik, namun memerlukan waktu perhitungan yang lama. Cara yang sering digunakan untuk menyingkat waktu *ray tracing* adalah dengan mengurangi jumlah pengecekan *intersection* antara *ray* dengan objek. Salah satu metode yang populer adalah *Bounding Volume Hierarchy* (BVH).

Pada aplikasi yang dibuat juga digunakan *Compute Unified Device Architecture* (CUDA) disamping penggunaan BVH. Penggunaan CUDA bertujuan untuk mempercepat proses dengan cara melakukan perhitungan secara *parallel* pada GPU. Perhitungan-perhitungan yang dilakukan pada GPU adalah penentuan arah *ray* ke setiap *pixel* dan warna pada setiap *pixel*.

Hasil dari aplikasi ini adalah pengurangan waktu *render* sebesar lebih dari 70% jika dibandingkan dengan aplikasi *ray tracing* dengan BVH. Pengurangan waktu maksimal adalah 86%.

Kata kunci: *Ray tracing*, BVH, CUDA, GPU, GPU *ray tracing*.

ABSTRACT

Ray tracing is a technique in computer graphics that produced image by traversing ray from eye to every pixels on screen, and then the intersection checks between ray and objects are done. Ray tracing can produce a photorealistic image, but needs a long time to do the computations. One of the ways to shorten the computation time is by reducing the number of intersection checks between ray and objects. One of the popular methods is Bounding Volume Hierarchy (BVH).

In this application, Compute Unified Device Architecture (CUDA) is also used beside BVH. The CUDA's usage purpose is to reduce computation time by doing parallel computing in the GPU. Computations done in GPU are computing ray directions from eye to every pixels and color of every pixels.

The result of this application is more than 70% rendering time reduced when compared to ray tracing application with only BVH. The maximm reduction is 86%.

Keywords: *Ray tracing*, BVH, CUDA, GPU, GPU *ray tracing*.

1. PENDAHULUAN

Ray tracing adalah teknik dalam grafika komputer yang menampilkan gambar atau citra dengan memancarkan sinar. Sinar dipancarkan dari mata ke setiap pixel pada screen lalu dicek apakah sinar tersebut mengenai objek primitif. Jika sinar tersebut mengenai objek, maka dihitung efek pencahayaan (*ambient*, *diffuse*, *specular*, *shadow*) dan efek optik pada *pixel* tersebut. Perhitungan ini menghasilkan warna pada *pixel* tersebut[8]. Perhitungan-perhitungan tersebut menghasilkan gambar yang fotorealistik, namun memerlukan perhitungan yang lama. Sehingga memperpendek *processing time* menjadi faktor penting dalam pengembangan *ray tracing*.

Salah satu cara untuk mempercepat *ray tracing* adalah dengan mengurangi banyak pengecekan *intersection* antara *ray* dengan objek. Salah satu cara yang digunakan adalah *Bounding Volume Hierarchy* (BVH). BVH membuat *bounding sphere* atau *box* yang berisi beberapa objek, sehingga *ray* akan terlebih dahulu mengecek *bounding sphere* atau *box*. Jika *ray* mengenai *bounding sphere* atau *box* tersebut, maka akan dilakukan pengecekan terhadap objek-objek di dalamnya. Jika tidak terjadi perpotongan, maka tidak akan dilakukan pengecekan terhadap objek-objek di dalam *bounding sphere* atau *box*[7]. BVH dipilih karena memiliki *memory footprint* yang dapat diprediksi, dapat diperbarui secara dinamis, dan memiliki perfornmance yang baik pada implementasi *Graphics Processing Units* (GPU) *ray tracing*[2].

Sementara *Compute Unified Device Architecture* (CUDA) adalah teknologi dari NVIDIA yang merupakan ekstensi untuk beberapa bahasa pemrograman, septerti C dan C++. CUDA adalah arsitektur *General-Purpose Computing on Graphics Processing Units* (GPGPU) yang resmi dan dikembangkan oleh NVIDIA. GPGPU dapat mengatasi workload yang lebih banyak dari CPU dan lebih efisien[6]. CUDA digunakan untuk melakukan perhitungan secara paralel sehingga waktu proses *ray tracing* menjadi lebih singkat.

2. LANDASAN TEORI

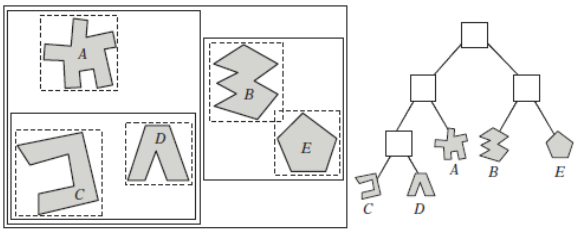
2.1 *Ray Tracing*

Ray tracing, khususnya *backward ray tracing* adalah sebuah metode *rendering* yang bekerja dengan cara memotongkan cahaya dari mata ke layar dengan objek-objek yang ada pada sebuah dunia. Metode ini akan menghitung warna pada setiap *pixel* pada layar 2 dimensi untuk menampilkan objek 3 dimensi. Dalam metode *ray tracing*, setiap *pixel* akan dikenai sinar yang akan diteruskan ke objek. Lalu, dilakukan perhitungan titik tabrak antara sinar dengan objek (jika ada). Jika sinar mengenai objek, maka warna *pixel* tersebut akan dihitung. Warna yang dihitung meliputi efek cahaya dan efek optik. Efek cahaya meliputi *ambient*, *diffuse*, dan *specular*. Sedangkan efek optik meliputi refleksi atau pemantulan yang terjadi saat cahaya mengenai permukaan yang reflektif dan efek refraksi yang terjadi saat cahaya mengenai permukaan yang bersifat transparan. Jika tidak, *pixel* tersebut diberi warna hitam[8]. *Ray tracing* menggunakan pendekatan *image-order* yang menentukan permukaan mana yang terlihat untuk setiap *pixel*[5].

2.2 Bounding Volume Hierarchy

2.2.1 *Definisi*

Dalam beberapa tahun terakhir, BVH telah menjadi struktur akselerasi yang populer karena memiliki performa yang baik, waktu pembuatan yang cepat, dan dapat diperbarui secara efisien[1]. Ide dasar dari BVH adalah menemukan volume yang dapat meliputi semua objek. Membungkus objek-objek dalam sebuah bounding volume dan melakukan tes *intersection* ke *bounding volume* terlebih dahulu sebelum objek dapat meningkatkan performa secara signifikan. Seluruh *bounding volume* yang digunakan akan disusun dalam bentuk *tree* yang selanjutnya disebut sebagai Bounding Volume Hierarchy (BVH). Dengan membuat BVH, waktu yang diperlukan untuk melakukan tes *intersection* dengan objek-objek yang ada menjadi lebih singkat[3]. Ilustrasi BVH dapat dilihat pada Gambar 1.



Gambar 1. Ilustrasi BVH dengan Menggunakan *Axis-Aligned Bounding Box*[3]

Di dalam BVH, *intersection test* antara *ray* dengan objek tidak akan terjadi jika *ray* tidak menabrak *bounding volume parent* dari objek tersebut. Pada sistem akan digunakan 3 (tiga) bentuk *bounding volume* yaitu *box*, *sphere*, dan *ellipse* yang akan dibandingkan performanya.

2.2.2 *Bentuk-bentuk Bounding Volume*

2.2.2.1 *Box*

Bentuk *box* yang digunakan merupakan AABB (*Axis-Aligned Bounding Box*). Algoritma yang sering digunakan untuk mengetahui apakah terjadi *intersection* antara *ray* dengan *bounding volume* adalah metode "*slab*". Cara kerja metode ini adalah dengan melakukan pengecekan antara *ray* dengan *boundary*, yang merupakan plane pada AABB 3 dimensi[7].

```
inline bool aabb::hit(const ray& r, float tmin, float tmax) const {
    for (int a = 0; a < 3; a++) {
        float invD = 1.0f / r.direction()[a];
        float t0 = (min()[a] - r.origin()[a]) * invD;
        float t1 = (max()[a] - r.origin()[a]) * invD;
        if (invD < 0.0f)
            std::swap(t0, t1);
        tmin = t0 > tmin ? t0 : tmin;
        tmax = t1 < tmax ? t1 : tmax;
        if (tmax <= tmin)
            return false;
    }
    return true;
}
```

Gambar 2. Pseudocode *Ray-Box Intersection*

2.2.2.2 *Sphere*

Pada penggunaan bentuk *sphere*, semua objek yang ada akan seolah-olah dimasukkan ke dalam sebuah bola. Dalam pembuatannya perlu dicari terlebih dahulu radius dan titik pusat dari *sphere*. Titik pusat didapatkan dengan merata-rata titik maksimum dan minimum. Sedangkan radiusnya merupakan jarak dari titik pusat ke titik maksimum atau minimum.

Menurut Hill[4], *intersection test* antara *ray* dengan persamaan:

$$S + \boldsymbol{c}t \tag{1}$$

Dimana: S : titik awal *ray* (x_0, y_0, z_0)

$\boldsymbol{c}$: vektor arah *ray* (c_x, c_y, c_z)

t : waktu tabrak

dengan *sphere* dapat dilakukan dengan menghitung nilai t dalam $At^2 + Bt + C = 0$.

$$A = |\boldsymbol{c}|^2 \tag{2}$$

$$B = S \bullet \boldsymbol{c} \tag{3}$$

$$C = |S|^2 - 1 \tag{4}$$

Setelah nilai *A*, *B*, dan *C* didapatkan maka dicari nilai diskriminan $D = B^2\text{-}4AC$. Jika diskriminan negatif, maka tidak terjadi *intersection*. Jika diskriminan adalah 0(nol), maka *ray* menyinggung *sphere*. Jika diskriminan adalah positif, maka ada 2(dua) waktu tabrak yaitu t1 dan t2. Setelah itu, dilakukan perhitungan nilai t. Jika t positif, maka terjadi *intersection* dan sebaliknya. Adapun cara mencari nilai t adalah:

$$t_h = \frac{-B \pm \sqrt{B^2 - 4AC}}{2A} \tag{5}$$

2.2.3 *Compute Unified Device Architecture*

CUDA dikembangkan oleh perusahaan IT NVIDIA mulai 2007 dan merupakan arsitektur GPGPU resmi. CUDA juga *excellent* dalam pemrosesan grafik. CUDA lebih efisien dari CPU karena CUDA merupakan GPU. GPU sering melakukan proses bersama-sama dengan CPU dan sangat efisien dalam komputasi-komputasi tertentu[6].

Terdapat 3 (tiga) jenis fungsi pada CUDA, yaitu *host*, *global*, dan *device*. Fungsi *host* adalah fungsi yang dijalankan pada CPU, biasanya dilakukan alokasi memori pada GPU dan juga penyalinan data yang diperlukan. Fungsi *global*, atau yang sering disebut dengan kernel, merupakan fungsi yang dijalankan pada GPU dan dapat dipanggil dari *host*. Fungsi *global* harus bertipe *void*. Sedangkan fungsi *device* merupakan fungsi yang dijalankan oleh GPU dan tidak dapat dipanggil dari fungsi *host*.

CUDA dalam aplikasi ini digunakan untuk menghitung arah *ray* dari *eye* ke setiap pixel pada *screen*. CUDA juga digunakan untuk menghitung warna setiap *pixel* pada *screen*.

3. METODE

3.1 *Bounding Volume Ellipse*

Persamaan *ellipse* pada titik pusat (p_x, p_y, p_z) adalah:

$$\left(\frac{x - p_x}{Rx}\right)^2 + \left(\frac{y - p_y}{Ry}\right)^2 + \left(\frac{z - p_z}{Rz}\right)^2 = 1 \tag{6}$$

Dimana: x, y, z : koordinat titik pada *ellipse*

p_x, p_y, p_z : koordinat titik pusat *ellipse*

Rx, Ry, Rz : jari-jari *ellipse*

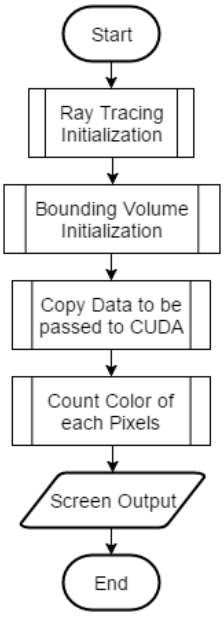
Lalu, *x*, *y*, dan *z* disubtitusi oleh *ray* $S + \boldsymbol{c}t$ menjadi:

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} x_0 + c_x t \\ y_0 + c_y t \\ z_0 + c_z t \end{pmatrix} \tag{7}$$

Persamaan hasil subtitusi dapat dikelompokkan menjadi $At^2 + Bt + C = 0$. Setelah *A*, *B*, dan *C* diketahui, diskriminan dan *t* dapat dicari. Rumus *t* ada pada persamaan (5).

3.2 Garis Besar Kerja Aplikasi

Dalam pembuatan aplikasi, diperlukan banyak proses. Proses-proses secara garis besar meliputi proses *Ray Tracing Initialization*, *Bounding Volume Initialization*, *Copy Data to be Passed to CUDA*, dan *Count Color of Each Pixels*. Hasil atau output dari proses-proses tersebut adalah gambar hasil *render*. Proses-proses besar yang ada dapat dilihat pada Gambar 3.



Gambar 3. Garis Besar Aplikasi

3.2.1 *Ray Tracing Initialization*

Pada proses ini dilakukan inisialisasi dilakukan konfigurasi ukuran *screen*, posisi mata dan lampu, serta *mesh* yang ingin dibaca. Jika *mesh* tidak ditemukan, maka aplikasi akan berhenti. Jika *mesh* ditemukan, *mesh* akan dibaca dan disimpan setiap *face*-nya. Bersamaan dengan itu, dilakukan pencarian titik berat maksimum dan minimum, yang diperlukan untuk pembuatan BVH *tree*. Setelah itu dilakukan pencarian arah *ray* ke *pixel* kiri bawah. Lalu dilanjutkan dengan proses penghitungan arah setiap *ray*. Selain itu, program akan langsung dihentikan jika *mesh* input tidak ada.

3.2.2 *Bounding Volume Initialization*

Proses ini bertujuan untuk membuat BVH *tree*. Pertama-tama *root* dari BVH *tree* akan diisi. Setelah *root* terisi, maka akan dilakukan pengisian pada *left* dan *right* dari *root*. Proses pengisian *left* dan *right* akan berhenti jika jumlah objek dalam *bounding volume* kurang dari atau sama dengan nilai yang sudah diset sebelumnya. Jika jumlah objek tidak memenuhi syarat, maka dilakukan pencarian *axis* yang paling panjang, lalu dilakukan pengisian *left* dan *right* dari *node* tersebut. Khusus pada *bounding volume sphere*, *axis* digilir secara berurutan, karena pada *sphere* tidak terdapat *axis* terpanjang.

3.2.3 *Copy Data to be Passed to CUDA*

Proses *Count Color of Each Pixel*s dilakukan pada GPU atau CUDA. Sehingga, perlu dilakukan penyalinan data yang diperlukan. Data yang disalin merupakan informasi dari BVH. Informasi dari BVH perlu disalin dengan sedemikian rupa sehingga nantinya GPU tidak perlu memroses secara rekursi, melainkan secara *loop*.

3.2.4 *Count Color of Each Pixels*

Proses ini bertujuan untuk menentukan warna setiap *pixel* dan dijalankan di dalam GPU. Pada proses ini, dicari objek mana yang tampak beserta perhitungan efek-efek warna. Efek-efek warna yang dihitung meliputi efek cahaya dan efek optik. *Ambient*, *diffuse*, *specular*, dan *shadow* merupakan efek cahaya. Sedangkan refleksi dan transparansi merupakan efek optik.

4. IMPLEMENTASI

Program yang digunakan diuji pada lingkungan sebagai berikut:

- Merk : Acer
- Sistem Operasi: Windows 7 Professional 64-bit
- Prosesor : Intel(R) Core(TM) i5-3340 @3.10Ghz (4 CPU)
- Memori : 16GB RAM
- VGA : Nvidia GeForce GTX 550 Ti
- CUDA *Library*: CUDA Toolkit 6.5
- *Software* : Microsoft Visual Studio 2010
- Resolusi : 200 *pixel* x 200 *pixel*

Pengujian yang dilakukan meliputi pengujian pengurangan waktu render dan pengujian *detail loss*.

4.1 Pengujian Waktu *Render*

Pada bagian ini, akan dibahas mengenai perbandingan waktu *rendering* dari berbagai *mesh* yang dijalankan pada berbagai aplikasi. Rumus pengurangan waktu adalah:

$$\text{Pengurangan waktu render (\%)} = \frac{\text{waktu 1} - \text{waktu 2}}{\text{waktu 1}} \text{ x } 100\% \quad (8)$$

Yang dibandingkan pertama kali adalah *basic ray tracing* dengan *ray tracing* menggunakan CUDA. Dalah perbandingan ini, waktu 1 merupakan waktu *render* dari *basic ray tracing* dan waktu 2 merupakan waktu *render* dari *ray tracing* menggunakan CUDA. Hasil perbandingan dapat dilihat pada Tabel 1.

Tabel 1. Tabel Perbandingan dan Pengurangan Waktu *Render* antara *Basic Ray Tracing* dengan *Ray Tracing* Menggunakan CUDA

No	Keterangan Mesh	Waktu Render(ms) dan Pengurangan Waktu(%)		
		Basic	CUDA	
1	triangle.obj 1 face	667	632	5.25%
2	triangle2.obj 2 faces	878	728	17.08%
3	triangle3.obj 3 faces	924	747	19.16%
4	triangle4.obj 4 faces	1087	818	24.75%

Tabel 2. Tabel Perbandingan dan Pengurangan Waktu *Render* antara *Basic Ray Tracing* dengan *Ray Tracing* Menggunakan CUDA (Lanjutan)

No	Keterangan Mesh	Waktu Render(ms) dan Pengurangan Waktu(%)		
		Basic	CUDA	
5	box1.obj 8 faces	1649	916	44.45%
6	diamond.obj 33 faces	7014	2655	62.15%
7	icosphere.obj 80 faces	14247	5203	63.48%
8	bunny500.obj 200 faces	42211	14115	66.56%
9	bunny1500.obj 200 faces	39592	13414	66.12%
10	jar.obj 204 faces	31073	10501	66.21%
11	cemara.obj 246 faces	38929	12933	66.78%
12	sword.obj 338 faces	55568	18600	66.53%
13	key_b.3ds 588 faces	98024	32114	67.24%
14	gourd.obj 648 faces	116987	40977	64.97%
15	sphere.obj 960 faces	159811	54165	66.11%
16	ellipse.obj 960 faces	168727	58074	65.58%
17	knob.obj 1344 faces	273141	80490	70.53%
18	dolphin.obj 1692 faces	285055	93924	67.05%
19	apel.obj 1704 faces	306869	84352	72.51%
20	camel.obj 4884 faces	775864	242459	68.75%
21	beethoven.ply 5030 faces	948919	339572	64.21%
22	teapot.obj 6320 faces	1254613	397596	68.31%

Tabel 3. Tabel Perbandingan dan Pengurangan Waktu *Render* antara *Basic Ray Tracing* dengan *Ray Tracing* Menggunakan CUDA (Lanjutan)

No	Keterangan Mesh	Waktu Render(ms) dan Pengurangan Waktu(%)		
		Basic	CUDA	
23	bunny.off 6966 faces	1203699	383018	68.18%
24	sample1.3DS 23578 faces	4089124	902385	77.93%
25	dragon.ply 37986 faces	7158953	901752	87.40%

Perbandingan kedua adalah perbandingan antara *basic ray tracing* dengan *ray tracing* menggunakan BVH. Dalam perbandingan ini, waktu 1 merupakan waktu *render* dari *basic ray tracing*, sedangkan waktu 2 merupakan waktu *render* dari *ray tracing* menggunakan BVH. Bentuk BVH yang digunakan adalah *box*, *sphere*, dan *ellipse*. Pada kolom BVH, terdapat waktu *render* dalam ms dan persentase pengurangan waktu (%) yang dihitung menggunakan Persamaan (8). *Mesh-mesh* yang digunakan sama dengan perbandingan pada Tabel 1. Hasil perbandingan dapat dilihat di Tabel 2.

Tabel 4. Tabel Perbandingan dan Pengurangan Waktu *Render* antara *Basic Ray Tracing* dengan *Ray Tracing* Menggunakan BVH

No	Keterangan Mesh	Waktu Render(ms) dan Pengurangan Waktu(%)			
		Basic	BVH		
1	triangle.obj 1 face	667	Box	607	9.00%
			Sphere	798	-19.64%
			Ellipse	852	-27.74%
2	triangle2.obj 2 faces	878	Box	733	16.51%
			Sphere	915	-4.21%
			Ellipse	961	-9.45%
3	triangle3.obj 3 faces	924	Box	631	31.71%
			Sphere	721	21.97%
			Ellipse	1266	-37.01%
4	triangle4.obj 4 faces	1087	Box	714	34.31%
			Sphere	798	26.59%
			Ellipse	1500	-37.99%
5	box1.obj 8 faces	1649	Box	1242	24.68%
			Sphere	1713	-3.88%
			Ellipse	1511	8.37%
6	diamond.obj 33 faces	7014	Box	3586	48.87%
			Sphere	6302	10.15%
			Ellipse	6119	12.76%

Tabel 2. Tabel Perbandingan dan Pengurangan Waktu *Render* antara *Basic Ray Tracing* dengan *Ray Tracing* Menggunakan BVH (Lanjutan)

No	Keterangan Mesh	Waktu Render(ms) dan Pengurangan Waktu(%)			
		Basic	BVH		
7	icosphere.obj	14247	Box	3803	73.31%
	80 faces		Sphere	5517	61.28%
			Ellipse	4913	65.52%
8	bunny500.obj	42211	Box	7038	83.33%
	200 faces		Sphere	21085	50.05%
			Ellipse	8609	79.60%
9	bunny1500.obj	39592	Box	4914	87.59%
	200 faces		Sphere	16257	58.94%
			Ellipse	6778	82.88%
10	jar.obj	31073	Box	13582	56.29%
	204 faces		Sphere	14040	54.82%
			Ellipse	12851	58.64%
11	cemara.obj	38929	Box	8906	77.12%
	246 faces		Sphere	6913	82.24%
			Ellipse	5414	86.09%
12	sword.obj	55568	Box	6718	87.91%
	338 faces		Sphere	2776	95.00%
			Ellipse	1226	97.79%
13	key_b.3ds	98024	Box	8329	91.50%
	588 faces		Sphere	8419	91.41%
			Ellipse	1875	98.09%
14	gourd.obj	116987	Box	17404	85.12%
	648 faces		Sphere	26882	77.02%
			Ellipse	9985	91.46%
15	sphere.obj	159811	Box	19988	87.49%
	960 faces		Sphere	8215	94.86%
			Ellipse	2770	98.27%
16	ellipse.obj	168727	Box	20967	87.57%
	960 faces		Sphere	14455	91.43%
			Ellipse	4748	97.19%
17	knob.obj	273141	Box	54315	80.11%
	1344 faces		Sphere	45134	83.48%
			Ellipse	23271	91.48%
18	dolphin.obj	285055	Box	35047	87.71%
	1692 faces		Sphere	38752	86.41%
			Ellipse	9585	96.64%

Tabel 2. Tabel Perbandingan dan Pengurangan Waktu *Render* antara *Basic Ray Tracing* dengan *Ray Tracing* Menggunakan BVH (Lanjutan)

No	Keterangan Mesh	Waktu Render(ms) dan Pengurangan Waktu(%)			
		Basic	BVH		
19	apel.obj	306869	Box	61153	80.07%
	1704 faces		Sphere	72912	76.24%
			Ellipse	32501	89.41%
20	camel.obj	775864	Box	140918	81.84%
	4884 faces		Sphere	109696	85.86%
			Ellipse	15625	97.99%
21	beethoven.ply	948919	Box	120614	87.29%
	5030 faces		Sphere	117617	87.61%
			Ellipse	54105	94.30%
22	teapot.obj	1254613	Box	240452	80.83%
	6320 faces		Sphere	371324	70.40%
			Ellipse	51374	95.91%
23	bunny.off	1203699	Box	296368	75.38%
	6966 faces		Sphere	223483	81.43%
			Ellipse	26837	97.77%
24	sample1.3DS	4089124	Box	502300	87.72%
	23578 faces		Sphere	1573880	61.51%
			Ellipse	185232	95.47%
25	dragon.ply	7158953	Box	1253351	82.49%
	37986 faces		Sphere	600198	91.62%
			Ellipse	341669	95.23%

Dari Tabel 1 dan Tabel 2, dapat dilihat bahwa BVH lebih efektif daripada CUDA. Untuk *mesh* kecil, CUDA lebih efektif. Namun pada *mesh* yang memiliki ukuran lebih besar, BVH jauh lebih efektif. Untuk *mesh* dengan jumlah *face* kurang dari 200, bentuk *box* lebih efektif. Tetapi, untuk *mesh* dengan jumlah *face* lebih dari 200, bentuk *ellipse* lebih efektif. Sedangkan pada bentuk *sphere*, akan lebih efektif dari *box* jika bentuk objek terlihat lebih cocok untuk di-*bounding* menggunakan *sphere*.

Perbandingan selanjutnya adalah perbandingan antara *ray tracing* menggunakan BVH dan *ray tracing* menggunakan BVH dan CUDA. Hasil perbandingan dapat dilihat pada Tabel 5.

Tabel 5. Tabel Perbandingan dan Pengurangan Waktu *Render* antara *Ray Tracing* dengan BVH dan *Ray Tracing* dengan BVH dan CUDA

No	Keterangan Mesh	Waktu Render(ms) dan Pengurangan Waktu(%)				
		BVH		BVH + CUDA		
1	triangle.obj	Box	607	Box	620	-2.1%
	1 face	Sphere	798	Sphere	807	-1.1%

Tabel 3. Tabel Perbandingan dan Pengurangan Waktu *Render* antara *Ray Tracing* dengan BVH dan *Ray Tracing* dengan BVH dan CUDA (Lanjutan)

No	Keterangan Mesh	Waktu Render(ms) dan Pengurangan Waktu(%)				
		BVH		BVH + CUDA		
		Ellipse	852	Ellipse	837	1.8%
2	triangle2.obj	Box	733	Box	688	6.1%
	2 faces	Sphere	915	Sphere	857	6.3%
		Ellipse	961	Ellipse	835	13.1%
3	triangle3.obj	Box	631	Box	625	1.0%
	3 faces	Sphere	721	Sphere	703	2.5%
		Ellipse	1266	Ellipse	837	33.9%
4	triangle4.obj	Box	714	Box	630	11.8%
	4 faces	Sphere	798	Sphere	735	7.9%
		Ellipse	1500	Ellipse	985	34.3%
5	box1.obj	Box	1242	Box	766	38.3%
	8 faces	Sphere	1713	Sphere	863	49.6%
		Ellipse	1511	Ellipse	877	42.0%
6	diamond.obj	Box	3586	Box	1142	68.2%
	33 faces	Sphere	6302	Sphere	1820	71.1%
		Ellipse	6119	Ellipse	1618	73.6%
7	bunny500.obj	Box	7038	Box	1678	76.2%
	200 faces	Sphere	21085	Sphere	5702	73.0%
		Ellipse	8609	Ellipse	1689	80.4%
8	bunny1500.obj	Box	4914	Box	2095	57.4%
	200 faces	Sphere	16257	Sphere	4677	71.2%
		Ellipse	6778	Ellipse	1440	78.8%
9	gourd.obj	Box	17404	Box	3639	79.1%
	648 faces	Sphere	26882	Sphere	8066	70.0%
		Ellipse	9985	Ellipse	2986	70.1%
10	knob.obj	Box	54315	Box	11592	78.7%
	1344 faces	Sphere	45134	Sphere	11504	74.5%
		Ellipse	23271	Ellipse	4793	79.4%
11	dolphin.obj	Box	35047	Box	7007	80.0%
	1692 faces	Sphere	38752	Sphere	13166	66.0%
		Ellipse	9585	Ellipse	2836	70.4%
12	apel.obj	Box	61153	Box	10493	82.8%
	1704 faces	Sphere	72912	Sphere	20220	72.3%
		Ellipse	32501	Ellipse	7118	78.1%
13	camel.obj	Box	140918	Box	33556	76.2%
	4884 faces	Sphere	109696	Sphere	34573	68.5%
		Ellipse	15625	Ellipse	5618	64.0%

Tabel 3. Tabel Perbandingan dan Pengurangan Waktu *Render* antara *Ray Tracing* dengan BVH dan *Ray Tracing* dengan BVH dan CUDA (Lanjutan)

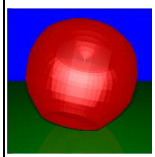
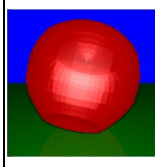
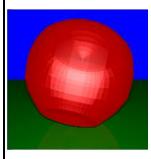
No	Keterangan Mesh	Waktu Render(ms) dan Pengurangan Waktu(%)				
		BVH		BVH + CUDA		
14	beethoven.ply	Box	120614	Box	25526	78.8%
	5030 faces	Sphere	117617	Sphere	34385	70.8%
		Ellipse	54105	Ellipse	15816	70.8%
15	teapot.obj	Box	240452	Box	52614	78.1%
	6320 faces	Sphere	371324	Sphere	115077	69.0%
		Ellipse	51374	Ellipse	18332	64.3%
16	bunny.off	Box	296368	Box	63693	78.5%
	6966 faces	Sphere	223483	Sphere	71831	67.9%
		Ellipse	26837	Ellipse	8503	68.3%
17	sample1.3DS	Box	502300	Box	105226	79.1%
	23578 faces	Sphere	1573880	Sphere	519273	67.0%
		Ellipse	185232	Ellipse	65486	64.6%
18	dragon.ply	Box	1253351	Box	260460	79.2%
	37986 faces	Sphere	600198	Sphere	166891	72.2%
		Ellipse	341669	Ellipse	97871	71.4%
19	ddboat.obj	Box	6796711	Box	901114	86.7%
	82666 faces	Sphere	6665842	Sphere	916287	86.3%
		Ellipse	6699465	Ellipse	906206	86.5%

Dari Tabel 5, dapat disimpulkan bahwa seiring bertambahnya jumlah *face* dalam *mesh*, pengurangan waktu pada bentuk *box* lebih stabil. Pada bentuk *sphere* dan *ellipse*, pengurangan waktu bergantung pada bentuk *mesh*. Pengurangan waktu maksimum yang dapat dicapai adalah 86%.

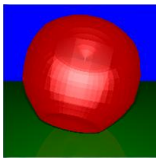
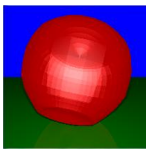
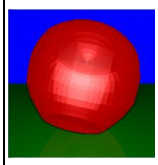
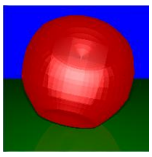
4.2 *Detail Loss*

Pada bagian ini akan dibahas mengenai ada atau tidaknya *detail loss* pada gambar atau citra yang dihasilkan oleh berbagai program *ray tracing*.

Tabel 6. Pengujian *Detail Loss* pada apel.obj

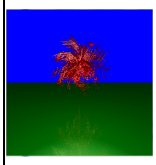
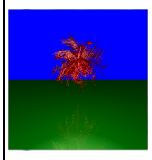
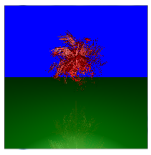
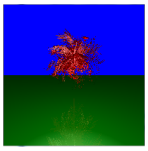
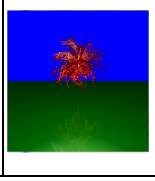
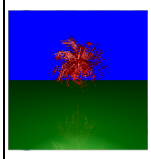
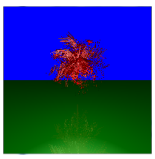
Nama *file*: apel.obj 1704 *faces* Koordinat mata: (0, 0, 250) Koordinat cahaya: (0, 0, 240)	*Ray Tracing*	*Ray Tracing* dengan BVH *Box*	*Ray Tracing* dengan BVH *Sphere*
			

Tabel 7. Pengujian *Detail Loss* pada apel.obj (Lanjutan)

Ray Tracing dengan BVH *Ellipse*	*Ray Tracing* dengan BVH *Box* dan CUDA	*Ray Tracing* dengan BVH *Sphere* dan CUDA	*Ray Tracing* dengan BVH *Ellipse* dan CUDA
			

Dari Tabel 6, dapat dilihat bahwa tidak terjadi *detail loss* pada apel.obj. Tujuh gambar tersebut terlihat sama persis. Adapun karakteristik dari apel.obj adalah segitiga-segitiga berkumpul pada 1(satu) buah *mesh* utuh.

Tabel 8. Pengujian *Detail Loss* pada sample1.3DS

Nama *file*: sample1.3DS 23578 *faces* Koordinat Mata: (0, 0, 20) Koordinat Cahaya: (0, 0, 18)	*Ray Tracing*	*Ray Tracing* dengan BVH *Box*	*Ray Tracing* dengan BVH *Sphere*
			
Ray Tracing dengan BVH *Ellipse*	*Ray Tracing* dengan BVH *Box* dan CUDA	*Ray Tracing* dengan BVH *Sphere* dan CUDA	*Ray Tracing* dengan BVH *Ellipse* dan CUDA
			

Karakteristik dari sample1.3DS adalah segitiga-segitiga yang ada membuat sebuah mesh yang bercabang-cabang. Pada sample1.3DS juga tidak terjadi detail loss.

5. KESIMPULAN

Berdasarkan hasil pengujian, dapat disimpulan berbagai hal sebagai berikut:

- Metode percepatan yang ada belum tentu lebih lambat daripada *parallel programming*
- Untuk jumlah *face* kurang dari 200(dua ratus) sebaiknya menggunakan BVH dengan bentuk *bounding volume box*
- Untuk jumlah *face* lebih dari 200(dua ratus) sebaiknya menggunakan BVH dengan bentuk *bounding volume ellipse*
- Bentuk *mesh* berpengaruh pada penggunaan bentuk *bounding volume*, terutama pada bentuk *box* dan *sphere*
- Pada aplikasi yang memanfaatkan BVH dan CUDA, CUDA mempercepat maksimal 86% waktu *render*
- Aplikasi *ray tracing* dengan BVH dan CUDA tidak didapatkan adanya *detail loss*. Gambar yang dihasilkan sama persis dengan gambar dari aplikasi *ray tracing* tanpa BVH dan CUDA.

6. TERIMA KASIH

Terima kasih kepada rekan-rekan mahasiswa Universitas Kristen Petra Surabaya yang telah mendukung dalam pembuatan aplikasi ini. Terima kasih juga kami tujukan kepada Program Studi Teknik Informatika yang telah mengizinkan penggunaan laboratorium guna pengembangan aplikasi pada penelitian karya tulis ini.

7. REFERENCES

[1] Afra, Attila T., Laszlo Szirmay-Kalos. 2014. Stackless Multi-BVH Traversal for CPU, MIC, and GPU Ray Tracing. Computer Graphics Forum, 33(1), 129-140.

[2] Bittnes, Jiri, Michal Hapala, Vlastimil Havran. 2015. Incremental BVH Construction for Ray Tracing. Computers & Graphics, 47(13), 135-144.

[3] Ericson, Christer. 2004. Real Time Collision Detection. United States of America: CRC Press.

[4] Hill, F. S., Jr, Stephen M. Kelley. 2007. Computer Graphics using OpenGL (3rd ed.). United Kingdom: Pearson Education International

[5] Mattausch, O. et al. 2015. CHC+RT: Coherent Hierarchical Culling for Ray Tracing. Computer Graphics Forum, 34(2), 537-548.

[6] Riley, Aaron. 2016. Getting Started with CUDA. United States of America: Kindle.

[7] Shirley, Peter. 2016. Ray Tracing: The Next Week. United States of America: Kindle.

[8] Suffern, Kevin. 2007. Ray Tracing from the Ground Up. United States of America: A K Peters, Ltd.