Konstruksi Bounding Volume Hierarchy dengan Metode Agglomerative Clustering untuk Meningkatkan Performa Ray Tracing

Arif Fathur Mahmuda, Anny Yuniarti, dan Wijayanti Nurul K.
Jurusan Teknik Informatika, Fakultas Teknologi Informasi, Institut Teknologi Sepuluh Nopember (ITS)
Jl. Arief Rahman Hakim, Surabaya 60111 Indonesia
e-mail: afmahmuda@gmail.com, ayuniarti@gmail.com, wijay.cs@gmail.com

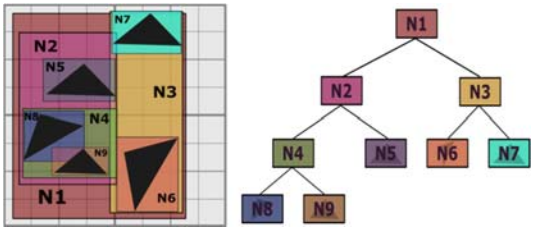
***Abstrak*—Ray Tracing sebagai algoritma *rendering* yang menghasilkan citra realistis memiliki beberapa kekurangan. Salah satu di antaranya adalah perhitungan persilangan *ray-object* pada tiap *pixel* yang memakan 75% waktu dari keseluruhan proses *rendering*. Penelitian ini menerapkan metode yang diharapkan dapat mempersingkat proses perhitungan persilangan *ray-object* dengan membangun struktur data berupa *binary tree*. *Tree* yang dibangun sering juga disebut sebagai Bounding Volume Hierarchy (BVH) di mana masing-masing *node*-nya adalah sebuah *container*. Struktur data tersebut akan dibangun dengan metode Approximate Agglomerative Clustering (AAC) yang merupakan metode *bottom-up clustering* dengan *top-down preprocessing*. 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 menunjukkan peningkatan kecepatan hingga 3 kali lipat dibandingkan tanpa menerapkan paralelisme. Pada hasil uji coba, juga didapatkan dua jenis parameter yang masing-masing memiliki karakteristik tersendiri (6= cepat, 12= kualitas baik).**

***Kata Kunci*—*Ray Tracing, Rendering, Agglomerative Clustering, Grafika, BVH*.**

I. PENDAHULUAN

DIANTARA banyak metode *advanced rendering*, Ray Tracing merupakan salah satu metode yang intuitif dan mudah diterapkan. Salah satu kelemahan Ray Tracing terletak pada proses *tracing* di mana setiap *ray* yang ditembakkan harus dicek persilangannya dengan seluruh objek. Proses tersebut memakan 75% waktu keseluruhan proses [1]. Salah satu solusi masalah tersebut adalah dengan membangun Bounding Volume Hierarchy (BVH). BVH adalah sebuah struktur data berupa *tree* yang setiap *node*-nya adalah sebuah *container* yang melingkupi beberapa *container* lain atau sebuah poligon (Gambar 1) [2].

Pembangunan BVH secara otomatis dapat dilakukan dengan melakukan *clustering* terhadap sejumlah *container* yang berisi sebuah poligon [3] [4]. Penggunaan Naive Agglomerative Clustering pada pembentukan BVH tidak *feasible* karena Agglomerative Clustering dapat memberikan kompleksitas $O(N^2)$ atau lebih buruk. Dengan optimalisasi yang tepat dan dengan memanfaatkan paralelisme, kita dapat membangun BVH menggunakan Agglomerative Clustring.



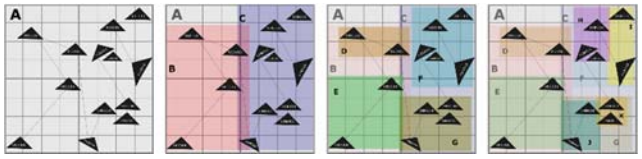
Gambar 1. Ilustrasi BVH pada bidang 2 dimensi (a) dan representasi *tree*-nya (b)

II. RAY TRACING DENGAN BVH

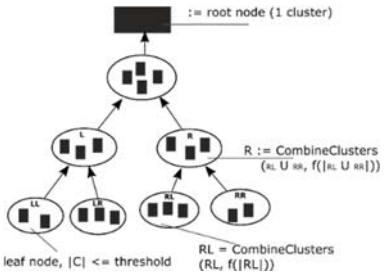
Pada algoritma Ray Tracing, kita memerlukan perhitungan persilangan *ray*-object sebanyak jumlah objek per *pixel*. BVH tersusun dari *container* dalam bentuk 3 dimensi yang membentuk sebuah *tree*. Dengan memanfaatkan BVH, objek yang *parent*-nya tidak bersilangan dengan objek tidak perlu diperhitungkan persilangannya (ditunjukkan pada Gambar 2). Pada penelitian kali ini diimplementasikan BVH dalam bentuk *binary tree*. Gambar 1 menunjukkan ilustrasi BVH pada bidang 2 dimensi dan dalam bentuk *tree*.

```
1.  traceRay(ray,bvh)
2.  In : a ray and a container
3.  Out : collided polygon and collision point
4.
5.    IF ray.intersect(bvh) THEN
6.       IF bvh is leaf THEN
7.          IF ray is intersecting polygon THEN
8.           CALCULATE collisionPoint
9.            IF collisionPoint is closer THEN
10.             SET collision object to polygon
11.           END IF
12.         END IF
13.      ELSE
14.         traceRay (ray, left  node)
15.         traceRay (ray, right node)
16.      END IF
17.   END IF
```

Gambar 2. *Pseudocode* Ray Tracing dengan BVH



Gambar 3. Tahap downward AAC. Pada tiap iterasi setiap *sublist* dibagi menjadi dua buah list baru hingga memenuhi *threshold*



Gambar 4. Tahap upward AAC. tiap kotak mewakili sebuah *binary tree*. Tiap *node cluster* mengandung n *tree*.

III. APPROXIMATE AGGLOMERATIVE CLUSTERING

A. *Agglomerative Clustring*

Agglomerative Clustering digunakan dalam berbagai bidang mulai dari *data mining*, kompresi data, dan lain-lain. Hal ini disebabkan kualitas *cluster* yang dihasilkan lebih baik jika dibandingkan dengan metode *divisive* dan pada implementasinya dapat menggunakan fungsi jarak yang sesuai dengan kebutuhan pengguna [3]. Namun, Agglomerative Clustering tanpa optimalisasi dengan kompleksitas $O(N^3)$ tidak praktikal.

B. *Approximate Agglomerative Clustering*

Pada kesempatan kali ini digunakan Approximate Agglomerative Clustering sebagai metode untuk membangun BVH. Secara garis besar AAC adalah Agglomerative Clustering dengan tahapan preprocessing yang membagi *list cluster* menjadi beberapa *sub-list* hingga perhitungan jarak tidak dilakukan secara global [4].

Fitur utama dari AAC adalah pembagian *subset* (tahap *downward* - Gambar 3) di mana sebuah *node* hanya perlu mencari tetangga terdekat dari *subset* tersebut [4]. Dengan *node* yang sudah tersortir, pembagian *subset* dapat dilakukan dengan cepat dan tidak akan mengurangi kualitas *cluster* secara signifikan. *Sub-list* tersebut lalu di satukan dengan menggunakan Agglomerative Clustering (tahap *upward* - Gambar 4).

Tahapan pertama AAC adalah tahap *preprocessing* data poligon. Proses yang dilakukan adalah menghitung Morton Code masing-masing poligon dan melakukan *sorting* dengan Radix Sort yang dijalankan pada awal buildBVH (Gambar 5). Morton Code digunakan untuk memudahkan proses *sorting* dengan menjaga lokalitas data. Radix Sort digunakan karena cocok digunakan untuk tipe data Morton Code.

Setelah tahapan *preprocessing*, *cluster* akan dibentuk pada metode buildTree (Gambar 6) yang akan menghasilkan *cluster* sesuai parameter algoritma. Proses ini akan membagi *list* poligon secara rekursif hingga memenuhi threshold tertentu. Tahap pembagian list memanfaatkan Morton Code sehingga hanya perlu mencari perubahan bit di posisi tertentu. Dikarenakan Morton Code menjaga *locality* data, hasil pembagian *list* tidak akan merusak kualitas *tree*.

Setelah jumlah *node* dalam suatu *subset* memenuhi kriteria, akan dilakukan Agglomerative Clustering dengan metode combineClusters (Gambar 7) yang merupakan implementasi Agglomerative Clustering dengan optimalisasi *memoization*. Tahap terakhir adalah menggabungkan seluruh *cluster* yang tersisa menjadi satu *cluster* (*root*). Tahap penggabungan *cluster* pada BVH bersifat *non-decremental* sehingga kita dapat menerapkan beberapa optimalisasi seperti menghitung jarak di awal pemanggilan *loop* dan tidak menghitung ulang jarak kecuali pada *node* yang membutuhkan. Penelitian kali ini mengimplementasikan algoritma AAC yang diterapkan oleh G. Yan, H. Yong, *et al*. [4].

```
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)
```

Gambar 5. *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    //C = container
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|))
```

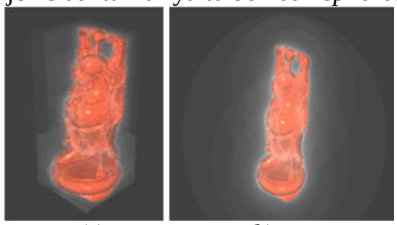
Gambar 6. *Pseudocode* buildTree.

```
1.  combineClusters(C,n)
2.  In : list of cluster (C)
3.  Out : list at most n clusters (C)
4.
5.    FOR EACH cluster in C DO
6.      findBestMatch(cluster,C)
7.    END FOR
8.    WHILE |C| > n DO
9.        Cl,Cr := bestMatch()
10.       Cn := newCluster(Cl,Cr)
11.       C = C – {Cl,Cr} + Cn
12.       findBestMatch(Cn,C)
13.       FOR EACH c in C DO
14.         IF c.closest = {Cl || Cr} THEN
15.           findBestMatch(c,C)
16.         END IF
17.       END FOR
18.   END WHILE
19.   RETURN C
```

Gambar 7. *Pseudocode* combineClusters.

C. *Container sebagai Bounding Volume*

Container adalah sebuah bangun tiga dimensi yang melingkupi objek lain (*container* lain atau poligon) dengan volume sekecil mungkin. *Container* digunakan sebagai *node* pada tree yang akan dibangun. Pada penelitian kali ini ditepkan dua jenis *container* yaitu *box* dan *sphere*.



(a) *(b)*

Gambar 8. Visualisasi BVH pada model “Buddha” **[5]** dengan *box container* (a) dan *sphere container* (b)

Implementasi *container* jenis *box* menggunakan konsep Axis Aligned Bounding Box (AABB) [3]. Karakteristik AABB adalah tiap sisinya paralel pada salah satu sisi yang dibentuk dari dua sumbu pada sistem koordinat. AABB dapat direpresentasikan dengan dua buah titik yaitu titik terjauh dan terdekat terhadap *origin* pada sistem koordinat. *Countainer* jenis *sphere* dibangun dengan membentuk lingkaran yang melingkupi bangun segitiga lalu membentuk bola berdasarkan lingkaran tersebut.

D. *Morton Code dan Radix Sort*

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 [6] sebagai metode untuk merepresentasikan *node* pada *quad tree* pada pembagian *scene* secara spasial.

Morton Code memudahkan perhitungan fungsi jarak antar dua titik pada dimensi lebih dari satu karena menjaga lokalitas tiap dimensi 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 tahap *downward* AAC [4].

Radix Sort adalah metode *sorting* yang secara iterasi mengurutkan 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 penelitian ini.

IV. HASIL UJI COBA DAN ANALISIS

Dalam penelitian ini dilakukan serangkaian uji coba untuk menganalisis beberapa parameter AAC dan pengaruhnya pada performa program secara keseluruhan. Dua parameter yang berhubungan langsung dengan performa program adalah jenis *container* dan *threshold* AAC.

Pada

Tabel **1** dan Tabel 2 terlihat bahwa *container* jenis *box* menunjukkan hasil 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 objek lain jika dibandingkan dengan *box* (Gambar 8). Hal ini menyebabkan jumlah persilangan *ray-container* yang lebih tinggi pada BVH pada *container* jenis *sphere* yang mengurangi efisiensi BVH secara umum.

Pada Gambar 9 terlihat bahwa nilai *threshold* berbanding lurus dengan kualitas BVH dan berbanding terbalik dengan kecepatan pembentukan BVH. Nilai *threshold* yang terlalu tinggi mengurangi efisiensi metode, terlihat nilai 12 memberikan nilai terbaik dan nilai 6 memberikan waktu pembentukan yang cepat. Berdasarkan hasil di atas ditetapkan dua jenis *threshold* AAC yaitu *fast* = 6 dan *HQ* =12.

Secara teori kualitas *tree* paling baik akan didapatkan dengan nilai *threshold* yang sama dengan jumlah poligon (*Naive Agglometarive Clustering*) namun waktu pembentukan yang lama membuat pendekatan tersebut tidak *feasible*. Kualitas *tree* juga dapat ditinjau dari jumlah pemanggilan fungsi *intersection* pada tiap *pixel*. Gambar 10 menunjukkan peningkatan kualitas *tree* yang berbanding lurus dengan nilai *threshold*. Pada Ray Tracing tanpa BVH, jumlah pemanggilan fungsi *intersection* akan konstan yaitu sesuai jumlah poligon, dari Gambar 10 dapat disimpulkan bahwa BVH dapat mengurangi kompleksitas Ray Tracing secara signifikan.

Tabel 1. Kecepatan berdasarkan jenis *container* BVH.

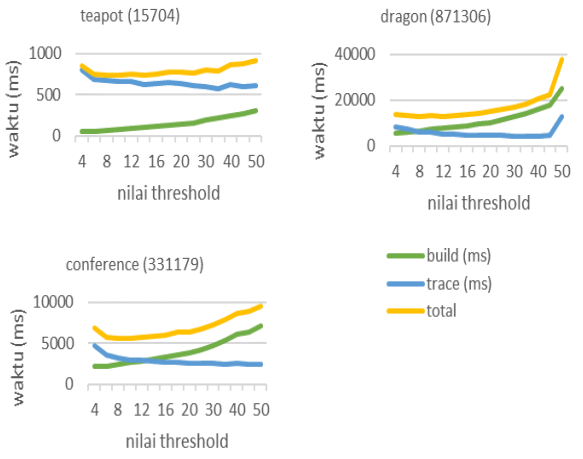
scene	box			sphere		
	build*	trace**	avg. intersects test ***	build*	trace**	avg. intersects test ***
teapot	92	626	19	126	3985	49
sponza	815	3373	119	1083	32462	430
car	1839	1526	42	2348	11418	114
conference	2904	2915	114	3634	51966	706
Buddha	4425	497	15	5774	2339	34
dragon	7807	5042	101	9824	30999	228

* *BVH build time (ms)*
** *scene tracing time (ms)*
*** *average ray-bin or ray-obj intersect function called per ray*

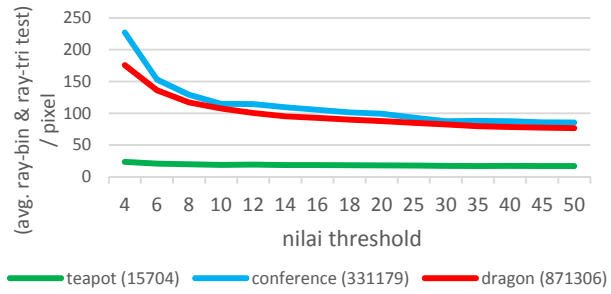
Tabel 2. Rasio kecepatan (*BOX* : *SPHERE).*

No	scene	Ratio (box : sphere)		
		build *	trace **	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

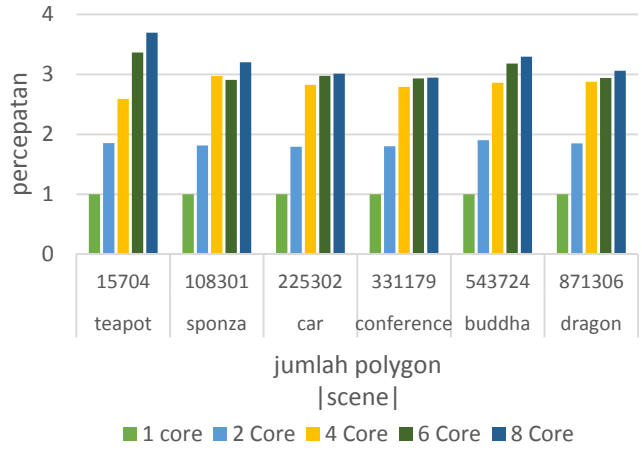
** BVH build time (ms)*
*** scene tracing time (ms)*
**** average ray-bin or ray-obj intersect function called per ray*



Gambar 9. Pengaruh *threshold* pada kecepatan konstruksi dan *rendering* pada tiga *scene* yang berbeda.



Gambar 10. Rerata jumlah *intersection test* tiap pixel pada uji coba threshold pada tiap scene



Gambar 11. Percepatan konstruksi BVH berdasarkan jumlah *core*, nilai tinggi lebih baik.

Hasil uji coba pada Gambar 11 menunjukkan peningkatan kecepatan yang signifikan pada perubahan jumlah *core*. Dikarenakan keterbatasan perangkat uji coba, tidak dapat ditentukan berapa besar peningkatan kecepatan pada jumlah *core* yang lebih besar. Berdasarkan hasil uji coba, penggunaan 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.

V. KESIMPULAN

Ray Tracing sebagai metode *rendering* realistis sangat terbantu oleh struktur data Bounding Volume Hierarchy. Pembentukan BVH dengan Approximate Agglomerative Clustering 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*).

Proses-proses yang menerapkan paralelisme akan meningkat performanya jika diterapkan pada (*Graphical Processing Unit*) GPU. Hal ini dikarenakan GPU sebagai unit dengan jumlah *core* yang besar dapat menjalankan banyak proses secara paralel dengan efisien.

REFERENCES

[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] B. Walter, K. Bala, M. Kulkarni dan K. Pingali, “Fast agglomerative clustering for rendering,” *IEEE Symposium on Interactive Ray Tracing 2008*, pp. 81-86, 2008.

[4] 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.

[5] M. McGuire, “Computer Graphics Archive,” August 2011. [Online]. Available: http://graphics.cs.williams.edu/data.

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