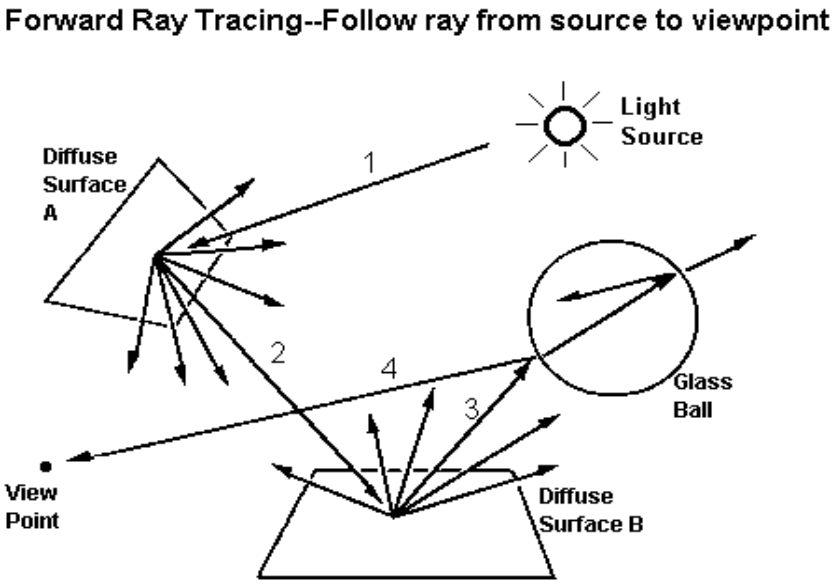
2. LANDASAN TEORI

2.1 *Ray Tracing*

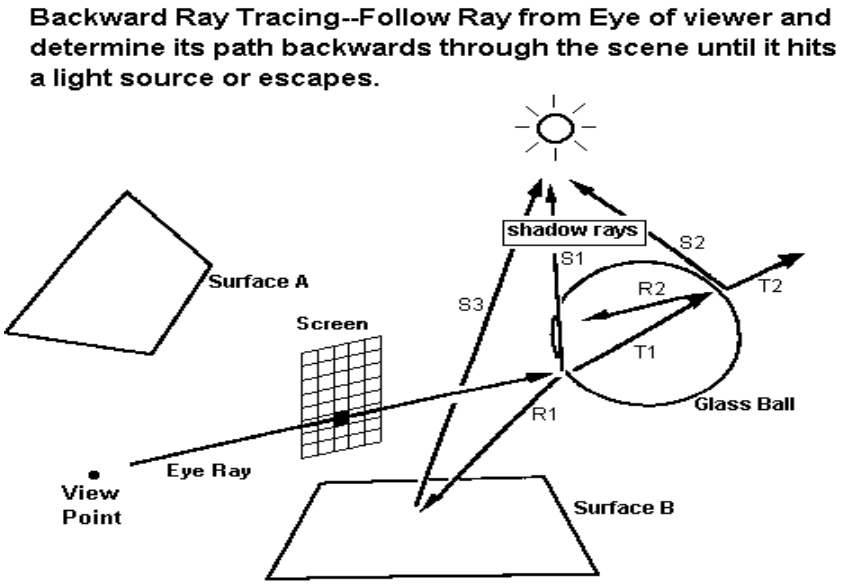
Dalam teori ilmu optik, mata dapat melihat karena objek yang dilihat yang mengeluarkan cahaya yang kemudian ditangkap oleh mata sehingga bisa terlihat (Ibnu Haytham). Setiap objek memiliki karakteristik tekstur dan warna yang merupakan hasil dari cara berinteraksi dengan objek. Contoh, sebuah obyek merah tampak mcrah karena cahaya merah dipantulkan sementara biru dan hijau diserap oleh obyek.

Ray tracing adalah suatu metode merender gambar dengan cara menelusuri sinar yang mengenai obyek. Jika sinar yang ditelusuri tersebut mengenai suatu obyek maka diperhitungkan intensitas warna obyek tersebut. Metode *ray tracing* dibagi menjadi dua jenis, yaitu forward ray tracing dan backward ray tracing. Forward ray tracing digunakan untuk mendapatkan intensitas obyek yang banyak dikenai sinar. Dengan metode ini didapatkan efek caustik dari objek-objek yang transparan sedangkan pada backward ray tracing sinar yang ditelusuri adalah sinar yang mengenai mata.



Gambar 2.1. Forward Ray tracing

Sumber: Eckert(1998, CS-460/560)



Gambar 2.2. Backward Ray tracing

Sumber: Eckert(1998, CS-460/560)

Metode forward ray tracing memperhitungkan semua sinar yang dipancarkan oleh sumber cahaya. Metode ini memperhitungkan keakuratan warna, namun tidak efektif karena jumlah yang dipancarkan oleh sumber cahaya sangat banyak (bisa mencapai jutaan sinar), dan jika sinar tersebut tidak mengenai mata maka sinar tersebut tidak akan diperhitungkan meskipun sebelumnya telah dihitung sebelumnya. Hal ini akan menimbulkan perhitungan yang sia-sia karena banyaknya sinar yang tidak diperhitungkan kemudian. Kelebihan dari metode ini adalah dapat memperoleh sinar yang lebih banyak dari pada metode *backward ray tracing.*

Seperti pada gambar 2.1 penelusuran sinar dilakukan mulai dari sumber cahaya seperti tertera pada no 1 menuju ke mata, sehingga semua sinar yang berasal dari sumber cahaya harus diperhitungkan. Metode penelusuran dari sumber cahaya ke mata inilah yang dinamakan metode *forward ray tracing.*

Cara kerja metode *backward ray tracing* adalah dcngan menelusuri sinar atau cahaya yang berasal dari mata. Sinar yang berasal dari mata dan melalui suatu pixel yang membentuk suatu layar gambar akan ditelusuri apakah sinar terscbut mengenai objek yang akan digambar. Jika sinar tersebut kena maka akan dilakukan perhitungan untuk menentukan intensitas titik tabrak objek tersebut.

Dalam proses perhitungan intensitas titik tabrak objek ini harus dipcrhatikan pula efek pcncahayaan dan efek visual yang ada. Intensitas ini berguna untuk mcmberi warna pada *pixel* yang bersangkutan. Jika sinar tersebut tidak mengenai objek yang akan digambar maka akan diberi warna seperti warna latar belakang *(background).*

Hal pertama yang harus dilakukan dalam proses perhitungan metode ini adalah mencari arah mata ke pixel yang membentuk layar gambar. Cara untuk mcncarinya adalah dengan menggunakan vektor acuan yang diberi nama PO. PO mcrupakan vcktor yang berasal dari mata yang menuju ke pixel pertama dcngan posisi kiri atas. Sctclah PO bcrhasil didapatkan, langkah selanjutnya adalah mcncari vcktor untuk *pixel-pixel* yang lain dcngan cara mcnambah atau mengurangkan vektor tersebut dcngan vektor acuan ke arah horisontal dan ke arah vertikal. Rumus yang digunakan untuk mendapatkan vektor acuan adalah (Hill, 1990):

$$\boldsymbol{P}0 = \boldsymbol{c} + (S/(2 * pix\ x)) * \boldsymbol{x} - (S/(2 * pix\ y)) * \boldsymbol{u} \qquad (2.1)$$

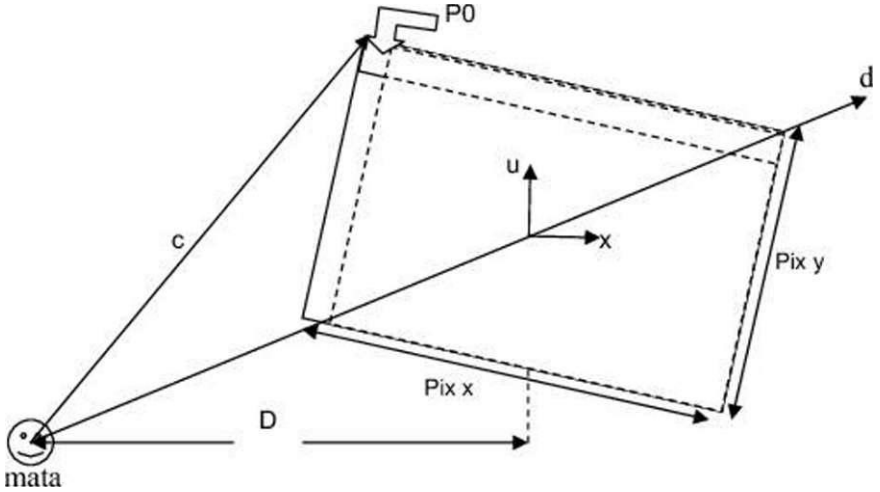
Pix x menyatakan lebar layar gambar dan pix y menyatakan tinggi layar gambar, sedangkan s menyatakan ukuran layar kamera. Vektor **u** merupakan vektor yang sejajar dengan arah atas kamera. Vektor **x** merupakan hasil *cross* antara vektor **u** dengan vektor arah pandang mata. Hasil perkalian *cross* tersebut merupakan vektor acuan untuk arah horisontal. Vektor c merupakan vektor arah ujung kiri atas layar gambar yang diperoleh dengan rumus (Hill, 1990):

$$\boldsymbol{c} = D * \boldsymbol{d} - (S/2) * \boldsymbol{x} + (S/2) * \boldsymbol{u} \qquad (2.2)$$

D menyatakan jarak antara mata ke layar gambar. Konstanta D ini sangat dipengaruhi oleh sudut pandang mata. Rumus yang digunakan untuk mencari konstanta D ini adalah (Hill, 1990):

$$D = S/2/tan(FOV * \pi/180) \qquad (2.3)$$

FOV merupakan sudut padang mata yang dinyatakan dengan satuan derajat. Untuk lebih jelas dapat dilihat pada gambar 2.3.



Gambar 2.3. Menentukan vektor pada setiap *pixel.*

Sumber: Sulian, M.L.S (2005)

2.1.1 Mencari Waktu tabrakan sinar dengan objek.

Pada pencarian titik tabrak terlebih dahulu harus diketahui waktu yang ditentukan sinar tersebut untuk objek yang bersangkutan. Untuk mendapatan waktu tabrakan tersebut adala dengan cara mensubtitusikan persamaan sinar dengan persamaan yang membentuk objek tersebut. Oleh karena itu dalam mencari titik tabrak dalam setiap objek berbeda. Rumus yang digunakan dalam adalah (Hill, 1990):

$$P(t) = S + dir * t \quad (2.4)$$

Keterangan: S = Posisi asal sinar

Dir = Arah sinar

T = Waktu sinar mencapai titik tabrak

P(t) = Titik tabrak

Bidang datar adalah suatu bentuk geometri dengan persamaan:

$$ax+by+cz = d \quad (2.5)$$

Dimana konstanta a, b, c merupakan bilangan arahnya yang diambil dari nilai pada sumbu x, y, z yang merupakan vektor normal bidang datar. Selain itu x, y, z merupakan komponen vektor yang berasal dari suatu titik yang terletak pada bidang.

Persamaan 2.5 dapat dituliskan:

$$N.P = d \quad (2.6)$$

Dengan mensubtitusikan persamaan 2.6 dan persamaan 2.4 maka akan mendapatkan persamaan baru untuk mencari waktu tabrakan dengan bidang datar adalah sebagai berikut:

$$N * (S + t * dir) = d \quad (2.9)$$

$$N * S + t * N \bullet dir = d \quad (2.10)$$

$$T = (d - N * S)/(N \bullet dir) \quad (2.11)$$

Dimana: t = Waktu tabrakan
d = Jarak bidang dengan koordinat
N = Normal bidang
S = Titik asal sinar yang menabrak
dir = Arah sinar

2.1.2 Efek Pencahayaan

Terdapat tiga macam efek pencahayaan, yaitu *ambient, diffuse* dan *specular.*

a) **Ambient**

Efek cahaya ambient berarti efek cahaya yang telah membaur dari lingkungan sekitar. Dalam kenyataan, semua sudut pada sebuah *scene* selalu diterangi oleh sinar yang dipantulkan oleh berbagai permukaan. Efek ini akan mempengaruhi terang atau tidaknya suatu lingkungan yang terlihat oleh mata.

Karena cahaya *ambient* adalah konstan, efek pencahayaannya menyinari semua permukaan dengan intensitas yang sama dan menghasilkan obyek-obyck monokrom. Kontribusi cahaya *ambient* kepada warna pixel dapat dihitung dengan menggunakan persamaan:

$$I = la * Ka \quad (2.12)$$

dimana,

I = Intcnsitas yang dihasilkan

la = Intcnsitas *ambient*

Ka = Kocfisicn *ambient*

b) **Diffuse**

Efek cahaya *diffuse* adalah pencahayaan yang bergantung pada sudut datang cahaya pada suatu permukaan. Scmakin besar sudutnya. semakin kecil efeknya. Hal ini tcrjadi karena energi sinar tersebar dalam area yang lebih luas.

Cahaya *diffuse* memodelkan permukaan yang cenderung untuk menyebarkan sinar ke segala arah. Hal ini terjadi karena pada permukaan yang tampaknya halus tersebut terdapat tonjolan-tonjolan mikroskopis yang normalnya tidak selaras dengan nonnal dari obyek yang bersangkutan secara makroskopis.

Obyek dengan permukaan buram memiliki nilai komponen difiise yang besar. permukaannya tampak sama terangnya dari semua arah pengamatan.

Intcnsitas *diffuse* dapat dicari dengan hukum Lambcrtian scbagai berikut

$$I = Ip * Kd \cos 6) \qquad (2.13)$$

Dari persamaan intcnsitas *diffuse* tcrscbut cos *0* dapat dihitung dengan melakukan dot product antara sinar dari lampu kc titik tabrak obyek dengan normal obyek itu, masing-masing merupakan unit vektor. Sehingga didapat persamaan baru yaitu :

$$I = Ip * Kd * (L \bullet N) \qquad (2.14)$$

dimana,

I = Intensitas yang dihasilkan

Ip = Intensitas *diffuse* dari sumber cahaya 'x'

Kd = Koofisicn *diffuse*

N = Vektor normal dari obyck

L = Vektor dari titik tabrak ke sumber cahaya

Q = Sudut antara N dan L

c) **Specular**

Jenis pencahayaan ketiga adalah refleksi *specular*. Pada permukaan yang berkilau seperti logam yang dipoles terdapat *highlight* atau titik yang lebih terang daripada sckitarnya. Refleksi jenis ini bersifat *view-dependant*, sehingga tidak seperti *ambient* dan *diffuse* ia harus dlhitung ulang ketika mata berpindah lokasi.

Pada dasamya, jika mata berada tepat pada arah refleksi, kontribusi dari komponen *specular* lebih tinggi daripada jika mata lebih jauh dari arah refleksi. Untuk menghitung intensitas *specular*, digunakan persamaan:

$$I = Ip*Ks(cos0)^n$$

(2.15)

Dari persamaan intensitas *specular* tersebut cos*0* dapat dihitung dcngan menggunakan dot product antara arah pantulan dengan negasi dari arah sinar.

$$I = Ip*Ks*(R \bullet V)^n \quad (2.16)$$

Dimana,

I = Intensitas yang dihasilkan

Is = Intensitas *specular* dari sumber cahaya 'x'

Ks = Koofisien *specular*

n = Variabel yang mencentukan luas area yang bcrkilau jika tcrkena cahaya yang dipancarkan olch sumber cahaya (bila n scmakin besar maka cahaya scmakin terfokus atau area yang berkilau menjadi Iebih kecil)

R = Arah pantulan, bcrupa unit vektor

V = Negasi dari arah sinar Sedangkan vektor R diperolch dari

$$-S + 2 * (S \bullet N) * N \quad (2.17)$$

dimana,

S = Vektor dari titik tabrak ke sumber cahaya

N = Vektor normal dari obyek

2.2 Percepatan dalam Ray tracing

Di dalam ray tracing terdapat 3 teknik untuk mempercepat proses ray tracing itu sendiri, teknik tersebut adalah *fast intersection*, *fewer rays*, dan *generalized rays*. Contoh dari teknik percepatan tersebut adalah *Object bounding volume, Bounding volume hierarchies, Adaptive tree-dept control.*

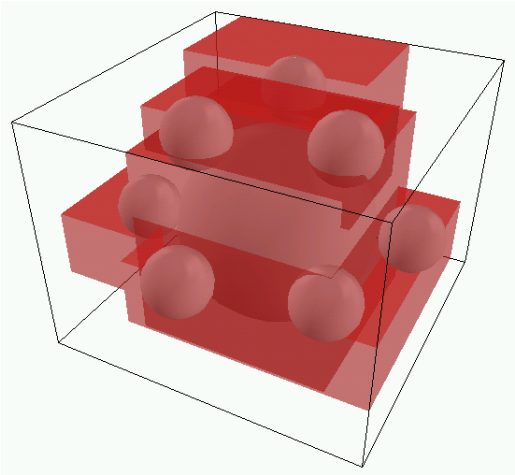
2.2.1 *Object Bounding Volume*

Object bounding volume adalah merupakan batasan dari kumpulan beberapa objek, Pada *ray tracing object bounding volume* digunakan untuk mengecek titik potong sinar. Jika sinar tidak bertabrakan dengan *bounding volume*

maka sinar tersebut tidak akan berpotongan dengan objek yang ada di *bounding volume* tersebut. Oleh karena cepatnya pengecekan dalam mencari titik tabrak di dalam menggunakan *bounding volume* maka hal ini dapat mempercepat dalam proses *ray tracing*.

Tipe umum dari *bounding volume* adalah sebagai berikut:

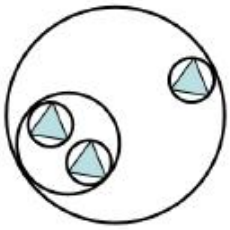
- Bounding Sphere yaitu sebuah adalah sebuah batasan yang berbentuk bola.
- Bounding Cylinder yaitu sebuah batasan yang berbentuk silinder
- Bounding Box yaitu sebuah batasan yang berbentuk kubus.



Gambar 2.4. *Bounding Volume Box*

2.2.2 *Bounding Volume Hierarchies*

Bounding volume hierarchies merupakan struktur hirarkis dari *bounding volume* untuk mempercepat proses *rendering* di dalam *ray tracing*. Penggunaannya adalah dengan membuat sebuah *bounding volume* yang terdiri dari beberapa objek secara menyeluruh. Kemudian membuat bounding volume yang lebih kecil terbagi dari dua atau lebih. Proses ini berlangsung secara rekursif hingga volume menjadi lebih kecil dan berisi lebih sedikit objek, hingga akhirnya berhenti jika batas maksimal jumlah objek dalam sebuah bounding tercapai.



Gambar 2.5.*Bounding volume hierarchies* menggunakan *Bounding sphere*

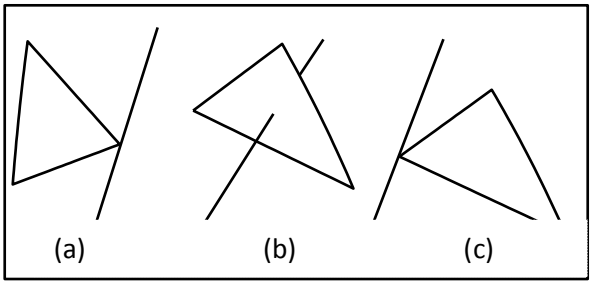
2.2.3 *Distance Comparison*

Metode *distance comparison* atau yang dikenal dengan perbandingan dengan jarak seperti yang telah disertakan pada jurnal *Acceleration rendering method on ray tracing with angle comparison and distance comparison* oleh Liliana dan Rudy Adipranata juga dapat mempercepat proses rendering dalam ray tracing.

Metode ini bertujuan untuk mengurangi persimpangan sinar dengan perbandingan jarak dengan objek. Gagasan dalam perbandingan sudut ialah jika jarak antara sinar dan satu titik dari tiga poin dalam segitiga cukup dekat, sinar dapat memotong segitiga. Ada tiga kondisi utama di mana sinar berpotongan segitiga:

- Kondisi pertama (a) adalah sinar memotong titik dalam segitiga yang akan diperiksa yang jarak dari sinar. Jika kondisi ini terjadi, jarak akan menjadi nol. Ini adalah kondisi terbaik.
- Kondisi kedua (b) adalah sinar memotong setiap titik di segitiga.
- Kondisi terakhir (c) adalah sinar memotong titik terjauh dari titik akan diperiksa jaraknya. Ini adalah kondisi terburuk.

Tiga kondisi ditampilkan dalam gambar 2.6.



Gambar 2.6. 3 Kondisi dari perpotongan sinar

Dari kondisi terburuk, dapat ditentukan bahwa jika jarak antara titik dalam segitiga dan ray tidak lebih panjang dari sisi terpanjang dari segitiga, maka akan memiliki probabilitas tinggi untuk memotong segitiga.

Untuk mengukur jarak, proyeksi titik ke garis (panah) yang digunakan, lihat Persamaan (2.14). Point (A) adalah salah satu dari tiga titik segitiga dan panah adalah sinar yang berpotongan segitiga.

$$d = | \text{sinar} \times A | / | A | \quad (2.14)$$

dimana: d: jarak antara ray untuk segitiga

sinar: sinar yang akan ditentukan jika memotong segitiga

J: vektor dari titik A ke Sumber sinar

| sinar x A |: besarnya vektor sinar produk dengan A.

| A |: besar vektor A Untuk menggunakan perbandingan jarak, pra-proses diperlukan.

2.2.4 *Spatial Hierarchies*

Teknik partisi ruang untuk mempercepat proses *rendering* dapat menggunakan *kd-tree*. Pada *kd-tree* objek di dalam gambar dipartisi sepanjang dimensi x, y dan z. Untuk mendapatkan struktur yang seimbang maka dimensi yang dipilih pada setiap langkah partisi di tentukan dari input sebuah objek di gambar. Struktur pencarian ini efektif karena menggabungkan effesiensi dalam pencarian geometri yang diambil dari data struktur seperi *octress* yang mempunyai fleksibilitas untuk mengadaptasi dalam sebuah gambar terhadap dirinya sendiri. Untuk lebih jelasnya *kd-tree* akan dijelaskan pada sub bab bagian dibawah ini.

2.3 *Kd-Tree*

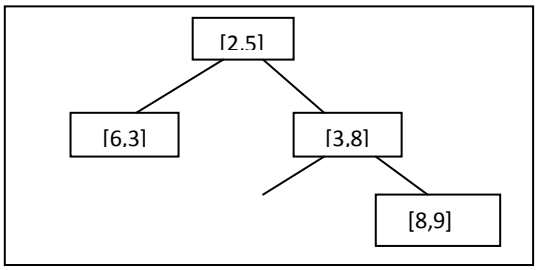
Kd-tree merupakan struktur data yang menyimpan suatu set titik terbatas dari sebuah ruang dimensi-*k*. *kd-tree* adalah sebuah pohon biner di mana setiap node adalah titik k-dimensi. Setiap simpul non-daun dapat dianggap sebagai secara implisit menghasilkan hyperplane membelah yang membagi ruang menjadi dua bagian, yang dikenal sebagai subspaces. Poin ke kiri dari hyperplane ini merupakan sub-pohon kiri dari simpul tersebut dan titik kanan hyperplane yang

diwakili oleh pohon-sub kanan. Arah hyperplane dipilih dengan cara berikut: setiap node di pohon dikaitkan dengan salah satu k-dimensi, dengan hyperplane tegak lurus dengan sumbu bahwa dimensi itu. Isi dari setiap node dapat dilihat di tabel 2.1.

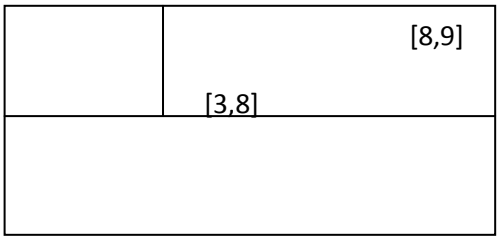
Tabel 2.1 Isi atribut dari *kd-tree*

Nama	Tipe	Deskripsi
Dom-elt	Domain-vektor	Titik dari ruang K_d-d
Range-elt	Range -vektor	Titik dari ruang K_r-d
split	Integer	Dimensi dari pembagian
left	*Kd-tree*	Sebuah kd tree yang mewakili titik-titik dari kiri bidang
right	*Kd-tree*	Sebuah kd tree yang mewakili titik-titik dari kanan bidang

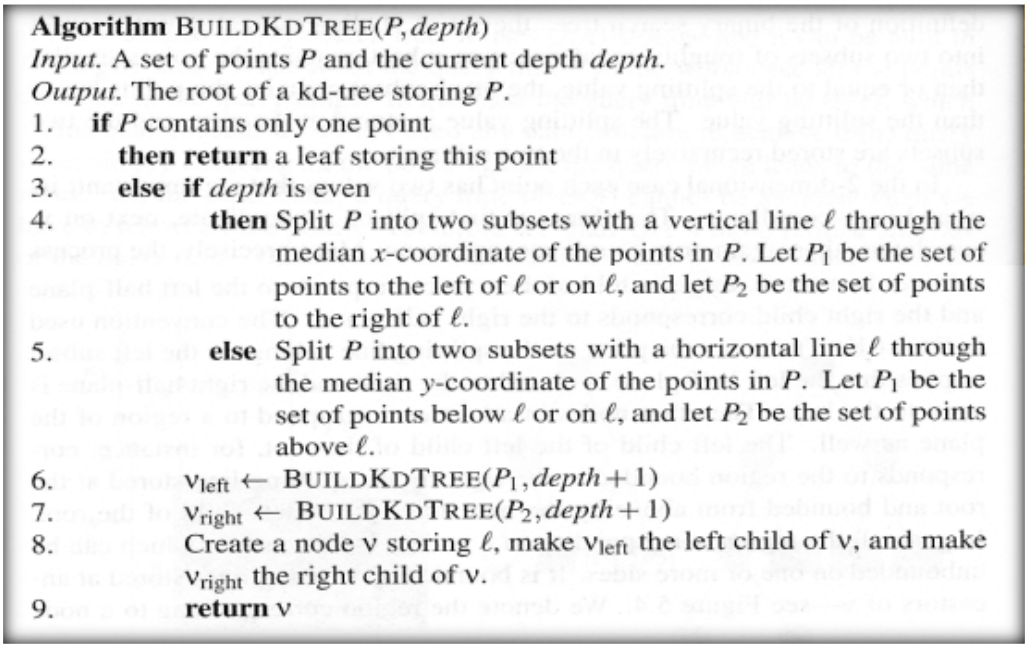
Penggunaan *kd-tree* yaitu dengan memasukan contoh dari beberapa titik (2,5) , (3,8) , (6,3) , dan (8,9). Akar dari node dengan **Dom-elt** (2,5) membagi bidang pada dimensi y menjadi 2 bagian sub-ruang. Titik (3,8) yang berada pada bawah sub-ruang, yaitu {(x,y)|y < 5} jadi meninggalkan dari sub-tree.



Gambar 2.7: Tree 2-dimensi dari 4 element



Gambar 2.8: Hasil kd-tree pada gambar 2.7 dibagi pada bidang x,y.



Algorithm BUILDKDTREE(P, *depth*)
Input. A set of points P and the current depth *depth*.
Output. The root of a kd-tree storing P.
1. **if** P contains only one point
2. **then return** a leaf storing this point
3. **else if** *depth* is even
4. **then** Split P into two subsets with a vertical line ℓ through the median x-coordinate of the points in P. Let P_1 be the set of points to the left of ℓ or on ℓ, and let P_2 be the set of points to the right of ℓ.
5. **else** Split P into two subsets with a horizontal line ℓ through the median y-coordinate of the points in P. Let P_1 be the set of points below ℓ or on ℓ, and let P_2 be the set of points above ℓ.
6. $v_{left} \leftarrow$ BUILDKDTREE(P_1, *depth* + 1)
7. $v_{right} \leftarrow$ BUILDKDTREE(P_2, *depth* + 1)
8. Create a node v storing ℓ, make v_{left} the left child of v, and make v_{right} the right child of v.
9. **return** v

Gambar 2.9 Tabel Algoritma untuk membangun *kd-tree*

2.4 *Mesh Object*

Mesh object adalah suatu objek komplek yang terbentuk dari bidang segitiga-segitiga sederhana yang digabungkan. *Mesh Object* termasuk objek dalam bentuk 3D.

2.4.1 Struktur *Mesh Object* dari file .obj

Mesh Objek terdiri dari komponen dasar yaitu *Vertex*, *Edge* dan *Face*.

- *Vertex*

  Adalah sebuah titik atau posisi pada ruang 3D.

- *Edge*

  Adalah sebuah garis yang menghubungkan dua *vertex*. *Edge* digunakan untuk membuat *Face.*

- *Face*

  Adalah bidang yang dibatasi oleh tiga *vertex.* Dimana setiap dua vertex dihubungkan oleh sebuah *edge.* Dengan demikian *Face* berbentuk segitiga. *Face* digunakan untuk menggambarkan permukaan dari sebuah objek. Pada saat merender objek maka *face* akan terlihat.

2.4.2 Struktur *Mesh Object* dari file 3DS.

File 3ds mengandung informasi yang digunakan untuk menggambarkan detail dari *scene* 3D yang terdiri satu atau lebih dari suatu objek. File 3ds berisi dari serangkaian blok yang disebut *Chunk*. Di dalam setiap blok berisikan Segala sesuatu yang diperlukan untuk menggambarkan *scene*: nama dari setiap objek, simpul koordinat, pemetaan koordinat, daftar poligon, wajah warna, *keyframes* animasi dan sebagainya. *Chunk* tidak memiliki struktur yang linear. Ini berarti chunk bergantung pada yang lain dan hanya bisa dibaca jika relatif dari induk-*chunk* telah dibaca terlebih dahulu.

Chunk terdiri dari 3 *field* yaitu:

- Identifier:

Dua nomor byte heksadesimal panjang yang mengidentifikasi sebuah chunk. Informasi ini memberitahu kita jika chunk ini berguna untuk tujuan kita. Jika kita perlu chunk kita kemudian dapat meramalkan kemungkinan informasi *scene* di dalamnya. Jika kita tidak perlu kita melewati *chunk* tersebut menggunakan informasi tersebut.

- Length of the chunk:

4 byte nomor yang merupakan jumlah dari panjang *chunk* dan semua panjang dari setiap potongan chunk yang terkandung.

- Chunk data:

Bidang ini memiliki panjang variabel dan mengandung semua data untuk *scene*.

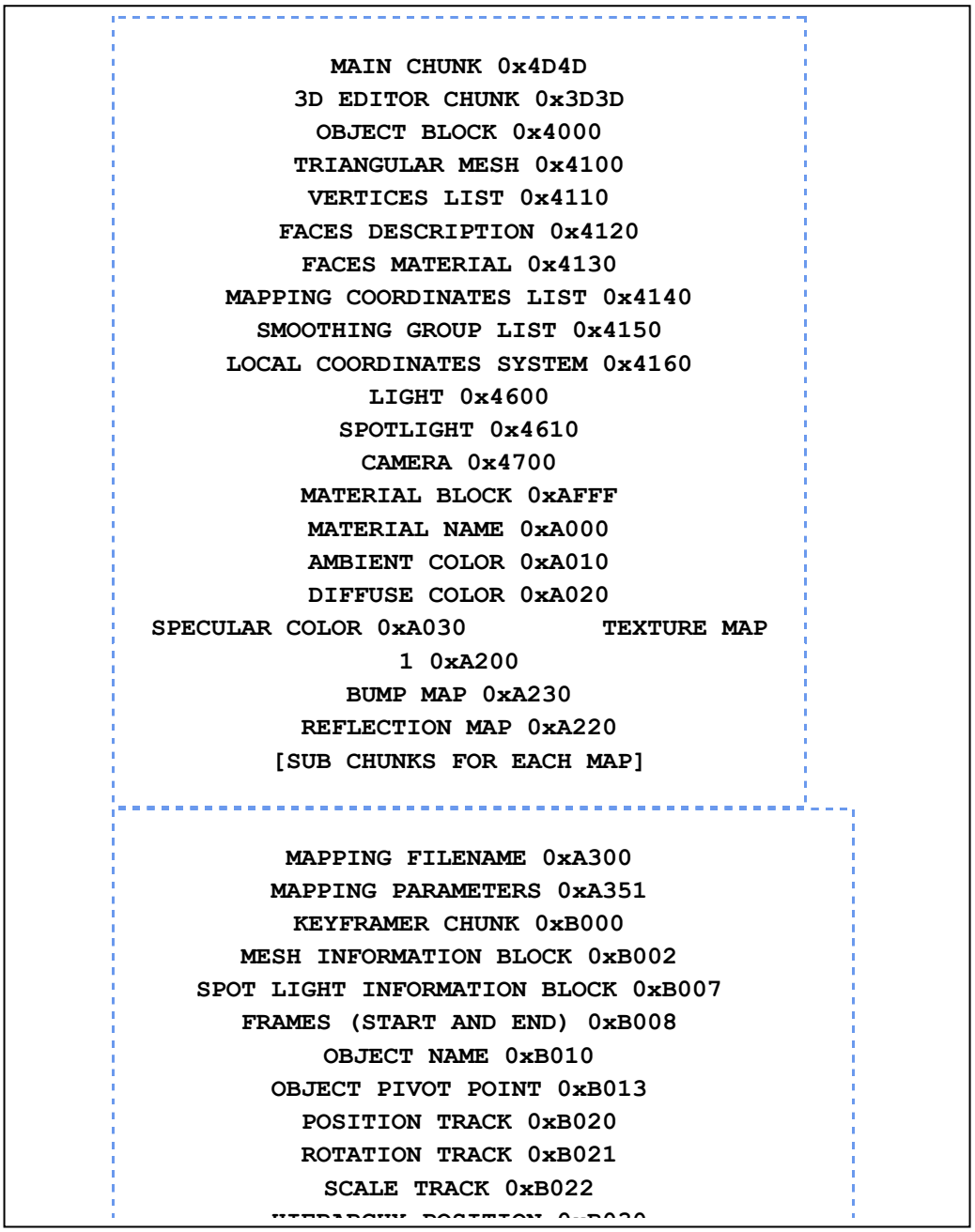
Pada tabel 2.3 ditunjukkan Offset (dalam *bytes*) dan Length (dalam *bytes*) dari setiap *field* di potongan *chunk*.

Tabel 2.3. Gambaran umum dari *chunk*.

Offset	Length	Description
0	2	Chunk identifier
2	4	Chunk length: chunk data + sub-chunks(6+n+m)
6	n	Data
6+n	m	Sub-chunks

Pada baris terakhir dalam tabel 2.4 terlihat bagaimana beberapa potongan tergantung pada yang lain: masing-masing anak-*chunk* sebenarnya terkandung di dalam field "Sub-*chunk*" dari induk-chunk.

Berikut Hirarki dalam berbagai variasi element di dalam file 3ds.



Gambar 2.10 Variasi element di dalam file 3ds.

Untuk dapat membaca *chunk* tertentu harus didahului dengan membaca induk-*chunk* terlebih dahulu. File 3ds mirip seperti tree dimana untuk membacanya harus dimulai pengurutan membaca mulai dari akar, batang sampai ke daun yang diinginkan. Seperti pada hirarki diatas untuk dapat memperoleh **VERTICES LIST**, kita harus membaca **MAIN CHUNK** dulu, kemudian **3D**

EDITOR CHUNK, **OBJECT BLOCK** dan yang terakhir **TRIANGULAR MESH** chunk. Chunk yang lain dapat dilewati. Detail dari hirarki diatas dapat dilihat pada tabel 2.4 dibawah ini.

Tabel 2.4. Detail dari chunk-chunk

MAIN CHUNK	
Identifier	0x4d4d
Length	0 + sub-chunks length
Chunk	None
Sub	3D EDITOR CHUNK
Data	None
3D EDITOR CHUNK	
Identifier	0x3D3D
Length	0 + sub-chunks length
Chunk	MAIN CHUNK
Sub	OBJECT BLOCK, MATERIAL BLOCK, KEYFRAMER
Data	None
OBJECT BLOCK	
Identifier	0x4000
Length	Object name length + sub-chunks length
Chunk	3D EDITOR CHUNK
Sub	TRIANGULAR MESH, LIGHT, CAMERA
Data	Object name

TRIANGULAR MESH	
Identifier	0x4100
Length	0 + sub-chunks length
Chunk	OBJECT BLOCK
Sub	VERTICES LIST, FACES DESCRIPTION, MAPPING
Data	None
VERTICES LIST	
Identifier	0x4110
Length	varying + sub-chunks length
Chunk	TRIANGULAR MESH
Sub	None
Data	Vertices number (unsigned short)
FACES DESCRIPTION	
Identifier	0x4120
Length	varying + sub-chunks length
Chunk	TRIANGULAR MESH
Sub	FACES MATERIAL
Data	Polygons number (unsigned short)
MAPPING COORDINATES LIST	
Identifier	0x4140

Length	varying + sub-chunks length
Chunk	TRIANGULAR MESH
Sub	SMOOTHING GROUP LIST
Data	Vertices number (unsigned short)