# IMPLEMENTASI OBYEK GRAFIS 3-D DENGAN POV-Ray

Rudy Hartanto[1]

***Abstract* — The development of 3-D graphics applications leads to the generation of photo-realistic graphics image, which is implemented many applications, such us design, simulation, games as well as to making animated films. Rendering techniques used to generate the graphics image is still les realistic, because lack of real-world lighting models. Subsequent research led to the global illumination models that used a model of the light rays as in the actual physical environment. The algorithm that can implement global illumination models are ray-tracing algorithm. This algorithm is capable of producing 3-D graphic image of a high quality photo-realistic, but requires a complex mathematical equation and computation. POV-Ray is a tool used to implement the ray-tracing models to produce photo-realistic graphics image. These results show that the POV-Ray can generate 3-D graphic image of high qualityt.**

***Intisari* — Perkembangan aplikasi grafis 3-D mengarah pada pembangkitan citra grafis foto-realistis, yang sangat diperlukan mulai dari aplikasi perancangan, simulasi, game sampai pembuatan film animasi. Teknik rendering yang digunakan untuk membangkitkan citra grafis masih belum realistis, karena belum didasarkan pada model pencahayaan dunia nyata. Penelitian selanjutnya mengarah pada model iluminasi global yang menggunakan model berkas cahaya seperti pada lingkungan fisik sebenarnya. Salah satu algoritma yang dapat mengimlementasikan model iluminasi global adalah algoritma ray-tracing. Algoritma ini mampu menghasilkan citra grafis 3-D kualitas tinggi mendekati foto-realistis, namun membutuhkan persamaan matematis dan komputasi yang kompleks. POV-Ray adalah salah satu tool yang digunakan untuk mengimplementasikan model ray-tracing untuk menghasilkan citra grafis foto-realistis. hasil penelitian ini menunjukkan bahwa POV-Ray dapat menghasilkan citra grafis 3-D kualitas tinggi.**

***Kata kunci*— Foto-realistis, iluminasi global, ray-tracing, POV-Ray.**

## I. PENDAHULUAN

Teknologi grafis 3-dimensi telah merambah ke berbagai bidang aplikasi, mulai dari aplikasi grafis sederhana pada handphone sampai pada aplikasi pemodelan dan visualisasi canggih pada film animasi yang diproses dengan menggunakan komputer super.

Aplikasi pemodelan grafis pada industri meliputi industri automotif, kimia, televisi, periklanan, film, game multimedia dan berbagai aplikasi yang menggunakan komputer untuk memvisualisasikan data [1].

[1] *Dosen, Jurusan Teknik Elektro dan Teknologi Informasi Fakultas Teknik Universitas Gadjah Mada, Jln. Grafika 2 Yogyakarta 55281 INDONESIA (telp: 0274-552305; fax: 0274-552305; e-mail: rudy@mti.gadjahmada.edu)*

Sejak berkembangnya grafika komputer pada tahun 1960 sudah banyak topik diskusi tentang bagaimana membangkitkan citra foto yang realistis dari model 3-D. Untuk menghasilkan citra foto realistis maka perlu secara tepat mensimulasikan fenomena cahaya dengan obyek dan teksturnya agar dapat terlihat realistis. Foto-realistis sangat diperlukan dalam berbagai aplikasi. Misalnya pada aplikasi perancangan obyek 3-D seperti perancangan mobil atau motor, pesawat terbang dan bangunan, simulator penerbangan, atau pada game dan film animasi. Teknik rendering cukup baik untuk mesin waktu-nyata karena cukup cepat dan hampir semua kartu grafis menggunakannya. Namun citra yang dihasilkan terlihat tidak terlalu realistis, karena tidak dapat mensimulasikan refleksi, refraksi, bayangan, dan aspek lain dalam rangka memberikan impresi yang diperlukan untuk citra foto-realistis. Hal ini karena algoritma yang digunakan dalam proses rendering dan rasterisasi tidak didasarkan pada model pencahayaan dunia nyata.

Kesulitan dalam mencapai hasil seperti citra foto-realistis mendorong para peneliti mengarah pada isu iluminasi global yang mengacu pada simulasi berbasis fisik dari berkas cahaya yang dihamburkan (scatered) dalam suatu lingkungan buatan [2]. Iluminasi global bukanlah suatu algoritma melainkan suatu efek yang ingin dicapai. Algoritma awal yang dikembangkan untuk mensimulasikan iluminasi gobal adalah algoritma ray-tracing. Dalam perkembangannya ray-tracing telah banyak digunakan dalam grafika komputer dan mampu menghasilkan gambar 3-D dengan kualitas tinggi yang mendekati foto-realistis. Pada makalah ini akan mengimplementasikan suatu obyek grafis 3-D foto-realistis dengan menggunakan POV-Ray.

Fenomena tentang bagaimana menghasilkan obyek 3-D yang terlihat realistis sudah menjadi salah satu topik penelitian yang intens pada bidang grafika komputer. Salah satu faktor yang menentukan agar suatu obyek garfis 3-D dapat terlihat realitstis adalah ray-tracing. Namun ray-tracing membutuhkan komputasi atas persamaan matematis yang cukup kompleks. Terdapat beberapa tool maupun pustaka grafis untuk implementasinya, seperti OpenGL, POVRay, dan Yafaray. Sudah banyak penelitian yang membahas tentang metoda untuk menghasilkan model 3-D yang terlihat realistis. Johnson [3] telah melakukan penelitian untuk meningkatkan citra yang dibangkitkan dengan komputer agar terlihat realistis dengan menggunakan sejumlah besar koleksi fotografi. Berdasarkan pada citra komputer grafis, mereka mengambil sejumlah gambar nyata dengan struktur global yang sama. Selanjutnya dilakukan proses identifikasi area yang sesuai antara citra komputer dengan citra nyata dengan menggunakan algoritma ko-segmentasi pergeseran-rerata. Pengguna selanjutnya dapat mentransfer warna, tone, dan tekstur secara otomatis dari area

yang cocok dengan citra grafis. Hasilnya berupa citra hibrida yang terlihat lebih realistis.

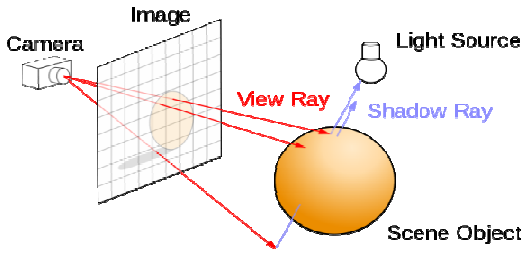
Roger Olsson dan Youzhi Xu [4] meneliti tentang simulasi lingkungan berbasis ray-tracing interaktif untuk membangkitkan runtun citra video terpadu. Mereka telah menciptakan lingkungan simulasi interaktif berbasis ray-tracing untuk membangkitkan runtun video Integral Imaging (II) sebagai suatu cara untuk membantu mengembangkan, mengevaluasi dan mengadopsi teknik baru untuk membangkitkan video 3-D. Model generik II juga dimaksudkan sebagai dasar dari lingkungan simulasi. Model ini memberi fasilitas untuk untuk merender II dengan menggunakan MegaPOV, yaitu versi custom dari POVRay. Artzi Aner Ben, Ravi Ramamoorthi, dan Maneesh Agrawala [5] melakukan penelitian untuk menyelesaikan permasalahan perhitungan bayangan yang efisien dari lingkungan dalam konteks ray-tracing. Penelitian ini menunjukkan bahwa koherensi pada ranah spasial dan sudut dapat digunakan untuk mengurangi jumlah berkas-bayangan yang harus dilacak sampai faktor 20 dengan mempertahankan pesat galat di bawah 0,01%.

Tushar Udeshi and Charles D. Hansen [6], mengembangkan suatu teknik untuk membangkitkan gambar sederhana dalam ruangan yang mendekati foto realistis, dengan menggunakan CPU dan perangkat keras grafis paralel. Sasaran yang ingin dicapai adalah dapat memanfaatkan sebanyak mungkin efek iluminasi global dengan mempertahankan pesat frame yang tinggi. Penelitian ini juga menjelaskan metoda untuk membangkitkan shoft shadow, pendekatan pantulan tunggal dari pencahayaan tidak langsung, dan efek refleksi serta refraksi.

## II. RAY TRACING

Ray-tracing adalah suatu teknik rendering yang menghitung citra dari adegan dengan mensimulasikan cara cahaya merambat dalam dunia nyata [7]. Ray-tracing menentukan visibilitas permukaan dengan melacak berkas imaginer dari mata pengamat ke obyek dalam adegan. Ide dari ray-tracing datang dari fisika. Untuk merancang lensa, fisikawan akan menggambar jalur berkas cahaya mulai dari sumber cahaya, yang disebut dengan ray-tracing.

Dalam dunia nyata berkas cahaya dipancarkan dari sumber cahaya dan menerangi obyek. Cahaya dapat memantul dari obyek atau melewati obyek yang transparan. Pantulan cahaya ini yang mengenai mata pengamat atau lensa kamera. Program ray-tracing seperti POVRay dimulai dengan mensimulasikan kamera dan melacak berkas cahaya ke belakang keluar dari adegan gambar. Pengguna dapat menentukan lokasi kamera, sumber cahaya dan sembarang media yang dilewati cahaya seperti seperti kabut, air, dan lain sebagainya. Untuk setiap piksel gambar, satu atau lebih berkas yang terlihat ditembakkan dari kamera kedalam adegan untuk melihat apakah terdapat potongan dengan obyek. Berkas cahaya ini berasal dari pengamat, yang direpresentasikan dengan kamera, dan melewati jendela pengamat yang merepresentasikan citra akhir seperti yang diperlihatkan pada Gbr. 1.




Gbr. 1. Ide dasar Ray-Tracing. [8]

Setiap kali obyek dikenai cahaya, maka warna permukaan pada titik tersebut akan dihitung. Untuk keperluan ini maka berkas cahaya akan dikirim balik ke masing-masing sumber cahaya untuk menentukan jumlah cahaya yang datang dari sumber cahaya. Warna dari piksel permukaan obyek akan ditentukan berdasarkan jalur berkas ke sumber cahaya. Teknik ini dapat menangani fenomena fisik dari cahaya serta refleksi, refraksi, atau difusi [8]. Berkas bayangan ini akan diuji untuk mengetahui apakah titik permukaan terletak dalam bayangan atau tidak. Jika permukaan obyek bersifat reflektif atau transparan, maka berkas yang baru akan diset dan dilacak untuk menentukan kontribusi dari refleksi atau refraksi cahaya pada warna akhir dari permukaan obyek.

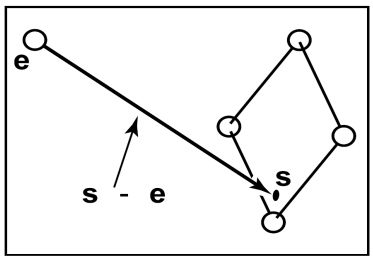
## III. HASIL DAN PEMBAHASAN

### A. *Menghitung Berkas Cahaya yang Terlihat*

Alat bantu dasar untuk pembangkitan berkas cahaya adalah titik pandang, dan bidang gambar. Untuk membangkitkan berkas cahaya, maka diperlukan representasi matematis untuk berkas cahaya. Suatu berkas dapat dinyatakan dengan titik pusat dan arah perambatan yang idealnya dinyatakan dengan garis parametris 3-D [9].

Pelacakan berkas mempunyai tiga tahap:

1. Pembangkitan berkas, untuk menghitung pusat dan arah masing-masing berkas piksel berdasarkan geometri kamera.
2. Titik perpotongan berkas, untuk mendapatkan titik perpotongan obyek terdekat dengan berkas yang kelihatan.
3. Bayangan, untuk menghitung warna piksel berdasarkan hasil titik perpotongan berkas.




Gbr. 2 Berkas dari mata ke titik pada bidang gambar [9].

Garis parametris 3-D dari mata **e** ke suatu titik **s** pada bidang gambar seperti yang diperlihatkan pada Gbr. 2, diberikan oleh persamaan:

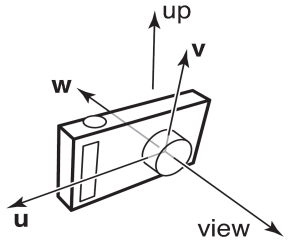
$$p(t) = e + t(s - e), \quad (1)$$

persamaan di atas diinterpretasikan sebagai "gerakan maju dari e sepanjang vektor (s - e) yaitu potongan jarak *t* untuk mendapatkan titik p". Jadi jika *t* diketahui maka dapat ditentukan titik p. Titik e adalah pusat berkas, dan s – e adalah arah berkas.

Untuk p(0) = e, dan p(1) = s, sehingga secara umum jika $0 < t_1 < t_2$, maka $p(t_1)$ akan lebih dekat ke mata dibanding $p(t_2)$. Dengan demikian jika $t < 0$, maka p(*t*) akan berada "di belakang" mata. Fakta ini berguna saat mencari obyek terdekat yang terkena berkas yang tidak berada di belakang mata. Untuk menghitung berkas yang terlihat diperlukan nilai e (diketahui) dan s.

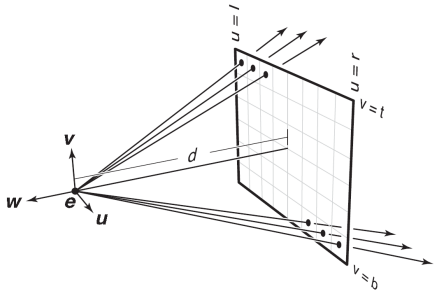
Metode pembangkitan berkas dimulai dari koordinat *orthonormal frame*, yang dikenal dengan *camera frame* yang ditandai dengan e untuk titik mata atau titik pandang, dan **u**, **v** serta **w** yang merupakan tiga vektor basis. Vektor basis diorganisasikan dengan **u** yang menunjuk ke kanan ( dari sudut pandang kamera), **v** menunjuk ke atas dan **w** menunjuk ke belakang sehingga membentuk sistem koordinat tiga dimensi, seperti yang diperlihatkan pada Gbr. 3.

Berkas yang terlihat harus dimulai dari bidang yang didefinisikan oleh titik **e** serta vektor **u** dan **v**. Dimensi gambar didefinisikan dengan empat angka, yaitu untuk keempat sisi gambar.




Gbr. 3 Vektor dari *frame* kamera [9].

*l* dan *r* adalah posisi ujung kiri dan kanan gambar yang diukur dari **e** sepanjang arah **u**, *b* dan *t* adalah posisi ujung bawah dan atas gambar yang diukur dari **e** sepanjang arah **v**. Umumnya $l < 0 < r$ dan $b < 0 < t$, seperti yang diperlihatkan pada Gbr. 4.




Gbr. 4 Proyeksi Perspektif, pusat sama tapi arah berbeda [9].

Untuk mengepaskan gambar dengan $n_x$ x $n_y$ piksel kedalam kotak berukuran $(r - l)$ x $(t - b)$, maka piksel-piksel akan berjarak sebesar $(r - l)/n_x$ horisontal dan $(t - b)/n_y$ vertikal. Hal ini berarti bahwa piksel pada posisi $(i, j)$ dalam gambar raster mempunyai posisi

$$u = l + (r - l)(i + 0.5)/n_x,$$
$$v = b + (t - b)(j + 0.5)/n_y, \quad (2)$$

dengan $(u, v)$ adalah koordinat dari posisi piksel pada bidang gambar, relatif terhadap pusat **e** dan sumbu {**u, v**}.

Dalam sudut pandang ortografik dapat disederhanakan dengan menggunakan posisi bidang gambar piksel sebagai titik awal berkas, dan seperti yang diketahui bahwa arah berkas adalah arah pandang. Maka prosedur untuk membangkitkan berkas ortografik menjadi

Hitung *u* dan *v* menggunakan persamaan (2)
Arah.berkas ← -dw + *u***u** + *v***v**
Pusat.berkas ← e

*B. Titik Perpotongan Obyek dan Berkas*

Setelah membangkitkan berkas e + *t*d, maka langkah berikutnya adalah mencari titik perpotongan dengan sembarang obyek untuk $t > 0$. Dalam praktek yang menjadi masalah umum adalah mencari titik perpotongan pertama antara berkas dan permukaan obyek yang terjadi pada saat *t* dalam rentang $[t_0, t_1]$. Titik perpotongan dasar adalah saat $t_0 = 0$ dan $t_1 = +\infty$.

Untuk suatu berkas $\mathbf{p}(t) = \mathbf{e} + t\mathbf{d}$ dan suatu permukaan implisit $f(\mathbf{p}) = 0$, maka titik perpotongan akan muncul saat titik berkas memenuhi persamaan implisit, sehingga nilai *t* yang dicari adalah penyelesaian dari persamaan

$$f(\mathbf{p}(t)) = 0 \text{ atau } f(\mathbf{e} + t\mathbf{d}) = 0.$$

Suatu bola dengan pusat $\mathbf{c} = (x_c, y_c, z_c)$ dan jari-jari *R* dapat dinyatakan dengan persamaan implisit

$$(x - x_c)^2 + (y - y_c)^2 + (z - z_c)^2 - R^2 = 0.$$

Persamaan di atas dapat ditulis dalam bentuk vektor

$$(\mathbf{p} - \mathbf{c}) \bullet (\mathbf{p} - \mathbf{c}) - R^2 = 0.$$

Sembarang titik p yang memenuhi persamaan ini berada pada bola. Jika titik pada berkas $\mathbf{p}(t) = \mathbf{e} + t\mathbf{d}$ diisikan dalam persamaan, maka akan diperoleh persamaan dalam *t* yang dipenuhi oleh nilai *t* yang menghasilkan titik pada bola

$$(\mathbf{e} + t\mathbf{d} - \mathbf{c}) \bullet (\mathbf{e} + t\mathbf{d} - \mathbf{c}) - R^2 = 0.$$

Dengan menyusun kembali persamaan di atas akan diperoleh

$$(\mathbf{d} \bullet \mathbf{d})t^2 + 2\mathbf{d} \bullet (\mathbf{e} - \mathbf{c})t + (\mathbf{e} - \mathbf{c}) \bullet (\mathbf{e} - \mathbf{c}) - R^2 = 0.$$

Pada persamaan di atas semua parameter telah diketahui nilainya kecuali *t*, sehingga dapat dipandang sebagai persamaan kuadrat klasik dalam *t* yaitu persamaan ABC yang mempunyai bentuk

$$\boldsymbol{At^2 + Bt + C = 0.}$$

Nilai dari akar persamaan di atas sangat bergantung pada nilai diskriminan yaitu $B^2 - 4AC$. Jika diskriminan negatif maka akar persamaannya adalah imajiner dan garis berkas dan bola tidak berpotongan. Jika diskriminan positif, maka terdapat dua solusi: pertama adalah berkas memasuki bola dan solusi kedua, berkas meninggalkan bola. Jika diskriminan bernilai nol, maka berkas mengenai permukaan bola, tepat menyentuh pada satu titik. Dengan menyusun kembali persamaan di atas maka diperoleh

$$t = \frac{-d.(e-c) \pm \sqrt{(d.(e-c))^2 - (d.d)((e-c).(e-c) - R^2}}{(d.d)}$$

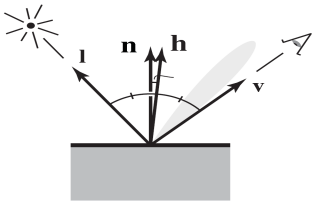
Vektor normal pada titik p diberikan oleh gradien n = 2(p−c). unit normalnya adalah (p − c)/*R*.

### C. *Shading*

Setelah permukaan yang terlihat untuk suatu piksel diketahui, maka nilai piksel dihitung dengan mengevaluasi model *shading*. Model *shading* dirancang untuk menangkap refleksi cahaya, tempat permukaan yang diterangi oleh sumber cahaya dan memantulkan sebagian cahaya ke kamera. Variabel terpenting dalam refleksi cahaya adalah arah sinar l, yang berupa vektor satuan yang menunjuk ke sumber cahaya; arah pengamat v yaitu vektor satuan yang menunjuk ke mata atau kamera; permukaan normal n yaitu vektor satuan yang tegak lurus terhadap permukaan pada titik tempat terjadinya refleksi; dan karakteristik dari permukaan yaitu warna, kilap, atau sifat yang lain bergantung pada model.

*1) Blinn-Phong Shading*

Banyak permukaan riel memperlihatkan beberapa variasi refleksi kilap, menghasilkan sorotan cahaya, atau refleksi *specular* yang muncul dan akan bergerak disekitar dengan berubahnya sudut-pandang. Model yang sangat sederhana dan digunakan secara luas untuk sorotan (*highlight*) *specular* adalah yang diusulkan oleh Phong tahun 1975 yang kemudian diperbaharui oleh Blinn tahun 1976. Idenya adalah menghasilkan refleksi sesuai dengan kecerahan saat v dan *l* diletakkan secara simetris melintasi permukaan normal, yaitu saat refleksi cermin akan muncul, refleksi akan menurun secara halus saat vektor menjauh dari konfigurasi cermin seperti yang diperlihatkan pada Gbr. 5.




Gbr. 5. Geometri untuk Blinn-Phong *sadhing* [9].

Seberapa dekat dengan konfigurasi cermin dapat ditentukan dengan membandingkan vektor separo h (garis bagi sudut antara v dan l) ke permukaan normal. Jika vektor separo dekat dengan permukaan normal, maka komponen *specular* akan menjadi cerah, namun jika semakin jauh maka akan redup. Hasil ini dapat diperoleh dengan menghitung *dot product* antara h dan n, selanjutnya menggunakan hasilnya untuk pangkat $p > 1$. Pangkat atau eksponen Phong mengendalikan kejelasan kilap permukaan.

Blinn-Phong *shading model* didefinisikan sebagai berikut.

$$h = \frac{v+1}{||v+1||}$$

$$L = kd\, I \max(0, \mathbf{n} \cdot \mathbf{l}) + k_s\, I \max(0, \mathbf{n} \cdot \mathbf{h})^p, \qquad (3)$$

dengan $k_s$ adalah *specular coefficient* atau warna *specular* dari permukaan.

*2) Ambient Shading*

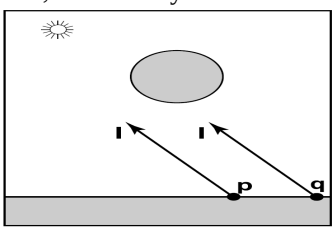
Permukaan yang tidak menerima penerangan sama sekali akan dirender sebagai hitam sempurna yang sering tidak diinginkan. Salah satu cara untuk menghindari bayangan hitam adalah dengan menambah suatu komponen konstanta ke model shading, untuk memberi kontribusi ke warna piksel yang bergantung pada obyek yang dikenai, dan sama sekali tidak bergantung pada geometri permukaan. Metode ini dikenal sebagai *ambient shading*, karena seakan-akan permukaan diterangi oleh cahaya lingkungannya yang datang secara seragam dari mana-mana. Untuk memudahkan dalam menyetel parameter, *ambient shading* biasanya dinyatakan sebagai hasil kali dari warna permukaan dengan warna cahaya lingkungan, sehingga dapat disetel untuk permukaan individual atau untuk seluruh permukaan bersama-sama. Bersama dengan model Blinn-Phong, *ambient shading* melengkapi versi lengkap dari model shading:

$$L = k_a\, I_a + k_d\, I \max(0, \mathbf{n} \cdot \mathbf{l}) + k_s\, I \max(0, \mathbf{n} \cdot \mathbf{h})^n, \qquad (4)$$

Dengan $k_a$ adalah koefisien permukaan lingkungan atau warna lingkungan, dan $I_a$ adalah intensitas cahaya lingkungan.

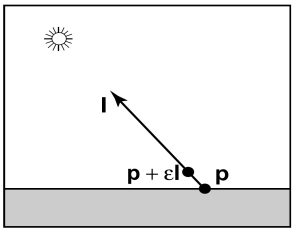
*3) Bayangan*

Setelah menentukan dasar *ray-tracing*, maka bayangan dapat ditambahkan dengan mudah. Bayangkan titik p pada permukaan yang tertutupi, titik akan berada dalam bayangan jika pengamat melihat pada arah l dan melihat obyek. Jika obyek tidak terlihat, maka cahaya tidak diblok.




Gbr. 6. Titik P tidak dalam bayangan sementara titip Q berada dalam bayangan [9].

Pada Gbr. 6 di atas, berkas **p** + *t***l** tidak mengenai obyek sehingga tidak dalam bayangan. Titik q berada dalam bayangan karena berkas **q** + *t***l** mengenai obyek. Vektor l dari kedua titik adalah sama karena sumber cahaya dianggap sangat jauh. Berkas yang menentukan bayangan keluar atau masuk disebut dengan berkas bayangan untuk membedakan dengan berkas terlihat.




Gbr. 7. Pengecekan interval awal mulai pada ε untuk menghindari perpotongan dengan permukaan [9].

Dalam implementasinya berkas bayangan akan mencek kondisi $t \in [\varepsilon, \infty)$, dengan $\varepsilon$ adalah suatu konstanta positif kecil, seperti yang diperlihatkan pada Gbr. 7.
SI

### *D. POVRay*

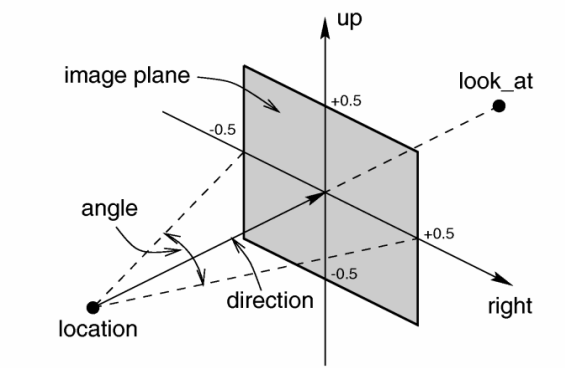
POVRay (*The Persistence of Vision Ray-tracer*) dikembangkan dari DKBTrace 2.12 (yang ditulis oleh David K. Buck dan Aaron A.Collins) oleh sekelompok orang yang membentuk suatu tim yang disebut POV_Team. Informasi tentang POV_Team dan segala sesuatu tentang aplikasi POVRay dapat dilihat di alamat http://www.povray.org [7]. PovRay adalah perangkat lunak untuk *ray-tracing* yang bersifat *open source*, untuk penggambaran grafika komputer 3-D. Terdapat beberapa aplikasi serupa seperti Lightwave Modeler, Rhinoceros 3-D, dan Moray. PovRay tersedia untuk platform komputer PC, Macintosh dan Unix, dan saat ini merupakan salah satu perangkat lunak *ray-tracing* yang banyak digunakan karena relatif mudah digunakan, murah dan mempunyai kualitas tinggi.

POVRay menciptakan gambar foto-realistis 3-dimensi dengan menggunakan teknik rendering yang disebut *ray-tracing*. Suatu file teks yang berisi informasi yang menggambarkan obyek dan pencahayaan dalam suatu adegan akan dibaca dan selanjutnya digunakan untuk membangkitkan gambar adegan tersebut dari sudut pandang kamera. *Ray-tracing* bukanlah algoritma yang dapat diproses dengan cepat, namun dapat menghasilkan citra kualitas tinggi yang realistis dengan efek refleksi, bayangan, perspektif dan efek yang lain.

*Ray-tracing* dimulai dari kamera. Berkas (*rays*) akan dilacak mundur dari kamera ke adegan. Definisi kamera menggambarkan posisi, tipe proyeksi dan bagaimana kamera melihat adegan. Kamera POVRay mempunyai sepuluh model berbeda, yang masing-masing menggunakan metode proyeksi berbeda untuk memproyeksikan adegan ke layar. Untuk mulai menggunakan perangkat lunak *ray-tracing* maka pengguna perlu menentukan posisi pengamat, selanjutnya menentukan dimana letak obyek dan sumber cahaya yang akan terlihat oleh kamera, dan juga menentukan sifat tekstur permukaan obyek, interiornya (jika obyek transparan) dan media yang akan dilalui cahaya seperti kabut, embun atau api. Seperti yang telah dijelaskan sebelumnya bahwa perangkat lunak *ray-tracing* akan melakukan pelacakan terbalik, yaitu dimulai dari piksel yang akan muncul sebagai gambar akhir.

### *E. Proyeksi Perspektif*

Proyeksi perspektif menentukan perspektif kamera yang mensimulasikan bagaimana cahaya yang dipantulkan obyek sampai ke film atau sensor pencitraan pada kamera. Sudut pengamat (horisontal) akan ditentukan oleh rasio antara panjang dari vektor *direction* dan panjang vektor *right*. Sudut pengamat harus lebih besar dari $0^0$ dan lebih kecil dari $180^0$, seperti yang diperlihatkan pada Gbr. 8.



Gbr. 8 Proyeksi Perspektif.

Definisi kamera menggambarkan posisi, tipe proyeksi dan sifat dari bagaimana kamera melihat adegan. Nilai *default* kamera:

DEFAULT CAMERA:
**camera { perspective**
**location <0,0,0>**
**direction <0,0,1>**
**right 1.33*x**
**up y**
**sky <0,1,0> }**

CAMERA TYPE: perspective
**angle : ~67.380 ( direction_length=0.5***
**right_length/tan(angle/2) )**
**confidence : 0.9 (90%)**
**direction : <0,0,1>**
**focal_point: <0,0,0>**
**location : <0,0,0>**
**look_at : z**
**right : 1.33*x**
**sky : <0,1,0>**
**up : y**
**variance : 1/128**

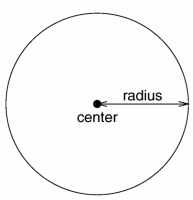
### *F. Obyek pada POVRay*

POVRay mendefinisikan hampir semua obyek secara matematis. Berikut ini akan dibahas beberapa obyek yang digunakan.

#### *1) Bola (Sphere)*

Sintaksis dari obyek bola adalah:
SPHERE:
**sphere { <Center>, Radius**
**[OBJECT_MODIFIERS...] }**

Dengan <Center> adalah vektor yang menentukan koordinat x,y,z dari pusat bola dan Radius berupa bilangan pecahan yang menentukan jari-jari, seperti yang diperlihatkan pada Gbr. 9. Bola dapat diskalakan secara tidak seragam untuk menghasilkan bentuk *elipsoide*.



Gbr. 9. Bola.

Struktur data bola adalah:

**struct Sphere_Struct**
**{ OBJECT_FIELDS VECTOR Center;**
**DBL Radius; }**

*2) Kotak (Box)*

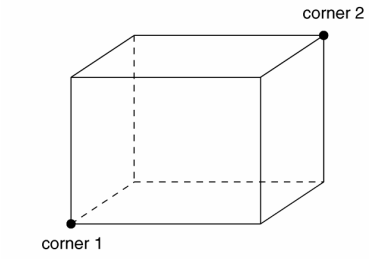
Suatu kotak sederhana dapat didefinisikan dengan daftar dua sudut kotak dengan menggunakan sintaks berikut.

BOX:

**box { <Corner_1>, <Corner_2>**
**[OBJECT_MODIFIERS...] }**

Dengan <Corner_1> dan <Corner_2> adalah vektor yang mendefinisikan koordinat x, y, z dari titik sudut kotak yang berlawanan (*corner* 1 dan *corner* 2), seperti yang diperlihatkan pada Gbr. 10. Sedangkan struktur kotak adalah sebagai berikut.

**struct Box_Struct**
**{ OBJECT_FIELDS VECTOR bounds[2]; }**

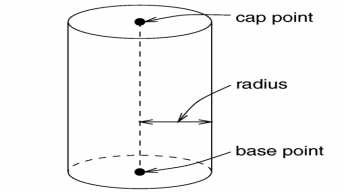


Gbr. 10. Kotak.

*3)Tabung (cylinder)*

Statemen *cylinder* akan menciptakan suatu tabung dengan panjang tertentu dengan kedua tutup paralel seperti yang diperlihatkan pada Gbr. 11. Sintaks untuk tabung adalah

**CYLINDER:**
**Cylinder { <Base_Point>, <Cap_Point>, Radius**
**[ open ][OBJECT_MODIFIERS...] }**



Gbr. 11. Tabung.

Struktur ideal suatu tabung adalah

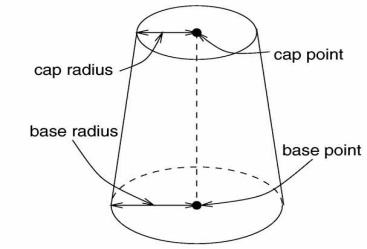
**Base(0,0,0)**
**Apex(0,0,1)**
**Apex_radius=base_radius=1**
**Dist=0**

*4) Kerucut*

Statemen cone akan menciptakan suatu kerucut dengan panjang tertentu, seperti yang diperlihatkan pada Gbr. 12. Sintaksnya adalah

CONE:

**cone { <Base_Point>, Base_Radius, <Cap_Point>,**
**Cap_Radius [ open ] [OBJECT_MODIFIERS...] }**



Gbr. 12. Kerucut (*Cone*).

Struktur kerucut sama seperti yang digunakan pada tabung. Kerucut ideal mempunyai struktur dengan nilai-nilai sebagai berikut.

**Base (0,0,0)**
**Apex (0,0,1)**
**Apex_radius = 1**
**Base_radius = 0**
**Dist = 0**

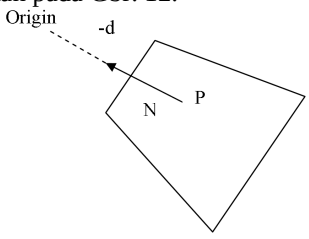
Kerucut yang didefinisikan dalam POV-Ray mempunyai dua nilai jari-jari yang berbeda.

*5) Bidang (Plane)*

Bidang primitif merupakan cara sederhana untuk mendefinisikan permukaan rata yang tidak terbatas. Bidang adalah obyek padat dengan ukuran tak terbatas, yang membagi ruang-POV menjadi dua bagian, di dalam dan di luar bidang. Spesifikasi bidang adalah sebagai berikut.

**PLANE:**
**plane{ <Normal>, Distance**
**[OBJECT_MODIFIERS...] }**

Vektor <normal> mendefinisikan normal permukaan dari bidang. Normal permukaan adalah suatu vektor yang menunjuk ke atas dari permukaan dengan sudut $90^0$, seperti yang diperlihatkan pada Gbr. 12.



Gbr. 12. Bidang (*Plane*)

Struktur bidang adalah sebagai berikut.

**Plane Structure**
**struct Plane_Struct**
**{ OBJECT_FIELDS VECTOR Normal_Vector;**
**DBL Distance; };**

*G. Implementasi*

Pada penelitian ini akan dibuat suatu citra grafis 3-D yang menggambarkan suatu ruang yang di dalamnya terdapat meja bulat dari kaca dengan kaki meja dari batu marbel putih. Di atas meja terdapat cangkir berisi coklat. Di depan meja ada kursi bulat, dan di sebelahnya terdapat hiasan bola yang menyerupai lampu berdiri. Pada dinding belakang terdapat gambar foto yang diberi pigura kayu.

Hampir semua adegan gambar pada dasarnya terdiri dari sumber cahaya, kamera atau mata pengamat, dan obyek. Seperti yang telah dibahas sebelumnya *ray-tracer* seperti POV-Ray bekerja dengan mengirim "berkas cahaya" dari kamera dan mengikuti berkas cahaya tersebut melalui refleksi, refraksi, dan absorbsi sampai berkas tersebut mencapai sumber cahaya atau hilang dalam bayangan.

*1) Kamera*

Langkah pertama adalah menentukan lokasi mata pengamat yang dalam hal ini diwakili oleh kamera. Informasi yang diperlukan oleh kamera adalah posisi kamera terhadap obyek dan posisi titik yang akan dilihat kamera.

```
camera { location <-17, 9, -16>
         look_at <0, 0, 0> }
```

Instruksi di atas dapat diartikan bahwa kamera berada posisi 17 satuan ke kiri, 9 satuan ke atas dan 16 satuan ke depan. Kamera menghadap ke depan pada pusat sumbu.

*2) Sumber cahaya*

Sumber cahaya memerlukan informasi tentang lokasi dan warna, yang dalam hal ini dipilih warna putih. Pada penelitian ini menggunakan dua sumber cahaya dengan posisi yang berbeda.

```
// Posisi Sumber Cahaya
  light_source {  <-7,9,-9>              // sumber cahaya 1
                color rgb <1,1,1>  } // warna cahaya putih
  light_source { <19,19,-10>           // sumber cahaya 2
                color rgb <1,1,1>  } // warna cahaya putih
```

Sumber cahaya pertama berada pada posisi koordinat (-7,9,-9) dan sumber cahaya kedua pada koordinat (19,19, -10).

*3) Perancangan Obyek*

Hal yang paling esensial selanjutnya adalah merancang obyek dalam gambar. Terdapat 5 obyek dalam gambar yang terdiri dari Meja kaca bulat, kursi bundar, hiasan bola dengan penyangga, foto dengan pigura yang menempel di dinding, dan cangkir berisi coklat. Selain itu terdapat dua dinding, yaitu dinding belakang dan dinding samping serta lantai.

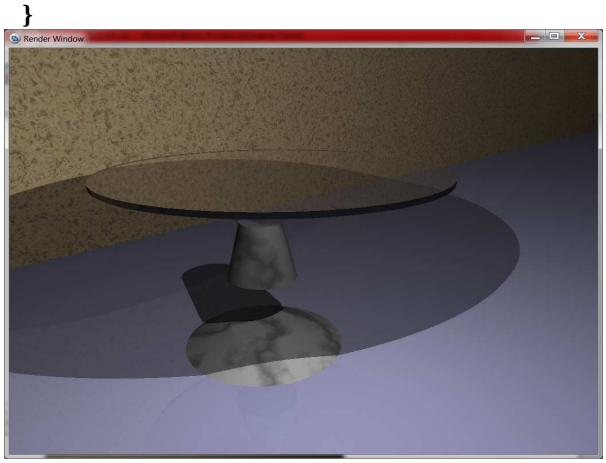
*4) Perancangan Meja*

Obyek meja terdiri dari gabungan dua bagian obyek, yaitu daun meja yang berbentuk bulat dan kaki meja bulat seperti diperlihatkan pada Gbr. 13. Daun meja dibuat dari obyek tabung dengan ketebalan 0,15 dan jari-jari 5 dengan tekstur kaca. Sedangkan kaki meja disusun dari dua buah objek kerucut terpotong dengan tekstur marbel. Sintaksisnya adalah sebagai berikut.

```
#declare daun_meja_bulat =
 cylinder { <0, 0, 0>, <0, .15, 0>, 5
            texture {T_Glass1 scale 3 } }        // tekstur
    kaca
```

Untuk kaki meja terdiri dari gabungan dua buah kerucut terpotong.

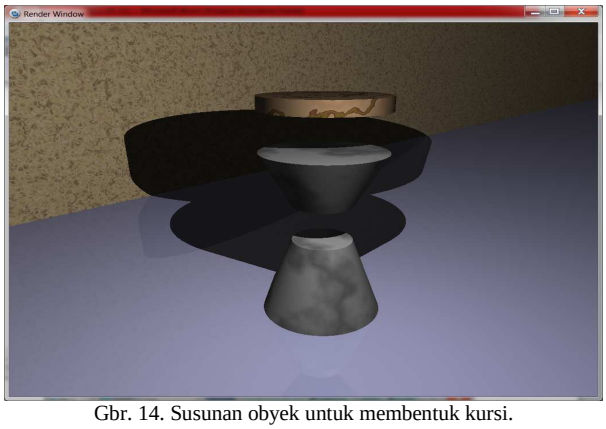
```
#declare kaki_meja =
union
 {  cone {  <0, 1, 0>, 1.7, <0, 4, 0>, 1        // Kaki meja atas
    texture { marble_putih scale 3} }
    cone {  <0,  1,  0>, 1.7, <0,  0,  0>, 4        // Kaki meja
bawah
    texture {marble_putih scale 3}  }
    }
```




Gbr. 13. Susunan obyek primitif yang membentuk meja.

*5) Perancangan Kursi*

Terdiri dari gabungan tiga obyek primitif, yaitu dua obyek kerucut terpotong dan satu obyek tabung, seperti yang diperlihatkan pada Gbr. 14.




Gbr. 14. Susunan obyek untuk membentuk kursi.

Kaki kursi terdiri dari gabungan dua buah kerucut yang saling berlawanan dengan tekstur marbel putih. Sedangkan untuk dudukan kursi dari obyek tabung dengan tekstur White_Wood.

```
#declare kursi_bulat=
 merge
 { cone {  <0, 0, 0>, 2,  <0, 3, 0>, 1        // Kaki kursi
   texture {marble_putih                      // bawah
                scale 3 } }
   cone {  <0, 3, 0>, 1,  <0, 5, 0>, 2        // Kaki kursi
   texture {marble_putih                      // atas
                scale 3 } }
    Cylinder { <0,5,0>,  <0,5.5, 0>, 2        //Dudukan
    texture { White_Wood
                        scale 3 }  }
}
```

*6) Perancangan Hiasan Bola*

Terdiri dari gabungan empat obyek primitif, berurutan dari atas ke bawah yaitu bola, kerucut, tabung dan kerucut seperti yang diperlihatkan pada Gbr. 15.
Bola dibuat dari obyek *Sphere* dengan radius 3.

```
#declare bola_kaca =          // Obyek lampu bentuk
Bola
Sphere {  <0,0,0>, 3
          texture { marble_putih } }
```

Untuk penyangga lampu dibuat dari obyek kerucut dengan jari-jari atas 2,5 dan jari-jari bawah sesuai dengan tiang penyangga yaitu 0,25. Tekstur menggunakan model Gold_Metal.

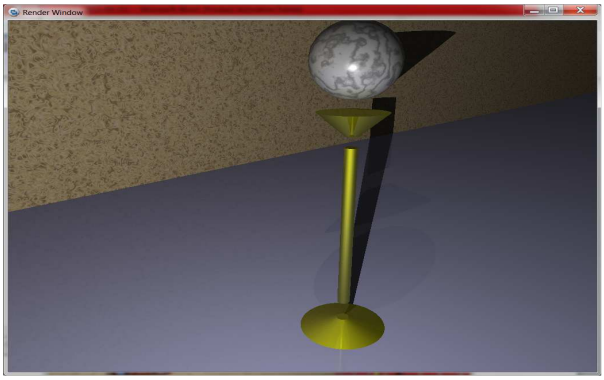
```
cone { <0, 17, 0>, 0.5,   <0, 19, 0>, 2.5
       texture {Gold_Metal  } }
```

Sedang untuk tiang lampu dibuat dari tabung dengan jari-jari 0,5 dan panjang 16. Teksturnya dibuat sama yaitu Gold_metal.

```
cylinder { <0,1,0>, <0,17,0>, 0.5
           texture {Gold_Metal  } }
```

Terakhir untuk alasnya dibuat dari kerucut dengan jari-jari atas sama dengan jari-jari tiang lampu dan jari-jari bawah 3,5.

```
cone { <0, 1, 0>, 0.5,   <0, 0, 0>, 3.5
       texture {Gold_Metal  }}
```



Gbr. 15. Susunan Obyek untuk membentuk hiasan lampu bola.

*7) Perancangan Foto dengan Pigura*

Terdiri dari gabungan tiga obyek primitif, yaitu dua kotak untuk membentuk bingkai pigura dan kotak untuk tempat foto, seperti yang diperlihatkan pada Gbr. 16. Untuk membentuk bingkai pigura dilakukan dengan mencari selisih antara dua kotak yang mempunyai ukuran berbeda. Kotak luar berukuran 10 x 10 dengan tebal 0.7 akan dikurangi dengan kotak dalam yang berukuran 9,2 x 9,2 dengan ketebalan 3. Agar hasilnya berupa bingkai sempurna, maka ukuran kotak dalam (pengurang) harus lebih tebal.

Gbr. 16. Susunan obyek pembentuk foto dan pigura.

Kode programnya adalah sebagai berikut.

```
#declare pigura=
difference
{ box { <0, 0, 0>, <10, 10, 0.7> }     // 10x10x1 Kotak Luar
        box { <0.4, 0.4, -1>, <9.6, 9.6, 2> } // Kotak Dalam
        texture { kayu }  }
```

Untuk gambar foto dilakukan dengan cara menjadikan foto sebagai *image-map* yang selanjutnya dijadikan sebagai struktur pigmen obyek. Dalam hal ini format foto yang digunakan adalah format GIF.

*8) Perancangan Cangkir*

Untuk merancang cangkir yang berisi coklat diperlukan 9 obyek primitif, seperti yang diperlihatkan pada Gbr. 17. Badan cangkir dibentuk dari selisih dua buah tabung dengan ukuran berbeda. Untuk bibir dan alas cangkir disusun masing-masing dari satu dan dua obyek donat (*torus*).

Pegangan cangkir disusun dari obyek donat dan dua obyek tabung. Dan untuk isi minuman coklat disusun dari obyek tabung dengan ukuran lebih kecil dari badan cangkir.

*9) Hasil Gabungan Obyek Gambar*

Setelah bagian demi bagian obyek gambar telah tersusun, selanjutnya digabungkan sehingga membentuk sebuah gambar utuh seperti diperlihatkan pada Gbr. 18. Proses penggabungan dilakukan dengan menggunakan koding yang relatif sederhana dan singkat.

Gbr. 17. Susunan obyek untuk membentuk cangkir.

Obyek hasil *rendering* dengan POV-Ray terlihat cukup realistis. Tekstur permukaan obyek yang dibuat sesuai dengan sifat obyek aslinya serta bayangan yang terjadi karena adanya dua sumber cahaya dapat diimplementasikan dengan baik. Gambar bayangan, pantulan, maupun refraksi cahaya dari dua sumber cahaya yang berseberangan mengenai obyek maupun latar belakang hasil penerapan algoritma *ray-tracing* dapat diimplementasikan dengan baik oleh POV-Ray.

Gbr. 18. Hasil penggabungan obyek membentuk gambar utuh hasil rendering dengan POV-Ray

## III. KESIMPULAN

Kesimpulan Dari hasil perancangan dan implementasi obyek dengan menggunakan POV-Ray, dapat ditarik suatu kesimpulan:

1. Algoritma *ray-tracing* untuk menghitung dan membentuk pola pantulan, bayangan, maupun refraksi dapat diimplementasikan dengan baik.
2. Obyek dengan berbagai bentuk maupun ukuran dapat dengan mudah disusun dengan menggunakan instruksi yang disediakan oleh POV-Ray.
3. Tekstur dan sifat pigmen dari masing-masing permukaan obyek terhadap sumber cahaya dapat diimplementasikan dengan baik oleh POV-Ray, sehingga gambar terlihat lebih realistis.

## REFERENSI


[1] D. P. Cleveland, "*A 3-Dimensional Geometric Modeler Using OpenGL*," Department of Computer Science and Engineering Auburn University, Alabama, 1996.

[2] D. Shklyar. [Online]. Available: http://www.cgsociety.org/index.php/cgsfeatures/cgsfeaturespecial/3d_rendering_history_pt.2_-_to_photorealism_and_beyond. [*Accessed 30 May 2012*].

[3] M. K. Johnson, K. Dale, S. Avidan, H. Pfister, W. T. Freeman and W. Matusik, "*CG2Real: Improving the realism of Computer Generated Images using a Large Collection of Photograps*," IEEE Transactions on Visualization and Computer Graphics, vol. 17, no. 9, September 2011.

[4] R. Olsson and Y. Xu, "*An Interactive Ray-tracing based simulation environment for generating integral imaging video sequences*," Three-Dimensional TV, Video and Display, vol. IV, 2005.

[5] A. Ben, Artzi, R. Ramamoorthi and M. Agrawala, "*Efficient complex shadows from environments maps*," ACM SIGGRAPH, vol. 4, 2004.

[6] TusharUdeshi and C. D. Hansen, "*Towards Interactive Photorealistic Rendering of Indoor Scenes: A Hybrid Approach*," in 10th Eurographics Workshop on Rendering, 1999.

[7] POV-Team, ""POV-Ray ™ Version 3.1, *User's Documentation*," 1999. [Online]. Available: http://POV.org.

[8] B. Feriel, S. Simon and W. Dongming, "*Ray Tracing*," in ING4 SI International, 2011/2012., 2012.

[9] S. Peter and S. Marschner, "*Fundamentals of Computer Graphics*", Third Edition, Massachusetts: A K Peters, Natick, Massachusetts, 2009.