Analisis Pendekatan Depth-First Search dan Algoritma Runut-balik dalam Path Tracing

Mikhael Andrian Yonatan - 13524051
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: mikhael.yonatan10@gmail.com , 13524051@std.stei.itb.ac.id

Abstrak—Grafika komputer dan simulasi pencahayaan realistis telah menjadi bidang yang berkembang dengan sangat pesat dan terus menuntut teknik komputasi yang semakin tinggi. Dalam perkembangannya, teknik *rendering* telah mencapai *path tracing*, yaitu sebuah metode yang mampu menghasilkan efek pencahayaan secara jauh lebih akurat. Namun, teknik ini dihadapkan dengan masalah performa (eksponensial). Penelitian ini bertujuan untuk menganalisis penerapan algoritma *Depth-First Search* (DFS) dan runut-balik (*backtracking*) sebagai mekanisme utama penelurusan sinar dalam *path tracing*, serta penggunaan teknik Monte Carlo sebagai strategi algoritma untuk mereduksi kompleksitas waktu dari eksponensial menjadi linear. Pada penelitian ini metode yang digunakan adalah analisis algoritma yang didukung oleh implementasi program sederhana. Hasil dari penelitian ini membuktikan bahwa pendekatan DFS dan runut-balik adalah fondasi dari *path tracing*, serta pengambilan sampel acak adalah kunci dalam menyeimbangkan kualitas visual dengan efisiensi performa.

Kata Kunci—*Depth-First Search*, grafika komputer, Monte Carlo, *path tracing*, runut-balik.

I. PENDAHULUAN

Grafika komputer merupakan proses menghasilkan gambar digital dari data maupun deskripsi matematis. Pada masa kini, grafika komputer memiliki peran yang begitu besar dalam kehidupan manusia. Berbagai produk digital yang mudah dijumpai, seperti seperti *video game*, film animasi (CGI), hingga visualisasi arsitektur, merupakan implementasi dari grafika komputer.

Teknologi *rendering* adalah salah satu cabang kajian dalam grafika komputer. *Rendering* adalah proses menghasilkan gambar (2D) dari suatu data non gambar (model 3D). Proses ini secara kompleks memperhitungkan tekstur, pencahayaan, bayangan, dan informasi geometris lainnya. Kualitas visual yang dihasilkan sangat bergantung pada algoritma dan model matematis yang digunakan.

Seiring dengan berkembangnya tenaga komputasi, metode *rendering* mulai bertransisi menuju simulasi pencahayaan berbasis fisika yang lebih komprehensif dan realistis, yaitu *path tracing*. Berbeda dengan *ray tracing* klasik (*whitted ray tracing)* yang hanya memperhitungkan jenis lintasan cahaya tertentu, *path tracing* berupaya memecahkan *Rendering Equation* untuk menyimulasikan *global illumination* dengan pencahayaan tidak langsung *(indirect lighting*) secara stokastik.

Gambar 1. Perbandingan gambar hasil *path tracing*, diambil dari [7]

Dari segi algoritma, DFS dan runut-balik (*backtracking*) adalah fondasi dari implementasi *path tracing*. Setiap piksel akan memiliki pohon pencariannya masing-masing yang disusun dari lintasan setiap sinar secara DFS. Ketika ujung lintasan sinar mencapai sumber emisi cahaya atau batas kedalaman maksimum yang, algoritma runut-balik berperan dalam mengakumulasikan warna akhir.

Memecahkan *Rendering Equation* secara langsung sangatlah berat secara komputasi karena perlu mengkalkulasikan persebaran sinar cahaya ke segala arah, lalu pada setiap titik tumbukan objek akan dilakukan hal yang sama sehingga perhitungannya sangatlah kompleks dan menciptakan pohon pencarian yang sangat masif dengan kompleksitas eksponensial.

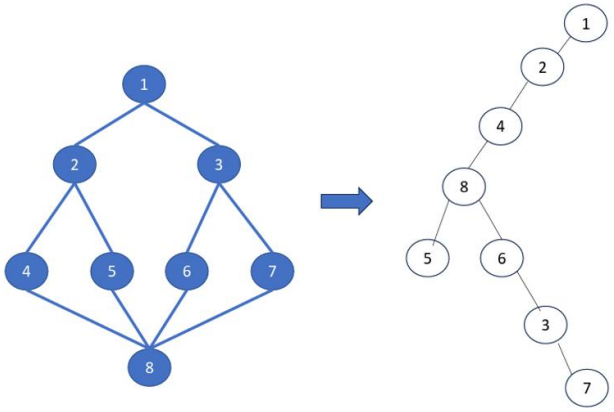
Salah satu solusi untuk mengatasinya adalah menerapkan penelusuran DFS yang didasarkan pada teknik Monte Carlo. Dibanding mengevaluasi seluruh kemungkinan lintasan sinar, algoritma ini hanya menembakkan dan mengevaluasi sejumlah N sampel sinar ke dalam satu area piksel. Setiap sampel sinar akan membentuk tepat satu lintasan acak saja dan warna akhir dari setiap piksel akan diperoleh dengan merata-ratakan nilai warna yang dibawa oleh setiap sampel. Pendekatan ini secara drastis memotong kompleksitas algoritma ruang dan waktu menjadi jauh lebih baik.

Meskipun konsep *path tracing* telah banyak diterapkan di industri film dan *video game* modern, pemahaman mengenai bagaimana logika algoritma dalam merepresentasikan sifat dan perhitungan fisikanya masih jarang dibahas secara struktural. Berdasarkan hal tersebut, makalah ini bertujuan untuk membahas penerapan algoritma *Depth-First Search* (DFS) dan runut-balik (*backtracking*) sebagai fondasi dari implementasi *path tracing*, serta mengevaluasi bagaimana teknik probabilistik Monte Carlo menyeimbangkan kualitas visual dan performa komputasi. Dengan demikian, diharapkan pemahaman terhadap mekanisme dan efisiensi dalam komputasi grafis dapat dipahami dengan lebih utuh.

II. LANDASAN TEORI

A. Depth-First Search dan Depth-Limited Search

Depth-First Search (DFS) merupakan metode penelusuran struktur data graf atau pohon dengan prinsip utamanya adalah perluasan simpul dilakukan secara mendalam terlebih dahulu. Algoritma ini akan terus bergerak menuju simpul anak (*child node*) terdalam di sepanjang salah satu cabang tunggal sebelum menelusuri cabang atau simpul tetangga lainnya. Secara komputasional, DFS umumnya diimplementasikan menggunakan pendekatan rekursif. Fungsi DFS akan merekam perjalanannya dalam *stack* sehingga dapat melakukan *backtrack* ke simpul sebelumnya ketika jalur saat ini telah selesai ditelusuri.



Gambar 2. Penelusuran secara DFS, diambil dari [8]

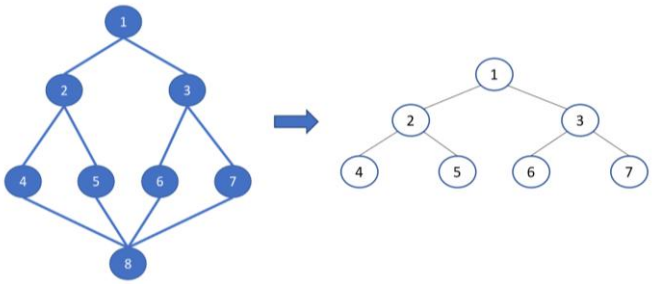
Langkah-langkah penelusuran DFS adalah sebagai berikut:

1. Kunjungi simpul *v*.
2. Jika *v* merupakan simpul tujuan, maka pencarian selesai.
3. Jika *v* bukan simpul tujuan, kunjungi simpul *w* yang bertetangga dengan simpul *v* dan panggil DFS untuk setiap *w*.
4. Ketika mencapai simpul *u* sedemikian sehingga seluruh simpul yang bertetangga dengannya telah dikunjungi, *backtrack* ke simpul terakhir yang dikunjungi sebelumnya dan masih mempunyai simpul bertetangga yang belum dikunjungi.
5. Pencarian berakhir apabila tidak ada lagi simpul yang belum dikunjungi.

Apabila langkah-langkah tersebut diterapkan, maka urutan penelusuran yang diperoleh pada Gambar 2 adalah 1, 2, 4, 8, 5, 6, 3, 7.

DFS memiliki kompleksitas ruang yang sangat efisien, yaitu $O(bm)$ untuk pohon berukuran terbatas. Namun, memiliki kompleksitas waktu $O(b^m)$, dengan b adalah maksimum percabangan dan m adalah kedalaman maksimum ruang pencarian. Algoritma ini memiliki kelemahan besar apabila dihadapkan pada ruang pencarian yang sangat besar, memiliki siklus, atau bahkan memiliki kedalaman yang tidak terhingga. Apabila graf tidak memiliki batas akhir atau pohon sangat dalam, DFS akan terus mengeksekusi pemanggilan rekursif tanpa henti. Jika kedalamannya tidak terbatas dan tidak ada penanganan *redundant paths*, maka algoritma ini tidak menjamin sifat *completeness* dan pada akhirnya akan mengalami kegagalan sistem akibat kehabisan memori (*stack overflow*).

Untuk menutupi kelemahan pada ruang pencarian yang tak terhingga, diperkenalkan varian algoritma yang disebut *Depth-Limited Search* (DLS). DLS mempertahankan mekanisme penelusuran mendalam milik DFS, namun memberikan batas kedalaman maksimum (l). Dengan membatasi penelusuran hanya sedalam l, kompleksitas ruang DLS menjadi $O(bl)$ dan kompleksitas waktunya menjadi $O(b^l)$. DLS tetap tidak menjamin *completeness* apabila solusinya berada di luar batas ($d > l$), namun menjamin diperolehnya suatu keputusan (berhenti) dibanding program terjebak dalam jalur penelusuran yang memakan waktu komputasi tanpa batas.



Gambar 3. Penelusuran secara DLS, diambil dari [8]

Langkah-langkah penelusuran DFS adalah sebagai berikut:

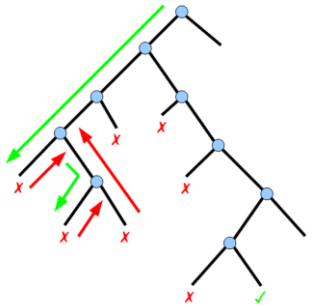
1. Kunjungi simpul *v*.
2. Jika *v* merupakan simpul tujuan, maka pencarian selesai.
3. Jika kedalaman *v* telah melebihi batas, maka pencarian pada cabang tersebut dihentikan.
4. Jika *v* bukan simpul tujuan dan kedalamannya belum melebihi batas, kunjungi simpul *w* yang bertetangga dengan simpul *v* dan panggil DFS untuk setiap *w*.
5. Pencarian berakhir apabila tidak ada lagi simpul sampai dengan batas kedalaman maksimum yang belum dikunjungi.

Apabila langkah-langkah tersebut diterapkan, maka urutan penelusuran yang diperoleh pada Gambar 2 dengan batas kedalamannya 2 adalah 1, 2, 4, 5, 3, 6, 7.

B. Algoritma Runut-balik (Backtracking)

Algoritma runut-balik (*backtracking*) adalah algoritma pencarian berbasis struktur data pohon ruang status (*state space tree*) yang memanfaatkan algoritma traversal DFS. Berbeda dengan DFS *exhaustive* yang akan menelusuri seluruh kemungkinan, algoritma *backtracking* menerapkan mekanisme pemangkasan (*pruning*). Algoritma ini tidak lagi fokus untuk menelusuri seluruh simpul, sebaliknya akan dilakukan evaluasi terlebih dahulu sebelum ditelusuri lebih jauh. Apabila suatu cabang dipastikan tidak akan mengarah pada solusi yang valid, maka penelusuran cabang tersebut akan dihentikan.

Algoritma *backtracking* sangat baik dalam memecahkan persoalan non-optimasi, yaitu persoalan keputusan "ya" atau "tidak". Sesuai dengan penelusuran secara DFS, setiap simpul akan dievaluasi terlebih dahulu menggunakan fungsi pembatas. Sementara, untuk persoalan optimasi, algoritma *backtracking* perlu dimodifikasi sedemikian rupa terlebih dahulu sehingga lebih umum menggunakan algoritma *Branch and Bound* (tidak dibahas dalam makalah ini).



Gambar 4. Penerapan *backtracking*, diambil dari [8]

Algoritma *backtracking* memiliki beberapa properti sebagai berikut,

1. Solusi Persoalan: Solusi akhir dinyatakan dalam bentuk vektor $X = (x_1, x_2, ..., x_n)$ dengan x_i mewakili komponen keputusan yang dipilih dari himpunan keputusan yang valid.
2. Fungsi Pembangkit: Fungsi yang bertugas membangkitkan nilai-nilai kandidat yang mungkin diekspansi untuk komponen x_k pada langkah ke-k.
3. Fungsi Pembatas (*Bounding Function*): Predikat bernilai *boolean* untuk mengevaluasi calon lintasan solusi. Jika bernilai *true*, maka *valid* mengarah ke solusi dan penelusuran dilanjutkan. Namun, jika bernilai *false*, maka simpul tersebut dimatikan (penelusuran tidak dilanjutkan) dan akan langsung dilakukan runut-balik ke simpul induk terdekat. Hasilnya simpul anak dari simpul yang dimatikan tidak dibangkitkan sehingga menghemat komputasi.

C. Rendering Equation

Dalam grafika komputer, salah satu permasalahan utama yang ingin diselesaikan adalah menentukan besarnya cahaya yang diterima kamera dari setiap titik pada objek. Untuk menghasilkan gambar yang realistis, maka tidak sesederhana menghitung cahaya yang dipancarkan oleh sumber cahaya saja karena suatu permukaan juga dapat menerima pantulan cahaya dari permukaan lain. Akibatnya, warna akhir dari suatu titik dipengaruhi oleh interaksi cahaya yang terjadi di seluruh *scene* (*global illumination*) yan bertumpu pada hukum fisika untuk menyimulasikan *transport* cahaya.

Untuk memodelkan fenomena tersebut, diperkenalkan *Rendering Equation* pada tahun 1986. Persamaan ini mendeskripsikan berapa banyak total energi cahaya yang dipancarkan dari suatu titik di permukaan objek menuju arah tertentu (seperti mata pengamat atau kamera), yaitu penjumlahan antara cahaya yang dipancarkan oleh titik itu sendiri dan akumulasi cahaya pantulan yang diterima dari lingkungan sekitarnya. Formulasinya adalah sebagai berikut,

$$L_o(x, \omega_o) = L_e(x, \omega_o) + \int_\Omega f_r(x, \omega_i, \omega_o) L_i(x, \omega_i) \, (\omega_i \cdot n) \, d\omega_i$$

dengan keterangan,

$L_o(x, \omega_o)$	Radiansi keluar, yaitu total cahaya yang meninggalkan titik x menuju ω_o
$L_e(x, \omega_o)$	Radiansi emisi, yaitu cahaya yang dipancarkan langsung oleh titik x
$L_i(x, \omega_i)$	Radiansi masuk, yaitu cahaya yang tiba di titik x dari arah masuk ω_i
$f_r(x, \omega_i, \omega_o)$	*Bidirectional Reflectance Distribution Function (BRDF)*, yaitu fungsi probabilitas yang mengatur bagaimana material memantulkan cahaya
$\int_\Omega$	Mengevaluasi seluruh arah cahaya yang datang melewati bidang setengah bola yang melingkup titik x
$(\omega_i \cdot n)$	Faktor kosinus sudut datang cahaya berdasarkan arah masuk (ω_i) terhadap garis normal permukaan (n).

Meskipun mampu memodelkan *transport* cahaya secara umum dan realistis, *Rendering Equation* memiliki tingkat kompleksitas yang sangat tinggi. Hal ini disebabkan oleh adanya operasi integral pada seluruh arah datang cahaya sehingga jumlah kemungkinan lintasan cahaya yang harus dipertimbangkan sangat masif. Selain itu, komponen $L_i(x, \omega_i)$ bergantung pada *Rendering Equation* itu sendiri sehingga membentuk hubungan rekursif.

Integral berdimensi tinggi dan rekursif menyebabkan *Rendering Equation* sulit diselesaikan secara eksak dengan pendekatan deterministik. Oleh karena itu, dibutuhkan suatu metode yang dapat mengestimasi nilai integral secara efisien. Salah satu pendekatan yang paling banyak digunakan adalah teknik Monte Carlo.

D. Teknik Monte Carlo

Dalam ilmu komputasi, terdapat banyak persoalan matematis yang melibatkan kalkulasi integral yang sangat kompleks, terutama ketika berdimensi tinggi. Ketika suatu fungsi integral terlalu rumit untuk diselesaikan secara konvensional, maka dibutuhkan suatu pendekatan stokastik. Teknik Monte Carlo adalah salah satu cara yang digunakan untuk mengestimasi nilai dari sebuah integral tentu dengan memanfaatkan prinsip pengambilan sampel acak (*random sampling*) dan probabilitas statistik.

Misalkan terdapat sebuah integral dari suatu fungsi $f(x)$ pada suatu domain kontinu,

$$F = \int_\Omega f(x) \, dx$$

dengan Ω adalah domain integral (seluruh kemungkinan sampel). Metode Monte Carlo tidak menghitung nilai integral tersebut secara eksak, melainkan menghampirinya dengan mengevaluasi pada sejumlah sampel acak $x_1, x_2, ..., x_N$ yang diperoleh dari suatu distribusi probabilitas. Estimator Monte Carlo dapat dirumuskan sebagai berikut,

$$F_N = \frac{1}{N} \sum_{i=1}^{N} \frac{f(x_i)}{p(x_i)}$$

Pada persamaan persamaan tersebut, F_N merupakan nilai estimasi integral yang dicari, N adalah jumlah sampel acak yang dibangkitkan, x_i adalah titik sampel acak ke-i, dan $p(x_i)$ mewakili Fungsi Kerapatan Probabilitas (*Probability Density*

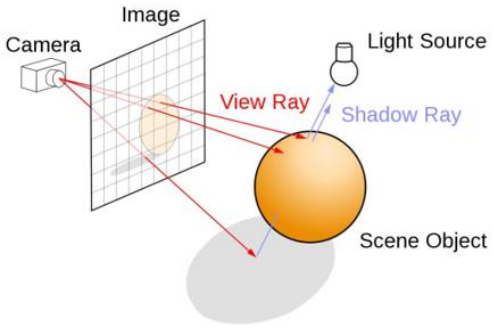
Function/PDF) yang menentukan peluang terpilihnya sampel x_i. Dengan formulasi ini, perhitungan integral kontinu tidak lagi dilakukan secara eksak, melainkan diestimasi melalui rata-rata dari sejumlah sampel acak. Pendekatan ini memungkinkan persoalan integral yang kompleks untuk dihitung secara efisien melalui operasi penjumlahan diskrit yang dijalankan secara iteratif. Dalam implementasi algoritma, notasi sigma (∑) pada estimator Monte Carlo dapat direalisasikan menggunakan struktur perulangan (*loop*) sehingga persoalan integral yang sulit dihitung secara langsung menjadi lebih mudah diimplementasikan dan dieksekusi oleh komputer.

Teknik Monte Carlo berdasar pada Hukum Bilangan Besar (*Law of Large Numbers*). Hukum ini menyatakan bahwa seiring dengan bertambahnya jumlah sampel N mendekati tak terhingga, nilai ekspetasi dari estimator F_N akan secara pasti menuju nilai integral yang sebenarnya. Meskipun metode ini akan selalu memiliki *error* atau varians akibat sifat pengambilannya yang acak, tingkat kesalahan tersebut dijamin akan terus menyusut seiring dengan bertambahnya jumlah sampel dengan laju sebesar $O(1/\sqrt{N})$. Jadi, dapat disimpulkan bahwa semakin banyak jumlah sampel yang digunakan, semakin baik pula estimasi yang dihasilkan oleh teknik Monte Carlo.

III. PEMBAHASAN

A. Algoritma dan Keterbatasan Ray Tracing

Ray tracing merupakan salah satu teknik *rendering* yang menyimulasikan *transport* cahaya dengan menelusuri lintasannya secara terbalik, yaitu dari kamera menuju objek-objek pada suatu *scene*. Pada metode ini, setiap piksel pada bidang gambar akan menghasilkan satu atau beberapa sinar (*ray*) yang ditembakkan dari posisi kamera. Sinar tersebut kemudian dievaluasi untuk menentukan titik perpotongan pertama. Informasi geometris dan material pada titik tumbukan akan digunakan untuk menentukan warna yang akan ditampilkan pada piksel yang bersangkutan.

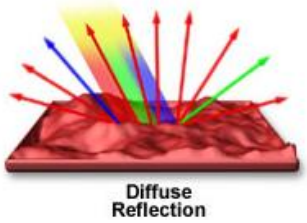


Gambar 5. Algoritma *ray tracing*, diambil dari [1]

Proses *ray tracing* diawali dengan pembangkitan *view ray* dari kamera menuju suatu piksel. Setelah titik perpotongan pertama ditemukan, akan dilakukan evaluasi terhadap sifat material pada titik tersebut. Jika objek bersifat reflektif, maka *reflection ray* akan dibangkitkan mengikuti hukum refleksi. Jika objek bersifat transparan, maka akan dibangkitkan *refraction ray* mengikuti hukum pembiasan. Proses ini dapat dilakukan secara rekursif hingga mencapai batas kedalaman tertentu, sinar tidak lagi berinteraksi dengan objek, atau sinar mencapai sumber cahaya. Warna akhir suatu piksel diperoleh dengan mengakumulasikan kontribusi warna dari seluruh sinar yang terlibat.

Ray tracing klasik bersifat deterministik, yaitu arah pantulan dan pembiasan cahaya ditentukan secara pasti berdasarkan hukum fisika dan parameter materialnya sehingga apabila arah datang sinar dan sifat permukaan diketahui, maka arah sinar berikutnya dapat langsung ditentukan secara eksak. Karakteristik ini memungkinkan *ray tracing* menghasilkan efek refleksi dan refraksi yang realistis.

Meskipun demikian, *ray tracing* klasik memiliki keterbatasan dalam *transport* cahaya secara menyeluruh. Interaksi cahaya yang bersifat *diffuse* dan pencahayaan tidak langsung akan diabaikan (dimatikan) karena kompleksitas komputasinya yang sangat berat.



Gambar 6. *Diffuse reflection*, diambil dari [11]

Akibatnya, *ray tracing* tidak mampu memperhitungkan akumulasi cahaya pantulan yang berasal dari lingkungan sekitarnya sebelum akhirnya mencapai kamera karena jalur-jalur cahaya tersebut memang tidak pernah dibangkitkan. Padahal, dalam dunia nyata, sebagian besar pencahayaan yang diterima suatu objek berasal dari rangkaian pantulan tidak langsung tersebut. Keterbatasan inilah yang kemudian mendorong pengembangan *path tracing*, yaitu teknik *rendering* berbasis probabilitas yang berupaya mengevaluasi kontribusi cahaya dari berbagai arah untuk mendekati solusi *Rendering Equation*.

B. Dasar Path Tracing

Berbeda dengan *ray tracing* yang umumnya hanya mengevaluasi jalur cahaya tertentu yang bersifat deterministik, *path tracing* merepresentasikan *transport* cahaya secara lebih menyeluruh dengan mempertimbangkan berbagai kemungkinan lintasan cahaya yang dapat terjadi pada suatu *scene*. Hal ini berarti, interaksi cahaya yang bersifat *diffuse* akan dibangkitkan dan diteruskan. Dengan demikian, metode ini mampu mewujudkan *global illumination*.

Path tracing berusaha untuk memenuhi *Rendering Equation* yang telah dibahas pada bagian sebelumnya. Pernyataan tersebut menyatakan bahwa cahaya yang keluar dari suatu titik dipengaruhi oleh cahaya yang dipancarkan oleh titik itu sendiri dan akumulasi cahaya yang datang dari lingkungannya. Dengan kata lain, untuk menentukan warna suatu titik dibutuhkan evaluasi terhadap seluruh kemungkina arah datang cahaya beserta interaksi yang terjadi sebelum cahaya tersebut sampai. Proses ini bersifat rekursif dan dapat terjadi berulang kali sehingga ruang pencariannya sangat besar.

Apabila seluruh kemungkinan arah cahaya dievaluasi secara eksplisit, maka jumlah lintasan yang harus dihitung akan bertumbuh secara eksponensial terhadap kedalaman pantulan. Kompleksitas tersebut menjadi penyelesaian *Rendering Equation* secara eksak sangat berat sehingga tidak praktis. Oleh

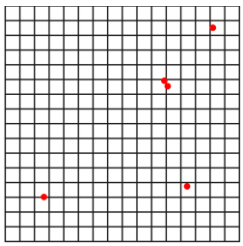
karena itu, *path tracing* memanfaatkan teknik Monte Carlo untuk mengestimasi solusi *Rendering Equation*.

Dibanding mengevaluasi seluruh kemungkinan arah cahaya, teknik Monte Carlo hanya memilih sejumlah sampel arah secara acak berdasarkan distribusi probabilitas tertentu. Setiap sampel kemudian ditelusuri sebagai satu lintasan cahaya (*path*) yang merepresentasikan sebagian kecil dari keseluruhan ruang kemungkinan yang ada. Meskipun hanya mengevaluasi sebagian kecil lintasan, hasil estimasi yang diperoleh akan semakin mendekati solusi sebenarnya seiring bertambahnya jumlah sampel yang digunakan. Melalui pendekatan ini, *path tracing* mampu menghasilkan simulasi pencahayaan yang lebih realistis dengan kompleksitas komputasi yang lebih efisien.

C. Algoritma Path Tracing

Secara umum, algoritma *path tracing* sangat mirip dengan *ray tracing*. *Path tracing* bertujuan untuk menentukan warna akhir setiap piksel dengan fokus utama mengestimasi solusi *Rendering Equation* menggunakan teknik Monte Carlo. Berbeda dengan *ray tracing* klasik yang hanya mengevaluasi jalur cahaya tertentu, *path tracing* akan mengizinkan sinar untuk terus dipantulkan pada berbagai jenis permukaan sehingga mampu merepresentasikan pencahayaan tidak langsung dengan lebih realistis.

Path tracing akan diawali dengan melakukan *pixel sampling*. Pada tahap ini, setiap piksel dipandang sebagai suatu area dua dimensi dan bukan sebagai satu titik tunggal. Sejumlah sampel acak dibangkitkan pada koordinat kontinu *(x, y)* yang berbeda di dalam area piksel tersebut.



Gambar 7. *Pixel sampling*, diambil dari [12]

Setiap sampel kemudian digunakan untuk membangkitkan sebuah *view ray* yang ditembakkan dari kamera menuju *scene*. Misalnya, pada Gambar 7, sebuah piksel dibagi menjadi area 16×16 (untuk visualisasi saja, koordinat pembangkitan adalah kontinu), lalu dibangkitkan 5 buah *ray* (titik merah). Penggunaan banyak sampel bertujuan untuk mengurangi *aliasing* dan meningkatkan akurasi warna piksel.

Setelah *view ray* dibangkitkan, untuk setiap *ray*, algoritma akan mencari titik perpotongan pertama antara sinar dengan objek pada *scene*. Apabila tidak ditemukan objek yang dipotong, maka *ray* akan mengembalikan warna latar belakang. Sebaliknya, apabila ditemukan titik tumbukan, maka informasi posisi, normal permukaan, dan sifat material pada titik tersebut akan digunakan untuk menentukan perilaku sinar selanjutnya.

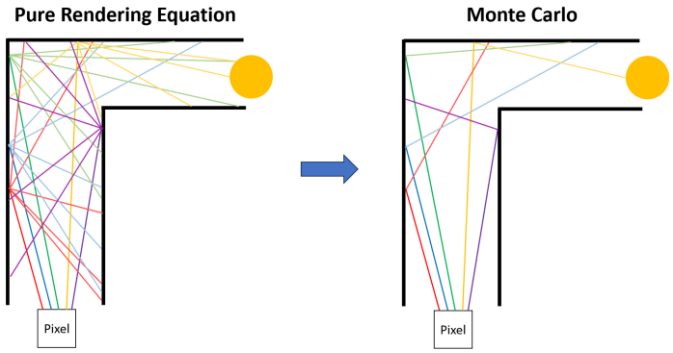
Berbeda dengan *ray tracing* klasik yang umumnya hanya mengikuti jalur refleksi dan pembiasan yang bersifat deterministik, *path tracing* melakukan *ray sampling* menggunakan teknik Monte Carlo. Pada setiap titik tumbukan, algoritma akan memilih satu arah datang cahaya secara acak berdasarkan distribusi probabilitas tertentu (menangani interaksi cahaya bersifat *diffuse*). Namun, penentuan arah tetap sesuai dengan hukum fisika sehingga untuk kasus reflektif akan terdapat arah dengan probabilitas 1, sementara arah lainnya 0 sehingga arah pantulan tetap bersifat deterministik, hal tersebut juga berlaku untuk refraksi. Arah yang dipilih kemudian digunakan untuk membangkitkan *ray* baru yang akan ditelusuri menuju titik tumbukan berikutnya. Proses ini dilakukan secara rekursif sehingga terbentuk suatu lintasan cahaya (*path*) yang terdiri atas rangkaian titik tumbukan dan arah pantulan.

Penelusuran lintasan akan berhenti apabila *ray* mencapai sumber cahaya, tidak lagi berinteraksi dengan objek lain, atau telah mencapai batas kedalaman maksimum yang ditentukan. Setelah kondisi terminasi tercapai, kontribusi cahaya yang diperoleh pada titik terdalam akan dikembalikan secara bertahap menuju titik-titik sebelumnya hingga akhirnya mencapai kamera. Mekanisme ini menghasilkan akumulasi kontribusi cahaya dari seluruh lintasan yang telah ditelusuri.

Tahap terakhir adalah menggabungkan hasil seluruh sampel pada piksel yang sama. Warna akhir suatu piksel diperoleh dengan menghitung rata-rata kontribusi cahaya dari seluruh *view ray* yang dibangkitkan pada area piksel tersebut.

D. Analisis DFS dan Backtracking

Untuk memperjelas analisis DFS dan *backtracking*, akan digunakan contoh berikut. Terdapat sebuah lorong dengan posisi sumber cahaya (lingkaran kuning) dan kamera (piksel) seperti pada gambar berikut,

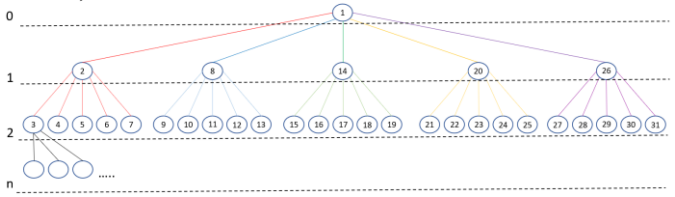


Gambar 8. *Contoh kasus*, diambil dari [12]

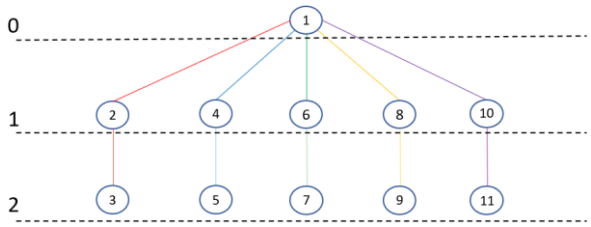
Rendering Equation direpresentasikan pada sisi kiri Gambar 8, gambar tersebut adalah bentuk penyederhanaan (bukan berarti hanya ada tepat 5 sampel ray) dengan tujuan menggambarkan *Rendering Equation* menembakkan tak hingga *ray* pada satu piksel dan untuk setiap *ray* akan dipantulkan kembali ke segala arah menjadi tak hingga *ray* lagi. Apabila seluruh kemungkinan arah dievaluasi secara eksplisit, maka jumlah lintasan yang harus dihitung akan bertumbuh secara eksponensial terhadap jumlah pantulan sehingga tidak praktis untuk diterapkan pada proses *rendering*.

Pada *path tracing* skenario atau pendekatan yang digunakan direpresentasikan sisi kanan Gambar 8, yaitu dengan optimasi teknik Monte Carlo. Pendekatan ini melakukan *pixel sampling*, di mana hanya menembakkan *N* buah sampel *ray* saja dan untuk setiap *ray* juga hanya dipilih satu arah pantulan yang akan dieksplorasi pada setiap titik tumbukan. Dengan demikian, satu sampel *ray* hanya akan menghasilkan satu lintasan linear yang terdiri atas rangkaian titik tumbukan dari kamera hingga kondisi terminasi tercapai.

Seperti yang sudah disebutkan sebelumnya bahwa tujuan kita adalah mencari warna akhir untuk setiap piksel, maka setiap piksel akan memiliki ruang pencariannya masing-masing. Pada setiap piksel akan dibangkitkan *N* sampel *ray* yang masing-masing membentuk lintasan cahaya tersendiri. Penelusuran lintasan dilakukan secara *Depth-First Search* (DFS), yaitu menelusuri satu lintasan hingga mencapai kondisi terminasi sebelum berpindah mengevaluasi lintasan lainnya. Pohon pencarian untuk kasus di Gambar 8 adalah sebagai berikut,



Gambar 9. Pohon pencarian *Pure Rendering Equation*, diambil dari [12]



Gambar 10. Pohon pencarian Monte Carlo, diambil dari [12]

Selain menerapkan penelusuran DFS, *path tracing* juga mengadopsi algoritma *backtracking*. Dalam prosesnya, setiap lintasan cahaya ditelusuri terlebih dahulu hingga mencapai kondisi terminasi. Setelah kondisi tersebut tercapai, kontribusi cahaya yang diperoleh pada simpul terdalam akan dikembalikan secara bertahap menuju simpul induknya hingga akhirnya mencapai kamera. Proses pengembalian nilai inilah yang menyerupai mekanisme runut-balik (*backtracking*).

Properti algoritma *backtracking* juga dapat didefinisikan dalam *path tracing*. Solusi akhir yang ingin diperoleh adalah warna suatu piksel, yaitu rata-rata kontribusi cahaya dari seluruh lintasan sampel yang dibangkitkan sekaligus sebagai ruang solusinya. Fungsi pembangkit direpresentasikan oleh mekanisme *sampling* Monte Carlo yang bertugas membangkitkan arah *ray* baru berdasarkan distribusi probabilitas tertentu. Sementara itu, fungsi pembatas (*bounding function*) digunakan untuk menentukan apakah suatu lintasan masih perlu dieksplorasi lebih lanjut. Apabila *ray* telah mencapai batas kedalaman maksimum, tidak lagi berinteraksi dengan objek, atau memenuhi kondisi terminasi lainnya, maka lintasan tersebut dihentikan dan proses runut-balik dilakukan untuk mengembalikan kontribusi cahaya menuju simpul induknya.

Mengenai kompleksitas waktunya, apabila seluruh kemungkinan arah pantulan pada *Rendering Equation* dievaluasi secara eksplisit, maka ruang pencarian akan membentuk pohon dengan faktor percabangan rata-rata *b* dan kedalaman maksimum *d* sehingga kompleksitas waktunya bersifat eksponensial, yaitu $O(b^d)$. Namun, pada *path tracing* yang sudah dioptimasi teknik Monte Carlo hanya dipilih satu arah pantulan yang dieksplorasi pada setiap titik tumbukan sehingga setiap sampel *ray* membentuk lintasan linear dengan kompleksitas *O(d)*. Jika terdapat *N* sampel untuk setiap piksel, maka kompleksitas waktunya menjadi *O(Nd)*.

IV. IMPLEMENTASI PROGRAM

Pada bagian ini, akan dibuat suatu program sederhana yang dapat menunjukkan implementasi DFS dan *backtracking* dalam *path tracing*. Namun, terlebih dahulu perlu ditekankan bahwa implementasi ini hanya berupa *path tracer* sederhana yang fokus utamanya adalah menyimulasikan lintasan cahaya dan bukan untuk mengkalkulasikan warna.

Program yang diimplementasikan mencoba untuk mereplikasi skenario seperti pada Gambar 8. Agar program tetap sederhana, maka tidak didefinisikan informasi posisi (geometris) pada *scene* begitu juga dengan distribusi probabilistik dalam menentukan arah, sebagai gantinya digunakan *randomiser* sederhana. Selain itu, untuk mengingat simpul yang sudah dilalui, akan digunakan struktur data *stack*. Mula-mula deklarasikan batas kedalaman, yang digunakan dalam program ini adalah 2.

```
// Parameter batas kedalaman
const int MAX_DEPTH = 2;
```

Gambar 11. Batas kedalaman, diambil dari [12]

Selanjutnya buat fungsi *trace* yang mengembalikan *float* berupa perhitungan cahaya dan memiliki parameter posisi, kedalaman, dan *stack* perekam. Definisikan juga basis rekursif, dalam hal ini kondisi terminasi, yaitu apabila sudah mencapai kedalaman maksimum (tidak bertemu sumber cahaya) atau bertemu sumber cahaya di perjalanannya.

```
float trace(string posisi, int depth, vector<string>& pathStack) {
    // Push ke dalam stack
    pathStack.push_back(posisi);

    bool isLeaf = false;
    float return_value = 0.0f;

    // Evaluasi kondisi
    if (depth >= MAX_DEPTH) {
        // Tidak sampai sumber cahaya, asumsikan gelap total
        isLeaf = true;
        return_value = 10.0f;
    } else if (posisi == "Lampu Ujung Lorong") {
        // Mencapai sumber cahaya, kembalikan cahaya 100%
        isLeaf = true;
        return_value = 100.0f;
    }
```

Gambar 12. Basis rekursif, diambil dari [12]

Apabila belum memenuhi basis, akan dilanjutkan penentuan arah secara acak (stokastik) menggunakan *randomiser* sederhana, lalu arah tersebut akan lanjut ditelusuri secara rekursif sesuai dengan prinsip DFS. Digunakan juga asumsi pelemahan cahaya apabila merupakan hasil pantulan dari objek lain.

```
// Teknik Monte Carlo dengan randomiser sederhana
int dadu = rand() % 100;
string next_node;

if (dadu < 40) next_node = "Tembok Kanan";
else if (dadu < 80) next_node = "Tembok Kiri";
else next_node = "Lampu Ujung Lorong";

// Pemanggilan rekursif
float incoming_light = trace(next_node, depth + 1, pathStack);

// Backtracking, asumsikan cahaya diserap 50%
float brdf = 0.5f;
float result = brdf * incoming_light;
```

Gambar 13. Rekursif, diambil dari [12]

Fungsi *trace* tersebut akan dipanggil di program utama. Program utama akan meminta jumlah *sampel* yang diinginkan (*N*). Kemudian, untuk setiap sampel akan dijalankan fungsi *trace*. Untuk kemudahan implementasi tidak dilakukan *pixel sampling* sehingga titik dan arah awalnya selalu sama (*hard coded*), yaitu ke "Tembok Kiri". Fungsi *trace* dari setiap sampel mengembalikan nilai yang akan diakumulasikan.

```
int N;
cout << "=== SIMULATOR PATH TRACING PIKSEL ===\n";
cout << "Masukkan jumlah sampel sinar (N): ";
cin >> N;

float total_akumulasi = 0.0f;
vector<string> memoryStack; // Vektor untuk menyimpan urutan DFS

for (int i = 1; i <= N; i++) {
    cout << "\n===== SAMPLE " << i << " =====\n\n";

    // Piksel awal selalu menembak ke Tembok Kiri terlebih dahulu
    float hasil_sampel = trace("Tembok Kiri", 0, memoryStack);

    // Kembali ke Kamera
    cout << "↑\n";
    cout << "Camera : " << hasil_sampel << "\n";

    total_akumulasi += hasil_sampel;
    cout << "\n[HASIL SAMPLE " << i << "] = " << hasil_sampel << "\n";
}
```

Gambar 14. Program utama, diambil dari [12]

Berikut adalah contoh *output* yang diperoleh,

```
=== SIMULATOR PATH TRACING PIKSEL ===
Masukkan jumlah sampel sinar (N): 2

===== SAMPLE 1 =====

PATH :
Camera
 └─ Tembok Kiri
      └─ Lampu Ujung Lorong

RETURN :
Lampu Ujung Lorong : 100
Tembok Kiri (depth=0) : 50
↑
Camera : 50

[HASIL SAMPLE 1] = 50

===== SAMPLE 2 =====

PATH :
Camera
 └─ Tembok Kiri
      └─ Tembok Kiri
           └─ Lampu Ujung Lorong (Cut-off)

RETURN :
Lampu Ujung Lorong : 10
Tembok Kiri (depth=1) : 5
Tembok Kiri (depth=0) : 2.5
↑
Camera : 2.5

[HASIL SAMPLE 2] = 2.5

====================================
HASIL RENDERING AKHIR
Jumlah Sampel (N) : 2
Warna Piksel      : 26.25
====================================
```

Gambar 14. Output N = 2, diambil dari [12]

```
====================================
HASIL RENDERING AKHIR
Jumlah Sampel (N) : 1000
Warna Piksel      : 12.4275
====================================
```

Gambar 15. Output N = 1000, diambil dari [12]

Berdasarkan *output* yang diperoleh, program yang dibuat berhasil menggambarkan penerapan algoritma DFS dan *backtracking* dalam *path tracing*. Selain itu, terbukti juga bahwa semakin banyak sampel, maka akan diperoleh hasil yang semakin konsisten.

V. KESIMPULAN

Path tracing merupakan teknik *rendering* berbasis fisika yang berupaya memperoleh solusi dari *Rendering Equation* untuk menyimulasikan pencahayaan global (*global illumination*) secara realistis. Berbeda dengan *ray tracing* klasik yang umumnya hanya mempertimbangkan interaksi cahaya yang bersifat deterministik seperti refleksi dan refraksi, *path tracing* turut memperhitungkan pencahayaan tidak langsung melalui proses pengambilan sampel arah cahaya secara stokastik. Pendekatan ini memungkinkan simulasi *transport* cahaya yang lebih mendekati fenomena di dunia nyata.

Namun, *Rendering Equation* memiliki bentuk integral multidimensi yang bersifat rekursif sehingga sangat kompleks untuk dihitung secara eksak. Teknik Monte Carlo menjadi solusi untuk mengatasi permasalahan tersebut dengan menggantikan evaluasi seluruh kemungkinan lintasan cahaya menjadi estimasi berbasis sejumlah sampel acak. Melalui pendekatan ini, ruang pencarian yang dapat berkembang secara eksponensial direduksi menjadi sekumpulan lintasan linear yang dapat dihitung secara efisien.

Dari sudut pandang algoritma, *path tracing* dapat dipandang sebagai penerapan algoritma *Depth-First Search* (DFS) dan runut-balik (*backtracking*). DFS digunakan untuk menelusuri satu lintasan cahaya hingga mencapai kondisi terminasi, sedangkan *backtracking* digunakan untuk mengembalikan dan mengakumulasikan kontribusi cahaya dari titik terdalam menuju kamera sehingga diperoleh warna akhir suatu piksel.

VI. UCAPAN TERIMA KASIH

Puji syukur saya panjatkan kepada Tuhan Yang Maha Esa atas segala rahmat dan karunia-Nya, sehingga saya dapat menyelesaikan makalah ini dengan baik. Saya juga ingin mengucapkan terima kasih kepada pihak-pihak yang telah memberikan dukungan, bantuan, dan motivasi selama proses penulisan makalah ini. Ucapan terima kasih saya sampaikan kepada:

1. Dr. Ir. Rinaldi, M.T. selaku dosen pengajar yang telah memberikan arahan dan referensi dalam mengerjakan makalah ini.
2. Orang tua saya yang selalu memberikan doa, semangat, dan dukungan moral.
3. Semua pihak yang tidak bisa saya sebutkan satu per satu, yang telah membantu baik secara langsung maupun tidak langsung.

Saya juga mengakui menggunakan generative AI dalam penyusunan makalah ini, dalam hal pengecekan tata bahasa, diskusi serta memperdalam pemahaman terhadap materi, dan implementasi program.

Semoga makalah ini dapat memberikan manfaat dan kontribusi yang positif bagi pembaca. Saya menyadari bahwa makalah ini masih jauh dari sempurna, oleh karena itu saran dan kritik yang membangun sangat saya harapkan.

VII. LAMPIRAN

Kode program secara lengkap dapat dilihat pada link github berikut: https://github.com/MikhaelYonatan/Path-Tracing.git

REFERENSI

[1] Yonatan, Mikhael. (2025). Analisis Pendekatan Vektor Euclidean dan Aljabar Geometri dalam Ray Tracing. 13 Juni 2026, dari https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2025-2026/MakalahAlgeo2025.htm

[2] NVIDIA. (n.d.). What is Path Tracing?. 13 Juni 2026, dari https://blogs.nvidia.com/blog/what-is-path-tracing/

[3] Kajiya, J. T. (1986). The Rendering Equation. 16 Juni 2026, dari https://www.cs.cmu.edu/afs/cs/academic/class/15462-s13/www/lec_slides/86kajiyaRenderingEquation.pdf

[4] Hanrahan, P. (n.d.). Monte Carlo Path Tracing. 16 Juni 2026, dari http://www.graphics.stanford.edu/courses/cs348b-01/course29.hanrahan.pdf

[5] YouTube (Computerphile). (2024). How Path Tracing Makes Computer Graphics Look Awesome - Computerphile. 16 Juni 2026, dari https://youtu.be/3OKj0SQ_UTw?si=uSH8cslyRdhpoQAO

[6] YouTube (IBM Technology). (2022). What is Monte Carlo Simulation?. 17 Juni 2026, dari https://youtu.be/7TqhmX92P6U?si=UjpURjKaZolcuaHY

[7] YouTube (Driviest). (2025). F1 25 Path Tracing vs Ray Tracing – Which Looks Better in Gameplay?. 18 Juni 2026, dari https://www.youtube.com/watch?v=3Y4HIbwN30U

[8] Munir, Rinaldi., & Maulidevi, N. U. (2026). Breadth/Depth First Search (BFS/DFS) (Bagian 1). 18 Juni 2026, dari https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2025-2026/13-BFS-DFS-(2026)-Bagian1.pdf

[9] Munir, Rinaldi., & Maulidevi, N. U. (2026). Breadth/Depth First Search (BFS/DFS) (Bagian 2). 18 Juni 2026, dari https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2025-2026/14-BFS-DFS-(2026)-Bagian2.pdf

[10] Munir, Rinaldi. (2026). Algoritma Runut-balik (Backtracking) (Bagian 1). 18 Juni 2026, dari https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2025-2026/15-Algoritma-backtracking-(2026)-Bagian1.pdf

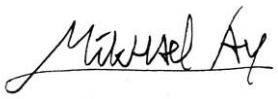
[11] Florida State University. (n.d.). Specular and Diffuse Reflection. 18 Juni 2026, dari https://micro.magnet.fsu.edu/primer/java/reflection/specular/

[12] Dokumen penulis

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 19 Juni 2026



Mikhael Andrian Yonatan
13524051