Aplikasi Algoritma Traversal Dalam *Binary Space Partitioning*

Pudy Prima (13508047)

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
if18047@students.if.itb.ac.id

Abstract*—Makalah ini membahas tentang aplikasi algoritma traversal dalam Binary Space Partitioning *(BSP), yaitu salah satu metode yang banyak digunakan dalam* computer grapics *untuk melakukan penanganan area yang terdiri atas beberapa poligon, sedemikian sehingga setiap poligon dapat ditangani terpisah dari poligon-poligon lain dalam area.* Binary Space Partitioning *dapat direpresentasikan dengan suatu struktur pohon yang biasa disebut* Binary Space Partitioning *Tree (BSP Tree). Algoritma traversal, yang terdiri atas algoritma pencarian mendalam (Depth First Search/DFS) dan algoritma pencarian melebar (Breadth First Search/BFS) dapat diterapkan dalam proses pencarian poligon dalam suatu objek yang ditangani dengan BSP. Jenis algoritma traversal yang akan dibahas dalam makalah ini adalah Depth First Search (DFS) secara spesifik, dan tidak membahas Breadth First Search (BFS).

Index Terms*—algoritma pencarian mendalam, algoritma traversal,** Binary Space Partitioning *(BSP), Depth First Search (DFS)

I. PENDAHULUAN

Salah satu konsep yang banyak digunakan dalam *computer graphics* adalah *Binary Space Partitioning* (BSP). Konsep ini bertujuan untuk menangani objek yang terdiri atas beberapa poligon sehingga setiap poligon dalam objek tersebut dapat ditangani secara terpisah. Penanganan terpisah terhadap poligon dalam suatu objek diperlukan agar proses perubahan terhadap suatu poligon dapat dilakukan.

Jika perubahan terhadap suatu poligon hanya memperngaruhi bentuk poligon tersebut dan tidak menyinggung atau tidak ikut mempengaruhi poligon lain di dalam objek, maka proses pemilihan tidak perlu dilakukan. Namun jika ternyata proses perubahan suatu poligon mempengaruhi keberadaan (ukuran maupun posisi) poligon lain, maka diperlukan suatu metode pemilihan poligon, baik poligon yang akan diubah maupun poligon lain yang akan mendapat pengaruh dari proses perubahan poligon tadi.

Pada kenyataannya, implementasi BSP terhadap suatu objek direpresentasikan dalam suatu struktur berupa pohon biner, yang sering disebut *Binary Space Partitioning Tree* (BSP *Tree*). Dalam BSP *tree*, sama seperti pohon biner lainnya, setiap simpul hanya boleh memiliki paling banyak dua simpul cabang. Ini menggambarkan pembangunan BSP *tree* yang didasarkan pada konsep BSP yang selalu membagi area objek menjadi dua bagian. Dengan menggunakan struktur BSP *tree* ini, proses pemilihan objek akan lebih mudah dilakukan, yaitu dengan memanfaatkan algoritma traversal, baik algoritma pencarian mendalam (DFS) maupun algoritma pencarian melebar (BFS).

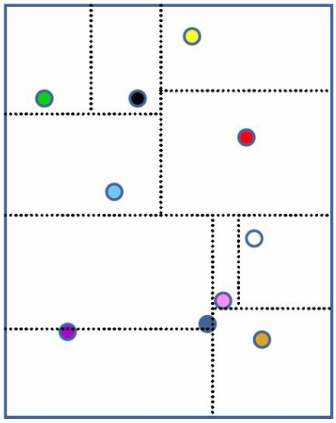
II. PEMBAHASAN

2.1. Binary Space Partitioning

Binary Space Partitioning (BSP) adalah salah satu metode yang digunakan untuk membagi suatu area yang terdiri dari beberapa poligon terpisah untuk mendapatkan bentuk yang diinginkan. Cara pembagian yang spesifik berbeda-beda, tergantung tujuan yang ingin dicapai. Tujuan yang banyak diinginkan adalah membagi area menjadi beberapa bagian yang belum tentu sama besar, namun hanya memiliki satu poligon di setiap bagiannya. BSP dilakukan dengan cara rekursif, dan melakukan pembagian area menjadi dua bagian dalam setiap kali pembagiannya. Sesuai namanya, *Binary Space Partitioning* selalu melakukan pembagian terhadap area suatu objek menjadi dua bagian.

Aplikasi BSP banyak ditemukan di bidang *computer graphics*, baik *two dimensions* (2D) maupun *three dimensions* (3D). Tujuannya adalah untuk membagi suatu objek menjadi objek-objek yang lebih kecil sehingga memudahkan pemrosesan setiap objek kecil tersebut, untuk melakukan modifikasi terhadap objek besar. Contoh modifikasi yang dapat dilakukan misalnya pewarnaan sebagian objek, penghapusan salah satu objek kecil,

pengubahan ukuran salah satu objek kecil, serta penyisipan objek baru di antara beberapa objek. Dalam modifikasi berupa penghapusan, pengubahan ukuran, maupun penambahan objek baru akan mempengaruhi keberadaan objek-objek kecil lainnya. Pendekatan BSP ini menangani perubahan yang dikenakan terhadap objek kecil dengan tetap menjaga keutuhan objek besar tersebut.



Gambar 1 Contoh hasil pembagian suatu objek menjadi objek-objek yang lebih kecil

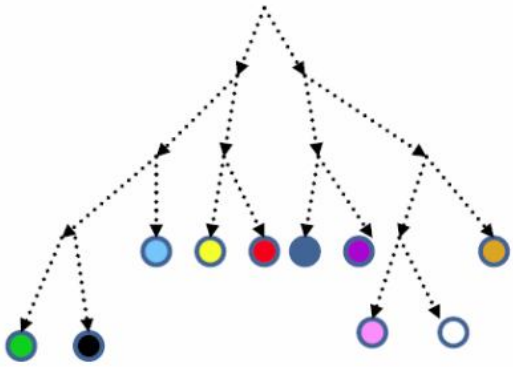
Pembagian suatu objek utuh dilakukan dengan membagi objek utuh tersebut menjadi dua bagian yang tidak harus sama besar. Konsep utamanya adalah melakukan pembagian objek menjadi tepat dua bagian, dan pembagian tidak bersinggungan dengan poligon yang terdapat di dalam objek. Bila pembagian yang dilakukan ternyata menyinggung atau menabrak poligon di dalam objek, maka pembagian harus diulang dengan cara lain. Selanjutnya, secara rekursif pembagian dilakukan terhadap objek-objek kecil yang di dalamnya masih terdapat lebih dari satu poligon. Pembagian dihentikan ketika setiap bagian hanya berisi satu poligon.

2.2. Binary Space Partitioning Tree

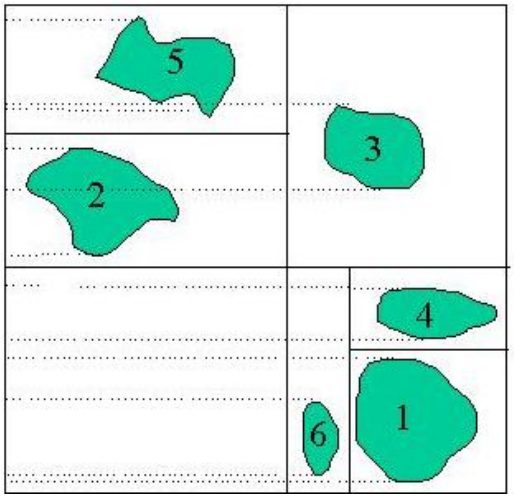
Seperti yang telah dipaparkan di atas, *Binary Space Partitioning* selalu melakukan pembagian terhadap suatu objek menjadi dua bagian objek yang lebih kecil. Di sini, konsep *binary tree* (pohon biner) dapat diterapkan. *Binary Space Partitioning Tree* (BSP *Tree*) adalah suatu struktur yang merepresentasikan pembagian suatu ruang atau area secara hierarki dan rekursif menjadi ruang-ruang yang lebih kecil.

Pembangunan BSP *Tree* mengikuti konsep BSP. Misalnya terdapat sejumlah poligon di dalam suatu area. Langkah-langkah pembangunan suatu BSP *Tree* adalah sebagai berikut.

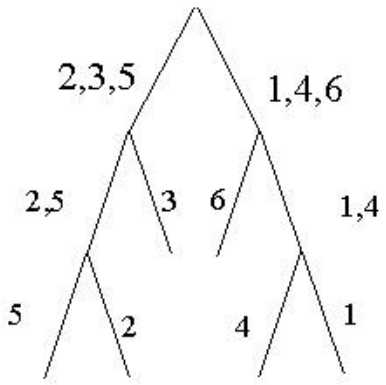
a. Buat satu simpul sebagai akar pohon.
   Akar pohon ini merepresentasikan suatu objek besar yang utuh yang akan dibagi ke dalam objek-objek kecil.
b. Pilih area yang akan dibagi.
   Dalam memilih area yang akan dibagi, misalnya pembagian area dua dimensi direpresentasikan dengan garis, pastikan garis pembagi tidak menyinggung poligon yang ada di dalam objek.
c. Lakukan pembagian terhadap area yang telah dipilih.
   Dengan melakukan pembagian, maka akar pohon yang dibuat di awal melahirkan dua simpul baru, yang masing-masing merupakan representasi area baru yang tercipta dari hasil pembagian.
d. Lakukan secara rekursif langkah (b) dan (c).
   Rekursif dilakukan terhadap simpul-simpul yang baru terbentuk jika di dalam area hasil pembagian tersebut masih terdapat lebih dari satu poligon.



Gambar 2 Hasil pembangunan BSP tree dari objek pada Gambar 1



Gambar 3 Contoh lain Binary Space Partitioning



Gambar 4 BSP Tree dari BSP pada Gambar 3

Dari Gambar 3 dan Gambar 4, kita dapat melihat urutan langkah pembangunan BSP *tree* dengan jelas. Pada awalnya, hanya ada satu area luas yang terdiri atas enam poligon, dan di BSP *tree* hanya terdapat satu simpul sebagai simpul akar. Pembagian area tersebut kemudian dilakukan secara horizontal, membagi area menjadi dua bagian. Bagian pertama terdiri atas poligon 2, 3, dan 5 (atas), sedangkan bagian kedua berisi poligon 1, 4, dan 6 (bawah). Untuk setiap bagian, pembagian diulangi kembali, karena setiap bagian masih memiliki lebih dari satu poligon. Untuk area bagian atas, pembagian selanjutnya dilakukan secara vertikal, yaitu membagi area menjadi dua bagian, dengan bagian pertama (kiri) terdiri atas poligon 2 dan 5, sedangkan bagian kedua (kanan) terdiri atas poligon 3. Karena bagian kiri masih terdiri atas dua poligon, pembagian diulangi kembali, kali ini secara horizontal dan memperoleh dua bagian, masing-masing terdiri atas poligon 5 (atas) dan poligon 2 (bawah). Bagian kiri tidak perlu dibagi lagi karena di dalamnya hanya terdapat poligon 3. Pembagian kemudian diulangi kembali terhadap area bagian bawah yang masih terdiri atas poligon 1, 4, dan 6. Pembagian untuk bagian ini dilakukan secara vertikal, membagi area menjadi bagian kiri (hanya terdiri atas poligon 6) dan bagian kanan (terdiri atas poligon 1 dan 4). Karena bagian kiri masih terdiri dari dua poligon, pembagian dilakukan kembali, yaitu secara horizontal, sehingga poligon 4 berada di bagian atas dan poligon 1 berada di bagian bawah. Kini, setiap bagian hanya memiliki tepat satu poligon. Representasi BSP *tree* dari proses tersebut dapat dilihat pada Gambar 4. Setiap simpul dalam BSP *tree* tersebut bertambah ketika terjadi pembagian daerah.

2.3. Algoritma Traversal

Algoritma traversal adalah suatu langkah pencarian solusi yang dilakukan dengan mengunjungi kandidat-kandidat solusi secara sistematik hingga berhasil menemukan solusi yang diinginkan. Algoritma traversal biasanya diterapkan pada penyelesaian masalah yang direpresentasikan dengan graf, yaitu dengan mengunjungi simpul-simpul pada graf berdasarkan urutan tertentu (secara sistematik). Terdapat dua buah algoritma yang termasuk dalam algoritma traversal, yaitu Algoritma Pencarian Melebar (*Breadth First Search* atau BFS) dan Algoritma Pencarian Mendalam (*Depth First Search* atau DFS). Baik BFS maupun DFS dapat digunakan untuk menyelesaikan masalah pencarian dengan cukup baik. Namun, kriteria setiap algoritma tersebut berbeda sehingga menyebabkan perbedaan performansi di antara keduanya.

Algoritma Pencarian Melebar (BFS) adalah suatu algoritma pencarian pada graf yang dimulai dengan mengunjungi sebuah simpul pada graf tersebut, dan mengunjungi setiap simpul yang bertetangga dengan simpul pertama tadi berurutan dari yang terdekat seterusnya hingga ditemukan solusi yang diinginkan. Pada algoritma ini, dikenal istilah laras atau tingkatan. Setiap simpul yang dibangkitkan secara bersamaan akan berada pada laras yang sama. Jika dimodelkan dalam bentuk pohon, maka semua simpul pada laras d akan dikunjungi terlebih dahulu sebelum simpul-simpul pada laras $d + 1$ dikunjungi.

Pencarian mendalam (DFS) adalah suatu algoritma pencarian pada graf yang dimulai dengan mengunjungi sebuah simpul pada graf tersebut, kemudian mengunjungi simpul terdekat yang bertetangga dengan simpul pertama tadi seterusnya hingga tujuan berhasil ditemukan atau melakukan runut balik (*backtrack*) untuk mencari jalur pencarian lain jika tujuan tidak berhasil ditemukan. *Backtrack* dilakukan hingga titik percabangan yang terakhir dilewati, kemudian penelusuran dilakukan melalui percabangan yang belum pernah dilewatinya. Hal ini dilakukan secara rekursif dan baru berhenti jika solusi ditemukan atau semua simpul telah dikunjungi.

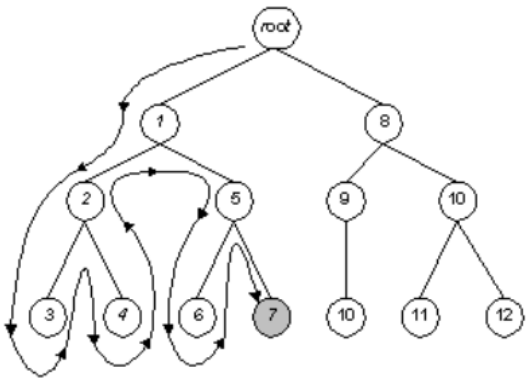
2.4. Algoritma Traversal Dalam BSP *Tree*

Modifikasi mungkin dilakukan terhadap suatu area yang telah terbagi menjadi area-area kecil yang terdiri atas sebuah poligon. Modifikasi tersebut antara lain mengubah ukuran, menghapus, serta menambah objek baru di dalam objek yang telah ada. Modifikasi-modifikasi yang dikenakan terhadap salah satu objek kecil di dalam objek besar mampu memberikan pengaruh terhadap objek-onjek lain di sekitarnya. Misalnya pada penyisipan objek baru di sebelah objek lama. Penyisipan objek baru

ini akan memberi pengaruh minimal terhadap objek lama tersebut, seperti pergeseran letak objek lama tadi jika objek baru yang disisipkan tidak cukup kecil menempati tempat yang tersedia. Tidak hanya itu, penyisipan objek baru mungkin membawa pengaruh juga terhadap objek-objek lain di sekitar lokasi objek baru tersebut akan disimpan.

Primitif-primitif utama yang digunakan dalam modifikasi ini di antaranya adalah memilih objek, membuat objek, mengubah objek, serta membuang/menghapus objek. Untuk memudahkan penyampaian, kita dapat menyebut simpul pada pohon Gambar 2 sebagai simpul 1 hingga simpul 10, berurutan dari simpul paling kiri. Pada primitif memilih objek, objek yang dipilih adalah objek yang akan dikenai perubahan. Misalnya kita ingin menambah objek baru di sebelah barat objek 6. Dengan hanya melihat, kita bisa dengan jelas mengetahui posisi objek 6. Namun tidak demikian halnya bagi komputer. Komputer tidak dapat melihat dan menentukan posisi objek tersebut selama tidak ada instruksi yang dijalankan padanya. Untuk menyisipkan objek baru di sebelah barat objek 6, maka pertama kali komputer perlu memilih objek 6. Pemilihan ini dilakukan dengan mencari objek tersebut di dalam objek besar, yaitu dengan memanfaatkan algoritma pencarian secara traversal. Algoritma traversal digunakan dalam kasus ini karena representasi yang digunakan adalah berbentuk struktur pohon, yaitu pohon biner.

Dalam BSP, pencarian dapat dilakukan terhadap struktur yang telah dimiliki, yaitu struktur BSP *Tree*. Algoritma traversal yang digunakan kali ini adalah algoritma Pencarian mendalam (*Depth First Search*/DFS). DFS melakukan pencarian secara traversal mulai dari akar pohon, lalu dilanjutkan ke salah satu simpul yang merupakan cabang dari akar, misalnya simpul yang ada di bagian kiri. Pencarian diteruskan terhadap cabang sebelah kiri dari simpul yang kita singgahi barusan. Jika ternyata simpul yang kita singgahi tidak memiliki simpul cabang di bagian kiri, maka kita dapat memilih memeriksa di simpul cabang bagian kanan. Pencarian terus dilakukan hingga seluruh bagian kiri suatu pohon telah dikunjungi atau simpul yang dicari telah ditemukan. Bila pencarian tidak ditemukan padahal seluruh bagian kiri pohon telah dikunjungi, maka runut balik (*backtracking*) dilakukan. Runut balik berarti pergerakan simpul kembali ke simpul ayah yang pernah dikunjungi sebelumnya. Setelah melakukan runut balik, di setiap simpul yang kita kunjungi lagi, kita dapat mengunjungi simpul anak dari simpul tadi yang belum kita kunjungi jika ada. Jika ternyata semua simpul anak dari simpul tadi telah kita kunjungi semua, makan kita lakukan runut balik menuju simpul lain di atasnya, begitu seterusnya hingga semua simpul dalam pohon berhasil dikunjungi atau simpul yang dicari berhasil ditemukan. Jika semua simpul dalam pohon sudah dikunjungi namun belum berhasil menemukan simpul yang dicari, itu berarti bahwa simpul yang dicari tidak ada di dalam pohon.



Gambar 5 Simulasi jalur pencarian simpul dengan Depth First Search (DFS)

Dari Gambar 5 di atas, kita dapat melihat simulasi pencarian simpul 7 di dalam pohon biner menggunakan metode pencarian DFS. Pencarian dilakukan terhadap bagian kiri pohon terlebih dahulu. Ketika sampai di simpul 3, *backtracking* dilakukan untuk melanjutkan pencarian di bagian kanan (simpul 4). Ketika ternyata simpul 4 bukan simpul yang dicari, *backtracking* dilakukan kembali hingga simpul 1, dan dilanjutkan pencarian di simpul bagian kanan, yaitu simpul 5. Ketika simpul yang dicari berhasil ditermukan di simpul 7, pencarian pun dihentikan.

Secara garis besar, algoritma traversal di dalam BSP ini memang hanya digunakan untuk mencari atau memilih poligon yang ada di dalam struktur BSP *tree*. Pengembangan dari algoritma traversal dalam BSP ini adalah dalam pencarian poligon lain yang akan ikut terpengaruh oleh proses perubahan yang dikenakan terhadap suatu poligon. Misalnya kita ingin melakukan perubahan terhadap bagian utara suatu objek. Jika dilihat langsung, kita bisa mengetahui objek lain yang berada di sebelah utara objek yang akan dikenai perubahan tadi yang mungkin akan ikut terpengaruh oleh adanya perubahan tersebut. Namun, secara fungsi atau prosedur, kita memerlukan algoritma pencariannya.

Untuk kasus tersebut, algoritma DFS masih dapat digunakan. Perubahan yang mungkin dilakukan adalah perubahan terhadap struktur BSP *tree* yang digunakan. Untuk setiap simpul yang memiliki cabang, perlu ditambah suatu status yang menyatakan cara perolehan cabang yang dimiliki. Selain itu, proses pembentukan simpul harus

dilakukan dengan mekanisme tertentu, seperti pembagian horizontal yang menghasilkan area atas dan area bawah harus menyimpan area atas sebagai simpul di bagian kiri dan area bawah sebagai simpul di bagian kanan. Cara perolehan cabang ada dua, yaitu dengan pembagian horizontal atau dengan pembagian vertikal. Dengan status tersebut, algoritma DFS dapat digunakan untuk mencari poligon berdasarkan posisinya terhadap poligon lain di dalam objek. Misalnya, untuk simpul A yang bersaudara dengan simpul B, jika diketahui simpul ayah keduanya memiliki status horizontal (artinya pembagian menjadi atas dan bawah) dan A adalah simpul kiri sedangkan B simpul kanan, maka dapat diambil kesimpulan bahwa A adalah poligon yang terletak di sebelah utara B (jika atas dapat dianggap sebagai utara).

III. KESIMPULAN

Binary Space Partitioning (BSP) merupakan metode pembagian suatu area, baik dua dimensi maupun tiga dimensi, ke dalam area-area yang lebih kecil. Hasil akhir pembagian ini bermacam-macam, tergantung tujuan yang ingin dicapai. Pada makalah ini, pembagian yang dilakukan bertujuan untuk membagia area sehingga setiap poligon yang ada di area tersebut berada di tepat satu bagian. Hal ini bertujuan untuk memudahkan pemrosesan setiap poligon yang ada di area tersebut, misalnya ketika akan dilakukan perubahan, penghapusam, maupun penambahan poligon. Aplikasi BSP banyak ditemukan di dalam *computer graphics*, terutama dalam penanganan dimensi ruang. Struktur BSP berupa pohon biner yang biasa disebut *Binary Space Partitioning Tree* (BSP *Tree*). Setiap pembagian area objek menjadi dua bagian menghasilkan pembangunan dua simpul cabang untuk suatu simpul. Pada akhirnya, setiap simpul daun pada BSP *tree* merepresentasikan setiap poligon tunggal yang berada di dalam suatu objek besar.

Algoritma traversal adalah suatu langkah pencarian solusi yang dilakukan dengan mengunjungi kandidat-kandidat solusi secara sistematik hingga berhasil menemukan solusi yang diinginkan. Algoritma traversal terdiri atas dua jenis, yaitu algoritma pencarian mendalam (*Depth First Search*/DFS) dan algoritma pencarian melebar (*Breadth First Search*/BFS).

Algoritma traversal dapat diterapkan terhadap suatu objek yang ditangani dengan BSP. Algoritma traversal yang dibahas kali ini adalah algoritma pencarian mendalam (*Depth First Search*/DFS). Dalam DSP, algoritma DFS digunakan untuk mencari keberadaan poligon di dalam objek, yaitu dengan mencari simpul yang merepresentasikan poligon tadi di dalam BSP *tree*.

Pengembangan dari penerapan algoritma DFS ini adalah untuk mencari poligon-poligon lain yang menjadi tetangga suatu poligon tertentu. Pengembangan ini memerlukan tambahan terhadap struktur BSP *tree*. Pada setiap simpul yang memiliki simpul cabang di BSP *tree* perlu ditambah status horizontal atau vertikal yang menyatakan pembagian area yang dilakukan sehingga melahirkan simpul anak. Selain itu, mekanisme pembangunan simpul juga perlu diatur, yaitu dengan menyimpan area yang ada di bagian atas atau kiri setelah pembagian sebagai simpul kiri, serta area yang ada di bagian bawah dan kanan setelah pembagian sebagai simpul kanan. Pengaturan ini dapat dibuat berbeda tergantung persepsi yang digunakan.

DAFTAR PUSTAKA

[1] Munir, Rinaldi. 2009. *Diktat Kuliah IF3051 Strategi Algoritma*. Program Studi Teknik Informatika, Sekolah Tinggi Teknik Elektro dan Informatika, Institut Teknologi Bandung.

[2] http://www.algolist.net/Algorithms/Graph/Undirected/Depth-first_search

[3] http://en.wikipedia.org/wiki/Binary_space_partitioning

[4] http://web.cs.wpi.edu/~matt/courses/cs563/talks/bsp/document.html

[5] http://www.devmaster.net/articles/bsp-*trees*/

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, 9 Desember 2010

Pudy Prima
13508047